

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ
УНІВЕРСИТЕТ

Сергій ОНИЩЕНКО

НАВЧАЛЬНИЙ ПОСІБНИК

WEB-ТЕХНОЛОГІЇ:
навчально-методичний комплекс

*Рекомендовано Радою
Бердянського державного педагогічного університету
Протокол № 17 від 30.06.2016 р.*

Бердянськ
«БДПУ»
2016

УДК 004.738.5(075.8)

ББК 32.973.202 я73

О - 58

*Рекомендовано Радою
Бердянського державного педагогічного університету
Протокол № 17 від 30.06.2016 р.*

Рецензенти:

Ткачук С. І. – доктор педагогічних наук, професор, декан технологічного-педагогічного факультету Уманського державного педагогічного університету.

Цина А. Ю. – доктор педагогічних наук, професор, завідувач кафедри виробничо-інформаційних технологій та безпеки життєдіяльності факультету технологій та дизайну Полтавського національного педагогічного університету імені В. Г. Короленка.

Жигір В. І. – доктор педагогічних наук, професор, завідувач кафедри професійної освіти Бердянського державного педагогічного університету.

ISBN: 978-617-7452-04-0

Онищенко С. В.

О - WEB-технології : навч.-метод. комплекс / Сергій Онищенко. –
58 Бердянськ : "БДПУ", 2016. – 500 с.

У навчальному посібнику подані: навчальна програма, лекції, рекомендації до виконання лабораторних робіт, рекомендації до виконання самостійної роботи, пакет ККР, РКР та програма екзамену з дисципліни "WEB-технології". Призначений для підготовки студентів спеціальності 6.010104 – професійна освіта денної форми навчання.

УДК 004.738.5(075.8)

ББК 32.973.202 я73

ISBN: 978-617-7452-04-0

© С.В.Онищенко, 2016

© Бердянськ «БДПУ», 2016

ВСТУП

WEB-технології є концепція роботи з інформацією. Вона відрізняється наступними особливостями:

- технічна основа веб-технологій – локальні і глобальні мережі, часто Інтернет;
- застосування особливого типу тонких клієнтів: веб-браузерів (типи та історія, сучасний стан браузерів віддаються на самостійне вивчення);
- переважно текстова і статично-графічна подача інформації (ослаблення цієї тенденції пов'язано з розвитком технологій зв'язку і ПЗ, експансією медіаконтенту);
- зміни в інформаційних джерелах миттєво відображаються в публікаціях;
- число споживачів інформації практично не обмежена. Публікатор сам може задати особливі умови на доступ до інформації, що публікується;
- в публікаціях можуть міститися посилання на інші публікації без обмеження на розташування і джерела матеріалів;
- активна робота пошукових машин (історія, сучасний стан та роль пошукових машин віддаються на самостійне вивчення);
- доставка і тиражування контенту практично безкоштовні.

Привабливість WEB-технологій як засобу доставки інформації багато в чому визначає універсальний інтерфейс між людиною і комп'ютером. Кожній людині зрозумілі написи, заголовки, посилання, картинки. WEB-інтерфейс як засіб доступу до інформації інтуїтивно зрозумілий. Наслідком простоти WEB-інтерфейсу є широко використання Інтернету як каналу комунікації. Браузер – програма для перегляду WEB-сторінок і роботи з інформацією в WEB-інтерфейсі. Браузери – програми,

якими забезпечені всі сучасні комп'ютери. Теоретично всі браузері повинні відображати всі сайти, зроблені за стандартами, однаково. Практично є безліч тонкощів і складнощів. Найбільш популярні браузері: Internet Explorer, Firefox, Opera, Safari, Chrome.

Значення WEB-технології, як для розробників програмного забезпечення, так і для звичайних користувачів, визначається тим, що це, перш за все – інтеграційна технологія. І важко знайти більш вдалий приклад того, як можна інтегрувати різні джерела інформації і різні її типи. WEB-технології дозволяють створювати прості для освоєння, легкодоступні, вкрай дешеві, швидко оновлювані інформаційні, діалогові, довідкові системи.

Розгляду принципів роботи в мережі, методики розробки WEB-сайтів, WEB-додатків та мобільних додатків присвячений даний навчальний посібник.

1. РОБОЧА ПРОГРАМА ДИСЦИПЛІНИ

Опис навчальної дисципліни

Найменування показників	Галузь знань, напрям підготовки, освітньо-кваліфікаційний рівень	Характеристика навчальної дисципліни	
		денна форма навчання	заочна форма навчання
Кількість кредитів – 3,5	Галузь знань <u>0101 Педагогічна освіта</u> (шифр і назва)	Нормативна (за вибором)	
	Напрямок підготовки <u>6.010104</u> <u>Професійна освіта</u> (шифр і назва)		
Модулів – 4	Спеціальність: <u>професійна освіта</u>	Рік підготовки	
Змістових модулів – 7		2-й	-й
Індивідуальне науково-дослідне завдання _____ (назва)		Семестр	
Загальна кількість годин – 105		4-й	-й
		Лекції	
Тижневих годин для денної форми навчання: аудиторних – 3 самостійної роботи студента – 5	Освітньо-кваліфікаційний рівень: <u>6.010101 бакалавр педагогічної освіти</u>	26 год.	год.
		Практичні, семінарські	
		год.	год.
		Лабораторні	
		26 год.	год.
		Самостійна робота	
		53 год.	год.
		Індивідуальні завдання:	
		год.	
		Вид контролю:	
екзамен			

Примітка.

Співвідношення кількості годин аудиторних занять до самостійної і індивідуальної роботи становить (%):

для денної форми навчання – 52:53.

1. Мета та завдання навчальної дисципліни

Метою викладання навчальної дисципліни «Web-технології» є ознайомлення студентів із сучасними Internet засобами, формування системи знань і навичок, необхідних для освоєння спеціальної області знань, пов'язаної з WEB-технологіями.

Основними завданнями вивчення дисципліни «Web-технології» є:

- навчити основним засобам, що використовуються для створення WEB-технологій;
- пояснити структуру побудови всесвітньої мережі Інтернет;
- ознайомити з принципами механізмів взаємодії складових мережі;
- ознайомити з принципами організації інформаційних баз даних у мережі Інтернет;
- ознайомити з принципами організації інформаційних баз даних у мережі Інтернет;
- розвивати логічне мислення у студентів тощо.

Основні результати навчання і компетентності згідно з вимогами освітньо-професійної (освітньо-наукової) програми:

№ з/п	Компетентності*	Результати навчання
1.	Загальні	Знати: структуру й принцип дії WEB-сторінок. Вміти: виконувати розробку базових клієнських та серверних сценаріїв та додатків.
2.	Фахові	Знати: структуру й принцип роботи серверних технологій. Вміти: створювати найпростіші статичні та динамічні сайти.

* На кожну дисципліну обсягом до 5 кредитів ЕКТС обирають 2-3 компетентності з освітньо-професійної програми (профілю програми), обсягом до 10 кредитів – 3-4 компетентності, але не більше 4.

2. Програма навчальної дисципліни

Змістовий модуль 1. Вступ до Web-технологій. Web-додатки.

Тема 1. Вступ до Web-технологій: структура і принципи Web.

Тема 2. Вступ до клієнт-сервісні технології Web. Протокол HTTP.

Тема 3. Клієнтські сценарії та додатки.

Тема 4. Серверні Web-додатки.

Змістовий модуль 2. Сценарії Perl і PHP.

Тема 5. Мови розробки сценаріїв Perl і PHP.

Тема 6. Вступ до C# і платформу Visual Studio.Net.

Змістовий модуль 3. Архітектура Web-додатків.

Тема 7. Архітектура Web-додатків ASP.NET. Розробка Web-додатків на платформі .NET.

Тема 8. Інтерфейси взаємодії Web-додатків з СУБД.

Змістовий модуль 4. Мова XML.

Тема 9. Вступ до XML.

Тема 10. Мови опису схем XML.

Тема 11. DOM XML. Перетворення XML документів.

Змістовий модуль 5. Робота з Web-контентом.

Тема 12. Інтеграція і взаємодія в мережі Web.

Тема 13. Організація процесу розробки Web-контента .CMS/CMF.

Тема 14. Синдикація та агрегування Web -контенту.

Змістовий модуль 6. Робота з Web-порталами та розробка Web-додатків.

Тема 15. Web-портали. Класифікація Web-порталів.

Тема 16. Вступ в технологію AJAX. Розробка мобільних Web-додатків.

Змістовий модуль 7. Мережа Web.

Тема 17. Вступ до Web 2.0.

Тема 18. Додатки для соціальних мереж.

3. Структура навчальної дисципліни

Назви змістових модулів і тем	Кількість годин											
	денна форма						Заочна форма					
	усього	у тому числі					усього	у тому числі				
		л	п	лаб	інд	с.р.		л	п	лаб	інд	с.р.
1	2	3	4	5	6	7	8	9	10	11	12	13
Модуль 1												
Змістовий модуль 1. Вступ до Web-технологій. Web-додатки.												
Тема 1. Вступ до Web-технологій: структура і принципи Web.	3	1		1		1						
Тема 2. Вступ до клієнт-сервісних технологій Web. Протокол HTTP.	4	1		1		2						
Тема 3. Клієнтські сценарії та додатки.	4	1		1		2						
Тема 4. Серверні Web-додатки.	4	1		1		2						
Разом за змістовим модулем 1	15	4		4		7						
Змістовий модуль 2. Сценарії Perl і PHP.												
Тема 1. Мови розробки сценаріїв Perl і PHP.	7	2		2		3						
Тема 2. Вступ до C# і платформи Visual Studio.Net.	8	2		2		4						
Разом за змістовим модулем 2	15	4		4		7						

Усього годин	30	8		8		14						
Модуль 2												
Змістовий модуль 3. Архітектура Web-додатків.												
Тема 1. Архітектура Web-додатків ASP.NET. Розробка Web- додатків на платформі .NET.	7	2		2		3						
Тема 2. Інтерфейси взаємодії Web- додатків з СУБД.	8	2		2		4						
Разом за змістовим модулем 3	15	4		4		7						
Змістовий модуль 4. Мова XML.												
Тема 1. Вступ до XML.	5	1		1		3						
Тема 2. Мови опису схем XML.	5	1		1		3						
Тема 3. DOM XML. Перетворення XML документів.	5	2		2		1						
Разом за змістовим модулем 4	15	4		4		7						
Усього годин	30	8		8		14						
Модуль 3												
Змістовий модуль 5. Робота з Web-контентом.												
Тема 1. Інтеграція і взаємодія в мережі Web.	5	2		2		1						
Тема 2. Організація процесу розробки Web-	5	1		1		3						

контента .CMS/CMF.												
Тема 3. Синдикація та аггрегування Web – контенту.	5	1		1		3						
Разом за змістовим модулем 5	15	4		4		7						
Змістовий модуль 6. Робота з Web-порталами та розробка Web-додатків.												
Тема 1. Web- портали. Класифікація Web-порталів.	7	2		2		3						
Тема 2. Вступ в технологію AJAX. Розробка мобільних Web-додатків.	8	2		2		4						
Разом за змістовим модулем 6	15	4		4		7						
Усього годин	30	8		8		14						
Модуль 4												
Змістовий модуль 7. Мережа Web.												
Тема 1. Вступ до Web 2.0.	7	1		1		5						
Тема 2. Додатки для соціальних мереж.	8	1		1		6						
Разом за змістовим модулем 7	15	2		2		11						
Усього годин	15	2		2		11						
Усього годин	105	26		26		53						

4. Теми лекцій

№ з/п	Назва теми	Кількість годин
		д/ф
1	Вступ до Web-технологій: структура і принципи Web.	1
2	Вступ до клієнт-сервісні технології Web. Протокол HTTP.	1
3	Клієнтські сценарії та додатки.	1
4	Серверні Web-додатки.	1
5	Мови розробки сценаріїв Perl і PHP.	2
6	Вступ до C# і платформу Visual Studio.Net.	2
7	Архітектура Web-додатків ASP.NET. Розробка Web-додатків на платформі .NET.	2
8	Інтерфейси взаємодії Web-додатків з СУБД.	2
9	Вступ до XML.	1
10	Мови опису схем XML.	1
11	DOM XML. Перетворення XML документів.	2
12	Інтеграція і взаємодія в мережі Web.	2
13	Організація процесу розробки Web-контенту .CMS/CMF.	1
14	Синдикація та агрегування Web -контенту.	1
15	Web-портали. Класифікація Web-порталів.	2
16	Вступ в технологію AJAX. Розробка мобільних Web-додатків.	2
17	Вступ до Web 2.0.	1
18	Додатки для соціальних мереж.	1
Усього годин		26

5. Теми лабораторних занять

№ з/п	Назва теми	Кількість годин
		д/ф
1	Службові утиліти для роботи в Інтернеті. Вивчення протоколу HTTP.	1
2	Принципи Web-дизайна. Знайомство з Microsoft Expression Web.	1
3	Вступ до JavaScript. Програмний взаємозв'язок з HTML документами на основі DOM API.	1
4	Клієнтські сценарії. Використання регулярних виразів.	1
5	Розробка CGI-додатків на Perl і PHP.	2
6	Знайомство з середовищем розробки Microsoft Visual Studio.NET. Структура програми на C#. Основи мови C#.	2
7	Основи мови C#. Робота з масивами і строками.	2

	Інтерфейси та колекції.	
8	Основи розробки Web-додатків за допомогою ASP.NET.	2
9	Серверні елементи управління ASP.NET.	1
10	Робота з джерелами даних у ASP.NET.	1
11	Структура XML документа. XML схеми.	2
12	Програмна обробка XML документів за допомогою XML DOM.	2
13	Формування та перетворення XML документу за допомогою XSL. XSLT перетворення XML документу.	1
14	Розробка Web-служби в ASP.NET.	1
15	Розробка Web-служби в ASP.NET. Створення проксі-зборки для Web-служби.	2
16	Приклади розробки RSS-джерел та RSS-рідерів.	2
17	Реалізація асинхронної взаємодії Web-браузера з Web-вузлом за допомогою технології AJAX	1
18	Створення мобільних Web-додатків за допомогою ASP.NET.	1
Усього годин		26

6. Самостійна робота

№ з/п	Назва теми	Кількість годин
		д/ф
1	Знайомство з HTML та CSS	4
2	Знайомство з текстовими елементами HTML	4
3	Знайомство з основними стильовими правилами CSS	5
4	Знайомство з основними правилами використання, позиціонування та оформлення зображень	5
5	Створення динамічного меню навігації за допомогою списків і правил CSS	5
6	Створення макета сторінки засобами правил CSS	5
7	Обробка подій за допомогою сценаріїв Javascript	5
8	Динамічний HTML	5
9	Створення форми з динамічною перевіркою	5
10	Знайомство з Silverlight	5
11	Знайомство з новими елементами HTML, та прикладами їх використання	5
Разом		53

7. Методи навчання

Лекція, бесіда, презентація, лабораторна робота, самостійна робота.

8. Методи контролю

Для поточного контролю застосовуються тестові завдання.

Для підсумкового контролю – екзамен.

9. Розподіл балів, які отримують студенти

для екзамену

Поточне тестування та самостійна робота																Підсумковий тест (екзамен)	Сума			
Змістовий модуль 1		Змістовий модуль 2		Змістовий модуль 3		Змістовий модуль 4		Змістовий модуль 5			Змістовий модуль 6		Змістовий модуль 7		Змістовий модуль 8		Змістовий модуль 9		50	100
T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18			
2	2	3	3	2	2	3	3	3	3	3	3	3	3	3	3	3	3			

T1, T2 ... T18 – теми змістових модулів.

Шкала оцінювання: національна та ЄКТС

Сума балів за всі види навчальної діяльності	Оцінка за національною шкалою	
	для екзамену, курсового проекту (роботи), практики	для заліку
90 – 100	відмінно	зараховано
65–89	добре	
50–64	задовільно	
1–49	незадовільно	не зараховано

10. Методичне забезпечення

1. Конспекти лекцій.
2. Комп'ютерні презентації.
3. Методичні рекомендації до виконання лабораторних робіт.
4. Методичні рекомендації до виконання самостійної роботи.
5. Критерії оцінювання знань студентів.
6. Питання до екзамену.

Лекція 1. Введення в веб-технології: структура та принципи Веб.

План:

- 1. Предмет курсу.**
- 2. Що таке Інтернет?**
- 3. Про роль стандартизації в Інтернет.**
- 4. Стек протоколи TCP / IP.**
- 5. Система доменних імен DNS.**
- 6. Структура і принципи WWW.**
- 7. Проксі-сервери.**
- 8. Протоколи Інтернет прикладного рівня.**

Література:

- 1. Храмцов П. Б. Основы WEB-технологий. / П. Б. Храмцов, С. А. Брик, А. М. Русак, А. И. Сурин – М. : ИТУИТ.РУ, 2003. – 512 с.**

1. Предмет курсу.

Предметом даного курсу є технології глобальної мережі World Wide Web (скорочено WWW або просто Web). Російською мовою поширеним варіантом є назва «Веб».

Зокрема, в рамках курсу будуть розглянуті такі питання як:

- 1. Структура і принципи Веб (базові поняття, архітектура, стандарти і протоколи);**
- 2. Технології мережі Веб (мови розмітки і програмування веб-сторінок, інструменти розробки та управління веб-контенту та програм для Веб, засоби інтеграції веб-контенту і додатків в Веб).**

Мережа Веб являє собою глобальний інформаційний простір, заснований на фізичній інфраструктурі Інтернету і протоколі передачі даних HTTP. Найчастіше, кажучи про Інтернет, мають на увазі саме мережу Інтернет.

2. Що таке Інтернет?

Оскільки фізичною основою мережі Веб є Інтернет, то для більш глибокого розуміння багатьох питань даного курсу буде потрібно коротко ознайомитися зі структурою та протоколами Інтернету.

Що ж таке Інтернет? По-суті, це найбільша в світі мережа, яка не має єдиного центру управління, але працює за єдиними правилами і надає своїм користувачам єдиний набір послуг. Інтернет можна розглядати як «мережа мереж», кожна з яких управляється незалежним оператором - постачальником послуг Інтернету (ISP, Internet Service Provider).

З точки зору користувачів Інтернет являє собою набір інформаційних ресурсів, розосереджених по різних мережах, включаючи ISP-мережі, корпоративні мережі, мережі та окремі комп'ютери домашніх користувачів. Кожен окремий комп'ютер в даній мережі називається хостом (від англійського терміну host).

Сьогоднішній Інтернет зобов'язаний своїй появі об'єднаній мережі ARPANET, яка починалася як скромний експеримент у новій тоді технології комутації пакетів (табл. 1.1). Мережа ARPANET була розгорнута в 1969 р і складалася спочатку всього з чотирьох вузлів з комутацією пакетів, що використовуються для взаємодії жменьки хостів і терміналів. Перші лінії зв'язку, що з'єднували вузли, працювали на швидкості всього 50 Кбіт/с. Мережа ARPANET фінансувалася управлінням перспективного планування науково-дослідних робіт ARPA (Advanced Research Projects Agency) міністерства оборони США і призначалася для вивчення технології та протоколів комутації пакетів, які могли б використовуватися для кооперативних розподілених обчислень.

Таблиця 1.1

Хронологія розвитку Інтернету (з 1966 по 2000 р)

Рік	Подія
1966	Експеримент з комутацією пакетів управління ARPA
1969	Перші працездатні вузли мережі ARPANET
1972	Винахід розподіленої електронної пошти
1973	Перші комп'ютери, підключені до мережі ARPANET за межами США
1975	Мережа ARPANET передана у відання управління зв'язку міністерства оборони США
1980	Починаються експерименти з TCP / IP
1981	Кожні 20 днів до мережі додається новий хост
1983	Завершено перехід на TCP / IP
1986	Створено магістраль NSFnet
1990	Мережа ARPANET припинила існування
1991	Поява Gopher
1991	Винахід Всесвітньої павутини. Випущена система RGP. Поява Mosaic
1995	Приватизація магістралі Інтернету
1996	Побудована магістраль OC-3 (155 Мбіт / с)
1998	Число зареєстрованих доменних імен перевищило 2 млн.
2000	Кількість індексованих веб-сторінок перевищило 1 млрд.

На рис. 1.1 представлений графік, що показує динаміку зростання кількості хостів (як формально зареєстрованих і так активно функціонуючих).

Чи можливо централізоване управління в такій глобальній мережі? Відповідь на дане питання буде негативним, оскільки, по-перше, дана мережа є транснаціональною і, по-друге, в силу історичних передумов її формування.

Тим не менш, в Інтернеті можуть проявлятися опосередковані форми централізації у формі єдиної технічної політики, узгодженому наборі технічних стандартів, призначення імен і адрес комп'ютерів і мереж, входять в Інтернет.

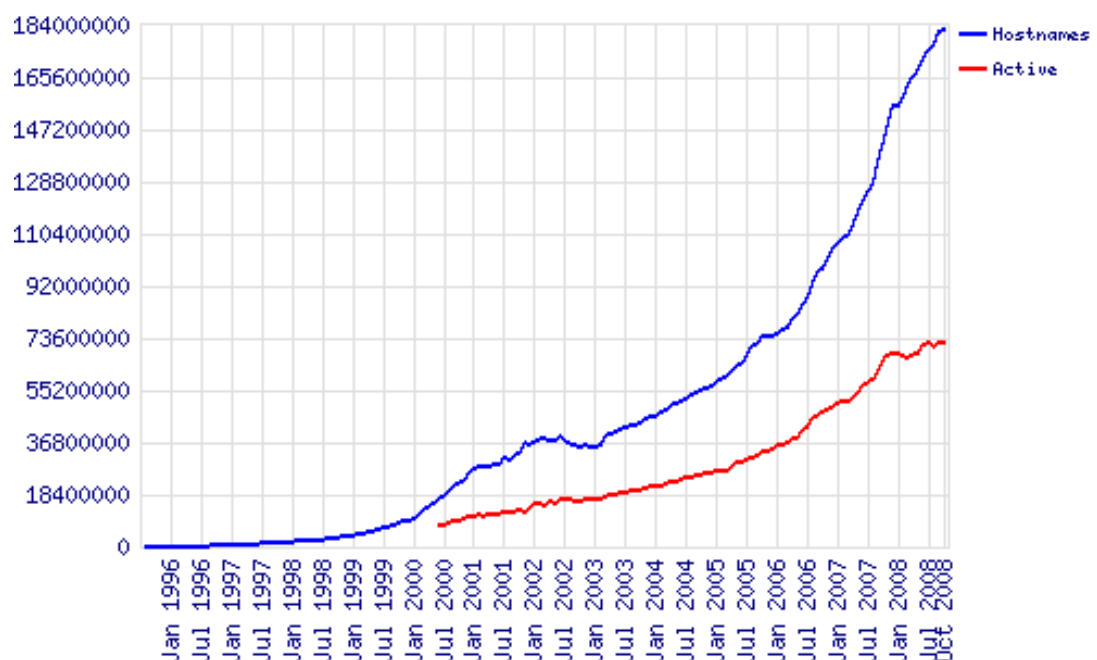


Рис. 1.1. Динаміка зростання кількості хостів в Інтернет (взято з сайту www.netcraft.com).

Тобто Інтернет є децентралізованою мережею, що має свої переваги і недоліки.

Переваги:

- Легкість нарощування Інтернету шляхом укладення угоди між двома ISP.

Недоліки:

- Складність модернізації технологій і послуг Інтернету, оскільки потрібні узгоджені зусилля всіх постачальників послуг.
- Невисока надійність послуг Інтернету.
- Відповідальність за працездатність окремих сегментів цієї мережі покладається на постачальників послуг Інтернету.

Існують різні типи постачальників послуг Інтернету:

- просто постачальник послуг Інтернету виконує транспортну функцію для кінцевих користувачів
- передачу їх трафіку в мережі інших постачальників послуг Інтернету;

- постачальник інтернет-контенту має власні інформаційно-довідкові ресурси, надаючи їх вміст у вигляді веб-сайтів;
- постачальник послуг хостингу надає свої приміщення, канали зв'язку і сервери для розміщення зовнішнього контенту;
- постачальник послуг з доставки контенту займається тільки доставкою контенту в численні точки доступу з метою підвищення швидкості доступу користувачів до інформації;
- постачальник послуг з підтримки додатків надає клієнтам доступ до великих універсальним програмним продуктам, наприклад SAP R3;
- постачальник білінгових послуг забезпечує оплату рахунків по Інтернету.

3. Про роль стандартизації в Інтернет.

Як впливає з усього вищевикладеного, Інтернет є дуже складною мережею, і відповідно такою ж складною є задача організації взаємодії між пристроями мережі. Для вирішення такого роду завдань використовується декомпозиція, тобто розбиття складної задачі на кілька простіших задач-модулів. Однією з концепцій, що реалізують декомпозицію, є багаторівневий підхід. Такий підхід дає можливість проводити розробку, тестування і модифікацію кожного окремого рівня незалежно від інших рівнів. Ієрархічна декомпозиція дозволяє, переміщаючись в напрямку від більш низьких до більш високих рівнів переходити до більш простому поданням розв'язуваної задачі.

Специфіка багаторівневого представлення мережевої взаємодії полягає в тому, що в процесі обміну повідомленнями беру участь як мінімум дві сторони, для яких необхідно забезпечити узгоджену роботу двох

ієрархій апаратно-програмних засобів. Кожен з рівнів повинен підтримувати інтерфейс з вище- і нищележачими рівнями власної ієрархії засобів і інтерфейс із засобами взаємодії іншого боку на тому ж рівні ієрархії. Даний тип інтерфейсу називається протоколом (див. рис. 1.2).

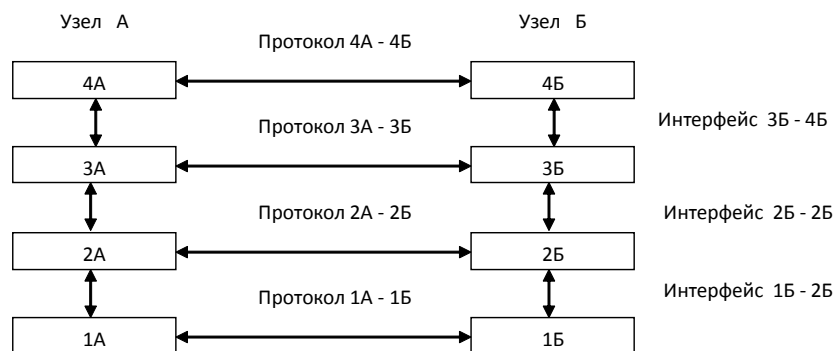


Рис. 1.2. Організація взаємодії між рівнями ієрархії при ієрархічного декомпозиції в мережі Інтернет.

Ієрархічно організований набір протоколів, достатній для організації взаємодії вузлів в мережі, називається стеком протоколів.

На початку 80-х років міжнародні організації зі стандартизації ISO (International Organization for Standardization), ITU (International Telecommunications Union) та інші розробили стандартну модель взаємодії відкритих систем OSI (Open System Interconnection). Призначення даної моделі полягає в узагальненому поданні засобів мережевої взаємодії. Її також можна розглядати в якості універсальної мови мережеских фахівців (довідкової моделі).

Оскільки мережа – це з'єднання різноманітного устаткування, актуальною є проблема сумісності, що в свою чергу, вимагає узгодження всіма виробниками загальноприйнятих стандартів. Відкритою є система, побудована відповідно до відкритих специфікацій.

Специфікація являє собою формалізований опис

апаратних (програмних) компонентів, способів їх функціонування, взаємодії з іншими компонентами, умов експлуатації, особливих характеристик. Під відкритими специфікаціями розуміються опубліковані, загальнодоступні специфікації, відповідні стандартам і прийняті в результаті досягнення згоди після всебічного обговорення всіма зацікавленими сторонами. Використання відкритих специфікацій при розробці систем дозволяє третім сторонам розробляти для цих систем апаратно-програмні засоби розширення і модифікації, а також створювати програмно-апаратні комплекси з продуктів різних виробників.

Якщо дві мережі побудовані з дотриманням принципів відкритості, це дає наступні переваги:

- можливість побудови мережі з апаратних і програмних засобів різних виробників, які дотримуються стандарту;
- безболісна заміна окремих компонентів мережі іншими, більш досконалішими;
- легкість сполучення однієї мережі з іншого.

В рамках моделі OSI засоби взаємодії діляться на сім рівнів: прикладний, уявлення, сеансовий, транспортний, мережевий, каналний і фізичний. У розпорядження програмістів надається прикладний програмний інтерфейс, що дозволяє звертатися із запитами до самого верхнього рівня, а саме, – рівню додатків.

Мережа Інтернет будувалася в повній відповідності з принципами відкритих систем. У розробці стандартів цієї мережі брали участь тисячі фахівців-користувачів мережі з вузів, наукових організацій і компаній. Результат роботи по стандартизації втілюється в документах RFC.

RFC (англ. Request for Comments) – документ із серії пронумерованих інформаційних документів Інтернету, що містять технічні специфікації і стандарти, що широко застосовуються у Всесвітній мережі. В даний час первинної

публікацією документів RFC займається IETF під егідою відкритої організації Товариство Інтернету (ISOC). Правами на RFC володіє саме Товариство Інтернет. Формат RFC з'явився в 1969 р при обговоренні проекту ARPANET. Перші RFC поширювалися в друкованому вигляді на папері у вигляді звичайних листів, але вже з грудня 1969, коли запрацювали перші сегменти ARPANET, документи почали поширюватися в електронному вигляді. У таблиці 1.2 наведені деякі з найбільш відомих документів RFC.

Таблиця 1.2.

Приклади популярних RFC-документів.

Номер RFC	Тема
RFC 768	UDP
RFC 791	IP
RFC 793	TCP
RFC 822	Формат електронної пошти, замінений RFC 2822
RFC 959	FTP
RFC 1034	DNS - концепція
RFC 1035	DNS - впровадження
RFC 1591	Структура доменних імен
RFC 1738	URL
RFC 1939	Протокол POP версії 3 (POP3)
RFC 2026	Процес стандартизації в Інтернеті
RFC 2045	MIME
RFC 2231	Кодування символів
RFC 2616	HTTP
RFC 2822	Формат електронної пошти
RFC 3501	IMAP версії 4 видання 1 (IMAP4rev1)

Основним організаційним підрозділом, що координує роботу по стандартизації Інтернет, є ISOC (Internet Society), що об'єднує близько 100 тисяч учасників, які займаються різними аспектами розвитку даної мережі. ISOC курирує роботу IAB (Internet Architecture Board), що включає дві групи:

- IRTF (Internet Research Task Force). Координує довгострокові дослідницькі проекти, що відносяться до TCP / IP;
- IETF (Internet Engineering Task Force). Інженерна група, яка визначає специфікації для подальших стандартів Інтернет.

Розробкою стандартів для мережі Інтернет, починаючи з 1994 року, займається Консорціум W3C (World Wide Web Consortium), заснований і досі очолюваний Тімом Бернерс-Лі.

Консорціум W3C – організація, яка розробляє та впроваджує технологічні стандарти для Інтернету та WWW. Місія W3C формулюється так: «Повністю розкрити потенціал Всесвітньої павутини шляхом створення протоколів і принципів, що гарантують довгостроковий розвиток Мережі». Два інших найважливіші завдання Консорціуму – забезпечити повну «інтернаціоналізацію Мережі» і зробити її доступною для людей з обмеженими можливостями.

W3C розробляє для WWW єдині принципи і стандарти, звані «Рекомендаціями», які потім впроваджуються розробниками програм і обладнання. Завдяки Рекомендаціям досягається сумісність між програмними продуктами та обладнанням різних компаній, що робить мережу WWW більш досконалою, універсальною і зручною у використанні.

Всі Рекомендації W3C відкриті, тобто, не захищені патентами і можуть впроваджуватися будь-якою людиною без будь-яких фінансових відрахувань Консорціуму.

Для зручності користувачів Консорціумом створені спеціальні програми-валідатори (англ. Online Validation Service), які доступні по мережі і можуть за кілька секунд перевірити документи на відповідність популярним Рекомендаціям W3C. Консорціумом також створені багато інших утиліти для полегшення роботи веб-майстрів та

програмістів. Більшість утиліт - це програми з відкритим вихідним кодом, всі вони безкоштовні. Останнім часом, підкоряючись світовим тенденціям, Консорціум, в цілому, набагато більше уваги приділяє проектам з відкритим вихідним кодом.

Перш ніж перейти до опису структури, принципів роботи та основних протоколів мережі Інтернет, розглянемо основний стік протоколів мережі Інтернет - стек TCP/IP.

4. Стек протоколів TCP/IP.

Ці протоколи спочатку орієнтовані на глобальні мережі, в яких якість сполучних каналів не ідеально. Він дозволяє створювати глобальні мережі, комп'ютери в яких з'єднані один з одним самими різними способами від високошвидкісних оптоволоконних кабелів і супутникових каналів до комутованих телефонних ліній. TCP/IP відповідає моделі OSI досить умовно і містить 4 рівня. Прикладний рівень стека відповідає трьом верхнім рівням моделі OSI: прикладному, подання та сеансовому.

У мережі дані завжди передаються блоками відносно невеликого розміру. Кожен блок має префіксну частину (заголовок), що описує вміст блоку, і суффіксну, що містить, наприклад, інформацію для контролю цілісності переданого блоку даних.

Назва стека протоколів TCP/IP складається з назв двох різних протоколів. Протокол IP (Internet Protocol) являє собою протокол нижнього (мережевого) рівня і відповідає за передачу пакетів даних в мережі. Він відноситься до так званих протоколами датаграм і працює без підтверджень. Останнє означає, що при його використанні доставка пакетів даних не гарантується і не підтверджується. Не гарантується також і те, що пакети досягнуть пункту призначення в тій послідовності, в якій вони були

відправлені.

До протоколів мережного рівня відноситься також протокол міжмережових керуючих повідомлень ICMP (Internet Control Message Protocol), призначений для передачі маршрутизатором джерелу інформації про помилки при передачі пакета.

Очевидно, що набагато зручніше передавати дані по каналу, який працює коректно, доставляючи всі пакети по порядку. Тому поверх протоколу IP працює протокол передачі даних більш високого (транспортного) рівня – TCP (Transmission Control Protocol). Посилаючи і беручи пакети через протокол IP, протокол TCP гарантує доставку всіх переданих пакетів даних у правильній послідовності.

Слід зазначити, що при використанні протоколу IP забезпечується більш швидка передача даних, оскільки не витрачається час на підтвердження прийому кожного пакета. Є й інші переваги. Одне з них полягає в тому, що він дозволяє розсилати пакети даних в ширококомовному режимі, при якому вони досягають всіх комп'ютерів фізичної мережі. Що ж стосується протоколу TCP, то для передачі даних з його допомогою необхідно створити канал зв'язку між комп'ютерами. Він і створюється з використанням протоколу IP.

Для ідентифікації мережових інтерфейсів використовуються 3 типи адрес:

- апаратні адреси (або MAC-адреси);
- мережові адреси (IP-адреси);
- символні (доменні) імена.

В рамках IP протоколу для створення глобальної системи адресації, що не залежить від способів адресації вузлів в окремих мережах, використовується пара ідентифікаторів, що складається з номера мережі і номера вузла. При цьому IP-адреса ідентифікує не окремий комп'ютер або маршрутизатор, а одне мережеве з'єднання в складі мережі, до якої він входить; тобто кінцевий вузол

може входити в кілька IP-мереж.

5. Система доменних імен DNS.

Незважаючи на те, що апаратне і програмне забезпечення в рамках TCP/IP мереж для ідентифікації вузлів використовує IP-адреси, користувачі віддають перевагу символічні імена (доменні імена).

Спочатку в локальних мережах з невеликого числа комп'ютерів застосовувалися плоскі імена, що складаються з послідовності символів без поділу їх на окремі частини, наприклад MYCOMP. Для встановлення відповідності між символічними іменами і числовими адресами використовувалися широкомовні запити. Однак для великих територіально розподілених мереж, що працюють на основі протоколу TCP/IP такий спосіб виявився неефективним. Тому для встановлення відповідності між доменним ім'ям та IP-адресою використовується спеціальна система доменних імен (DNS, Domain Name System), яка заснована на створюваних адміністраторами мережі таблиць відповідності.

У мережах TCP/IP використовується доменна система імен, що має ієрархічну (у вигляді дерева) структуру. Дана структура імен нагадує ієрархію імен, використовувану в багатьох файлових системах. Запис доменного імені починається з наймолодшої складової, потім після точки слід наступна за старшинством символічна частина імені і так далі. Послідовність закінчується кореневим ім'ям, наприклад: company.yandex.ru.

Побудована таким чином система імен дозволяє розділяти адміністративну відповідальність за підтримки унікальності імен у межах свого рівня ієрархії між різними людьми або організаціями.

Сукупність імен, у яких кілька старших складових частин збігаються, утворюють домен імен.

Кореневої домен управляється центральними органами Інтернету: IANA і Internic.

Домени верхнього рівня призначаються для кожної країни, а також для різних типів організацій. Імена цих доменів повинні слідувати міжнародним стандартом ISO 3166. Для позначення країн використовуються дволітерні аббревіатури, наприклад ua (Україна), us (США), it (Італія), fr (Франція), ru (Росія).

Для різних типів організацій використовуються трьохбуквені аббревіатури:

- net – мережеві організації;
- org – некомерційні організації;
- com – комерційні організації;
- edu – освітні організації;
- gov – урядові організації.

Адміністрування кожного домена покладається на окрему організацію, яка делегує адміністрування піддоменів іншим організаціям.

Для отримання доменного імені необхідно зареєструватися у відповідній організації InterNIC, яка делегувала свої повноваження з розподілу доменних імен.

Реєстратором доменних імен в зоні ru до 2005 р був Російський науково-дослідний інститут розвитку суспільних мереж (РосНИИРОС). В даний час реєстрація доменів здійснюється одним з діючих реєстраторів.

В TCP/IP мережах відповідність між доменними іменами та IP-адресами може встановлюватися як локальними засобами, так і централізованими службами. Спочатку відповідність задавалося за допомогою створюваного вручну на хості файлу hosts.txt, що складається з рядків, що містять пару виду «доменне ім'я – IP-адреса». Однак з активним зростанням Інтернету таке рішення виявилось немасштабованим.

Альтернативне рішення – централізована служба DNS, що використовує розподілену базу відображень

«доменне ім'я – IP-адреса». Сервер домену зберігає тільки імена, які закінчуються на наступному нижче по дереву рівні. Це дозволяє розподіляти більш рівномірно навантаження з вирішення імен між усіма DNS-серверами. Кожен DNS-сервер крім таблиці відображення імен містить посилання на DNS-сервери своїх піддоменів.

Існують дві схеми дозволу DNS-імен.

Нерекурсивна процедура:

1. DNS-клієнт звертається до кореневого DNS-сервера із зазначенням повного доменного імені.
2. DNS-сервер відповідає клієнту, вказуючи адресу наступного DNS-сервера, який обслуговує домен верхнього рівня, заданий в наступній старшій частини імені.
3. DNS-клієнт робить запит наступного DNS-сервера, який відсилає його до DNS-сервера потрібного піддомена і т.д., поки не буде знайдений DNS-сервер, у якому зберігається відповідність запитаного імені IP-адресою. Сервер дає остаточну відповідь клієнту.

Рекурсивна процедура:

1. DNS-клієнт запитує локальний DNS-сервер, що обслуговує піддомен, якому належить клієнт.
2. Далі.
3. Якщо локальний DNS-сервер знає відповідь, він повертає його клієнту.
4. Якщо локальний сервер не знає відповідь, то він виконує ітеративні запити до кореневого сервера. Після отримання відповіді сервер передає його клієнту.

Таким чином, при рекурсивної процедури клієнт фактично передоручає роботу своєму серверу. Для прискорення пошуку IP-адрес DNS-сервери широко застосовують кешування (на час від годин до декількох днів) проходять через них відповіді.

6. Структура і принципи WWW.

Мережа WWW утворюють мільйони веб-серверів, розташованих по всьому світу. Веб-сервер є програмою, що запускається на підключеному до мережі комп'ютері і передавальній дані по протоколу HTTP.

Для ідентифікації ресурсів (найчастіше файлів або їх частин) в WWW використовуються ідентифікатори ресурсів URI (Uniform Resource Identifier). Для визначення місцезнаходження ресурсів у цій мережі використовуються локатори ресурсів URL (Uniform Resource Locator). Такі URL-локатори являють собою комбінацію URI і системи DNS.

Доменне ім'я (або IP-адреса) входить до складу URL для позначення комп'ютера (його мережевого інтерфейсу), на якому працює програма веб-сервер.

На клієнтському комп'ютері для перегляду інформації, отриманої від веб-сервера, застосовується спеціальна програма – веб-браузер. Основна функція веб-браузера – відображення гіпертекстових сторінок (веб-сторінок). Для створення гіпертекстових сторінок в WWW спочатку використовувався мову HTML. Безліч веб-сторінок утворюють веб-сайт.

7. Проксі-сервери.

Проксі-сервер (proxy-server) – служба в комп'ютерних мережах, що дозволяє клієнтам виконувати непрямі запити до інших мережних служб.

Спочатку клієнт підключається до проксі-сервера і запитує який-небудь ресурс, розташований на іншому сервері. Потім проксі-сервер або підключається до вказаного серверу і отримує ресурс у нього, або повертає ресурс з власного кешу (якщо є). У деяких випадках запит клієнта або відповідь сервера може бути змінений проксі-

сервером в певних цілях. Також проксі-сервер дозволяє захищати клієнтський комп'ютер від деяких мережесих атак.

Найчастіше проксі-сервери застосовуються для наступних цілей:

- забезпечення доступу з комп'ютерів локальної мережі в Інтернет;
- кешування даних: якщо часто відбуваються звернення до одних і тих же зовнішніх ресурсів, то можна тримати їх копію на проксі-сервері і видавати за запитом, знижуючи тим самим навантаження на канал у зовнішню мережу і прискорюючи отримання клієнтом запитаної інформації;
- тиснення даних: проксі-сервер завантажує інформацію з Інтернету і передає інформацію кінцевому користувачеві в стислому вигляді;
- захист локальної мережі від зовнішнього доступу: наприклад, можна налаштувати проксі-сервер так, що локальні комп'ютери будуть звертатися до зовнішніх ресурсів тільки через нього, а зовнішні комп'ютери не зможуть звертатися до локальних взагалі (вони «бачать» тільки проксі-сервер);
- обмеження доступу з локальної мережі до зовнішньої: наприклад, можна заборонити доступ до певних веб-сайтів, обмежити використання інтернету якимось локальним користувачем, встановлювати квоти на трафік або смугу пропускання, фільтрувати рекламу і віруси;
- анонімізація доступу до різних ресурсів. Проксі-сервер може приховувати відомості про джерело запиту або користувача. У такому випадку цільовий сервер бачить лише інформацію про проксі-сервер, наприклад, IP-адресу, але не має можливості визначити дійсне джерело запиту.

Існують також спотворюють проксі-сервери, які передають цільового сервера неправдиву інформацію про справжній користувача.

8. Протоколи Інтернет прикладного рівня.

Самий верхній рівень в ієрархії протоколів Інтернет займають наступні протоколи прикладного рівня:

- DNS – розподілена система доменних імен, яка за запитом, що містить доменне ім'я хоста повідомляє IP адреса;
- HTTP – протокол передачі гіпертексту в Інтернет;
- HTTPS – розширення протоколу HTTP, що підтримує шифрування;
- FTP (File Transfer Protocol – RFC 959) – протокол, призначений для передачі файлів в компютерних мережах;
- Telnet (TELEcommunication NETwork – RFC 854) – мережевий протокол для реалізації текстового інтерфейсу по мережі;
- SSH (Secure Shell – RFC 4251) – протокол прикладного, що дозволяє проводити віддалене управління операційною системою та передачу файлів. На відміну від Telnet шифрує весь трафік;
- POP3 – протокол поштового клієнта, який використовується поштовим клієнтом для отримання повідомлень електронної пошти з сервера;
- IMAP – протокол доступу до електронної пошти в Інтернет;
- SMTP – протокол, який використовується для відправки пошти від користувачів до серверів і між серверами для подальшої пересилки до одержувача;
- LDAP – протокол для доступу до служби каталогів

X.500, є широко використовуваним стандартом доступу до служб каталогів;

- XMPP (Jabber) – заснований на XML розширюваний протокол для миттєвого обміну повідомленнями в майже реальному часі;
- SNMP – базовий протокол управління мережі Internet.

Розглянемо більш детально деякі з цих протоколів.

1. FTP.

FTP дозволяє підключатися до серверів FTP, переглядати вміст каталогів і завантажувати файли з сервера або на сервер; крім того, можливий режим передачі файлів між серверами; FTP дозволяє обмінюватися файлами і виконувати операції над ними через TCP-мережі. Даний протокол працює незалежно від операційних систем. Історично протокол FTP запропонував відкриту функціональність, забезпечуючи прозорий перенесення файлів з одного комп'ютера на інший через мережу. Це не так тривіально, як може здатися, так як у різнотипних комп'ютерів можуть відрізнятися розміри слів, біти в словах можуть зберігатися в неоднаковому порядку або використовуватися різні формати слів.

2. Telnet.

Назва «telnet» мають також деякі утиліти, що реалізують клієнтську частину протоколу. Протокол telnet працює відповідно до принципів архітектури «клієнт-сервер» і забезпечує емуляцію алфавітно-цифрового терміналу, обмежуючи користувача режимом командного рядка. Додаток telnet надало мову для спілкування терміналів з віддаленими комп'ютерами. Коли з'явилася мережа ARPANET, для кожної комп'ютерної системи потрібні власні термінали. Додаток telnet стало спільним знаменником для терміналів. Досить було написати для кожного комп'ютера програмне забезпечення, що підтримує «термінал telnet», щоб один

термінал міг взаємодіяти з комп'ютерами всіх типів.

3. SSH.

Подібний по функціональності з протоколами telnet і rlogin, але, на відміну від них, шифрує весь трафік, включаючи і передані паролі. SSH-клієнти та SSH-сервери є для більшості операційних систем.

4. Поштові протоколи.

Хоча telnet і FTP були (і залишаються) корисними, першим додатком, які вчинили переворот у свідомості користувачів комп'ютерів мережі ARPANET, стала електронна пошта. До мережі ARPANET існували системи електронної пошти, але всі вони були однокомп'ютерними системами. У 1972 р Рей Томлінсон (Ray Tomlinson) з компанії BBN написав перший пакет, що надає розподілені поштові послуги в комп'ютерній мережі з декількох комп'ютерів. Вже до 1973 дослідження управління ARPA показали, що три чверті всього трафіку мережі ARPANET становила електронна пошта. Користь електронної пошти виявилася настільки велика, що все більше користувачів прагнуло підключитися до мережі ARPANET, в результаті чого зростала потреба в додаванні нових вузлів і використанні високошвидкісних ліній. Таким чином, з'явилася тенденція, що зберігається і донині.

POP3 (Post Office Protocol Version 3 – RFC 1939) – протокол, який використовується поштовим клієнтом для отримання повідомлень електронної пошти з поштового сервера;

IMAP (Internet Message Access Protocol – RFC 3501) – протокол доступу до електронної пошти. Аналогічний POP3, проте надає користувачеві багаті можливості для роботи з поштовими скриньками, що знаходяться на центральному сервері. Електронними листами можна маніпулювати з комп'ютера користувача (клієнта) без необхідності постійної пересилання з сервера і назад файлів з повним змістом листів.

SMTP (Simple Mail Transfer Protocol – RFC 2821) – протокол, призначений для передачі електронної пошти. Використовується для відправки пошти від користувачів до серверів і між серверами для подальшої пересилки до одержувача. Для прийому пошти поштовий клієнт повинен використовувати протоколи POP3 або IMAP.

Лекція 2. Введення в клієнт-серверні технології Веб. Протокол HTTP.

План:

- 1. Протокол HTTP.**
- 2. Забезпечення безпеки передачі даних HTTP.**
- 3. Cookie.**

Література:

1. Храмцов П. Б. Основы WEB-технологий. / П. Б. Храмцов, С. А. Брик, А. М. Русак, А. И. Сурин – М. : ИТУИТ.РУ, 2003. – 512 с.
2. Шеперд Д. Освой самостоятельно XML за 21 день / Д. Шеперд. – М. : «Вильямс», 2002. – 432 с.

1. Протокол HTTP.

Базовим протоколом мережі гіпертекстових ресурсів Веб є протокол HTTP. У його основу покладено взаємодію «клієнт-сервер», тобто передбачається, що:

- споживач-клієнт ініціювавши з'єднання з постачальником-сервером посилає йому запит;
- Постачальник-сервер, отримавши запит, проводить необхідні дії і повертає назад клієнтові відповідь з результатом.

При цьому можливі два способи організації роботи комп'ютера-клієнта:

1. Тонкий клієнт – це комп'ютер-клієнт, який переносить всі завдання з обробки інформації на сервер. Прикладом тонкого клієнта може служити комп'ютер з браузером, що використовується для роботи з веб-додатками.
2. Товстий клієнт, навпаки, робить обробку

інформації незалежно від сервера, використовує останній в основному лише для зберігання даних.

Перш ніж перейти до конкретних клієнт-серверним веб-технологіям, розглянемо основні принципи та структуру базового протоколу HTTP.

1. Протокол HTTP.

HTTP (HyperText Transfer Protocol – RFC 1945, RFC 2616)
– протокол прикладного рівня для передачі гіпертексту.

Центральним об'єктом в HTTP є ресурс, на який вказує URI в запиті клієнта. Зазвичай такими ресурсами є зберігаються на сервері файли. Особливістю протоколу HTTP є можливість вказати в запиті і відповіді спосіб представлення одного і того ж ресурсу за різними параметрами: формату, кодуванні, мови і т. Д. Саме завдяки можливості вказівки способу кодування повідомлення клієнт і сервер можуть обмінюватися двійковими даними, хоча спочатку даний протокол призначений для передачі символічної інформації. На перший погляд це може здатися зайвою тратою ресурсів. Дійсно, дані в символічному вигляді займають більше пам'яті, повідомлення створюють додаткове навантаження на канали зв'язку, однак подібний формат має багато переваг. Повідомлення, що передаються по мережі, зручні для читання, і, проаналізувавши отримані дані, системний адміністратор може легко знайти помилку і усунути її. При необхідності роль одного з взаємодіючих додатків може виконувати людина, вручну вводячи повідомлення в необхідному форматі.

На відміну від багатьох інших протоколів, HTTP є протоколом без пам'яті. Це означає, що протокол не зберігає інформацію про попередніх запитах клієнтів і відповідях сервера. Компоненти, що використовують HTTP, можуть самостійно здійснювати збереження інформації

про стан, який пов'язаний з останніми запитами і відповідями. Наприклад, клієнтський веб-додаток, посилаючи запити, може відстежувати затримки відповідей, а веб-сервер може зберігати IP-адреси та заголовки запитів останніх клієнтів.

Все програмне забезпечення для роботи з протоколом HTTP поділяється на три основні категорії:

1. *Сервери* – постачальники послуг зберігання і обробки інформації (обробка запитів).
2. *Клієнти* – кінцеві споживачі послуг сервера (відправка запитів).
3. *Проксі-сервери* для підтримки роботи транспортних служб.

Основними клієнтами є браузері наприклад: *Internet Explorer, Opera, Mozilla Firefox, Netscape Navigator* та інші. Найбільш популярними реалізаціями веб-серверів є: *Internet Information Services (IIS), Apache, lighttpd, nginx*. Найбільш відомі реалізації проксі-серверів: *Squid, UserGate, Multiproxy, Naviscope*.

«Класична» схема HTTP-сеансу виглядає так:

1. встановлення TCP-з'єднання;
2. запит клієнта;
3. відповідь сервера;
4. розрив TCP-з'єднання.

Таким чином, клієнт посилає серверу запит, отримує від нього відповідь, після чого взаємодія припиняється. Зазвичай запит клієнта являє собою вимога передати HTML-документ або який-небудь інший ресурс, а відповідь сервера містить код цього ресурсу.

До складу HTTP-запиту, переданого клієнтом серверу, входять наступні компоненти:

1. Рядок стану (іноді для її позначення використовують також терміни рядок-статус, або рядок запиту).
2. Поля заголовку.

3. Порожній рядок.

4. Тіло запиту.

Рядок стану разом з полями заголовка іноді називають також заголовком запиту.

Рядок стану має наступний формат:

метод_запросу URL_ресурсу версія_протоколу_HTTP

Розглянемо компоненти рядка стану, при цьому особливу увагу приділимо методам запиту.

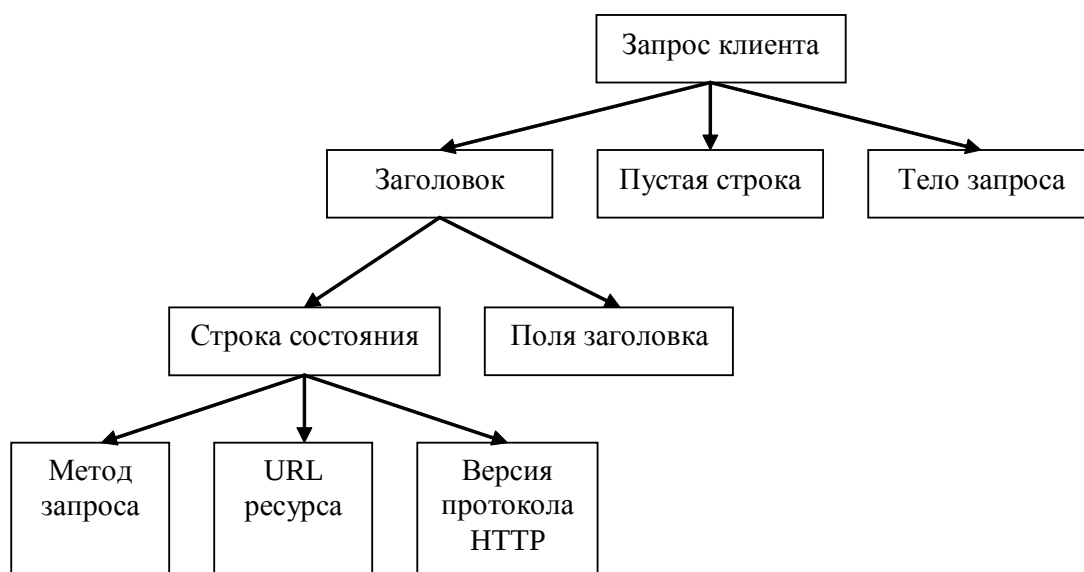


Рис. 2.1. Структура запиту клієнта.

Метод, вказаний в рядку стану, визначає спосіб впливу на ресурс, *URL* якого заданий в тому ж рядку. Метод може приймати значення *GET*, *POST*, *HEAD*, *PUT*, *DELETE* і т. д. Незважаючи на велику кількість методів, для веб-програміста по-справжньому важливі лише два з них: *GET* і *POST*.

Метод *GET*.

Згідно формального визначення, метод *GET* призначається для отримання ресурсу з вказаним *URL*. Отримавши запит *GET*, сервер повинен прочитати зазначений ресурс і включити код ресурсу до складу

відповіді клієнту. Ресурс, URL якого передається в складі запиту, не обов'язково повинен являти собою HTML-сторінку, файл із зображенням або інші дані. URL ресурсу може вказувати на виконуваний код програми, який, при дотриманні певних умов, повинен бути запущений на сервері. У цьому випадку клієнтові повертається не код програми, а дані, згенеровані в процесі її виконання. Незважаючи на те що, за визначенням, метод GET призначений для отримання інформації, він може застосовуватися і в інших цілях. Метод GET цілком підходить для передачі невеликих фрагментів даних на сервер.

Метод *POST*.

Згідно з тим же формального визначення, основне призначення методу POST – передача даних на сервер. Однак, подібно методу GET, метод POST може застосовуватися по-різному і нерідко використовується для отримання інформації з сервера. Як і у випадку з методом GET, URL, заданий у рядку стану, вказує на конкретний ресурс. Метод POST також може використовуватися для запуску процесу.

Методи *HEAD* і *PUT* є модифікаціями методів GET і POST.

Версія протоколу HTTP, як правило, задається в наступному форматі:

HTTP / версія.модифікація

Поля заголовку, наступні за рядком стану, дозволяють уточнювати запит, тобто передавати серверу додаткову інформацію. **Поле заголовку** має наступний формат:

Ім'я_поля: Значення

Призначення поля визначається його ім'ям, яке відділяється від значення двокрапкою.

Імена деяких найбільш часто зустрічаються в запиті клієнта полів заголовка і їх призначення наведені у таблиці 2.1.

У багатьох випадках при роботі в Веб, тіло запиту відсутнє. При запуску CGI-сценаріїв дані, передані для них у запиті, можуть розміщуватися в тілі запиту.

Нижче представлений приклад HTML-запиту, згенерованого браузером.

Таблиця 2.1.

Поля заголовка запиту HTTP.

Поля заголовка HTTP-запиту	Значення
Host	Доменне ім'я або IP-адресу сайту, до якого звертається клієнт
Referer	URL документа, який посилається на ресурс, зазначений у рядку стану
From	Адреса електронної пошти користувача, що працює з клієнтом
Accept	MIME-типи даних, оброблюваних клієнтом. Це поле може мати кілька значень, відокремлюваних одне від іншого запитом. Часто поле заголовка Асепт використовується для того, щоб повідомити сервер про те, які типи графічних файлів підтримує клієнт
Accept-Language	Набір двосимвольних ідентифікаторів, розділених комами, які позначають мови, що підтримуються клієнтом
Accept-Charset	Перелік підтримуваних наборів символів
Content-Type	MIME-тип даних, що містяться в тілі запиту (якщо клопотання не складається з одного заголовка)
Content-Length	Число символів, що містяться в тілі запиту (якщо клопотання не складається з одного заголовка)
Range	Присутній в тому випадку, якщо клієнт запитує не

	весь документ, а лише його частина
Connection	Використовується для управління TCP-з'єднанням. Якщо в полі міститься Close, це означає, що після обробки запиту сервер повинен закрити з'єднання. Значення Keep-Alive пропонує не закривати TCP-з'єднання, щоб воно могло бути використано для подальших запитів
User-Agent	Інформація про клієнта

Отримавши від клієнта запит, сервер повинен відповісти йому. Знання структури відповіді сервера необхідно розробнику веб-додатків, так як програми, які виконуються на сервері, повинні самостійно формувати відповідь клієнту.

```
GET http://oak.oakland.edu/ HTTP / 1.0
Connection: Keep-Alive
User-Agent: Mozilla / 4.04 [en] (Win95; I)
Host: oak.oakland.edu
Accept: image / gif, image / x-xbitmap, image / jpeg,
image / pjpeg, image / png, * / *
Accept-Language: en
Accept-Charset: iso-8859-1, *, utf-8
```

Подібно запиту клієнта, відповідь сервера також складається з чотирьох перерахованих нижче компонентів.

1. Рядок стану.
2. Поля заголовку.
3. Порожній рядок.
4. Тіло відповіді.

Відповідь сервера клієнту починається з рядка стану, яка має наступний формат:

Версія_протоколу Код_відповіді Пояснювальне_повідомлення

Версія_протоколу задається в тому ж форматі, що і в

запиті клієнта, і має той же зміст.

Код_відповіді – це тризначне десяткове число, що представляє в закодованому вигляді результат обслуговування запиту сервером.

Пояснювальне_повідомлення дублює код відповіді в символьному вигляді. Це рядок символів, який не обробляється клієнтом. Воно призначена для системного адміністратора або оператора, що займається обслуговуванням системи, і є розшифровкою коду відповіді.

З трьох цифр, складових коду відповіді, перша (старша) визначає клас відповіді, інші дві являють собою номер відповіді усередині класу. Так, наприклад, якщо запит був оброблений успішно, клієнт отримує наступне повідомлення:

HTTP / 1.0 200 OK

Як видно, за версією протоколу HTTP 1.0 слідує код 200. У цьому коді символ 2 означає успішну обробку запиту клієнта, а інші дві цифри (00) – номер даного повідомлення.

У використовуваних в даний час реалізаціях протоколу HTTP перша цифра не може бути більше 5 і визначає наступні класи відповідей:

- **1** – спеціальний клас повідомлень, званих інформаційними. Код відповіді, що починається з 1, означає, що сервер продовжує обробку запиту. При обміні даними між HTTP-клієнтом і HTTP-сервером повідомлення цього класу використовуються досить рідко.
- **2** – успішна обробка запиту клієнта.
- **3** – перенаправлення запиту. Щоб запит був обслужений, необхідно вжити додаткові дії.
- **4** – помилка клієнта. Як правило, код відповіді, що

починається з цифри 4, повертається в тому випадку, якщо в запиті клієнта зустрілася синтаксична помилка.

- 5 – помилка сервера. З тих чи інших причин сервер не в змозі виконати запит.

Приклади кодів відповідей, які клієнт може отримати від сервера, і пояснювальне повідомлення наведені в таблиці 2.2.

Таблиця 2.2.

Класи кодів відповіді сервера.

Код	Розшифровка	Інтерпретація
100	Continue	Частина запиту прийнята, і сервер очікує від клієнта продовження запиту
200	OK	Запит успішно оброблений, і у відповіді клієнта передаються дані, зазначені в запиті
201	Created	В результаті обробки запиту був створений новий ресурс
202	Accepted	Запит прийнятий сервером, але обробка його не закінчена. Даний код відповіді не гарантує, що запит буде оброблений без помилок.
206	Partial Content	Сервер повертає частину ресурсу у відповідь на запит, що містив поле заголовка Range
301	Multiple Choice	Запит вказує більш ніж на один ресурс. У тілі відповіді можуть міститися вказівки на те, як правильно ідентифікувати запитуваний ресурс
302	Moved Permanently	Викликаний ресурс більше не розташовується на сервері
302	Moved Temporarily	Викликаний ресурс тимчасово змінив свою адресу
400	Bad Request	У запиті клієнта виявлена синтаксична помилка
403	Forbidden	Наявний на сервері ресурс недоступний для даного користувача
404	Not Found	Ресурс, зазначений клієнтом, на сервері відсутній

405	Method Not Allowed	Сервер не підтримує метод, зазначений у запиті
500	Internal Server Error	Один з компонентів сервера працює некоректно
501	Not Implemented	Функціональних можливостей сервера недостатньо, щоб виконати запит клієнта
503	Service Unavailable	Служба тимчасово недоступна
505	HTTP Version not Supported	Версія HTTP, що указана в запиті, не підтримується сервером

У відповіді використовується така ж структура полів заголовка, як і в запиті клієнта. Поля заголовка призначені для того, щоб уточнити відповідь сервера клієнту. Опис деяких з полів, які можна зустріти в заголовку відповіді сервера, наведено в таблиці 2.3.

Таблиця 2.3.

Поля заголовка відповіді веб-сервера.

Ім'я поля	Опис змісту
Server	Ім'я та номер версії сервера
Age	Час у секундах, що минув з моменту створення ресурсу
Allow	Список методів, допустимих для даного ресурсу
Content-Language	Мови, які повинен підтримувати клієнт для того, щоб коректно відобразити переданий ресурс
Content-Type	MIME-тип даних, що містяться в тілі відповіді сервера
Content-Length	Число символів, що містяться в тілі відповіді сервера
Last-Modified	Дата і час останньої зміни ресурсу
Date	Дата і час, що визначають момент генерації відповіді
Expires	Дата і час, що визначають момент, після якого інформація, передана клієнту, вважається застарілою
Location	У цьому полі вказується реальне розташування ресурсу. Воно використовується для

	перенаправлення запиту
Cache-Control	Директиви управління кешуванням. Наприклад, <i>no-cache</i> означає, що дані не повинні керуватися

У тілі відповіді міститься код ресурсу, переданого клієнтові у відповідь на запит. Це не обов'язково має бути HTML-текст веб-сторінки. У складі відповіді можуть передаватися зображення, аудіо-файл, фрагмент відеоінформації, а також будь-який інший тип даних, підтримуваних клієнтом. Про те, як слід обробляти отриманий ресурс, клієнту повідомляє вміст поля заголовка *Content-type*.

На рис. 2.2 представлений приклад відповіді серверу на запит, наведений у попередньому розділі. У тілі відповіді міститься вихідний текст HTML-документа.

Поля заголовка і тіло повідомлення можуть бути відсутні, але рядок стану є обов'язковим елементом, оскільки вказує на тип запиту / відповіді.

Поле з ім'ям *Content-type* може зустрічатися як в запиті клієнта, так і у відповіді сервера. Як значення цього поля вказується MIME-тип вмісту запиту або відповіді. MIME-тип також передається в поле заголовка *Accept*, присутнього в запиті.

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
X-Powered-By: ASP.NET
Date: Mon, 20 OCT 2008 11:25:56 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Sat, 18 Oct 2008 15:05:44 GMT
ETag: "b66a667f948c92:8a5"
Content-Length: 426

<html>
<body>
  <form action='http://localhost/Scripts/test.pl'>
    <p>Operand1: <input type='text' name='A'></p>
```

```
<p>Operand2: <input type='text' name='B'></p>
<p>Operation:<br>
<select name='op'>
    <option value='+'>+</option>
    <option value='-'>-</option>
    <option value='*'>*</option>
    <option value='/'>/</option>
</select></p>
<input type='submit' value='Calculate!'>
</from>
</body>
</html>
```

Рис. 2.2. Відповіді серверу на запит.

Специфікація *MIME* (*Multipurpose Internet Mail Extension* – багатоцільове поштове розширення Internet) спочатку була розроблена для того, щоб забезпечити передачу різних форматів даних у складі електронних листів. Однак застосування *MIME* не вичерпується електронною поштою. Засоби *MIME* успішно використовуються в WWW і, по суті, стали невід’ємною частиною цієї системи.

Стандарт *MIME* розроблений як розширювана специфікація, в якій передбачається, що число типів даних буде рости в міру розвитку форм представлення даних. Кожен новий тип в обов’язковому порядку повинен бути зареєстрований в *IANA* (*Internet Assigned Numbers Authority*).

До появи *MIME* комп’ютери, які взаємодіють за протоколом HTTP, обмінювалися виключно текстовою інформацією. Для передачі зображень, як і для передачі будь-яких інших довічних файлів, доводилося користуватися протоколом FTP.

Відповідно до специфікації *MIME*, для опису формату даних використовуються *тип* і *підтип*. *Тип* визначає, до якого класу належить формат вмісту HTTP-запиту або

HTTP-відповіді. *Підтип* уточнює формат. Тип і підтип відокремлюються друг від друга косою рисою:

тип/підтип

Оскільки в переважній більшості випадків у відповідь на запит клієнта сервер повертає вихідний текст HTML-документа, то в поле *Content-type* відповіді зазвичай міститься значення *text/html*. Тут ідентифікатор *text* визначає тип, повідомляючи, що клієнту передається символічна інформація, а ідентифікатор *html* описує підтип, тобто вказує на те, що послідовність символів, що міститься в тілі відповіді, являє собою опис документа на мові HTML.

Перелік типів і підтипів *MIME* досить великий. У таблиці 2.4 наведені приклади *MIME*-типів, найбільш часто зустрічаються в заголовках HTML-запитів і відповідей.

Для однозначної ідентифікації ресурсів в мережі Веб використовуються унікальні ідентифікатори *URL*.

Однаковий ідентифікатор ресурсу *URI* (*Uniform Resource Identifier*) являє собою коротку послідовність символів, що ідентифікує абстрактний або фізичний ресурс. *URI* не вказує на те, як отримати ресурс, а тільки ідентифікує його. Це дає можливість описувати за допомогою *RDF* (*Resource Description Framework*) ресурси, які не можуть бути отримані через Інтернет (імена, назви тощо). Найвідоміші приклади *URI* – це *URL* і *URN*.

Таблиця 2.4.

MIME типи даних.

Тип/підтип	Розширення файлу	Опис
application/pdf	.pdf	Документ, призначений для обробки Acrobat Reader
application/msexcel	.xls	Документ у форматі Microsoft Excel

application/postscript	.ps, .eps	Документ у форматі PostScript
application/x-tex	.tex	Документ у форматі TeX
application/msword	.doc	Документ у форматі Microsoft Word
application/rtf	.rtf	Документ у форматі RTF, який відображається за допомогою Microsoft Word
image/gif	.gif	Зображення у форматі GIF
image/jpeg	.jpeg, .jpg,	Зображення у форматі JPEG
image/tiff	.tiff, .tif	Зображення у форматі TIFF
image/x-xbitmap	.xbm	Зображення у форматі XBitmap
text/plain	.txt	ASCII-текст
text/html	.html, .htm	Документ у форматі HTML
audio/midi	.midi, .mid	Аудіофайл у форматі MIDI
audio/x-wav	.wav	Аудіофайл у форматі WAV
message/rfc822		Поштове повідомлення
message/news		Повідомлення в групі новин
video/mpeg	.mpeg, .mpg, .mpe	Відеофрагмент у форматі MPEG
video/avi	.avi	Відеофрагмент у форматі AVI

URL (Uniform Resource Locator) – це URI, який, крім ідентифікації ресурсу, надає ще й інформацію про місцезнаходження цього ресурсу.

URN (Uniform Resource Name) – це URI, який ідентифікує ресурс в певному просторі назв, але, на відміну від URL, URN не вказує на місцезнаходження цього ресурсу.

URL має наступну структуру:

<Схема>:// <логін>: <пароль> @ <хост>: <порт>/ <URL-шлях>

де:

- *схема* – схема звернення до ресурсу (зазвичай мережевий протокол);
- *логін* – ім'я користувача, що використовується для доступу до ресурсу;
- *пароль* – пароль, асоційований з вказаним ім'ям користувача;
- *хост* – повністю прописане доменне ім'я хоста в системі DNS або IP-адреса хоста;
- *порт* – порт хоста для підключення;
- *URL-шлях* – уточнююча інформація про місце знаходження ресурсу.

Загальноприйняті схеми (протоколи) URL включають протоколи: *ftp*, *http*, *https*, *telnet*, а також:

- *gopher* – протокол Gopher;
- *mailto* – адреса електронної пошти;
- *news* – новини Usenet;
- *nntp* – новини Usenet через протокол NNTP;
- *irc* – протокол IRC;
- *prospero* – служба каталогів Prospero Directory Service;
- *waits* – база даних системи WAIS;
- *xmpp* – протокол XMPP (частина Jabber);
- *file* – ім'я локального файлу;
- *data* – безпосередні дані (Data: URL);

2. Забезпечення безпеки передачі даних HTTP.

Оскільки протокол HTTP призначений для передачі символічних даних у відкритому (незашифрованому) вигляді, то особи, які мають доступ до каналу передачі даних між клієнтом і сервером, можуть без зусиль переглядати весь трафік і використовувати його для вчинення несанкціонованих дій. У зв'язку з цим запропоновано ряд розширень базового протоколу

спрямованих на підвищення захищеності інтернет-трафіку від несанкціонованого доступу.

Найпростішим є розширення *HTTPS*, при якому дані, передані по протоколу *HTTP*, «упаковуються» в криптографічний протокол *SSL* або *TLS*, тим самим забезпечуючи захист цих даних. На відміну від *HTTP*, для *HTTPS* за замовчуванням використовується TCP-порт 443. Щоб підготувати веб-сервер для обробки *HTTPS* з'єднань, адміністратор повинен отримати і встановити в систему сертифікат для цього веб-сервера.

SSL (Secure Sockets Layer) – криптографічний протокол, що забезпечує безпечну передачу даних по мережі Інтернет. При його використанні створюється захищене з'єднання між клієнтом і сервером. *SSL* спочатку розроблений компанією *Netscape Communications*. Згодом на підставі протоколу *SSL 3.0* був розроблений і прийнятий стандарт *RFC*, що отримав назву *TLS*. Цей протокол використовує шифрування з відкритим ключем для підтвердження автентичності передавача та одержувача. Підтримує надійність передачі даних за рахунок використання коригувальних кодів і безпечних хеш-функцій. На нижньому рівні багаторівневого транспортного протоколу (наприклад, *TCP*) він є протоколом запису і використовується для інкапсуляції різних протоколів (*POP3*, *IMAP*, *SMTP* або *HTTP*). Для кожного інкапсульованого протоколу він забезпечує умови, за яких сервер і клієнт можуть підтверджувати один одному свою автентичність, виконувати алгоритми шифрування і проводити обмін криптографічними ключами, перш ніж протокол прикладної програми почне передавати і отримувати дані.

Для доступу до веб-сторінок, захищеним протоколом *SSL*, в URL замість схеми *http*, як правило, підставляється схема *https*, яка вказує на те, що буде використовуватися *SSL*-з'єднання. Стандартний TCP-порт для з'єднання за

протоколом *https* – 443. Для роботи SSL потрібно, щоб на сервері мався SSL-сертифікат.

У мережі Веб підтримуються 3 типи аутентифікації при клієнт-серверних взаємодіях:

- *Basic* – базова аутентифікація, при якій ім'я користувача та пароль передаються в заголовках http-пакетів. Пароль при цьому не шифрується і присутній в чистому вигляді в кодуванні *base64*. Для даного типу аутентифікації використання SSL є обов'язковим.
- *Digest* – дайджест-аутентифікація, при якій пароль користувача передається в хешированому вигляді. За рівнем конфіденційності паролів цей тип мало чим відрізняється від попереднього, так як атакуючому все одно, чи дійсно це справжній пароль або тільки хеш від нього: перехопивши посвідчення, він все одно отримує доступ до кінцевої точки. Для даного типу аутентифікації використання SSL є обов'язковим.
- *Integrated* – інтегрована аутентифікація, при якій клієнт і сервер обмінюються повідомленнями для з'ясування автентичності один одного за допомогою протоколів NTLM або Kerberos. Цей тип аутентифікації захищений від перехоплення посвідчень користувачів, тому для нього не потрібно протокол SSL. Тільки при використанні даного типу аутентифікації можна працювати за схемою *http*, у всіх інших випадках необхідно використовувати схему *https*.

3. Cookie.

Оскільки HTTP-сервер не пам'ятає передісторії запитів клієнтів, то кожен запит обробляється незалежно від інших, і у сервера немає можливості визначити, чи

виходять запити від одного клієнта або різних клієнтів.

Якщо сервер перевірятиме TCP-з'єднання і запам'ятовувати IP-адреси комп'ютерів-клієнтів, він все одно не зможе розрізнити запити від двох браузерів, що виконуються на одній машині. І навіть якщо припустити, що на комп'ютері працює лише одна клієнт-програма, то ніхто не може стверджувати, що в проміжку між двома запитами вона не була завершена, а потім запущена знову вже іншим користувачем.

Тим не менше, якщо ви коли-небудь користувалися поштовою скринькою на *mail.ru* або на іншому сервері, що надає поштові послуги користувачам Веб, згадайте, як повадився клієнт після того, як ви створили для себе поштову скриньку на сервері. Коли ви наступного разу звернулися з того ж комп'ютера до *mail.ru*, ви, мабуть, помітили, що після завантаження веб-сторінки ваше реєстраційне ім'я вже відображалось у відповідному полі введення.

Такі відомості дозволяє отримати додатковий засіб під назвою *cookie*. Механізм *cookie* дозволяє серверу зберігати інформацію на комп'ютері клієнта і витягувати її звідти.

Ініціатором записи *cookie* виступає сервер. Якщо у відповіді сервера присутній поле заголовка *Set-cookie*, клієнт сприймає це як команду на запис *cookie*. Надалі, якщо клієнт звертається до сервера, від якого він раніше прийняв поле заголовка *Set-cookie*, крім іншої інформації він передає серверу дані *cookie*. Для передачі зазначеної інформації сервера використовується поле заголовка *Cookie*.

Для того щоб в загальних рисах уявити собі, як відбувається обмін даними *cookie*, розглянемо наступний приклад. Припустимо, що клієнт передає запити на сервери А, В і С. Припустимо також, що сервер В, на відміну від А і С, передає клієнту команду записати *cookie*. Послідовність запитів клієнта сервера і відповідей на них буде виглядати приблизно наступним чином:

1. Передача запиту серверу А.
2. Отримання відповіді від сервера А.
3. Передача запиту серверу В.
4. Отримання відповіді від сервера В. До складу відповіді входить поле заголовка *SetCookie*. Отримавши його, клієнт записує *cookie* на диск.
5. Передача запиту серверу С. Незважаючи на те що на диску зберігається запис *cookie*, клієнт не робить ніяких спеціальних дій, оскільки значення *cookie* було записано з ініціативи іншого сервера.
6. Отримання відповіді від сервера С.
7. Передача запиту серверу А. У цьому випадку клієнт також ніяк не реагує на той факт, що на диску зберігається *cookie*.
8. Отримання відповіді від сервера А.
9. Передача запиту серверу В. Перед тим як сформулювати запит, клієнт визначає, що на диску зберігається запис *cookie*, створена після отримання відповіді від сервера В. Клієнт перевіряє, чи задовольняє даний запит деяким вимогам, і, якщо перевірка дає позитивний результат, включає в заголовок запиту поле *Cookie*.

Таким чином, процедуру запису та отримання *cookie* можна уявити собі як своєрідний «запит» серверу, інкапсульований в його відповіді клієнту. Відповідно отримання *cookie* також можна уявити собі як відповідь клієнта, інкапсульований у складі запиту того ж сервера.

Розглянемо докладніше, які дані передаються в поле заголовка *Set-cookie* і як вони впливають на поведінку клієнта.

Поле *Set-cookie* має наступний формат:

Set-cookie: ім'я = значення; expires = дата; path = шлях; домен = ім'я_домена, secure

де:

- *Set-cookie: ім'я = значення* – іменовані дані, які зберігаються за допомогою механізм *cookie*. Ці дані повинні зберігатися на клієнт-машині і передаватися сервера у складі чергового запиту клієнта;
- *expires = дата* – дата, яка є значенням параметра *expires*, визначає час, після закінчення якого інформація *cookie* втрачає свою актуальність. Якщо ключове слово *expires* відсутня, дані *cookie* видаляються після закінчення поточного сеансу роботи браузера.
- *path = шлях* – значення параметра *domain* визначає домен, з яким пов'язуються дані *cookie*. Щоб дізнатися, чи слід передавати у складі запиту дані *cookie*, браузер порівнює доменне ім'я сервера, до якого він збирається звернутися, з доменами, які пов'язані із записами *cookie*, що зберігаються на клієнт-машині. Результат перевірки буде вважатися позитивним, якщо сервер, якому направляється запит, належить домену, пов'язаному з *cookie*. Якщо відповідність не виявлено, дані *cookie* не передаються.
- *домен = ім'я_домена* – шлях, зазначений у якості значення параметра *path*, дозволяє виконати подальшу перевірку і прийняти остаточне рішення про те, чи слід передавати дані *cookie* у складі запиту. Крім домену із записом *cookie* зв'язується шлях. Якщо браузер виявив відповідність імені домена значенням параметра *domain*, він перевіряє, чи відповідає шлях до ресурсу шляху, пов'язаному з *cookie*. Порівняння вважається успішним, якщо ресурс міститься в каталозі, вказаному допомогою ключового слова *path*, або в одному з його підкаталогів. Якщо і ця

перевірка дає позитивний результат, дані *cookie* передаються сервера. Якщо параметр *path* в поле Set-Cookie відсутній, то вважається, що запис *cookie* пов'язана з URL конкретного ресурсу, переданого сервером клієнтові.

- *secure* – вказує на те, що дані *cookie* повинні передаватися по захищеному каналу.

Для передачі даних *cookie* сервера використовується поле заголовка *Cookie*. Формат цього поля досить простий:

Cookie: ім'я = значення; ім'я = значення; ...

За допомогою поля *Cookie* передається одна або декілька пар імен = значенню. Кожна з цих пар належить записи *cookie*, для якої URL запитуваного ресурсу відповідають імені домена і шляхи, зазначеним раніше в полі *Set-cookie*.

Лекція 3. Клієнтські сценарії та додатки.

План:

1. Програми, що виконуються на клієнт-машині.
2. Програми, що виконуються на сервері.
3. Насичені інтернет-додатки.
4. Введення в JScript.
5. Коротка характеристика VBScript.
6. Java-аплети.
7. ActionScript - загальна характеристика.
8. XAML і Microsoft Silverlight.
9. Поняття про DOM.
10. DHTML.
11. Регулярні вирази.

Література:

1. Храмцов П. Б. Основы WEB-технологий. / П. Б. Храмцов, С. А. Брик, А. М. Русак, А. И. Сурин – М. : ИТУИТ.РУ, 2003. – 512 с.
2. Шеперд Д. Освой самостоятельно XML за 21 день / Д. Шеперд. – М. : «Вильямс», 2002. – 432 с.

Як правило, Веб-додаток – додаток, в якому *клієнтом* виступає *браузер*, а *сервером* – *веб-сервер*.

Розглянемо типи програм, що забезпечують роботу Веб і використовують HTTP-протокол.

Ніякий HTTP-обмін неможливий без клієнта і сервера. Однак крім клієнта і сервера в веб-сеансі можуть брати участь й інші програми, які є об'єктом веб-програмування.

Результатом роботи веб-додатки є веб-сторінка, яка відображається у вікні браузера. При цьому саме веб-

додаток може виконуватися як на комп'ютері клієнта, так і на комп'ютері серверу.

Розглянемо докладніше обидві схеми.

1. Програми, що виконуються на клієнт-машині.

Одним з типів програм, призначених для виконання на клієнт-машині, є *сценарії*, наприклад, JavaScript (VBScript). Оригінальний текст сценарію являє собою частину веб-сторінки, тому сценарій JavaScript передається клієнту разом з документом, до складу якого він входить. Обробляючи HTML-документ, браузер виявляє вихідний текст сценарію і запускає його на виконання.

До всіх програм, які передаються з сервера на клієнт-машини і запускаються на виконання, пред'являється одна загальна вимога: *ці програми повинні бути позбавлені можливості звертатися до ресурсів комп'ютера, на якому вони виконуються*. Така вимога цілком обґрунтована. Адже передача по мережі і запуск Java-апплетів і JavaScript-сценаріїв відбувається автоматично без участі користувача, тому робота цих програм повинна бути *абсолютно безпечною для комп'ютера*. Іншими словами, мови, призначені для створення програм, що виконуються на клієнт-машині, повинні бути *абсолютно непридатні для написання вірусів і подібних програм*.

2. Програми, що виконуються на сервері.

Код програми, що працює на сервері, не передається клієнту. При отриманні від клієнта спеціального запиту, який передбачає виконання такої програми, сервер запускає її і передає параметри, що входять до складу запиту. Засоби для генерації подібного запиту зазвичай входять до складу HTML-документа.

Результати своєї роботи програма оформляє у вигляді HTML-документа і передає їх веб-серверу, а останній, у свою чергу, доповнює отримані дані HTTP-заголовком і передає їх клієнту. Взаємодія клієнта і сервера в цьому випадку показано на рис. 3.1.

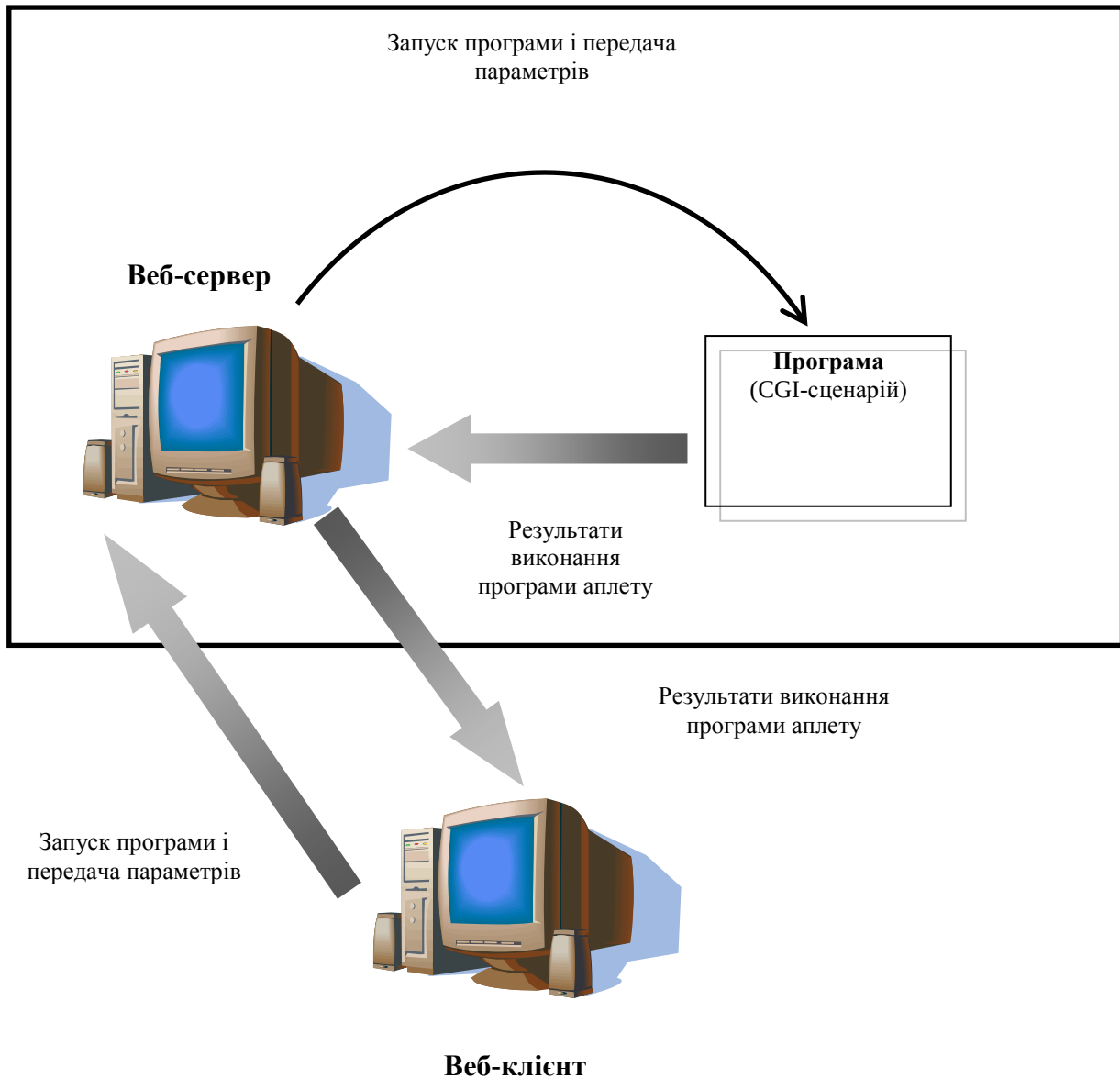


Рис. 3.1. Взаємодія клієнта з програмою, що виконується на сервері.

3. Насичені інтернет-додатки.

Насичений інтернет-додаток (Rich Internet application) - ще один підхід, який полягає у використанні *Adobe Flash* або *Java-апплетів* для повної або часткової реалізації інтерфейсу користувача, оскільки більшість браузерів підтримує ці технології (як правило, за допомогою плагінів).

Виникнення даного підходу обумовлена тим, що в рамках веб-додатків з «тонким» клієнтом взаємодія користувача з додатком реалізується в істотному ступені через сервер, що вимагає відправки даних на сервер, отримання відповіді від сервера і перезавантаження сторінки на стороні клієнта.

При використанні *Java-апплетів* до складу HTML-документа включається спеціальний дескриптор, що описує розташування файлу, що містить код апплету, на сервері. Після того як клієнт отримує HTML-код документа, що включає апплет, він генерує додатковий запит серверу. Після того як сервер пересилає клієнту код апплету, сам апплет запускається на виконання. Взаємодія між клієнтом і сервером при отриманні апплету показано на рис. 3.2.

При використанні насичених інтернет-додатків доводиться стикатися з такими проблемами:

- необхідність забезпечення безпечного середовища виконання («пісочниця»);
- для виконання коду повинно бути дозволено виконання сценаріїв;
- втрата в продуктивності (тому виконуємо на клієнтській стороні);
- потрібно багато часу на завантаження.

Для розробки насичених інтернет-додатків використовуються пакети *Curl*, *Adobe Flex* і *Microsoft Silverlight*.

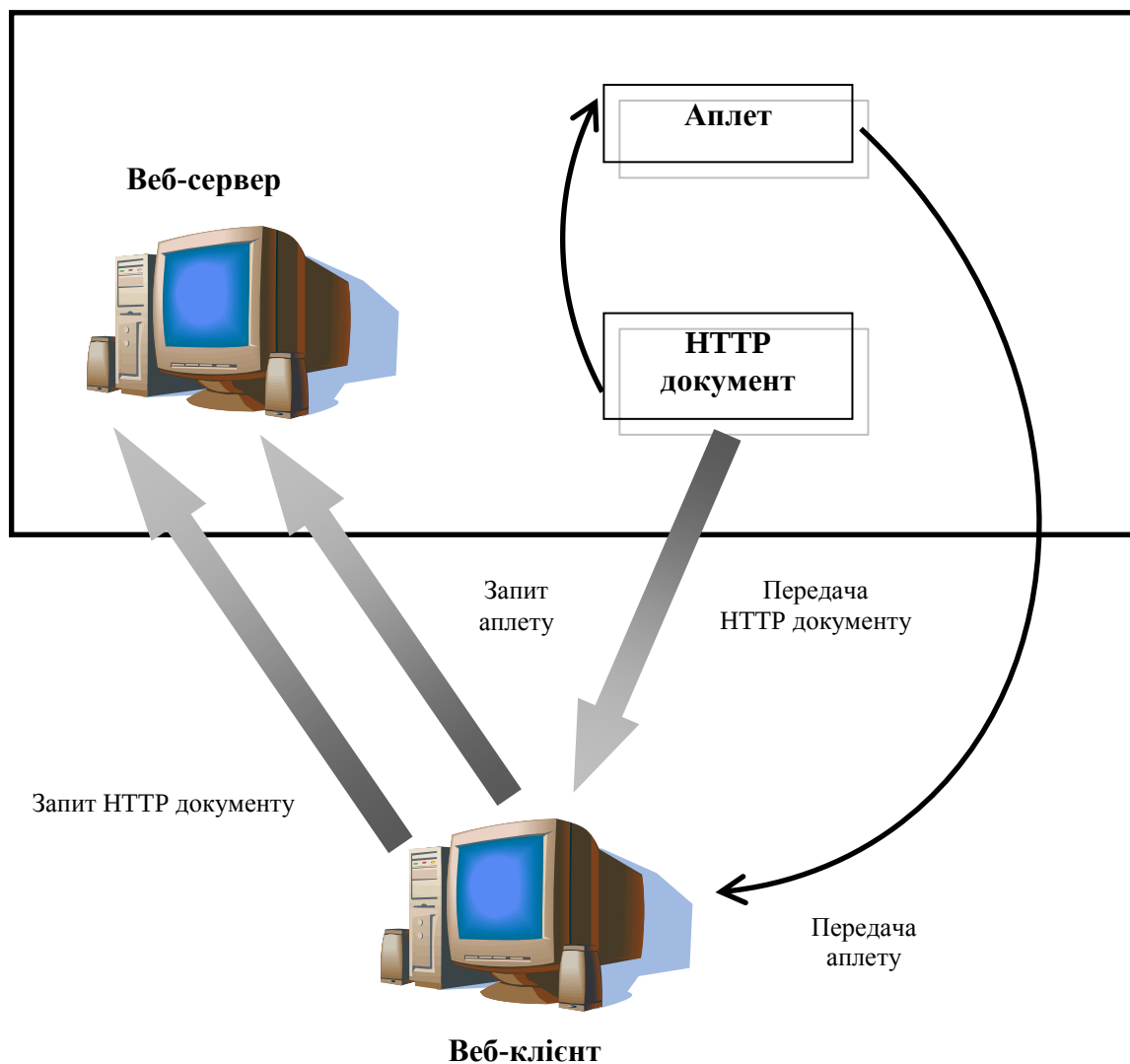


Рис. 3.2. Передача клієнтові Java-аплету.

4. Введення в JScript.

JavaScript – інтерпретована мова програмування, стандартизований міжнародною організацією ECMA в специфікації ECMA-262. Мови *JavaScript*, *JScript* і *ActionScript* є розширенням стандарту ECMA-262.

Назва «ECMAScript» стало фактично компромісом між організаціями, залученими в процес стандартизації, зокрема Netscape і Microsoft. Хоча *JavaScript* і *JScript* прагнули до сумісності з ECMAScript, вони мають ряд

додаткових можливостей не передбачених специфікацією ECMA.

Синтаксис JScript багато в чому аналогічний мови JavaScript, однак, крім додавання клієнтських скриптів на веб-сторінки і деяких інших функцій, JScript може використовуватися і для інших цілей, наприклад:

- автоматизація адміністрування систем Microsoft Windows;
- створення сторінок ASP.

Мова JScript отримав подальший розвиток у вигляді мови JScript.NET, який орієнтований на роботу в рамках платформи Microsoft.NET

JScript – інтерпретована, об'єктно-орієнтована мова. Хоча він має істотно меншу кількість можливостей, ніж такі об'єктно-орієнтовані мови як C++ і Java.

Можливості мови істотно обмежені:

- мова не дозволяє розробляти окремі програми;
- сценарії на JScript можуть виконуватися тільки за допомогою інтерпретатора, зокрема веб-браузером;
- JScript - мова без суворого контролю типів. Тому не потрібно оголошувати тип змінних явно. Крім того, у багатьох випадках JScript виконує перетворення автоматично, коли вони необхідні. Наприклад, при додаванні рядка і числа, число буде перетворено в рядок.

Код на JScript пишеться в текстовому форматі, і організований в *інструкції, блоки*, що складаються з пов'язаних наборів *інструкцій, і коментарів*. В межах інструкції можна використовувати змінні і дані, такі як рядки, числа і вирази. Для оголошення кінця інструкції крапку з комою (;). Група JScript-інструкцій, укладена у фігурні дужки {}, називається блоком.

Коментарем в JScript є текст, розташований після `//` до кінця рядка. Багаторядковий коментар починається з `/*`, і закінчується `*/`.

Знак рівності (`=`) використовується в JScript як присвоювання. Наступний код

```
Pi = 3.14;
```

мається на увазі «Присвоїти значення 3.14 змінної Pi».

При порівнянні двох значень на рівність застосовується подвійний знак рівності (`==`).

JScript вирази можна розділити на логічні або числові. Вирази містять деякі особливості, наприклад, символ `+` означає «додати до ...». Будь допустима комбінація значень, змінних, операторів, і інших виразів є виразом.

Оголошення змінної перед використанням є необов'язковим. Для цього використовується інструкція `var`. Інструкція `var` є обов'язковою при оголошенні локальної змінної всередині функції. Дозволяється оголошення змінної неявно – без інструкції `var`. Однак, у виразах застосовувати неоголошені змінні не допускається. JScript розрізняє регістр в імені змінної. `Name` і `name` розглядаються як різні змінні.

Типи даних.

JScript – мову з нестрогим контролем типів, змінні в JScript не мають строго фіксованого типу. Змінні мають тип, еквівалентний типу значення, яке вони містять. Проте, в деяких випадках, необхідно примусове перетворення змінної в певний тип. Числа можуть бути оголошені як рядки, а рядки необхідно перетворити в числовий тип. Для цього застосовують функції `parseInt ()` і `parseFloat ()`.

В JScript використовується шість типів даних. Основні з них – числа, рядки, об'єкти, логічний. Інші два – *null* і *undefined* (тобто невизначений).

Рядки оголошуються за допомогою подвійних лапок або апострофів. Рядок може складатися з нуля або більше символів *unicode*. Коли кількість символів дорівнює нулю, це називається порожнім рядком (*""*).

JScript підтримує числа як цілі, так і з плаваючою комою. Також існують спеціальні представлення чисел, наприклад *NaN* (не числом).

Приклади чисел:

3.14 // Дійсне число

15 // Ціле число

0177 // Вісімкове число 177

0XA8 // шістнадцятирічне число A8

Логічний тип допускає значення – *true* і *false*. Будь-яке вираження, рівне 0, вважається еквівалентним *false*, а будь-який вираз, рівне числу, відмінному від 0 буде еквівалентним *true*.

Undefined – означає, що тип не визначений. Значення *undefined* має змінна після її оголошення і до присвоєння їй якогось певного значення.

Змінна типу *null* – не має ніякого певного значення.

Оператори.

Мова підтримує умовні вирази *if* і *if ... else*. При використанні декількох умов одночасно можна використовувати оператори *||* (АБО) або *&&* (І).

В JScript підтримується кілька типів циклів: *for*, *for ... in*, *while*, *do ... while* і *switch*. Також існує інструкція зупинення виконання циклу. Оператор завершення *break* може використовуватися, щоб зупинити цикл, при виконанні якого-небудь умови. Інструкція *continue* використовується, щоб негайно перейти до виконання

наступної ітерації, пропускаючи іншу частину виконання коду поточної ітерації, але оновлюючи змінну-лічильник.

Функції та об'єкти.

В JScript є два види функцій: *вбудовані* і *зумовлені*. Програміст має можливість створювати власні функції. Визначення функції складається з оголошення параметрів і блоку інструкцій JScript.

Об'єкти в JScript, по-суті, є сукупність методів і властивостей. Всі об'єкти можна розділити на три види: вбудовані, створені і браузерні. Обробка об'єктів і масивів ідентична. Можна звернутися до будь-якої частини об'єкту (його властивостей і методів) або по імені, або за індексом. Нумерація індексів в JScript починається з нуля.

5. Коротка характеристика VBScript.

Visual Basic Scripting Edition (зазвичай просто VBScript) – сценарна мова програмування, що інтерпретується компонентом *Windows Script Host*. Він широко використовується при створенні скриптів в операційних системах сімейства Microsoft Windows.

Мова була створена компанією Microsoft як заміна застарілого пакетної мови, інтерпретованих додатком *command.com*. Синтаксис VBScript є спрощеною версією синтаксису мови *Visual Basic*.

Сценарії мови VBScript найчастіше використовуються в наступних областях, що використовують програмні продукти Microsoft:

- автоматизація адміністрування систем Windows;
- серверний програмний код в сторінках ASP;
- клієнтські сценарії в браузері Internet Explorer.

6. Java-аплети.

Java-апплет – це програма, написана на мові Java і відкомпілювати в байт-код. Виконується в браузері з використанням віртуальної Java-машини (JVM). Аплети використовуються для надання інтерактивних можливостей веб-додатків, які не можливі в HTML. Так як байт-код Java платформо-незалежний, то Java-аплети можуть виконуватися браузерами на багатьох операційних платформах.

Java-сервери є серверними додатками, але вони відрізняються від аплетів мовою, функціями та іншими характеристиками.

Призначені Java-аплети для виконання в безпечному середовищі з метою запобігання їх доступу до локальних ресурсів комп'ютера клієнта.

Код аплету завантажується з веб-сервера, і браузер:

- або вставляє аплет у веб-сторінку;
- або відкриває окреме вікно з власним користувача інтерфейсом аплету.

Аплет може бути впроваджений у веб-сторінку за допомогою використання HTML тега `<applet>`, або (що рекомендується) тега `<object>`.

Можна назвати наступні переваги Java-аплетів:

- працюють практично на більшості операційних платформ;
- підтримуються більшістю браузерів;
- кешируються в більшості браузерів, що істотно прискорює їх завантаження при поверненні на веб-сторінку;
- після першого запуску аплету, коли Java-машина вже виконується і швидко запускається, виконання аплетів відбувається значно швидше;
- завантажуються зі швидкістю порівнянної з програмами на інших компільованих мовах,

наприклад C ++, але у багато разів швидше ніж на JavaScript.

При цьому у Java-апплетів є й недоліки:

- потрібна установка Java-розширення, які доступні за замовчуванням не у всіх браузерах;
- проблеми реалізації Java-розширень для 64-розрядних процесорів;
- не можуть запускатися до першого завантаження віртуальної Java-машина, що може займати значний час;
- розробка користувацького інтерфейсу з використанням апплетів є більш складним завданням в порівнянні з HTML;
- не мають прямого доступу до локальних ресурсів компютера клієнта;
- деякі апплети прив'язані до використання певного середовища часу виконання Java (JRE).

7. ActionScript – загальна характеристика.

ActionScript – об'єктно-орієнтована мова програмування, один з діалектів EcmaScript, який додає інтерактивність, обробку даних та багато іншого у вміст Flash-додатків. ActionScript виконується віртуальною машиною (*ActionScript Virtual Machine*), яка є складовою частиною додатка *Flash Player*. ActionScript компілюється в байткод, який включається в SWF-файл.

SWF-файли виконуються Flash Player. Сам Flash Player існує у вигляді плагіна до веб-браузера, а також як самостійне виконуваний додаток. У другому випадку можливе створення виконуваних exe-файлів, коли swf-файл включається у Flash Player.

За допомогою ActionScript можна створювати інтерактивні мультимедіа-додатки, ігри, веб-сайти та багато іншого.

8. XAML i Microsoft Silverlight.

XAML (*eXtensible Application Markup Language*) – мова інтерфейсів платформи Windows Vista.

Модель додатків Vista включає об'єкт *Application*. Його набір властивостей, методів і подій дозволяє об'єднувати веб-документи в пов'язану програму. Об'єкт *Application* контролює виконання програми і генерує події для користувацького коду. Документи додатки створюються за допомогою мови XAML, який описує, насамперед, користувацький інтерфейс. Логіка додатка управляється процедурним кодом (C #, VB та ін.). XAML включає основні чотири категорії елементів: панелі, елементи управління, елементи, пов'язані з документом і графічні фігури.

Microsoft Silverlight є офіційною назвою заснованої на XML і .NET технології під кодовим ім'ям WPF / E (*Windows Presentation Foundation Everywhere*), що є альтернативною Adobe Flash. Являє собою підмножину Windows Presentation Foundation, в якому реалізовані векторна графіка, анімація і засоби відтворення відео. Версія 1.1 включає в себе повну версію .NET CLR – так звану CoreCLR, що дозволить розробляти *Silverlight* додатки на будь-якому з мов .NET. *Silverlight v.1.0* містить модуль браузера для обробки XAML і кодеки для відтворення мультимедійного вмісту в форматах WMV, WMA і MP3.

Microsoft Silverlight представляє браузеру внутрішню модель DOM, керовану з JavaScript коду. Оскільки мова XAML заснований на XML, то документ, що визначає завантажувється клієнту користувацький інтерфейс – текстовий і тому цілком придатний для індексування пошуковими системами. Використовуючи модель DOM, JavaScript може динамічно оновлювати вміст *Silverlight*, аналогічно DHTML.

Також можна викликати методи управління презентацією (запуску анімації або призупинити відтворення відео, наприклад).

Silverlight-додаток починається з виклику об'єкту Silverlight з HTML сторінки, який завантажує XAML файл. XAML файл містить об'єкт *Canvas*, який виступає підкладкою для інших елементів. Об'єкти XAML здатні генерувати події, перехоплювати з JavaScript.

9. Поняття про DOM.

DOM (*Document Object Model*) – об'єктна модель документа. Це незалежний від платформи і мови програмний інтерфейс, що дозволяє програмам отримувати доступ до вмісту документів, а також змінювати вміст, структуру і вид документів.

В рамках DOM будь-який документ представляється у вигляді дерева вузлів. Кожен вузол являє собою елемент, атрибут, текстовий, графічний або будь-який інший об'єкт. Вузли між собою перебувають у відношенні «батько-нащадок».

Спочатку різні браузері мали власні моделі DOM, не сумісні з іншими. Для того, щоб забезпечити взаємну і зворотну сумісність, консорціум W3C класифікував цю модель по рівнях, для кожного з яких була створена своя специфікація. Всі ці специфікації об'єднані в загальну групу, що носить назву W3C DOM:

1. Рівень 0. Включає в себе всі специфічні моделі DOM, які існували до появи Рівня 1, наприклад `document.images`, `document.forms`. Ці моделі формально не є специфікаціями DOM, опублікованими W3C, а швидше відображають те, що існувало до початку процесу стандартизації.
2. Рівень 1. Базові функціональні можливості DOM (HTML і XML) в документах, такі як отримання

дерева вузлів документа, можливість змінювати і додавати дані.

3. Рівень 2. Підтримка простору назв XML, *filtered views* і подій.
4. Рівень 3. Складається з шести різних специфікацій:
 - DOM Level 3 *Core*;
 - DOM Level 3 *Load and Save*;
 - DOM Level 3 *XPath*;
 - DOM Level 3 *Views and Formatting*;
 - Level 3 *Requirements*;
 - DOM Level 3 *Validation*.

Поточним рівнем специфікацій DOM є Рівень 2, але, тим не менш, деякі частини специфікацій Рівня 3 є рекомендованими W3C.

10. DHTML.

Динамічний HTML або DHTML являє собою набір технологій, які спільно дозволяють створювати інтерактивні веб-сайти на основі статичного мови розмітки (HTML), мови створення клієнтських сценаріїв (JavaScript), мови опису представлення документа (CSS) і документної об'єктної моделі (DOM).

DHTML дозволяє сценарним мовам змінювати змінні мови опису представлення документа, таким чином, змінюючи вигляд і поведінка перш статичного вмісту HTML документа вже після повного завантаження документа і в процесі перегляду його користувачем. Таким чином, динамічність, вноситься DHTML, проявляє себе в процесі перегляду сторінки, але не має ніякого відношення до генерації вмісту сторінки при кожній її завантаженні.

На противагу DHTML, динамічно генерується сторінка – більш широке поняття, що припускає, наприклад генерацію вмісту веб-сторінки індивідуально

для кожного користувача. Це досягається створенням сторінок за допомогою клієнтських або серверних (наприклад, на PHP або Perl) сценаріїв.

11. Регулярні вирази.

Регулярні вирази – система пошуку текстових фрагментів в електронних документах, заснована на спеціальній системі запису зразків для пошуку. Зразок, що задає правило пошуку, називається «шаблоном». Застосування регулярних виразів принципово перетворило технології електронної обробки текстів.

Багато мови програмування підтримують регулярні вирази для роботи з рядками або у вигляді окремих функцій, або мають вже вбудований в їх синтаксис механізм обробки регулярних виразів, наприклад, *Perl* і *Tcl*. Популяризації поняття регулярних виразів сприяли утиліти, що поставляються в дистрибутивах *Unix*.

За допомогою регулярних виразів можна задавати структуру шуканого шаблону і його позицію всередині рядка (наприклад, на початку або в кінці рядка, на кордоні або не на межі слова).

При описі структури шаблону використовуються:

- гнучка система квантифікаторів (операторів повторення);
- оператори опису наборів символів і їх типу (числові, нечислові, спеціальні).

Лекція 4. Серверні веб-додатки.

План:

1. Стандарт CGI.
2. Сценарії веб-додатків.
3. Мова Python.
4. Мова Ruby.
5. Веб-додаток ASP.
6. Веб-додаток ISAPI.

Література:

1. Беллиньясо М. Разработка Web-приложений в среде ASP.NET 2.0: с примерами на С# / М. Беллиньясо. – М.: «Диалектика», 2007. – 640 с.
2. Старыгин А. XML: разработка Web-приложений / А. Старыгин. – СПб.: «БХВ-Петербург», 2003. – 585 с.
3. Храмцов П. Б. Основы WEB-технологий. / П. Б. Храмцов, С. А. Брик, А. М. Русак, А. И. Сурин – М.: ИТУИТ.РУ, 2003. – 512 с.

Для розширення можливостей клієнт-серверної взаємодії в рамках протоколу HTTP крім створення на клієнтській стороні розширень стандартних можливостей, що надаються мовами розмітки і браузерами, можна також розробляти на стороні веб-сервера *додатки, плагіни і сценарії*, що розширюють можливості самого веб-сервера.

Плагін (plug-in) – незалежно компільований програмний модуль, що динамічно підключається до основної програми, призначений для розширення або використання її можливостей. Зазвичай виконуються у вигляді поділюваних бібліотек.

Сценарій (скрипт, script) – програма, яка автоматизує деяку задачу, яку користувач зазвичай виконує вручну, використовуючи інтерфейси програми.

1. Стандарт CGI.

Коло завдань, що вирішуються Web-сервером, обмежене. В основному воно зводиться до підтримки HTTP-взаємодії та доставки клієнту Web-документів. Будь-які «нестандартні» дії реалізуються за допомогою спеціальної програми, яка взаємодіє з веб-сервером і клієнтом. Ця взаємодія підпорядковується певним правилам.

Основний набір таких правил – *стандарт CGI* (Common Gateway Interface – інтерфейс загального шлюзу), який визначає порядок запуску програми на комп'ютері-сервері, способи передачі програмою параметрів і доставки результатів її виконання клієнту. Програма, написана за правилами CGI, називається CGI-сценарієм (script CGI), хоча це не означає, що на сервері не може виконуватися двійковий файл.

Завдяки цьому інтерфейсу для розробки додатків можна використовувати будь-яку мову програмування, яка володіє засобами взаємодії зі стандартними пристроями введення/виведення. Такими можливостями володіють також сценарії для вбудованих командних інтерпретаторів операційних систем.

Виконання будь-якої програми (у тому числі CGI-сценарію) можна умовно розділити на п'ять етапів:

1. Запуск програми.
2. Ініціалізація і читання вихідних даних.
3. Обробка даних.
4. Виведення результатів виконання.
5. Завершення програми.

Відмінності між CGI-сценарієм і консольним додатком стосуються першого, другого і четвертого етапів виконання.

Кожен раз, коли *веб-сервер* отримує запит від клієнта, він аналізує зміст запиту і повертає відповідну відповідь:

- Якщо запит містить вказівку на *файл*, що знаходиться на жорсткому диску, то сервер повертає в складі відповіді цей *файл*.
- Якщо запит містить вказівку на *програму* і необхідні для неї *аргументи*, то сервер виконує програму і *результат* її роботи повертає клієнту.

CGI визначає:

- яким чином *інформація* про сервер і запит клієнта передається програмі у формі *аргументів* і *змінних оточення*;
- яким чином програма може передавати назад додаткову інформацію про результати (наприклад про тип даних) у формі *заголовків* відповіді сервера.

У переважній більшості випадків запуск CGI-сценарію здійснюється клацанням на кнопці *Submit*, сформованої за допомогою дескриптора `<input type = "submit">`, який знаходиться на HTML-сторінці між `<form>` і `</ form>`. Не знаючи призначення атрибутів *action* і *method*, неможливо зрозуміти, як відбувається виклик програми і передача параметрів.

Значенням атрибута *action* дескриптора `<form>` є URL файл, що містить код CGI-сценарію. Так, наведений нижче вираз означає, що файл з кодом CGI-сценарію знаходиться на сервері *www.myhp.edu* в каталозі *cgi-bin* у файлі *script.pl*.

```
<Form action="http://www.myhp.edu/cgi-bin/script.pl"
method="post">
```

Як веб-сервер розрізняє, що треба зробити з файлом, на який вказує URL, – передати його зміст клієнтові або запустити файл на виконання? Існує два способи розпізнавання файлів, що містять тексти CGI-сценаріїв.

Перший спосіб полягає в тому, що при установці веб-серверу один з каталогів спеціально виділяється для зберігання сценаріїв. Зазвичай такий каталог отримує ім'я *cgi-bin* (або *Scripts* для веб-серверу *IIS*). У цьому випадку, якщо клієнт запитує файл з каталогу *cgi-bin*, сервер сприймає такий запит як команду на запуск сценарію. Файли з інших каталогів інтерпретуються як HTML-документи.

Другий спосіб використовує розширення файлу. При налаштування сервера вказується, що файли з певними розширеннями містять коди сценаріїв.

Ідентифікація з розширення використовується відносно рідко. Найчастіше всі сценарії поміщаються в */cgi-bin*, */Scripts* або в інший каталог, спеціально виділений для їх зберігання.

Виведення результатів виконання CGI-сценарію здійснюється надзвичайно просто. Для того щоб дані були передані клієнту, досить вивести їх у стандартний вихідний потік. Однак, розробляючи CGI-сценарій, не слід забувати про те, що він все ж відрізняється від консольної програми і має такі особливості.

Інформація, передана клієнту, повинна відповідати протоколу HTTP, тобто складатися із заголовка і тіла відповіді. Як правило, отримавши дані від сценарію, сервер самотійно додає перший рядок заголовка.

HTTP/1.0 200 OK

Формування інформаційних полів, що входять до складу заголовка – завдання сценарію. Щоб дані, передані сценарієм, були правильно інтерпретовані клієнтом,

необхідно, щоб в заголовку було присутнє як мінімум поле *Content-type*. За заголовком повинен слідувати порожній рядок. При відсутності полів заголовка реакція браузера буде непередбачуваною. У подібних випадках браузер зазвичай намагається відобразити отриману інформацію як текстовий файл.

Самий природний формат для браузера – формат HTML. Результати роботи сценарію зазвичай оформляються у вигляді веб-сторінки, тобто повертаються дані слід доповнити дескрипторами HTML. Таким чином, відповідь CGI-сценарію клієнту зазвичай виглядає так:

```
Content-type: text/html

<html>
<head>
<title> відповідь сценарію</title>
</head>
<body>
.....
</body>
</html>
```

Зверніть увагу на порожній рядок після вираження *Content-type: text/html*. Він обов'язково має бути присутній у відповіді, в іншому випадку клієнт сприйме всі наступні дані як продовження заголовка.

Після компіляції програми необхідно скопіювати виконуваний файл в каталог *cgi-bin* (або в інший каталог, призначений для розміщення виконуваних файлів) з якого він може запускатися веб-сервером на виконання за запитом клієнта.

Для виклику даного сценарію досить включити у веб-сторінку цей фрагмент HTML-коду:

```
<form method="post" action="/cgi-bin/hello.exe">  
<input type="submit">  
</form>
```

Якщо сценарій викликається з форми, йому передаються ті дані, які користувач ввів за допомогою інтерактивних елементів, що відображаються на веб-сторінці – передача інформації CGI-сценарію здійснюється в два етапи: спочатку браузер передає дані веб-серверу, потім веб-сервер передає їх сценарію.

У більшості випадків крім кнопки *Submit* форма містить інші інтерактивні елементи, кожен з яких має ім'я (атрибут NAME) і значення (атрибут VALUE, або послідовність символів, введена користувачем). З імен елементів і їх значень формується рядок параметрів, який має наступний формат:

$$ім'я=значення\&ім'я=значення\&... \&ім'я=значення$$

Кожен параметр являє собою ім'я керуючого елемента та його значення, розділені знаком рівності, а кілька таких пар об'єднують рядок за допомогою символу «&». Якщо до складу імені або значення входить символ «&» або «=», то подібні символи кодуються в послідовність знака відсотка «%», за яким слідують дві шістнадцятирічні цифри, що визначають код символу. Так, наприклад, послідовністю «% 21» кодується знак оклику «!». Як правило, при передачі параметрів трьохсимвольними послідовностями замінюються всі знаки, крім латинських букв, цифр та символу пробілу (останній замінюється знаком «+»).

Таким чином, перед використанням рядка параметрів його треба декодувати. Алгоритм декодування надзвичайно простий і включає в себе наступні дії:

1. Виділити з рядка параметрів пари ім'я = значення.

2. Виділити з кожної пари ім'я і значення.
3. У кожному імені та кожному значенні замінити символи «+» пробілами.
4. Кожну послідовність з символу «%» і двох шістнадцятирічних перетворити в ASCII-символ.

Атрибут *method* дескриптора <form> має або значення «GET», або значення «POST». Значення «GET» і «POST» визначають два різних методи передачі параметрів сценарію:

1. Якщо атрибут *method* має значення «GET», рядок параметрів передається разом з URL сценарієм, що викликається. Роздільником між URL і рядком параметрів є символ «?».
2. Якщо атрибут *method* має значення "POST", рядок параметрів передається в тілі HTTP-запиту.

Розглянемо, як повинен вести себе CGI-сценарій, щоб правильно обробити дані залежно від методу, використаного при передачі даних, рядок параметрів доставляється CGI-сценарію різними способами.

Якщо атрибут METHOD дескриптора <FORM> мав значення «GET», рядок параметр передається серверу в якості значення змінної оточення QUERY_STRING.

При використанні методу POST дані доставляються сценарієм по-іншому. Вони передаються через стандартний потік введення (STDIN). Щоб сценарій зміг визначити, скільки символів слід читати зі стандартного вводу, веб-сервер встановлює значення змінної оточення CONTENT_LENGTH, рівним довжині рядка параметрів.

Отримавши управління, сценарій в першу чергу повинен з'ясувати, за допомогою якого методу виконувалася передача параметрів. Ця інформація міститься в змінній оточення REQUEST_METHOD.

Таким чином, в простому випадку, щоб виконати обробку рядка параметрів, досить знати призначення трьох

змінних оточення: REQUEST_METHOD, QUERY_STRING і CONTENT_LENGTH.

```
$method = $ENV{'REQUEST_METHOD'};

if ($method eq "GET")
{ $pars = $ENV{'QUERY_STRING'}; }
else
{ $length = $ENV{'CONTENT_LENGTH'}; }

read (STDIN, $pars, $length);

print "Content-type: text/html\n\n";
print "<HTML><BODY>\n";
print "<P>METHOD = ", $method;
print "<P>String of parameters: <P>\n";
print $pars;
print "</HTML></BODY>\n";
```

Приклад сценарію на мові Perl, який повертає клієнту рядок параметрів, наведено нижче. Сценарій визначає, який метод використовувався для передачі даних, читає рядок параметрів і передає її клієнту, попередньо доповнивши HTML-дескрипторами.

При розробці більш складних сценаріїв може знадобитися додаткова інформація. Інформація про типи сервера і браузера, адресу клієнт-машини і багато інших відомості передаються за допомогою змінних оточення. Деякі з них перераховані нижче:

REMOTE_ADDR	IP-адреса вузла, з якого надійшов запит
REMOTE_HOST	Доменне ім'я вузла, з якого надійшов запит
SERVER_PORT	Номер порту, який використовувався при зверненні до сервера
SERVER_SOFTWARE	Ім'я та версія сервера, за допомогою якого був запущений сценарій
SERVER_NAME	Ім'я або адреса вузла, на якому виконується

	сервер
SERVER_PROTOCOL	Назва і версія протоколу, за допомогою якого був переданий запит
HTTP_USER_AGENT	Клієнтська програма, що відправила запит серверу
HTTP_REFERER	URL документа, що відображається браузером при виклику сценарію

2. Сценарії веб-додатків.

До основних переваг розробки додатків на стороні веб-сервера у формі сценаріїв можна віднести наступні:

- оскільки сценарії не компілюються а інтерпретуються, то помилки в сценарії викличуть тільки діагностичне повідомлення, але не приведуть до дестабілізації веб-сервера або операційної системи;
- кращі виразні можливості. Мова сценаріїв як правило має власний проблемно-орієнтований набір команд, і один рядок сценарію може робити те ж, що кілька десятків рядків традиційною мовою. Як наслідок, на цій мові може писати програміст низької кваліфікації
- підтримка багатоплатформеності.

Оскільки сценарії інтерпретуються з вихідного коду динамічно при кожному виконанні, вони виконуються зазвичай значно повільніше готових програм, що транслювалися в машинний код на етапі компіляції.

У плані швидкодії *сценарні мови* можна розділити на:

1. *Мови динамічного розбору* (наприклад *command.com*). Інтерпретатор зчитує інструкції з файлу програми мінімально необхідними блоками, і виконує ці блоки, не читаючи подальший код.
2. *Попередньо компільовані* (наприклад *Perl*). Спочатку зчитується вся програма, потім компілюється або

в машинний код, або в один із внутрішніх форматів, після чого отриманий код виконується.

Коротко розглянемо найбільш відомі мови розробки сценаріїв для веб-додатків.

3. Мова Python.

Python – високорівнева мова програмування загального призначення з акцентом на продуктивність і читаність коду. Мова *Python* поєднує в собі мінімалізм синтаксису ядра і великий обсяг корисних функцій стандартної бібліотеки.

Python підтримує структурну, об'єктно-орієнтовану, функціональну, імперативну і аспектно-орієнтовану парадигми.

Його основні архітектурні риси:

- динамічна типізація;
- автоматичне управління пам'яттю;
- повна інтроспекція; механізм обробки виключень;
- підтримка багатопотокових обчислень;
- зручні високорівневі структури даних

Код в *Python* організовується у функції і класи, які можуть об'єднуватися в модулі (які в свою чергу можуть бути об'єднані в пакети).

Для всіх основних платформ *Python* має підтримку характерних для даної платформи технологій (наприклад, *Microsoft COM/DCOM*). Існує навіть спеціальна версія *Python* для віртуальної машини *Java* – *Jython*, що дозволяє інтерпретатору виконуватися на будь-якій системі, що підтримує *Java*, при цьому класи *Java* можуть безпосередньо використовуватися з *Python* і навіть бути написаними на *Python*. Кілька проектів забезпечують інтеграцію з платформою *Microsoft.NET*, основні з яких – *IronPython* і *Python.Net*.

Стандартна бібліотека *Python* має засоби для роботи з багатьма мережевими протоколами і форматами інтернету, наприклад, модулі для написання HTTP-серверів і клієнтів, для розбору і створення поштових повідомлень, для роботи з XML і таке інше. Набір модулів для роботи з операційною системою дозволяє писати крос-платформні додатки. Існують також модулі для роботи з регулярними виразами, текстовими кодуваннями, мультимедійними форматами, криптографічними протоколами, архівами, серіалізаційними даних, підтримкою юніт-тестування та ін.

Крім стандартної бібліотеки існує безліч бібліотек, що надають інтерфейс до всіх системним викликам на різних платформах. Є велика кількість прикладних бібліотек для *Python* в найрізноманітніших областях (веб, бази даних, обробка зображень, обробка тексту, чисельні методи, додатки операційної системи і таке інше.).

4. Мова Ruby.

Ruby – інтерпретована мова високого рівня для швидкого і зручного об'єктно-орієнтованого програмування. Ruby володіє незалежною від операційної системи реалізацією багатопоточності, суворою динамічною типізацією, «складальником сміття» і багатьма іншими можливостями. Багато особливостей синтаксису і семантики мови *Perl* запозичені в Ruby.

Перша загальнодоступна версія Ruby з'явилася в 1995 р.

Ruby – повністю об'єктно-орієнтована мова:

1. Всі дані є об'єктами, на відміну від багатьох інших мов, де існують примітивні типи.
2. Кожна функція є методом.
3. Змінні Ruby містять не самі об'єкти, а посилання на них.

4. Присвоєння – це не передача значення, а копіювання *посилання* на об'єкт.
5. В Ruby можна додавати методи не тільки в будь-які класи, але й у будь-які об'єкти. Наприклад, можна додати до деякого рядку довільний метод.

Масиви в Ruby можуть автоматично змінювати розмір, можуть містити будь-які елементи і мова надає потужні засоби для їх обробки.

Ruby поставляється з великою стандартною бібліотекою. Це, насамперед, бібліотеки для роботи з різними *мережними протоколами* на стороні *сервера* і *клієнта*, засоби для роботи з різними форматами представлення даних (XML, XSLT, YAML, PDF, RSS, CSV, WSDL). Також є бібліотеки для роботи з *архівами*, *датами*, *кодуваннями*, *матрицями*, засоби для *системного адміністрування*, *розподілених обчислень*, підтримки *багатопоточності* і таке інше.

У мові Ruby також реалізований простий і зручний механізм для розширення мови за допомогою бібліотек, написаних на C, що дозволяє легко розробляти додаткові бібліотеки. Для уніфікованого доступу до баз даних розроблена бібліотека *Ruby DBI*.

До недоліків інтерпретатора Ruby можна віднести наступні:

1. Невисока швидкість роботи.
2. Відсутність підтримки потоків операційної системи (для Unix-подібних операційних систем є підтримка процесів ОС), є в експериментальній версії 1.9.
3. Відсутність вбудованої підтримки юнікода (можлива робота з використанням додаткових бібліотек, є в експериментальній версії 1.9).
4. Відсутність компіляції в байткод. (При цьому є можливість компілювати Ruby в Java і .NET байткод, використовуючи компілятор JRuby і

Ruby.NET). В експериментальну версію 2.0 входить віртуальна машина YARV, компілює Ruby в байткод і суттєво прискорює виконання.

5. Веб-додаток ASP.

ASP (Active Server Pages) – технологія, розроблена компанією *Microsoft*, що дозволяє легко створювати додатки для Веб.

Програмування на *ASP* дає розробникам доступ до інтерфейсу програмування додатків *Internet Information Server* за допомогою мови сценаріїв *VBScript* і *JScript*.

ASP працює на платформі операційних систем лінії *Windows NT* і на веб-сервері *Microsoft IIS*.

Архітектура *ASP* представлена на рис. 4.1.

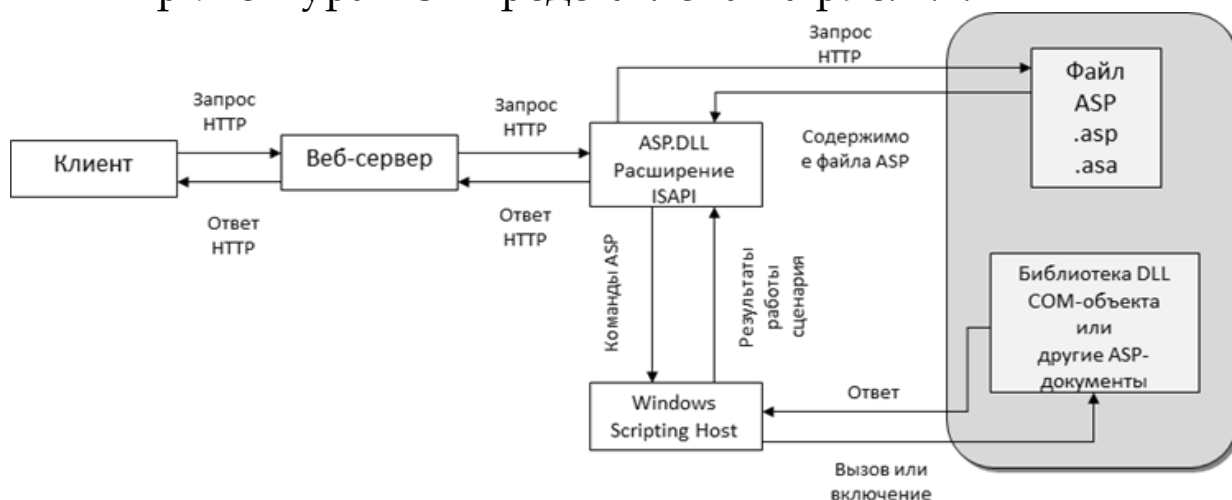


Рис. 4.1. Архітектура ASP.

Файли *ASP* являють собою сценарії, що інтерпретуються в міру надходження запитів. *ISAPI*-розширення *ASP.DLL* пов'язано в *IIS* з розширеннями файлів *.asp* або *.asa*.

Порядок обробки таких файлів виглядає наступним чином:

1. *ASP.DLL* переглядає файли із зазначеними розширеннями на наявність тегів, що позначають впроваджений код для виконання на сервері і передає знайдений код в *Windows Script Host (WSH)*.
2. *WSH* виконує цей код і повертає результат файлу *ASP.DLL*.
3. *ASP.DLL* передає *IIS* цей результат і вміст самого файлу *ASP*.
4. *IIS* повертає відповідь клієнту, від якого надійшов запит.

Розглянемо основи синтаксису *ASP*.

IIS розрізняє код, що виконується на сервері, і вміст, що відправляється клієнтові за допомогою *ASP.DLL*, аналізуючи файл *ASP* на наявність початкового «<%» і кінцевого «%>» тегів і виконуючи код, розташований між ними, за допомогою *WSH*.

Розглянемо приклад:

```
<% Language=VBScript %>
<HTML>
  <BODY>
    <%
      Response.Write("<p>Hello world!</p>")
    %>
  </BODY>
</HTML>
```

У прикладі перший рядок коду `<%Language=VBScript%>` повідомляє про необхідність використовувати інтерпретатор мови *VBScript*. Для вставки рядка в документ був використаний метод *Write* стандартного об'єкту *Response*.

Подія веб-запиту в *ASP* обробляється за допомогою наступних об'єктів:

- *Response*. Використовується для запису даних в запит *HTTP*, що повертається клієнту;
- *Application*. Містить параметри і конфігурації з налаштування роботи ASP для даного веб-сайту;
- *Request*. Зберігає вміст *HTTP*-запиту і забезпечує допоміжні функції для обробки даних *HTTP*-запиту;
- *Server*. Містить інформацію про веб-сервер, веб-сайт, а також забезпечує підтримку викликаючої програми;
- *Session*. Являє собою стан заданого веб-сеансу із заданим хостом клієнтом.

6. Веб-додаток ISAPI.

Для веб-сервера IIS (Internet Information Server). був розроблений спеціальний програмний інтерфейс для створення додатків, що розширюють стандартні можливості веб-сервера.

ISAPI (Internet Server Application Programming Interface) – багатоланковий API для IIS.

ISAPI також реалізований у вигляді модуля *mod_isapi* для веб-сервера *Apache*. Таким чином, серверні додатки, розроблені для MS IIS можуть також виконуватися в *Apache* та інших веб-серверах.

На противагу CGI – *ISAPI*-додаток завантажується в тому ж адресному просторі, що й веб-сервер IIS. Це дозволяє підвищити продуктивність додатків завдяки скороченню витрат на запуск окремих процесів. Однак збій *ISAPI*-додатку може привести до нестійкої роботи самого веб-сервера. У другій версії IIS є можливість запуску додатків в рамках окремого процесу.

ISAPI включає в себе 2 компоненти: розширення і фільтри.

Таким чином, все різноманіття розроблюваних *ISAPI*-додатків зводиться тільки до цих двох типів. І *фільтри* і *розширення* компілюються в *DLL* файли, що динамічно запускаються веб-сервером.

ISAPI додатки можуть розроблятися за допомогою будь-яких мов, що підтримують експорт стандартних *C*-функцій, наприклад *C*, *C++*, *Delphi Pascal*. Для розробки є обмежене число бібліотек для розробки *ISAPI* додатків, наприклад *Intraweb*-компоненти *Delphi Pascal*, спеціальні *MFC*-класи, спеціальна *C++* бібліотека серверних технологій *ATL*.

До найбільш важливих особливостей *ISAPI*-розширень можна віднести наступні:

- *ISAPI*-розширення мають доступ до всіх функціональних можливостей *IIS*;
- реалізуються у вигляді *DLL*-модулів, що завантажуються в просторі процесу, контрольованого *IIS*;
- клієнти можуть звертатися до *ISAPI*-розширень так само як до статичних *HTML* сторінок;
- *ISAPI*-розширення можуть бути асоційовані з окремими розширеннями файлів, з цілими каталогами або сайтами.

ISAPI-фільтри необхідні для зміни або вдосконалення функціональності *IIS*. Вони зазвичай працюють з *IIS*-сервером і фільтрують кожен запит. Фільтри застосовуються для аналізу і модифікації вхідних і вихідних потоків даних.

Фільтри також як і розширення реалізуються у вигляді *DLL* файлів.

Зазвичай *ISAPI*-фільтри використовуються для вирішення наступних завдань:

- Зміна даних в запиті клієнта (*URL* або заголовків).
- Управління відображенням *URL* в фізичні файли.

- Управління іменами і паролями користувачів при анонімної або базової аутентифікації.
- Аналіз та модифікація запитів по завершенні аутентифікації.
- Модифікація відповіді веб-сервера.
- Ведення журналів та аналіз трафіку.
- Реалізація власної аутентифікації.
- Управління шифруванням і стисненням.

Варто зазначити, що існують реалізації у вигляді ISAPI-розширень для таких інструментальних засобів як:

- ASP (Active Server Pages);
- ASP.NET;
- ColdFusion;
- Perl ISAPI (Perlis);
- PHP.

Лекція 5. Мови розробки сценаріїв Perl і PHP.

План:

1. Мова Perl.
2. Мова PHP.

Література:

1. Мельтцер К. Разработка CGI-приложений на Perl / К. Мельтцер. – М. : «Вильямс», 2001. – 395 с.
2. Шапошников И. PHP 5.1: учебный курс / И. Шапошников. – СПб.: «Питер», 2007.– 192 с.

1. Мова Perl.

Мова *Perl* (*Practical Extraction and Report Language*) – це мова програмування, сильними сторонами якого вважаються його багаті можливості для роботи з текстом, у тому числі реалізовані за допомогою регулярних виразів. Також мова відома тим, що має величезну колекцію додаткових модулів CPAN.

Щоб запустити програму мовою Perl на її виконання компіляція не потрібна, вона цілком може виконуватися під управлінням інтерпретатора. Щоб файл з вихідним текстом Perl можна було запускати на виконання, треба щоб перший його рядок виглядав так:

#!/ шлях_до_інтерпретатора_Pperl

Основними типами даних у мові є: скалярні, масиви (скалярні), хеш-таблиці (асоціативні масиви), функції, файлові дескриптори і константи.

Скалярні змінні використовуються для зберігання одиночних значень. Вони можуть містити числа, рядки і

посилання на інші об'єкти. Перед ім'ям скалярної змінної необхідно ставити знак долара «\$». Тип скалярної змінної не фіксований і визначається динамічно залежно від контексту.

Змінні різних типів відрізняються знаком, який стоїть перед ім'ям змінної:

\$ A – скаляр або показник

@b – скалярний масив

% C – асоціативний масив (хеш-таблиця)

& D – функція

F – дескриптор введення-виведення або константа

Скалярний масив є впорядкованим списком скалярів. Кожен елемент масиву має порядковий номер (індекс), за допомогою якого до нього можна отримати доступ. Нумерація елементів починається з нуля.

Перед ім'ям змінного типу скалярний масив вказується знаком «@», а для доступу до певного елемента масиву необхідно ставити знак «\$», так як певний елемент масиву є скаляром:

```
winter = ("грудень", "січень", "лютий");  
print "Другий місяць зими", $ winter [1], "\ n";
```

Хеш-таблиця являє собою асоціативний масив, що дозволяє асоціювати рядок (ключ) зі скаляром (значення). Рядок при цьому називається ключем, а скаляр в хеш-таблиці – значенням. Перед ім'ям змінної-списку необхідно ставити знак відсотка «%», а для доступу до певного елемента масиву ставлять знак «\$».

Фактично хеш-таблиця являє собою масив, де в непарних позиціях знаходяться ключі, а на парних – значення.

Використання асоціативних масивів нагадує використання масивів скалярних значень, проте індексація проводиться не цілими числами, а ключовими словами. Крім того, індекси полягають не в квадратні, а у фігурні дужки.

Так, наприклад, для того щоб привласнити значення трьом елементам масиву *%dict* з індексами *first*, *second* та *third*, можна скористатися одним із двох способів, зазначених нижче.

```
$ Dict {'first'} = "перший";  
$ Dict {'second'} = "другий";  
$ Dict {'third'} = "третій";  
або  
% Dict {'first', 'second', 'third'} = "перший", "другий",  
"третій";
```

Крім того, існує спосіб одночасно записати в асоціативний масив і ключові слова, і їх значення. Зробити це можна за допомогою наступного виразу:

```
Ім'я_масиву = (ключ 1, значення 1, ключ 2, значення 2, ...);
```

Для прикладу, який наведено вище, цей вираз буде виглядати так:

```
% Dict = ("first", "перший", "second", "другий", "third",  
"третій");
```

Розглянемо наведений наступний фрагмент програми мовою *Perl*:

```
while (<STDIN>)  
{Print; }
```

Незважаючи на те, що формально програма складена правильно, вона на перший погляд може здатися безглуздою. Однак при запуску вона поводить себе точно так само, як і одна з програм «відлуння». У мові Perl існує зумовлена скалярна змінна «\$_», використовувана за замовчуванням. Саме в неї поміщаються дані, прочитані зі стандартного вводу, і з неї береться значення для виведення в STDOUT.

Крім «\$_» в Perl є й інші зумовлені змінні:

- `$/` – номер версії Perl.
- `$.` – Номер рядка, прочитаної з файлу останнім.
- `$!` – Повідомлення про помилку.
- `$$` – ідентифікатор поточного процесу.
- `$ ^ T` – час у секундах з початку 1970 року до запуску даної програми.
- `$ O` – ім'я файлу, в якому міститься виконувана програма.
- `$ 1 ... $ 9` – фрагменти тексту, відмічені при виконанні операції зіставлення з шаблоном.

Подібно зумовленим скалярним змінним, в Perl існують масиви, що мають спеціальне значення. Найбільш важливий з них – асоціативний масив `%ENV`, що містить поточні значення змінних оточення. Щоб отримати значення змінної оточення, треба звернутися до елементу даного масиву, вказавши в якості індексу ім'я змінної оточення. Так, наведений нижче вираз записує в скалярну змінну `$path_string` значення змінної оточення `PATH`.

```
$ Path_string = $ ENV {'PATH'};
```

Одна з перших рядків CGI-сценарію на Perl, може виглядати так:

```
$ Method = $ ENV {'REQUEST_METHOD'};
```

Для роботи з файлами і потоками в Perl передбачені спеціальні файлові дескриптори.

Файлові дескриптори являють собою вказівник на файл, пристрій або PIPE канал, відкриті для запису, читання або для запису і читання. Оператор «<>» в Perl називається діамантовим оператором (*diamond operator*). Він визначає операцію читання рядка з потоку, дескриптор якого міститься в кутових дужках:

```
$ Str = <STDIN>; # Читання рядки з дескриптора STDIN
(стандартного потоку введення)
```

```
@ Lines = <F>; # Читання всіх рядків з пов'язаного з
дескриптором файлу F.
```

```
print STDOUT $ str; # Друк в STDOUT (стандартний
потік виводу)
```

Для зв'язування файлу з файловим дескриптором використовується функція *open*. Нижче наводяться варіанти використання цієї функції:

<i>open</i> дескриптор_потoku > і'мя_файлу	файл відкривається для виведення даних. Якщо файл з вказаним ім'ям відсутній, створюється новий файл.
<i>open</i> дескриптор_потoku >> і'мя_файлу	файл відкривається в режимі, який дозволяє записувати дані в кінець файлу.
<i>open</i> дескриптор_потoku +> і'мя_файлу	відкритий файл стає доступним для читання і для запису.

Функція

close дескриптор_файлу

закриває файл, пов'язаний із зазначеним дескриптором.

До складу мови Perl входять засоби пошуку і заміни, причому, задаючи шаблон для пошуку, можна використовувати регулярні вирази. Це означає, що складні операції, що зустрічаються в спеціалізованих додатках, можна легко реалізувати в будь-якій Perl-програмі.

Оператор пошуку *m//* записується таким чином:

m/ шаблон /

Якщо значення змінної «\$_*»* містить підрядок, відповідну вказаним шаблоном, оператор пошуку повертає значення *true*.

Розглянемо наступний приклад:

```
$ _ = <INPUT>;  
if (m / Scripts /)  
{Print "В URL є каталог Scripts \ n"; }  
else  
{Print "В URL немає каталогу Scripts \ n"; }
```

Оператор заміни *s///* записується таким чином:

s / шаблон пошуку / вираз для заміни / [набір модифікаторів]

При виконанні оператора *s///* проводиться пошук відповідності шаблоном, і якщо пошук завершується успішно, знайдений підрядок замінюється зазначеним виразом. Подібно оператору *m //*, оператор *s///* використовує змінну «\$_*»*. Нижче наведено найпростіший приклад застосування оператора *s///*.

```
$ _ = "CGI-сценарій написаний на мові C";  
s/ C $ / Perl /;  
print;
```

В результаті виконання сценарію на консоль буде виведений наступний рядок:

CGI-сценарій написаний на мові Perl

Оскільки символ *C* міститься в аббревіатурі *CGI*, тому в шаблоні пошуку вказано, що він повинен бути останнім у рядку.

За останніми роздільником в операторі *s///* можуть слідувати один або кілька модифікаторів. Призначення деяких модифікаторів наведено нижче.

- *g* – глобальний пошук. Якщо цей модифікатор не вказаний, після виявлення першого відповідності оператор *s///* закінчить свою роботу. Тому за відсутності модифікатора *g* буде вироблено не більше однієї заміни.
- *i* – вказує, що при пошуку слід ігнорувати регістр символів.
- *e* – вказує, що послідовність символів для заміни слід інтерпретувати не як підрядок, а як вираження *Perl*.

У виразі для підстановки можуть бути присутніми змінні *\$1* – *\$9*, і в цьому випадку необхідно вказати модифікатор *e*. Так, наприклад, якщо потрібно інтерпретувати десяткове число як код символу, можна використати наступним виразом:

s/([0-9]+)/chr(\$1)/e;

Щоб пошук або заміна вироблялися в рядку, що міститься в потрібній змінній, треба використовувати такий вираз:

Мінлива = ~ оператор_пошуку_або_заміна

Так, наприклад, для перетворення шістнадцятиричних чисел, що містяться в змінній *\$string*, в десяткове подання можна використовувати інструкцію:

```
$String = ~ s/([0-9A-Fa-f]+)(H|h)/hex($1)/ge;
```

2. Мова PHP.

Мова PHP (*PHP: Hypertext Preprocessor*) – одна з найбільш популярних сценарних мов зважаючи на свою простоту, швидкість виконання, багаті функціональності і поширеності початкових кодів на основі ліцензії PHP.

PHP складається з ядра і набору розширень, що підключаються: для роботи з базами даних, сокетамі, динамічною графікою, криптографічними бібліотеками, документами формату PDF та ін. Можлива розробка своїх власних розширень з їх подальшим підключенням. Хоча і існують сотні розширень, проте в стандартне постачання входить лише декілька десятків розширень, що добре зарекомендували себе.

Інтерпретатор PHP підключається до веб-сервера або через DLL модуль, створений спеціально для цього сервера, або у вигляді CGI- застосування.

Нині PHP використовується сотнями тисяч розробників. Близько 20 мільйонів сайтів повідомляють про роботу з PHP, що складає більше п'ятої долі доменів Інтернету

Синтаксис PHP подібний до синтаксису мови C. При цьому деякі елементи, як наприклад асоціативні масиви і цикл *foreach*, запозичені з мови Perl.

Для роботи програми на PHP не потрібно описувати будь-які змінні, використовувані модулі, і т. п. Будь-яка програма може починатися безпосередньо з оператора PHP.

```
<?Php  
echo 'Hello,world!';  
?>
```

Крім обмежувачів `<? Php?>`, допускається використання додаткових варіантів, таких як `<? ?> I <script language = "php"> </ script>`. Крім того, до версії 6.0 допускається використання обмежувачів мови програмування ASP `<% %>`.

Імена змінних починаються з символу \$, тип змінної оголошувати не потрібно. На відміну від імен функцій і класів, імена змінних *чутливі до регістру*. Змінні обробляються в рядках, які укладено в подвійні лапки.

Інструкції завершуються крапкою з комою (;)

PHP підтримує два типи коментарів:

- в стилі мови C (обмежені / * * /),
- C ++ (починаються з // і йдуть до кінця рядка)

PHP є мовою програмування з *динамічною типізацією*, що не вимагає вказівки типу при оголошенні змінних, так само як і самого оголошення змінних. Перетворення між скалярними типами може здійснюватися автоматично (хоча і є можливості для явного перетворення типів).

До *скалярних типів* даних відносяться:

- цілий тип (*integer*);
- дійсний тип даних (*float, double*);
- логічний тип (*boolean*);
- строковий тип (*string*);
- спеціальний тип NULL.

До *нескалярних типів* відносяться:

- «ресурс» (*resource*);
- масив (*array*);
- об'єкт (*object*).

Тип NULL призначений для змінних без певного значення. Значення NULL приймають неініціалізовані

змінні, змінні початкові константою `NULL`, а також змінні, віддалені за допомогою конструкції `unset()`.

Посилання на зовнішні ресурси мають тип *resource*. Змінні даного типу, як правило, являють собою дескриптор, що дозволяє управляти зовнішніми об'єктами, такими як файли, динамічні зображення, результуючі таблиці бази даних і т. П.

Масиви підтримують числові і рядкові ключі і є гетерогенними. Масиви можуть містити значення будь-яких типів, включаючи інші масиви. Суперглобальними масивами (*superglobal arrays*) в PHP називаються зумовлені масиви, які видно в будь-якому місці вихідного коду без використання ключового слова *global*:

- `$GLOBALS` – масив всіх глобальних змінних (у тому числі і для користувача).
- `$_SERVER` – містить безліч інформації про поточний запит і сервері.
- `$_ENV` – поточні змінні середовища. Їх набір специфічний для кожної конкретної платформи, на якій виконується сценарій.
- `$_GET` – асоціативний масив з параметрами GET-запиту. У початковому вигляді ці параметри доступні в `$_SERVER ['QUERY_STRING']` і в `$_SERVER ['REQUEST_URI']` у складі URI.
- `$_POST` – асоціативний масив значень полів HTML-форми при відправці методом POST.
- `$_FILES` – асоціативний масив з відомостями про надіслані методом POST файлах. Кожен елемент має індекс ідентичний значенню атрибуту «name» у формі *i*, в свою чергу, також є масивом з наступними елементами:
 - `['name']` – вихідне ім'я файлу на комп'ютері користувача.
 - `['type']` – зазначений агентом користувача MIME-тип файлу.

- ['size'] – розмір файлу в байтах.
- ['tmp_name'] – повний шлях до файлу в тимчасовій папці.
- ['error'] – код помилки.
- \$_COOKIE – асоціативний масив з переданими агентом користувача значеннями cookie.
- \$_REQUEST – загальний масив ввідних даних запиту користувача як в масивах \$_GET, \$_POST, \$_COOKIE. Починаючи з версії PHP 4.1 включається і вміст \$_FILES.
- \$_SESSION – інформація про поточної сесії користувача.

PHP підтримує широкі об'єктно-орієнтовані можливості, повна підтримка яких була введена в п'ятій версії мови. Клас в PHP оголошується за допомогою ключового слова *class*. Методи і поля класу можуть бути загальнодоступними (*public*, за замовчуванням), захищеними (*protected*) і прихованими (*private*). PHP підтримує всі три основні механізми ООП – інкапсуляцію, поліморфізм і успадкування (батьківський клас вказується за допомогою ключового слова *extends* після імені класу). Підтримуються інтерфейси (ставляться у відповідність з допомогою *implements*). Дозволяється оголошення фінальних, абстрактних методів і класів. Множинне спадкування класів не підтримується, проте клас може реалізовувати кілька інтерфейсів. Для звернення до методів батьківського класу використовується ключове слово *parent*. Примірники класу створюються за допомогою ключового слова *new*, звернення до полів і методів об'єкту проводиться з використанням символів *->*. Для доступу до членів класу з його методів використовується змінна *\$ this*.

Серед найбільш часто використовуваних можливостей PHP варто відзначити наступні:

- є великий набір функцій для роботи з рядками;
- робота з регулярними виразами PCRE.

- робота з базами даних, що здійснюється за допомогою модулів:
 - *php5-mysql* для MySQL;
 - *php5-pgsql* для PostgreSQL;
 - та ін.
- для PHP розроблені засоби шаблонування веб-сторінок, що дозволяють ефективно розділити подання від моделі, наприклад *Smarty*;
- є бібліотека для роботи з графічними зображеннями GD, що дозволяє виробляти перетворення з графічними файлами, і створювати зображення «на льоту».

Лекція 6. Введення в С# і платформу Visual Studio.Net.

План:

1. Платформа Visual Studio.NET.
2. Основи С#.
3. Типи даних С#.
4. Класи і методи в С#.
5. Вирази та оператори С#.
6. Інтерфейси С#.
7. Серіалізація об'єктів.

Література:

1. Беллиньясо М. Разработка Web-приложений в среде ASP.NET 2.0 : с примерами на С# / М. Беллиньясо. – М. : «Диалектика», 2007. – 640 с.
2. Роджерс Д. Программирование на Microsoft JScript.NET / Д. Роджерс. – М. : «Вильямс», 2002. – 352 с.
3. Троелсен Э. С# и платформа .NET. Библиотека программиста / Э. Троелсен. – СПб. : «Питер», 2007. – 796 с.
4. Шорт С. Разработка XML Web-сервисов средствами Microsoft.NET / С. Шорт. – СПб. : «БХВ-Петербург», 2003. – 480 с.

1. Платформа Visual Studio.NET.

Microsoft Visual Studio.NET – це інтегроване середовище розробки для створення, документування, запуску та налагодження програм, написаних на мовах .NET.

Це середовище розробки є відкритим мовним середовищем. Поряд з мовами програмування, спочатку

включеними в середовище – C++, C#, J#, *Visual Basic*, – в неї можуть додаватися будь-які мови програмування, компілятори які створюються сторонніми розробниками. Необхідною умовою для включення мов в середовище Visual Studio.Net є використання єдиного каркасу – платформи *Framework.Net*.

Платформа *Framework.Net* дозволяє:

- легко використовувати компоненти, розроблених на різних мовах;
- розробляти єдиний додаток з декількох частин на різних мовах.

Платформа *Framework.Net* містить два основних компоненти:

- *FCL* (Framework Class Library) – бібліотеку класів каркаса;
- *CLR* (Common Language Runtime) – загальномовне виконавчу середовище.

В рамках даної платформи використовується стандартна система типів *Common Type System* (CTS), яка повністю описує всі типи даних, які підтримуються середовищем виконання, визначає взаємодію типів даних та їх представлення у форматі метаданих .NET.

Набір правил, що визначають підмножину загальних типів даних, щодо яких гарантується, що вони небезпечні при використанні у всіх мовах .NET, описується в рамках специфікації *Common Language Specification* (CLS). Для того щоб класи, розроблені на різних мовах, можна було спільно використовувати в рамках однієї програми, вони повинні відповідати певним обмеженням, що задаються CLS. Клас, що задовольняє CLS, називається *CLS-сумісним*. Він доступний для використання в інших мовах, класи яких можуть бути клієнтами або спадкоємцями сумісного класу.

Платформа .NET надає в розпорядження програміста бібліотеку базових класів, доступну з будь-якої мови програмування .NET. Оскільки число класів бібліотеки FCL

сягає кількох тисяч, то в цілях структуризації функціонально близькі класи об'єднуються в групи, звані *простором імен* (Namespace).

Основним простором імен бібліотеки FCL є простір *System*, що містить як класи, так і інші вкладені простори імен. Наприклад, в просторі *System.Collections* знаходяться класи та інтерфейси, що підтримують роботу з колекціями об'єктів – списками, чергами, словниками. Простір *System.Windows.Forms* містить класи, використовувані при створенні windows-додатків.

Слід пам'ятати, що C# генерує код, призначений для виконання тільки в середовищі виконання .NET (керований код). Сам двійковий файл, що містить керований файл, називається збіркою. Збірка містить код на проміжній мові MSIL (Microsoft Intermediate Language) або просто IL. Аналогічно байт-коду Java, IL-код компілюється в платформено-специфічні інструкції при безпосередньому зверненні середовища виконання .NET до блоку IL-інструкцій. Двійкові модулі .NET на додаток до інструкцій містять також *метадані*. *Метадані* описують не тільки типи, використовувані в збірці, а й саму збірку. Дана частина метаданих називається *маніфестом*.

У більшості випадків між двійковим файлом .NET і збіркою існує відношення «один-до-одного». Однак збірка може складатися як з одного, так і з кількох двійкових файлів.

Збірка з одного файлу містить і маніфест, і метадані, та інструкції IL.

Двійкові файли, що утворюють разом загальну збірку називаються *модулями*. При цьому один з двійкових файлів повинен містити маніфест збірки. Інші модулі можуть містити тільки метадані типів та інструкції IL.

Багатофайлові збірки дозволяють середовищу виконання вибірково завантажувати тільки ті збірки, які в даний момент роботи програми дійсно необхідні, що

дозволяє скорочувати мережевий трафік і збільшувати швидкість роботи програм.

2. Основи C#.

Що нового принесла мова C# в порівнянні з вже існуючими до неї мовами?

- У програмах на C#, як правило, немає необхідності в роботі з покажчиками (при збереженні цієї можливості), оскільки в ньому реалізовано автоматичне управління пам'яттю.
- Передбачені вбудовані синтаксичні конструкції для роботи з перерахуваннями, структурами і властивостями класів.
- Є повна підтримка програмних інтерфейсів. Використання двійкових модулів .NET дозволяє передавати об'єкти (за посиланням або за значенням) через кордони програмних модулів.
- Повна підтримка об'єктно-орієнтованих технологій.

Для розробки додатків в Visual Studio.Net використовуються *проекти*.

Проект (Project) – це основна одиниця, з якою має справу розробник. Спочатку він повинен вибрати тип проекту, після чого Visual Studio створює каркас проекту відповідно за обраним типом. Проект складається з *класів*, зібраних в одному або декількох просторах імен. *Простори імен* (Namespaces) дозволяють структурувати проекти, що містять велику кількість класів, об'єднуючи в одну групу близькі класи.

Кілька проектів можуть об'єднуватися в *рішення* (Solution), яке також може включати ресурси, необхідні цим проектам.

З точки зору розробника кінцевим результатом його роботи, одержаних після компіляції вихідного програмного

коду, є рішення, а з точки зору CLR (Common Language Runtime – загальномовного середовища виконання) – збірка (assembly), що містить PE файл, тобто модуль у форматі виконуваного файлу PE (Portable Executable) для 32-розрядної ОС Windows або DLL (Dynamic Link Library) файл.

Visual Studio.Net пропонує велику різноманітність можливих типів проектів.

3. Типи даних C#.

Стандарт мови C++ включає наступний набір фундаментальних типів:

- Логічний тип (*bool*).
- Символьний тип (*char*).
- Цілі типи. Вони можуть відрізнятися розміром: *short*, *int*, *long*, а також можуть бути знаковими (*signed*) або беззнаковими (*unsigned*).
- Типи з плаваючою крапкою. Вони також можуть відрізнятися розмірами: *float*, *double* і *long double*.

Тип *void* вказує на відсутність інформації.

До конструйованим типів відносяться наступні:

- Показчики (наприклад, *char**).
- Посилання (наприклад, *char&*).
- Масиви (наприклад, *char[]*).

Також мова дозволяє розробнику конструювати власні типи:

- перелічуваних типи (*enum*);
- структури (*struct*);
- класи.

У мові C# всі типи можна розглядати і під іншим ракурсом, розділивши їх на чотири категорії:

1. Типи-значення (*value*).
2. Посилальні (*reference*).
3. Показчики (*pointer*).

4. Типові *void*.

Для посилального типу значення задає посилання на область пам'яті в «купі» (heap), де розташований відповідний об'єкт. Для типу-значення значенням є власне дані, а пам'ять для них виділяється в стек.

Логічний, арифметичний, структури, перерахування відносяться до типів-значень. Масиви, рядки і класи відносяться до *посилальних* типів.

І посилальні, і звичайні типи є похідними від базового класу *object*. У тих випадках, коли звичайний тип повинен вести себе як об'єкт, створюється оболонка (wrapper), яку можна розглядати як контрольний об'єкт, поміщений в купу, і в неї копіюється значення змінної звичайного типу. Оболонка автоматично позначається таким чином, що система знає, яке значення вона містить. Цей процес називається *упаковкою* (boxing), а зворотний процес – *розпакуванням* (unboxing).

Упаковка відбувається автоматично, для цього потрібно лише привласнити значення звичайного типу змінної типу *object*. Упаковка та розпакування дозволяють обробляти будь-який тип як об'єкт. Наприклад, у виразі

7.ToString();

ціле число 7 упаковується шляхом виклику функції *Int32.ToString()*.

Масиви в С# можуть бути багатовимірними (multidimensional) або *невирівняні* (jagged). Більш складні структури даних такі, як *стек* і *хеш-таблиця* визначені в просторі імен System.Collections.

У мові С# визначено клас *char[]*, і його можна використовувати для представлення рядків постійної довжини. Однак масив *char[]* – це звичайний масив, тому його не можна ініціалізувати рядком символів. У С# не визначене перетворення з класу *char[]* в клас *String*. У *String*

є динамічний метод *ToCharArray*, що задає подібне перетворення в *char[]*.

Клас *String* не дозволяє змінювати існуючі об'єкти. Клас *StringBuilder* дозволяє виправити цей недолік. Цей клас належить до змінюваних класам і його можна знайти в просторі імен *System.Text*.

4. Класи і методи в С#.

У Visual Studio.Net, і в С# зокрема, будь-яка програмна система розглядається як сукупність класів, об'єднаних в проекти, простору імен, рішення.

Опис класу має наступний синтаксис:

[Атрибути] [модифікатори] **class** ім'я_класу [:список_батьків]
{тіло_класу}

У тілі класу можуть бути зазначені:

- константи;
- поля;
- конструктори і деструктори;
- методи;
- події;
- делегати;
- класи (структури, інтерфейси, перерахування).

Поля класу синтаксично є звичайними змінними (об'єктами) мови. Їх опис задовольняє звичайним правилам оголошення змінних. Поля характеризують *властивості об'єктів* класу.

Методи класу синтаксично є звичайними процедурами і функціями мови. Методи містять описи операцій, доступних над об'єктами класу. Методи, звані *властивостями* є спеціальною синтаксичною конструкцією, призначеної для забезпечення ефективної роботи з класами.

Конструктор являє собою спеціальний метод класу, що дозволяє створювати об'єкти класу. Його ім'я повинно збігатися з ім'ям класу. Якщо розробник не визначає конструктор класу, то до класу автоматично додається конструктор за замовчуванням – конструктор без аргументів.

Делегат в С# є опис випадку класу і задає визначення функціонального типу (класу) даних. Екземплярами класу є функції. Кожен делегат описує безліч функцій із заданою сигнатурою. Кожна функція (метод), сигнатура якого збігається з сигнатурою делегата, може розглядатися як екземпляр класу, заданого делегатом. Синтаксис оголошення делегата має наступний вигляд:

[<Специфікатор доступу>] *delegate* <тип результату> <ім'я класу> (<список аргументів>);

5. Вирази та оператори С#.

Вирази будуються з операндів – констант, змінних, функцій, об'єднаних знаками операцій і дужками. При обчисленні виразу визначається його значення і тип.

У таблиці 6.1 наведений список операцій С#.

Таблиця 6.1.

Список операцій С#.

Категорія операцій	Операції
Арифметичні	+ - * / %
Логічні (<i>boolean</i> і побітові)	& ^ ! ~ &&
Рядкові	+
Інкремент і декремент	++ --
Здвиг	>> <<
Порівняння	== != < > <= >=
Присвоєння	= += -= *= /= %= &= = ^= <<= >>=
Звернення до члена класу	.

Індексація	[]
Приведення типу (Cast)	()
Умова	?:
Створення об'єкту	new()
Інформація про тип	is sizeof typeof
Керування винятками	checked unchecked
Побічним і адресації	* -> [] &

Ім'я та тип змінної задаються при її оголошенні і залишаються незмінними протягом усього часу її життя. Особливістю мови С# є вимога *обов'язкової ініціалізації* змінної до початку її використання. Спроба використати неініціалізовану змінну призводить до помилок, що виявляється ще на етапі компіляції.

З використовуваних виразів, оператор С# схожий на С++. Так в програмах на С# використовуються такі оператори як:

- Оператор присвоювання (=).
- Складовий оператор ({}).
- Оператори вибору: *if-else* і *switch*.
- Оператори циклу: *for*, *while*, *оператор*.
- Оператори *break* і *continue*.
- Оператор *return*.
- Оператор переходу *goto*

Крім того, введені кілька нових інструкцій. Наприклад, оператор *foreach* дозволяє отримати доступ до всіх елементів масиву чи колекції по черзі, в порядку зростання індексів. Його синтаксис:

foreach (*тип ідентифікатор* **in** *контейнер*) *оператор*

6. Інтерфейси С#.

Інтерфейс являє собою повністю абстрактний клас, всі методи якого абстрактні. Проте методи *інтерфейсу* оголошуються без вказівки модифікатора доступу, і клас,

що успадковує *інтерфейс*, зобов'язаний повністю реалізувати всі методи *інтерфейсу*. У цьому – відмінність від класу, що посяде абстрактний клас, де нащадок може реалізувати лише деякі методи батьківського абстрактного класу, залишаючись абстрактним класом.

Інтерфейс дозволяє описувати деякі бажані властивості, якими можуть володіти об'єкти різних класів.

Серед інтерфейсів, вбудованих в бібліотеку базових класів .NET, можна особливо виділити такі як:

- *IEnumerable* (для роботи з наборами об'єктів, в т. ч. з використанням оператора *foreach*).
- *IClonable* (копіювання об'єктів).
- *IComparable* (для порівняння і сортування об'єктів).

Простір імен *System.Collections*, призначений для роботи з наборами об'єктів, підтримує інтерфейси:

- *ICollection* (визначає загальні характеристики класу набору елементів).
- *IComparer*, *IDictionary* (дозволяє представляти вміст об'єкта у вигляді пар ім'я-значення).
- *IDictionaryEnumerator* (нумерація вмісту об'єкта, що підтримує *IDictionary*).
- *IEnumerable*, *IEnumerator*.
- *IHashCodeProvider* (повертає хеш-код за допомогою обраного алгоритму хешування).
- *IList* (забезпечує методи додавання, видалення та індексування елементів у списку об'єктів).

7. Сериалізація об'єктів.

При роботі з програмною системою часто виникає необхідність в серіалізації об'єктів. Під *серіалізацією* розуміють процес збереження об'єктів в довгостроковій пам'яті (файлах) в період виконання системи. Під *десериалізацією* розуміють зворотний процес – відновлення стану об'єктів, що зберігаються в довгостроковій пам'яті.

Механізми серіалізації C# і Framework.Net підтримують два формати збереження даних – в бінарному файлі і XML файлі. У першому випадку дані при серіалізації перетворюються в бінарний потік символів, який при десеріалізації автоматично перетвориться в потрібний стан об'єктів. Інший можливий перетворювач запам'ятовує стан об'єкту у форматі XML.

Якщо клас оголосити з атрибутом *[Serializable]*, то в нього вбудовується стандартний механізм серіалізації, що підтримує глибоку серіалізацію. Якщо з яких-небудь причин стандартна серіалізація розробника не влаштовує, то клас слід оголосити спадкоємцем інтерфейсу *ISerializable*, реалізація методів якого дозволить управляти процесом серіалізації.

Лекція 7. Архітектура веб-додатків ASP.NET. Розробка веб-додатків на платформі .NET.

План:

- 1. Платформа .NET Framework.**
- 2. Серверні елементи керування ASP.NET.**
- 3. Робота з джерелами даних в ASP.NET.**

Література:

1. Беллиньясо М. Разработка Web-приложений в среде ASP.NET 2.0 : с примерами на С# / М. Беллиньясо. – М. : «Диалектика», 2007. – 640 с.
2. Роджерс Д. Программирование на Microsoft JScript.NET / Д. Роджерс. – М. : «Вильямс», 2002. – 352 с.
3. Троелсен Э. С# и платформа .NET. Библиотека программиста / Э. Троелсен. – СПб. : «Питер», 2007. – 796 с.
4. Шорт С. Разработка XML Web-сервисов средствами Microsoft.NET / С. Шорт. – СПб. : «БХВ-Петербург», 2003. – 480 с.

1. Платформа .NET Framework.

Платформа .NET Framework надає можливість розробки та інтеграції веб-додатків. ASP.NET є однією зі складових структури .NET Framework і фактично є платформою для створення веб-додатків і веб-сервісів, що працюють під керуванням IIS.

ASP.NET зовні багато в чому нагадує більш стару технологію ASP, але в той же час внутрішній пристрій ASP.NET істотно відрізняється від ASP. Компанія Майкрософт ASP.NET побудувала на базі CLR (Common Language Runtime), який є основою всіх додатків .NET.

Розробники можуть створювати код для ASP.NET, використовуючи мови програмування, що входять до .NET Framework: C #, Visual Basic.NET, JScript.NET та інші.

Розглянемо більш докладно, чим відрізняється ASP.NET від ASP.

Класичний ASP має такі недоліки:

1. Використовуються тільки *мови сценаріїв*, які дають великий програш в продуктивності (через їх інтерпретованість) і не підтримують багато можливості об'єктно-орієнтованого програмування.
2. *Логіка уявлення* (у вигляді коду HTML) не відділена від *бізнес-логіки* (виконуваного коду), що призводить перемішуванню в одному файлі коду HTML з кодом сценарію.
3. Неможливо *повторно використовувати готові рішення* в інших проектах (можливо тільки копіювання коду сценаріїв).

У файлах ASP.NET включається код на таких мовах програмування як C#, JScript.NET, VisualBasic.NET, що дозволяє застосовувати безпосередньо у веб-додатках можливості об'єктно-орієнтованого програмування. Також істотно скорочується обсяг коду, написаного вручну за рахунок застосування серверних об'єктів, автоматично генеруючих код елементів керування HTML. Можливе використання стандартного середовища розробки *Visual Studio.NET*, тобто ASP.NET має перевагу в швидкості в порівнянні зі сценарними технологіями, так як при першому зверненні код компілюється і поміщається в спеціальний кеш, а згодом тільки виконується, не вимагаючи витрат часу на парсинг, оптимізацію, і таке інше.

Незважаючи на можливість спільної роботи ASP і ASP.NET на одному веб-сервері, вони не можуть використовувати загальний сеанс. Файли ASP.NET

обробляються бібліотекою *aspnet_isapi.dll* (а не *asp.dll*), яка, в свою чергу, використовує для виконання коду технологію *.NET*.

Бібліотека базових класів *.NET* містить простір імен з основних груп:

- елементи *web-додатків* (протоколи, безпека та ін.);
- елементи *графічного інтерфейсу* (WebForms);
- *web-служби*.

Як вже зазначалося раніше, ASP.NET використовує можливості стандартної середовища розробки *Visual Studio.Net*, і зокрема класи бібліотеки FCL (Framework Class Library).

Розробнику веб-додатків на ASP.NET доступні класи (таблиця 7.1).

Таблиця 7.1

Класи веб-додатків ASP.NET

Простір імен	Зміст
System.Web	Організація взаємодії web-клієнта (браузера) з web-сервером (запит-відповідь, cookie і та ін.)
System.Web.Caching	Підтримка кешування при роботі web-додатків
System.Web.Configuration	Налаштування web-додатків відповідно до файлів конфігурації проекту
System.Web.Security	Реалізація системи безпеки web-додатків
System.Web.Services	Організація роботи web-сервісів
System.Web.Services.Description	
System.Web.Services.Discovery	
System.Web.Services.Protocols	
System.Web.UI	Побудова графічного інтерфейсу користувачів web-додатків
System.Web.UI.WebControls	
System.Web.HtmlControls	

У свою чергу простір імен *System.Web* включає в себе простори імен, назви яких знайомі розробникам веб-додатків на ASP (таблиця 7.2).

Таблиця 7.2

Простір імен System.Web

Простір імен	Зміст
HttpApplication	Даний клас визначає загальні для всіх web-додатків члени
HttpApplicationState	У цьому класі міститься загальна інформація web-додатки для безлічі запитів, сеансів і каналів передачі даних
HttpBrowserCapabilities	Цей клас використовується для отримання інформації про можливості клієнтського браузера, що звертається до web-серверу
HttpCookie	Підтримка механізму безпечної роботи з об'єктами HTTP cookie
HttpRequest	Надає доступ до інформації, переданої web-клієнтом
HttpResponse	Використовується для формування HTTP-відповіді серверу

В основу розробки веб-додатків на ASP.NET покладено модель *поділу коду представлення і коду реалізації*, яка рекомендована Майкрософт при створенні динамічних документів за допомогою програмних кодів. Це робиться шляхом розміщення програмного коду або в окремий файл, або всередині спеціального тегу для сценаріїв. Файл такого роду зазвичай має розширення **.aspx.cs* (**.aspx.vb*) і має ім'я, що збігається з ім'ям основного ASPX файлу. В принципі такий підхід дозволяє веб-дизайнеру сконцентруватися на роботі з кодом розмітки документа з мінімальними змінами програмного коду, в звичайному ASP, який впроваджено безпосередньо в код розмітки.

Взаємодія користувача з веб-додатком, реалізованому на ASP.NET включає в себе такі процеси:

1. При запиті сторінки *ASPX* ініціюється подія *Page_Init*, що виробляє початкову ініціалізацію сторінки та її об'єкту.
2. Далі ініціюється подія *Page_Load*, яка може бути використана, наприклад для установки початкових значень для елементів керування. При цьому також можна визначити чи була завантажена сторінка вперше або звернення до неї здійснюється повторно в рамках зворотного відсилання у відповідь на події, пов'язані з елементами керування, розміщеними на сторінці; тобто перевірити властивість *Page.IsPostBack*.
3. Далі виконується перевірка валідності елементів сторінки з погляду коректності введених користувачем даних.
4. І, нарешті, виконується обробка всіх подій, пов'язаних з діями користувача з моменту останнього зворотного відсилання.

Для збереження даних веб-сторінки в проміжках між зверненнями до неї в ASP.NET використовуються стани відображення (view state).

Якщо дані, введені в веб-форму, необхідно зробити доступними іншим веб-формам того ж додатка, ці дані необхідно зберегти в об'єктах *Application* і *Session*. Об'єкти *Application* доступні всім користувачам програми та можуть розглядатися як глобальні змінні, звернення до яких можливо з будь-яких сеансів. Об'єкти *Session* доступні тільки в рамках одного сеансу, і тому вони виявляються доступними тільки одному користувачеві.

2. Серверні елементи керування ASP.NET.

Важливою особливістю ASP.NET є використання серверних елементів керування на веб-сторінці (елементи *WebForm*), які є фактично тегами, зрозумілими веб-серверу.

Ці елементи визначені в просторі імен *System.Web.UI.WebControls*.

Прийнято виділяти три типи серверних елементів керування:

1. Серверні елементи керування HTML – звичайні HTML теги.
2. Елементи керування веб-сервера – нові теги ASP.NET.
3. Серверні елементи керування для перевірки даних (валідації) – застосовуються для валідації вхідних даних від клієнтського застосування (зазвичай веб-браузера).

Переваги від використання таких елементів при розробці веб-додатків:

1. Скорочується кількість коду, написаного вручну (що особливо помітно для складних елементів документа). Елемент просто «перетягується» з панелі інструментів, після чого виконується налаштування його параметрів у спеціальному вікні. При цьому всі зміни автоматично заносяться безпосередньо в *.aspx файл.
2. З програмної точки зору кожного з цих елементів керування відповідає певний клас в бібліотеці базових класів .NET, що дозволяє писати для них такий самий код як і для будь-яких інших класів.
3. Для будь-якого елементу керування WebForm визначений набір подій, оброблюваних на веб-сервері.
4. Для будь-якого елементу керування WebForm надається можливість для перевірки введення даних користувачем.

За замовчуванням серверні елементи керування HTML в ASP.NET файлах розглядаються як текст. Для їх програмування потрібно додавання атрибуту *runat="server"* у відповідний HTML елемент. Крім того, всі серверні елементи керування HTML повинні бути розміщені

всередині області дії тегу *<form>*, який також має атрибут *runat="server"*.

Подібно серверним елементам керування HTML *елементи керування веб-сервера* також створюються на веб-сервері і припускають додавання атрибуту *runat="server"*. Однак вони можуть і не відповідати конкретним елементам HTML, але представляти більш складні елементи.

Загальний синтаксис для опису таких елементів:

```
<asp:тип_елемента id="ідентифікатор" runat="server"/>
```

Серверні елементи валідації застосовуються для перевірки введених користувачем даних.

Вони мають наступний синтаксис:

```
<asp:тип_елемента id="ідентифікатор" runat="server" />
```

3. Робота з джерелами даних в ASP.NET.

У ASP.NET використовуються два елементи керування WebForm для керування відображенням даних, одержуваних з джерела даних:

1. *DataGrid* – елемент керування, що відображає вміст об'єкту ADO.NET *DataSet* у вигляді таблиці.
2. *DataList* – елемент керування для вибору значень, що заповнюються з джерела даних.

Якщо необхідно відобразити дані, отримані за запитом користувача з джерела даних, у вигляді таблиці на веб-сторінці, то ASP.NET надає в розпорядження веб-програміста зручний елемент керування *DataGrid*.

Лекція 8. Інтерфейси взаємодії веб-додатків з СКБД.

План:

1. Взаємодія веб-додатків з СКБД.
2. Додаток ADO.NET.

Література:

1. Роджерс Д. Программирование на Microsoft JScript.NET / Д. Роджерс. – М. : «Вильямс», 2002. – 352 с.
2. Шорт С. Разработка XML Web-сервисов средствами Microsoft.NET / С. Шорт. – СПб. : «БХВ-Петербург», 2003. – 480 с.

1. Взаємодія веб-додатків з СКБД.

Сьогодні більшість інформаційних систем в тій чи іншій мірі використовують бази даних. Не є винятком і системи, засновані на веб-технологіях. Тому організація взаємодії веб-додатків з СКБД є невід'ємною складовою частиною веб-технологій.

До початку 90-х років існувало кілька різних постачальників баз даних, кожен з яких мав власний інтерфейс. Якщо додатку було необхідно обмінюватися даними з декількома джерелами даних, для взаємодії з кожною з баз даних було необхідно написати окремий код. З метою вирішення цієї проблеми Майкрософт і ряд інших компаній створили *стандартний інтерфейс* для отримання і відправки даних джерел даних різних типів. Цей інтерфейс отримав назву *open database connectivity* (ODBC).

З допомогою ODBC прикладні програмісти змогли розробляти програми з використанням єдиного інтерфейсу доступу до даних, не враховуючи тонкощі взаємодії з різними джерелами даних. Це досягається завдяки тому, що

постачальники різних баз даних розробляють *драйвери*, що враховують специфіку конкретних джерел даних при реалізації стандартних функцій з ODBC API. При цьому додатки використовують функції такого API, які реалізовані у відповідному конкретному джерелі даних драйверу.

По-суті, інтерфейс ODBC є звичайним *процедурним* API. ODBC підтримується великою кількістю операційних систем.

Є також ODBC-драйвери і для не реляційних даних, таких як *електронні таблиці, текст і XML файли*.

Типовий сценарій роботи веб-додатків з джерелом даних виглядає наступним чином:

1. Встановлення з'єднання та підключення до джерела даних.
2. Виконання запитів, необхідних для вибірки, вставки або зміни наборів даних джерела.
3. Відключення від джерела даних.

Компанією Майкрософт був запропонований прикладний програмний інтерфейс для доступу до даних, розроблений і заснований на технології компонентів *ActiveX* – ADO (ActiveX Data Objects), який дозволяє представляти дані з різноманітних джерел (реляційних баз даних, текстових файлів і таке інше.) в об'єктно-орієнтованому вигляді. Компоненти ADO застосовуються при розробці додатків на таких мовах як VBScript в ASP і Visual Basic.

2. Додаток ADO.NET.

В рамках Microsoft .NET основною моделлю доступу додатків до джерел даних є ADO.NET. Вона не є розвитком ADO і являє собою абсолютно самостійну технологію. Компоненти *ADO.NET* входять в поставку *.NET Framework*.

ADO.NET включає в себе дві основні частини:

1. *Data provider* – набір класів для доступу до джерел даних. Кожен з джерел даних має свій власний набір об'єктів, проте всі вони мають спільну множину класів: *Connection*, *Command*, *Parameter*, *DataAdapter*, *DataReader*.
2. *DataSets* об'єкти – група класів, що описують прості реляційні бази даних, які розміщуються в пам'яті. Містить ієрархію таких класів як: *DataTable*, *DataRow*, *DataColumn*, *DataRowView*, *DataRelation*, *Constraint*.

Об'єкт *DataSet* заповнюється даними з БД за допомогою об'єкту *DataAdapter*, у якого задані властивості *Connection* і *Command*. *DataSet* може зберігати свій вміст також в XML (опціонально разом з XSD схемою) або отримувати дані з XML.

ADO.NET підтримує роботу з *від'єднаними наборами даних*, що вкрай важливо при використанні масштабованих веб-додатків. Така можливість реалізується за допомогою класу *DataSet* спільно з класом *DataAdapter*.

Однією з найважливіших складових технології ADO.NET є *постачальник даних*. По-суті, це набір класів, призначених для взаємодії з джерелом даних певного типу. Використання різних постачальників даних робить ADO.NET дуже гнучкою і розширюваною.

Лекція 9. Введення в XML.

План:

1. Мова SGML.
2. Мова XML.

Література:

1. Крейн Д. AJAX в действии: технология – Asynchronous JavaScript and XML = AJAX in Action / Д. Крейн, Э. Паскарелло, Д. Джеймс. – М. : «Вильямс», 2006. – 640 с.
2. Ньюкомер Э. Веб-сервисы: XML, WSDL, SOAP и UDDI / Э. Ньюкомер. – СПб. : Питер, 2003. – 345 с.
3. Старыгин А. XML : разработка Web-приложений / А. Старыгин. – СПб. : «БХВ-Петербург», 2003. – 585 с.
4. Шеперд Д. Освой самостоятельно XML за 21 день / Д. Шеперд. – М. : «Вильямс», 2002. – 432 с.
5. Шорт С. Разработка XML Web-сервисов средствами Microsoft.NET / С. Шорт. – СПб. : «БХВ-Петербург», 2003. – 480 с.

1. Мова SGML.

У 1986 році, задовго до того, як ідея створення мережі Веб була втілена в життя, *універсальна стандартизована мова розмітки* SGML (Standardized Generalized Markup Language) була затверджена як міжнародний стандарт (ISO 8879) визначення мов розмітки, хоча SGML існував ще з кінця шістдесятих . Він використовувався для того, щоб описувати мови розмітки, надаючи при цьому автору можливість давати формальні визначення кожному елементу і атрибуту мови.

Мова HTML спочатку була всього лише одна з SGML-додатків. Вона описувала правила, за якими повинна бути підготовлена інформація для World Wide Web. Таким чином, мова HTML – це набір приписів SGML, сформульованих у вигляді визначення типу документа (DTD), які пояснюють, що саме позначають теги й елементи. Схема DTD для мови HTML зберігається у веб-браузері.

До *недоліків* мови HTML можна віднести наступні:

1. HTML має *фіксований набір тегів*. Не можна створювати свої теги, зрозумілі іншим користувачам.
2. HTML – це виключно *технологія представлення даних*. HTML не несе інформації про значення змісту, укладеного в тегах.
3. HTML – *«плоска» мова*. Значимість тегів в ньому не визначена, тому за її допомогою не можна описати ієрархію даних.
4. У вигляді платформ для додатків використовуються *браузери*. HTML не володіє достатньою потужністю для створення веб-додатків на тому рівні, до якого в даний час прагнуть веб-розробники. Наприклад, на мові HTML неможливо розробити додаток для професійної обробки і пошуку документів.
5. *Великі обсяги трафіку мережі*. Існуючі HTML-документи, використовувані як додатки, перевантажують Інтернет великими обсягами трафіку в системах клієнт-сервер. Прикладом може служити пересилання по мережі великого за обсягом документа, в той час як користувачу необхідна тільки невелика частина цього документа.

Таким чином, з одного боку, мова HTML є дуже зручним засобом розмітки документів для використання у

веб, а з іншого – документ, розмічений в HTML, має мало інформації про свій зміст. Якщо той чи інший документ несе досить повну інформацію про свій зміст, з'являється можливість порівняно легко провести автоматичну узагальнену обробку і пошук у файлі, що зберігає документ. Мова SGML дозволяє зберігати інформацію про зміст документа, однак внаслідок особливої складності він ніколи не використовувався так широко, як HTML.

Для створення підмножини мови SGML, яка могла б бути прийнята Веб-спільнотою, була організована спеціальна група експертів з мови SGML, очолювана Джоном Боузеком (Jon Bosak) з компанії Sun Microsystems. Вирішено було видалити багато несуттєвих можливостей SGML. Перебудовану таким чином мову назвали XML. Спрощений варіант виявився найбільш доступним, ніж оригінал, його специфікації займали всього 26 сторінок в порівнянні з більш ніж 500 сторінками специфікацій SGML.

2. Мова XML.

Розглянемо більш детально структуру й особливості мови XML.

XML (*eXtensible Markup Language*) – рекомендована W3C мова розмітки. XML – текстовий формат, призначений для зберігання структурованих даних, для обміну інформацією між програмами, а також для створення на його основі спеціалізованих мов розмітки. XML є спрощеною підмножиною мови SGML.

Мова XML має такі *переваги*:

1. Це людино-орієнтований формат документа, він зрозумілий як людині, так і комп'ютеру.
2. Підтримує Юнікод.
3. У форматі XML можуть бути описані основні структури даних – такі як записи, списки і дерева.

4. Це самодокументований формат, який описує структуру й імена полів також як і значення полів.
5. Має суворо певний синтаксис і вимоги до аналізу, що дозволяє йому залишатися простим, ефективним і несуперечливим.
6. Широко використовується для зберігання і обробки документів.
7. Це формат, заснований на міжнародних стандартах.
8. Ієрархічна структура XML підходить для опису практично будь-яких типів документів.
9. Являє собою простий текст, вільний від ліцензування і будь-яких обмежень.
10. Не залежить від платформи
11. Є підмножиною SGML, для якого накопичений великий досвід роботи і створені спеціалізовані додатки.

До відомих *недоліків* мови можна віднести наступні:

- Синтаксис XML надмірний.
 - Розмір XML документа істотно більше бінарного уявлення тих же даних (порядку 10 разів).
 - Розмір XML документа істотно більше, ніж документа в альтернативних текстових форматах передачі даних (наприклад *JSON*, *YAML*) і особливо в форматах даних, оптимізованих для конкретного випадку використання.
 - Надмірність XML може вплинути на ефективність програми. Зростає вартість зберігання, обробки і передачі даних.
 - Для великої кількості завдань не потрібна вся міць синтаксису XML, і можна використовувати значно простіші і продуктивні рішення.

- Простори назв XML складно використовувати і їх складно реалізовувати в XML парсер.
- XML не містить вбудованої в мову підтримки типів даних. У ньому немає понять «цілих чисел», «рядків», «дат», «булевих значень» і таке інше.
- Ієрархічна модель даних, пропонована XML, обмежена в порівнянні з реляційною моделлю і об'єктно-орієнтованими графами.

Простіше кажучи, XML можна розглядати не тільки як нову мову розмітки, але і як основу для цілого сімейства технологій:

XML	Технічні рекомендації про використання XML
DTD	Визначення типу документа (схема)
XDR	Формат XML Reduced (схема Microsoft)
XSD	Визначення схеми XML (схеми W3C)
Простір імен	Метод визначення імен елементів і атрибутів
XPath	Мова шляхів XML
XLink	Мова посилань XML
XPointer	Мова покажчиків XML
DOM	API для об'єктної моделі документу
SAX	Простий API для XML
XSL	Розширювана мова таблиць стилів
XSL-FO	Об'єкту форматування XSL
XSLT	Мова перетворень XSL
XInclude	Синтаксис XML Include
XBase	Синтаксис XML Base URI

По-суті, XML служить *метамовою* для опису структури інших мов. Взаємозв'язок між SGML, XML, HTML і деякими іншими мовами показана на рис. 9.1.

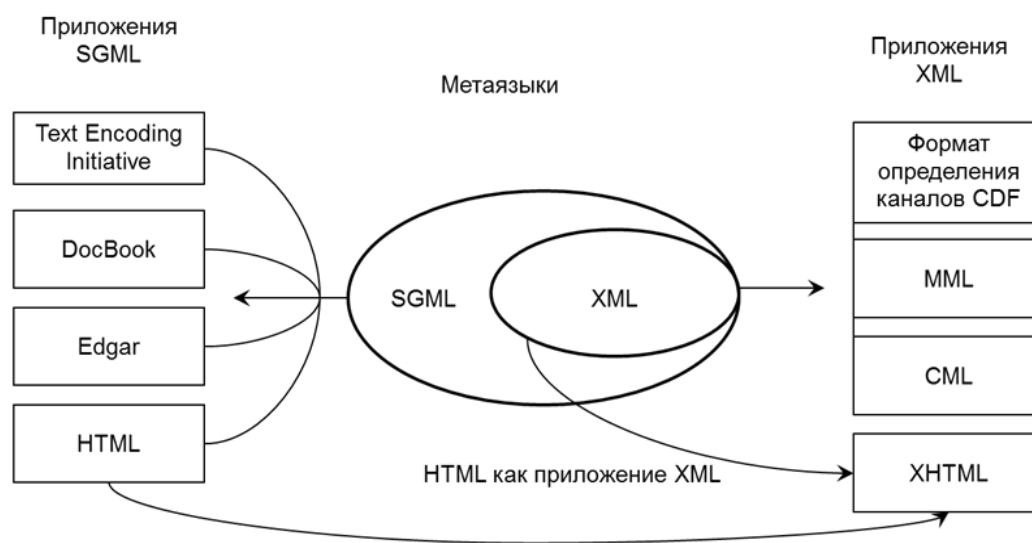


Рис. 9.1. Взаємозв'язок між SGML, XML, HTML.

Важливою відмінністю XML від HTML є та велика увага, яка приділяється контролю за тим, наскільки точно дотримуються правила мови при розмітці документів. Залежно від цього прийнято виділяти *правильно побудовані і дійсні XML документи*.

Документ XML вважається *правильно побудованим*, якщо він відповідає всім синтаксичним правилам XML.

Перевірка дійсності документа передбачає виконання таких дій:

1. Перевірка використання тільки заданого набору дескрипторів.
2. Перевірка повної відповідності порядку проходження елементів і атрибутів змісту документа або певними правилами.
3. Контроль типів даних (досягається при використанні відповідної схеми).
4. Контроль цілісності даних для забезпечення оптимального обміну інформацією через Веб за допомогою транзакцій.

Розглянемо тепер основні синтаксичні правила побудови XML документів:

1. XML документ містить один і тільки один *кореневий елемент*, що містить всі інші елементи.
2. *Дочірні елементи*, що містяться в *кореновому елементі*, повинні бути правильно вкладені.
3. *Імена елементів* підкоряються правилам:
 - Ім'я починається з букви, знака підкреслення або двокрапки.
 - Після першого символу в імені можуть бути букви, цифри, знаки переносу, підкреслення, крапка або двокрапка.
 - Імена не можуть починатися з буквосполучення XML.

XML документ має наступну *структуру*:

- Перший рядок XML документа називається *оголошенням XML*. Це необов'язковий рядок, що вказує версію стандарту XML (зазвичай це 1.0). Також тут може бути вказане кодування символів і зовнішні залежності.
- Коментар може бути розміщений в будь-якому місці дерева. XML коментарі розміщуються всередині пари тегів `<!--` і закінчуються `-->`. Два знака дефіс (-) не можуть бути застосовані ні в якій частині всередині коментаря.
- Інша частина цього XML-документа складається з вкладених елементів, деякі з яких мають атрибути і вміст.
- Елемент зазвичай складається з *відкриваючих* і *закриваючих* тегів, що обрамляють текст та інші елементи. *Відкриваючий* тег складається з імені елемента в кутових дужках. *Закриваючий* тег складається з того ж імені в кутових дужках, але перед ім'ям ще додається коса риска. *Вмістом* елемента називається все, що розташоване між

відкриваючим і закриваючим тегами, включаючи текст та інші (вкладені) елементи.

- Крім *вмісту* в елемента можуть бути атрибути – пари *ім'я* = *значення*, що додаються всередину відкриваючого тегу після назви елемента.
- Значення атрибутів завжди беруться в лапки (одинарні або подвійні), одне і те ж ім'я атрибута не може зустрічатися двічі в одному елементі.
- Не рекомендується використовувати різні типи лапок для значень атрибутів одного тегу.
- Для позначення *елемента без вмісту*, називають *порожнім* елементом, тому необхідно застосовувати особливу форму запису, що складається з одного тегу, в якому після назви *елементу* ставиться коса риса «/».

На жаль, описані вище правила дозволяють контролювати тільки формальну правильність XML документу, але не змістовну. Для вирішення другого завдання використовуються так звані схеми.

Схема чітко визначає *ім'я* та *структуру* кореневого елемента, включаючи *специфікацію* всіх його *дочірніх елементів*. Програміст може задати, які елементи і в якій кількості *обов'язкові*, а які – *необов'язкові*. Схема також визначає, які елементи містять *атрибути*, *допустимі значення* цих атрибутів, в т. ч. *значення за замовчуванням*.

Найчастіше для опису схеми використовуються наступні специфікації:

- DTD (*Document Type Definition*) – мова визначення типу документів.
- XDR (*XML Data Reduced*) – діалект XML, розроблений Microsoft.
- XSD (*мова визначення схем XML*) – рекомендована консорціумом W3C.

XML документ відрізняється від HTML документа також і тим, як він відображається у веб-браузері. Без

використання CSS або XSL, XML-документ відображається як простий текст в більшості веб-браузерів. Деякі веб-браузери, такі як *Internet Explorer*, *Mozilla* і *Firefox* відображають структуру документа у вигляді дерева, дозволяючи згортати і розгортати вузли за допомогою натискань клавіші миші.

Найбільш поширені три способи перетворення XML-документа в відображуваний користувачеві вигляд:

1. Застосування стилів CSS.
2. Застосування перетворення *XSLT*.
3. Написання на якійсь мові програмування обробника XML-документу.

Лекція 10. Мови опису схем XML.

План:

1. DTD схема.
2. XDR схема.

Література:

1. Крейн Д. AJAX в действии: технология – Asynchronous JavaScript and XML = AJAX in Action / Д. Крейн, Э. Паскарелло, Д. Джеймс. – М. : «Вильямс», 2006. – 640 с.
2. Ньюкомер Э. Веб-сервисы: XML, WSDL, SOAP и UDDI / Э. Ньюкомер. – СПб. : Питер, 2003. – 345 с.
3. Старыгин А. XML : разработка Web-приложений / А. Старыгин. – СПб. : «БХВ-Петербург», 2003. – 585 с.
4. Шеперд Д. Освой самостоятельно XML за 21 день / Д. Шеперд. – М. : «Вильямс», 2002. – 432 с.
5. Шорт С. Разработка XML Web-сервисов средствами Microsoft.NET / С. Шорт. – СПб. : «БХВ-Петербург», 2003. – 480 с.

Ідея створення власних тегів, що мають спеціальне значення і допомагають описати зміст документа, сама по собі просто чудова. Але якщо кожен користувач може створювати свої власні описи, яким чином їх розпізнавати? З цією метою в специфікації XML для опису подібних «самодіяльних» тегів використовуються *схеми*. Вони необхідні для того, щоб:

- описати, що саме є розміткою;
- описати точно, що означає розмітка.

Найбільш відомими мовами опису схем є наступні:

- DTD (*Document Type Definition*) – мова визначення типу документів, який спочатку використовувався в якості мови опису структури SGML-документа.
- XDR (*XML Data Reduced*) – діалект схеми XML, розроблений Microsoft, який підтримувався в Internet Explorer 4 і 5 версій.
- XML Schema або просто XSD (*мова визначення схем XML*) – рекомендація консорціуму W3C з 2001 року.

Розглянемо докладніше перші два з них. Третя мова опису схем розглядається в лабораторній роботі №11.

1. DTD схема.

Схема DTD надає *шаблон* розмітки документа, в якому зазначаються *наявність*, порядок проходження і *розташування елементів* та їх атрибутів в документі XML.

В рамках DTD модель змісту XML документа можна описати (рис. 10.1):

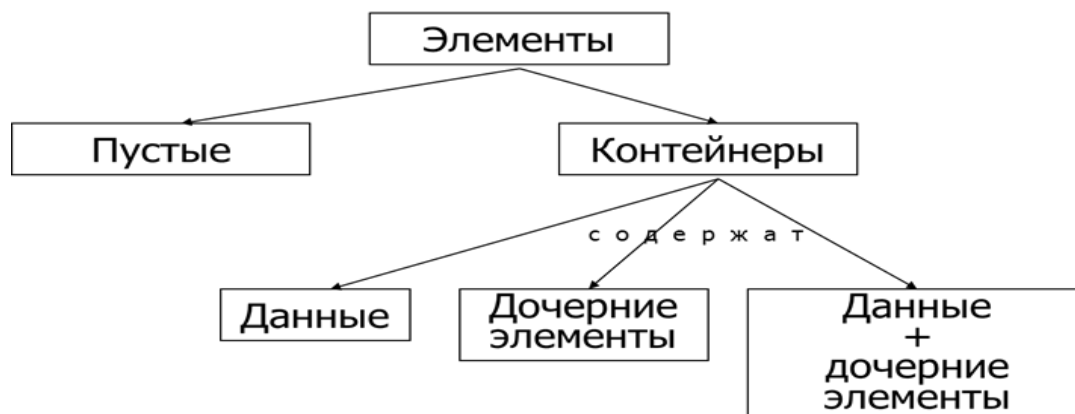


Рис. 10.1. DTD схема моделі змісту XML документа.

Кожен елемент документа може мати один з типів (табл. 10.1).

Таблиця 10.1.

Типи елементів XML документу

Зміст	Синтаксис	Коментар
Дані	<!ELEMENT ім'я (#PCDATA)>	Містить тільки текстові дані
Інші елементи	<!ELEMENT ім'я (дочірній елемент 1, дочірній елемент 2)>	Містить тільки дочірні елементи
Змішане	<!ELEMENT ім'я (#PCDATA, дочірній елемент)*>	Містить комбінацію текстових даних і дочірніх елементів
EMPTY	<!ELEMENT ім'я EMPTY>	Нічого не містить
ANY	<!ELEMENT ім'я ANY>	Може містити текстові дані або дочірні елементи

Атрибути, що знаходяться всередині тегів документу, описуються окремо за допомогою синтаксису:

```
<! ATTLIST
    Ім'я_елемента ім'я_атрибута1 (тип) значення_за_замовчуванням
    Ім'я_елемента                ім'я_атрибутаN                (тип)
    значення_за_замовчуванням>
```

При цьому атрибут в DTD може мати один з трьох типів:

1. Рядок.
2. Марковані атрибути.
3. Атрибут з перерахуванням

Крім типу атрибута можна також задавати і його модальність (табл. 10.2).

Таблиця 10.2.

Модальність атрибуту

Значення	Опис
#REQUIRED	Атрибут обов'язково повинен бути вказаний

#FIXED	Значення атрибута не повинне відрізнятися від вказаного
#IMPLIED	Необов'язкове значення

Розглянемо як приклад опис атрибутів *строкового* типу для елемента, що описує деяке повідомлення:

<!ATTLIST message			
number	CDATA	#REQUIRED	
date	CDATA	#REQUIRED	
from	CDATA	#FIXED	
status	CDATA	#IMPLIED	>

Якщо цей елемент містить атрибути з *перерахуванням*, то їх опис може виглядати, наприклад, таким чином:

<!ATTLIST message			
number	ID	#REQUIRED	
from	CDATA	#REQUIRED	
alert	(low normal urgent)	"normal"	>

Марковані атрибути елемента можуть бути чотирьох типів (табл. 10.3).

Таблиця 10.3.

Марковані атрибути елементу

Значення	Опис
ID	Унікальний ідентифікатор елемента (починається з букви, двокрапки або підкреслення)
IDREF	Посилання на елемент, що містить атрибути ID
ENTITIES	Посилання на зовнішній елемент
NMTOKEN	Містить букви, цифри, крапки, знаки підкреслення, переноси, двокрапки, але не прогалини

І, нарешті, в DTD можна використовувати такі індикатори входження послідовностей (табл. 10.4).

Таблиця 10.4.

Індикатори входження послідовностей в DTD мові

Символ	Приклад	Опис
,	(a, b, c)	Послідовне використання елементів списку
	(a b c)	Використовується один з членів списку
	date	Використовується один і тільки один елемент
?	subject?	Необов'язкове використання (0 або 1 раз)
+	paragraph+	Використовується один або кілька разів
*	brother*	Використовується нуль або декілька разів

Як приклад наведемо DTD схему, що описує структуру електронної поштової скриньки:

```

<!ELEMENT mailbox (message*)>
<!ELEMENT message (head, body)>
<!ATTLIST message uid CDATA #REQUIRED>
<!ELEMENT head ( from,to+, subject?, CC*, notify?) >
<!ELEMENT from (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT CC (#PCDATA)>
<!ELEMENT notify EMPTY>
<!ELEMENT body (#PCDATA)>

```

Вихідний XML документ, що задовольняє даній схемі, може виглядати так:

```

<?xml version="1.0" ?>
<!DOCTYPE mailbox SYSTEM "mailbox.dtd">
<mailbox>
  <message uid="1">
    <head>
      <from>user1@myhp.edu</from>
      <to>user2@myhp.edu</to>
      <subject>Re:</subject>
    </head>
    <body>

```

```
        What's up!
    </body>
</message>
<message uid="2">
  <head>
    <from>user3@myhp.edu</from>
    <to>user2@myhp.edu</to>
    <subject>Remind</subject>
    <CC> user1@myhp.edu </CC>
    <notify/>
  </head>
  <body>
    Remind me about meeting.
  </body>
</message>
</mailbox>
```

Зверніть увагу на 2-й рядок документу, в якому вказується зовнішнє посилання на файл, що містить DTD схему.

В принципі, DTD допускає два способи використання в XML документі:

1. Оголошення внутрішньої схеми:

```
<!DOCTYPE корневий_елемент [  
<!ELEMENT корневий_елемент (модель змісту)>  
>
```

2. Оголошення зовнішньої схеми:

```
<!DOCTYPE корневий_елемент SYSTEM "name.DTD">
```

Недоліки DTD схем:

1. Не є екземплярами XML. Потрібно вивчення абсолютно іншої мови.
2. Не надають контроль за типами даних, за винятком найпростіших текстових даних.
3. Не є екземплярами XML, тому їх не можна легко розширити або перетворити до інших мов розмітки - HTML або DHTML.
4. Не забезпечують підтримки просторів імен XML.

2. XDR схема.

XML-Data – повне ім'я мови опису схем, запропонованого Майкрософт, а *XML-Data Reduced* – це «частина» повної рекомендації. Схема XDR – це екземпляр XML, тобто відповідає всім синтаксичним правилам і стандартам XML.

Реалізуючи перевірки даних на рівні документа за допомогою схеми, додатки, що генерують і приймають транзакції, можна оптимізувати для забезпечення максимальної швидкодії. Відповідність полів і правильність записів перевіряються на рівні примірників XML.

Кореневим елементом у схемі XDR завжди є елемент Schema:

```
<Schema
  name="ім'я_схеми" xmlns="urn:schemas-microsoft-com:xml-
data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <-- Оголошення інших елементів -->
</Schema>
```

Елемент *ElementType* має синтаксис:

```
<ElementType
  content="{empty | texOnly | eltOnly | mixed}">
  dt:type "datatype"
  model="{open | closed}"
  name = "idref"
  order="{one | seq | many}"
>
```

Елемент *ElementType* може мати такі атрибути (табл. 10.5).

Таблиця 10.5.

Атрибути елементу *ElementType*

Ім'я атрибута	Опис
name	Ім'я елемента
content	Зміст елемента. Допустимі значення: <i>empty</i> (порожній елемент), <i>eltOnly</i> (може бути тільки контейнером для інших елементів), <i>textOnly</i> (тільки текстові дані), <i>mixed</i> (змішані дані).
dt:type	Тип даних елементу
model	Може приймати значення:
	<i>Open</i> - дозволено використовувати елементи, які не визначені в схемі
	<i>Closed</i> - заборонено використовувати елементи, які не визначені в схемі
order	Порядок проходження дочірніх елементів у примірнику XML. Допустимі значення:
	<i>one</i> – передбачається наявність одного документа
	<i>many</i> – будь-яка кількість елементів в будь-якому порядку
	<i>seq</i> – елементи вказуються в строго заданому порядку.

В якості дочірніх елементів для *ElementType* використовують (табл. 10.6).

Таблиця 10.6.

Дочірні елементи *ElementType*

Ім'я елемента	Опис
element	Оголошує дочірній елемент
description	Забезпечує опис елемента <i>ElementType</i>
datatype	Забезпечує тип даних елемента <i>ElementType</i>
group	Визначає порядок проходження елементів
AttributeType	Визначає атрибут
attribute	Визначає відомості про дочірній елемент <i>AttributeType</i>

Для оголошення атрибутів використовується синтаксис:

```
<AttributeType
    default="default-value"
    dt:type="primitive-type"
    dt:values="enumerated-values"
    name="idref"
    required="{yes | no}"
>
```

У свою чергу елемент `AttributeType` може мати атрибути (табл. 10.7).

Таблиця 10.7.

Атрибути елементу `AttributeType`

Значення	Опис
default	Значення за замовчуванням
dt:type	Один з наступних типів: <i>entity, entities, enumeration, id, idref, nmtoken, nmtokens, notation, string</i>
dt:values	Допустимі значення
name	Ім'я атрибута
required	Вказує на обов'язкову наявність атрибута в описі

Синтаксис для опису елемента *attribute* виглядає наступним чином:

```
<attribute
    default="default-value"
    type="attribute-type"
    [required="{yes | no}"
]>
```

а його можливі значення можуть бути такими (табл. 10.8).

Таблиця 10.8.

Можливі значення елементу *AttributeType*

Значення	Коментар
default	Значення за замовчуванням
type	Ім'я елементу <i>AttributeType</i> , визначеного в даній схемі. Має відповідати атрибуту <i>name</i> елемента <i>AttributeType</i>
required	Вказує на обов'язкову наявність атрибута в описі

На відміну від DTD схем XDR підтримує типи даних. Елемент *Schema* має наступний атрибут:

Xmlns:dt="urn=schemas-microsoft-com:datatypes"

З повним списком типів даних можна ознайомитись на сторінці за адресою:

[http://msdn.microsoft.com/en-us/library/ms256121\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms256121(VS.85).aspx)

Індикатори входження в схемах XDR мають синтаксис:

```
<element
  type="element-type"
  [minOccur="{0 | 1}"]
  [maxOccur="{1 | *}"]
>
```

XDR схема дозволяє визначати *групи змісту*. Так, в елементі *ElementType* може міститися елемент *group*, що має синтаксис:

```
<group          order="(one | seq | many)"          minOccur="(0 | 1)"
  maxOccur="(1 | *)">
  <element type="ElementType/">
  <element type="ElementType/">
  <element type="ElementType/">
```

```
<element type="ElementType"/>
</group>
```

На закінчення наведемо приклад XSD схеми, яка описує структуру XML документа, що містить листи електронної пошти:

```
<?xml version = "1.0"?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">

  <xsd:element name="m_box">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="message" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="message">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="head" minOccurs="1"
maxOccurs="1"/>
        <xsd:element ref="body" minOccurs="1"
maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="uid" use="required"
type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="head">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="to" minOccurs="1"
maxOccurs="unbounded"/>
        <xsd:element ref="from" minOccurs="1"
```

```
maxOccurs="1"/>
    <xsd:element          ref="date"          minOccurs="1"
maxOccurs="1"/>
    <xsd:element          ref="subject"        minOccurs="1"
maxOccurs="1"/>
    <xsd:element          ref="cc"            minOccurs="0"
maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="to" type="xsd:string"/>
<xsd:element name="from" type="xsd:string"/>
<xsd:element name="date" type="xsd:string"/>
<xsd:element name="subject" type="xsd:string"/>
<xsd:element name="cc" type="xsd:string"/>

</xsd:schema>
```

Для перевірки дійсності XML документа можна використовувати спеціальні валідатори, наприклад W3C валідатор (<http://validator.w3.org/>).

Для перевірки схем також існують спеціальні валідатори, наприклад *XML Schema валідатор* (<http://www.w3.org/2001/03/webdata/xsv>).

Згідно специфікації W3C XML програма повинна припинити обробку XML документа, як тільки буде виявлена помилка в цьому документі.

Лекція 11. DOM XML. Перетворення XML документів.

План:

1. Модель XML DOM.
2. Мови XSLT и XPath.
3. Мова XSL-FO.
4. Мова XQuery.

Література:

1. Ньюкомер Э. Веб-сервисы: XML, WSDL, SOAP и UDDL / Э. Ньюкомер. – СПб.: Питер, 2003. – 345 с.
2. Старыгин А. XML: разработка Web-приложений / А. Старыгин. – СПб.: «БХВ-Петербург», 2003. – 585 с.

1. Модель XML DOM.

Для програмної обробки XML документів використовується модель XML DOM, яка визначає об'єкти і властивості всіх XML елементів і методи (інтерфейс) для доступу до них. Інакше кажучи, XML DOM описує яким чином необхідно отримувати, змінювати, додавати і видаляти XML елементи.

Згідно DOM:

- все, що міститься всередині XML документу, є вузлом;
- весь документ представляється *вузлом документу*;
- кожен XML елемент – *вузол елементу*;
- текст всередині XML елементів – *текстовим вузлом*;
- кожен атрибут – *вузол атрибуту*;
- коментарі – *вузли коментарів*.

XML документ відповідно до моделі XML DOM представляється як дерево з вузлів, при цьому:

1. Усі вузли дерева знаходяться в певних відносинах один з одним.
2. Усі вузли доступні через дерево. Їх зміст може бути змінено, видалено; нові елементи можуть бути додані в дерево.
3. Дерево починається з кореневого вузла і розгалужується вниз аж до текстових вузлів на самому нижчому рівні дерева.
4. Усі вузли знаходяться в *ієрархічних відносинах* між собою.
5. Ці відносини описуються за допомогою понять батько, дочірній і нащадок (всі дочірні на одному рівні).

Альтернативним інтерфейсом для обробки XML документів є SAX.

SAX (*Simple API for XML*) – прикладний програмний інтерфейс для парсеру з послідовним доступом до XML. Цей інтерфейс надає механізм читання даних з XML документа.

SAX парсер є потоковим і керуючий *подіями*. Завданням користувача SAX API полягає в описі методів, що викликаються подіями, які виникають при аналізі документу.

Такими подіями можуть бути: текстовий вузол; вузол елемента XML; інструкція обробки XML; коментар XML.

Події викликаються появою як відкриваючого тега, так і закриваючого тега будь-якого з цих елементів документу. *Атрибут* XML також розглядається як подія. Аналіз документа є односпрямованим (тобто без повернень по дереву).

На відміну від DOM формальної специфікації для SAX не існує. У якості нормативної розглядається Java реалізація SAX.

Слід відзначити наступні переваги і недоліки SAX.

Переваги:

1. Витрати пам'яті істотно менше (залежить від максимальної глибини дерева документа та кількості атрибутів у вузлі елемента), ніж у випадку DOM (потрібно зберігати в пам'яті все дерево документу).
2. Швидкість роботи вище за рахунок скорочення витрат часу на виділення пам'яті для елементів дерева в разі DOM.
3. Потокowe читання даних з диска в разі DOM неможливо. Якщо для розміщення всього документа в пам'яті недостатньо місця, то використання SAX є безальтернативним.

Недоліки:

1. Процедура перевірки правильності передбачає доступ до всього документу одночасно.
2. Це також потрібно і в разі XSLT перетворення.

Якщо завантажити «чистий» XML-документ у веб-браузер, то можна буде побачити деревоподібну структуру цього документу (рис. 11.1).

У цьому якраз і полягає головна відмінність між XML і HTML, а саме поділ структури документа і його представлення в браузері. Конкретний вид XML-документу описується окремо за допомогою CSS або XSL.

CSS і XSL – принципово різні технології, що мають лише часткове пересічення областей застосування. CSS-форматування застосовується до HTML-документу браузером на *клієнтській стороні*, а XSL-перетворення виконується, як правило, *на сервері*, після чого результат відправляється браузеру клієнта. XSL базується на XML, завдяки чому XSL більш *гнучкий і універсальний*. У розробників є можливість використовувати засоби контролю за коректністю складання стильових списків (використовуючи схеми XML).

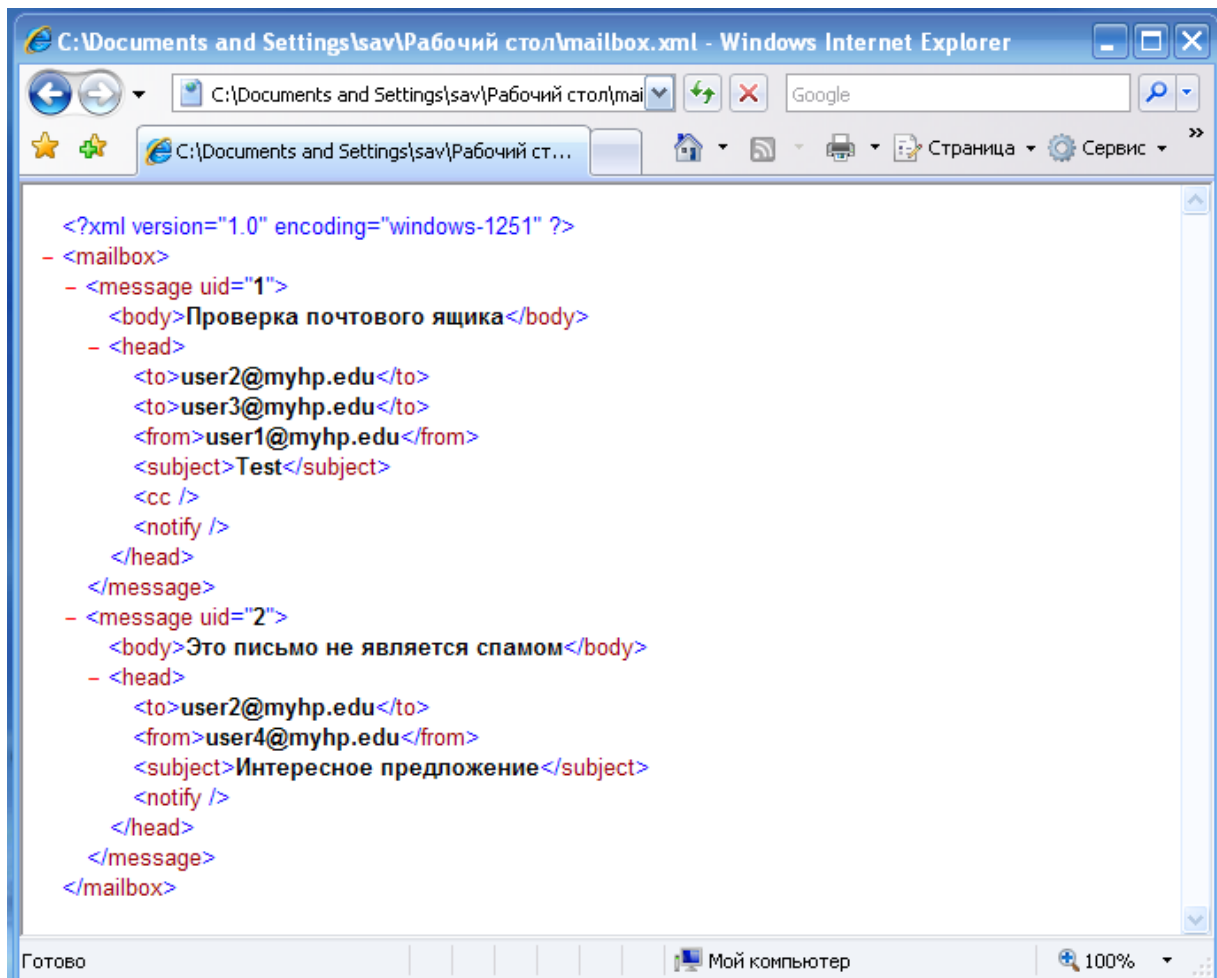


Рис. 11.1. Деревоподібна структура XML-документу.

За допомогою XSL можна перетворити XML-документ у формат *HTML*, *WML*, *RTF*, *PDF*, *SQL*, *SWF*, і так само в *інший XML* і *XSL* документ. XSL вказує як буде оформлений документ, де і яким чином повинні розміщуватися дані.

Специфікація XSL складається з трьох частин:

1. *XSLT* (XSL Transformations) – мова для перетворення XML.
2. *XPath* – мова шляхів і виразів, використовуваний в XSLT для доступу до окремих частин XML-документа.
3. *XSL-FO* (XSL Formatting Objects) – мова для верстки XML.

Найбільш поширеним механізмом XSLT перетворень для систем, що працюють на платформі Microsoft Windows є MSXML; для систем на основі GNU - *xsltproc*.

Для того, щоб обробити XML документ за допомогою XSL, необхідно в XML-документі написати наступну інструкцію:

```
<? Xml version = "1.0" ?>
<? Xml-stylesheet type = "text / xsl" href = "my-style.xsl" ?>
<Root>
    <! - ... ->
</ Root>
```

Повертаючись до розглянутого раніше прикладу, додавши в XML файл посилання на XSL файл, отримаємо наступний код розмітки:

```
<?xml version="1.0" encoding="windows-1251" ?>
<!DOCTYPE mailbox SYSTEM "mailbox.dtd">
<?xml-stylesheet href="mailbox.xsl" type="text/xsl" ?>

<mailbox>
  <message uid='1'>
    <body>Перевірка поштової скриньки</body>
    <head>
      <to>user2@myhp.edu</to>
      <to>user3@myhp.edu</to>
      <from>user1@myhp.edu</from>
      <subject>Test</subject>
      <cc></cc>
      <notify></notify>
    </head>
  </message>
  <message uid='2'>
    <body>Цей лист не є спамом </body>
    <head>
      <to>user2@myhp.edu</to>
      <from>user4@myhp.edu</from>
```

```

<subject>Цікава пропозиція </subject>
<notify></notify>
</head>
</message>
</mailbox>

```

Після завантаження даного документа в веб-браузері його вид кардинально зміниться (рис. 11.2).

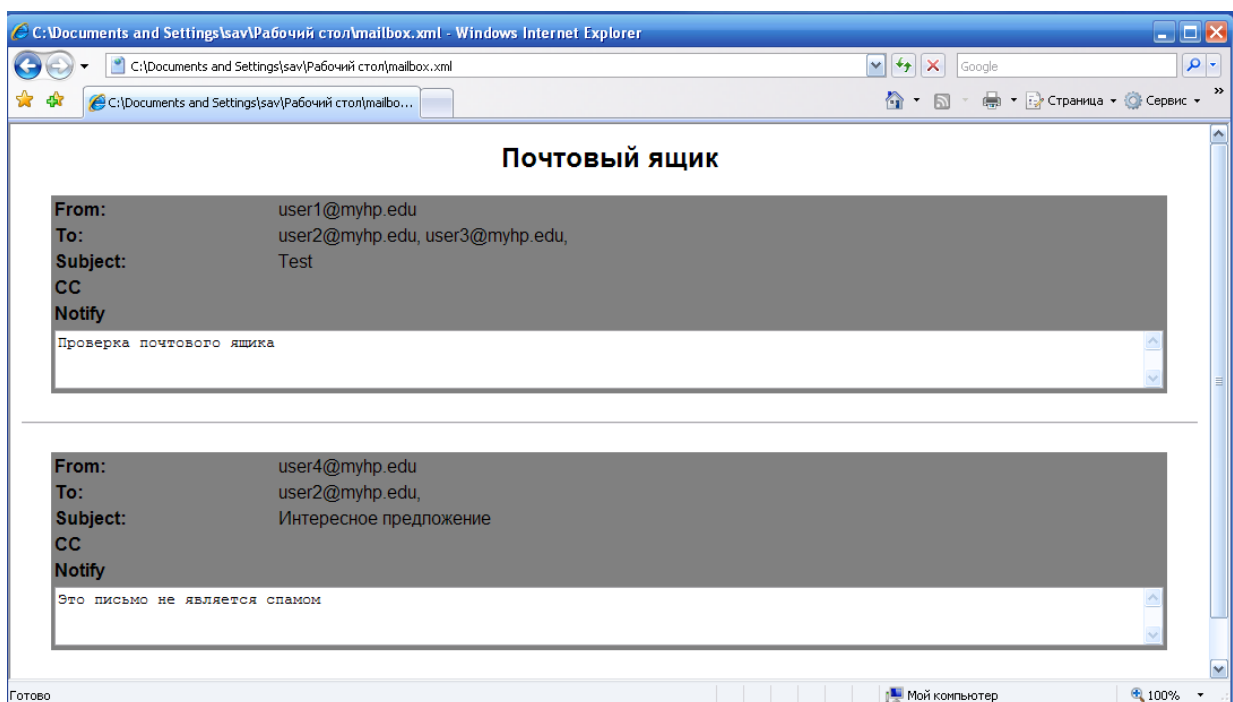


Рис. 11.2. Вид XML-документу в веб-браузері.

Зміст XSL файлу mailbox.xsl наводиться нижче:

```

<?xml version="1.0" encoding="windows-1251"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match = "mailbox">
  <h2 align="center" color="red">Поштова скринька </h2>
  <xsl:apply-templates/>
</xsl:template>

```

```
<xsl:template match = "message">
  <table align="center" border="0" width="80%" bgcolor="gray">
    <tr><td width="20%"><b>From:</b></td><td><xsl:value-of
select="head/ from"/></td></tr>
    <tr>
      <td width="20%"> <b>To:</b> </td>
      <td> <xsl:for-each select="head/to">
        <xsl:apply-templates/>,
      </xsl:for-each>
    </td>
  </tr>
  <tr><td width="20%"><b>Subject:</b></td><td><xsl:value-of
select="head/ subject"/></td></tr>
  <tr><td width="20%"><b>CC</b></td><td><xsl:value-of
select="head/cc"/></td></tr>
  <tr><td width="20%"><b>Notify</b></td><td><xsl:value-of
select="head/ notify"/></td></tr>
  <tr><td colspan="2">
    <textarea cols="120%" rows="3"> <xsl:value-of
select="body"/> </textarea>
  </td></tr>
</table>
<br/><hr/><br/>
</xsl:template>

</xsl:stylesheet>
```

2. Мови XSLT и XPath.

Специфікація XSLT є рекомендацією W3C.

В результаті застосування таблиці стилів XSLT, що складається з набору шаблонів, до XML-документу (вихідне дерево) утворюється кінцеве дерево, яке може бути іншою XML-структурою, HTML-документом або звичайним текстом. Правила вибору даних з вихідного дерева записуються на мові запитів XPath. XSLT застосовується в основному в веб-програмуванні і для генерації звітів.

Завдяки XSLT реалізується відділення даних від їх подання в рамках парадигми MVC (*Model-view-controller*).

XPath (XML Path Language) – мова запитів до елементів XML-документа. XPath був розроблений для організації доступу до частин документа XML в файлах трансформації XSLT і є стандартом консорціуму W3C. У мові XPath використовується компактний синтаксис, відмінний від прийнятого в XML. Починаючи з версії 2.0, XPath є складовою частиною мови *XQuery*. XPath покликаний допомогти обходити всілякі дерева, отримувати необхідні елементи з іншої гілки відносно точки обходу, розпізнавати предків, нащадків, атрибути елементів. Це повноцінна мова навігації по дереву.

Для перебування елемента(ів) в дереві документа використовуються шляхи адресації.

Кожен крок адресації складається з трьох частин:

- осі, наприклад *child::*;
- умови перевірки вузлів, наприклад імена елементів документа *body, html*;
- предиката, наприклад *attribute::class*.

Доповненням до ядра мови є набір функцій, які діляться на 5 груп: системні функції, функції з множинами, рядкові функції, логічні функції, числові функції.

3. Мови XSL-FO.

XSL-FO (eXtensible Markup Language Formatting Objects) – рекомендована W3C мова розмітки переддрукарських матеріалів. По-суті, XSL-FO – це уніфікована мова уявлення. Вона не має семантичної розмітки (як в HTML) і зберігає всі дані документа всередині себе (на відміну від CSS, який модифікує уявлення за замовчуванням для зовнішнього HTML або XML-документу).

В результаті застосування *XSLT-перетворення* до вихідного XML-документу виходить його опис мовою *XSL-FO*. *FO-процесор* конвертує XSL-FO-документ в який-небудь читаний і/або друкований формат. Найбільш часто використовується перетворення в PDF або PS; деякі FO-процесори можуть давати на виході RTF-файли або просто показувати документ у вікні.

4. Мови XQuery.

XQuery – мова запитів, розроблений для обробки даних у форматі XML.

В даний час робочими групами консорціуму W3C ведуться роботи з розвитку цього стандарту даної мови, з додаванням виразів для вільного пошуку по тексту і для внесення змін в XML документи і бази даних, а також для процедурних операцій.

В рамках стандарту *SQL:2006* розроблені механізми для вбудовування *XQuery*-запитів безпосередньо в SQL-запити.

Лекція 12. Інтеграція та взаємодія в мережі Інтернет.

План:

1. Інтеграція на основі XML.
2. Веб-сервіси (веб-служби).
3. Специфікація WSDL.
4. Протокол SOAP.
5. Стандарт DISCO.
6. Специфікація UDDI.

Література:

1. Ньюкомер Э. Веб-сервисы: XML, WSDL, SOAP и UDDI / Э.Ньюкомер. – СПб.: Питер, 2003. – 357 с.
2. Шорт С. Разработка XML Web-сервисов средствами Microsoft .NET / С.Шорт. – СПб.: «БХВ-Петербург», 2003. – 480 с.

У багатьох компаніях вже склалася тенденція надавати своїм співробітникам, партнерам і клієнтам доступ до всіх типів інформації та сервісів у вигляді мережі Інтернет. Однак в корпоративних мережах компаній функціонує величезна кількість різнорідних бізнес-додатків, створених в різний час, різними організаціями, на базі різних технологій. Завдання веб-інтеграції полягає в тому, щоб об'єднати різнорідні веб-додатки і системи в єдине середовище на базі мережі Інтернет.

Практикуються такі підходи до веб-інтеграції:

1. Інтеграція на рівні уявлення. Даний рівень дозволяє користувачеві взаємодіяти з додатком. Інтеграція на рівні уявлення дає *доступ до інтерфейсу користувача віддалених програм*.
2. Інтеграція на рівні функціональності. Дана інтеграція передбачає забезпечення прямого

доступу до бізнес-логіки додатків. Це досягається безпосередньою взаємодією додатків з *API* (програмного інтерфейсу додатків) або ж взаємодією у вигляді *веб-сервісів*.

3. Інтеграція на рівні даних. У даному випадку передбачається доступ до однієї або декількох *баз даних*, використовуваних віддаленим додатком.
4. Комплексна інтеграція. Комерційні рішення з веб-інтеграції, як правило, включають всі три типи інтеграції

Використання веб-інтеграції вигідно з багатьох причин:

1. *Веб-інтеграція* дозволяє розгортати інформаційні системи на базі сторонніх додатків без необхідності розбиратися в їх батьківських системах, програмних середовищах і архітектурах баз даних.
2. *SOA* і *веб-сервіси* використовують програмну мову і платформи-незалежні інтерфейси між додатками корпоративної інфраструктури ІТ. Це дає очевидні переваги в підтримці, керованості, розгортанні інформаційних мереж.
3. Веб-інтеграція дозволяє конструювати комплексну функціональність, комбінуючи різноманітні компоненти у вигляді протоколів веб-сервісів.
4. Веб-інтеграція дозволяє використовувати веб-сервіси розробників.
5. Веб-інтеграція дозволяє розвивати програмні інтерфейси додатків через протоколи веб-сервісів без програмування.

Для веб-інтеграції зазвичай використовується комерційне ПЗ або популярні технології, такі як *PHP/Python/Perl, XForms, SOAP* і таке інше.

1. Інтеграція на основі XML.

Велика кількість систем, стандартів і технологій призводить до того, що ефективно зв'язати різні джерела даних в одну систему не виходить. Навіть такі, на перший погляд однорідні джерела, як системи управління базами даних, застосовують мови запитів і формати подання обраних інформації, які рідко повністю сумісні між собою. Як наслідок, проекти інтеграції в таких умовах вимагають великих зусиль – потрібно вникати в деталі різних баз даних, протоколів, операційних систем і так далі. В результаті інтеграція декількох додатків або систем реалізується за схемою, представленій нижче:

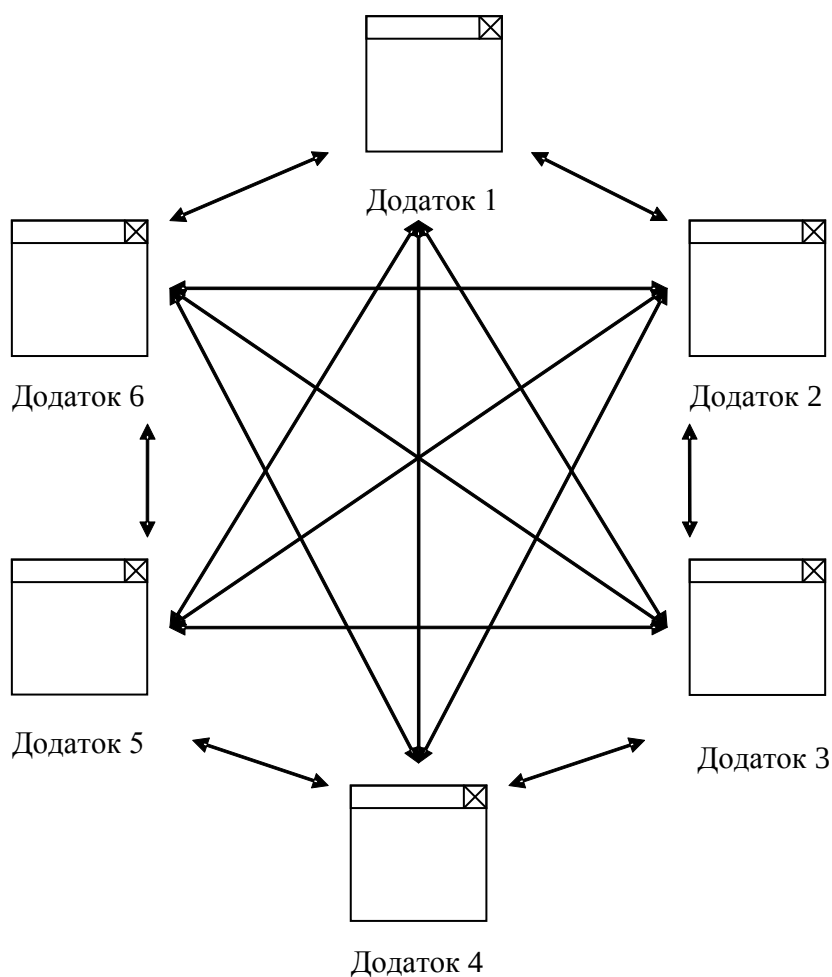


Рис. 12.1. Схема додатків.

Змусити різні системи працювати разом – надзвичайно трудомістке завдання. Ідея використання XML в інтеграції інформаційних систем зводиться до створення загальної XML-мови, якою могла б користуватися кожна з них.

Таке рішення відразу ж набагато спрощує проект. Замість реалізації взаємодії між кожною парою систем слід всього лише навчити кожну з них «говорити» XML мовою. Інакше кажучи, все зводиться до розробки декількох *вранперів* (wrapper – пакувальник, програмний засіб створення системної оболонки для стандартизації зовнішніх звернень та зміни функціональної орієнтації діючої системи), які будуть перекладати зі стандартної XML-мови інтегрованої системи на мову, зрозумілу кожній системі окремо:

- засоби розробки та стандартні бібліотеки для XML існують практично на всіх платформах і для більшості популярних мов програмування;
- методи роботи з XML досить стандартні для того, щоб в різних системах можна було користуватися однаковими прийомами;
- інформація, оформлена у вигляді XML, може оброблятися не тільки машинами, а й людиною (що набагато полегшує налагодження).

Інтеграція на основі XML практично реалізується у рамках протоколів:

- *XML-RPC*. Це протокол віддаленого виклику процедур з передачею даних у форматі XML через TCP-порт 80, тобто HTTP-порт.
- *WDDX* (Web Distributed Exchange). Являє собою механізм обміну складними структурами даних по протоколу HTTP. Протокол базується не на структурах, а на події.
- *ebXML* (electronic business XML) - XML для електронного бізнесу. Основне призначення –

надання відкритої XML-інфраструктури, що забезпечує безпечне глобальне використання інформації електронного бізнесу.

- Веб-сервіси (веб-служби).

В принципі, інтеграція за XML-схемою не відрізняється докорінно від інтеграції на основі будь-якого іншого загального стандарту. Разом з тим, вона має цілий ряд вагомих переваг:

- XML мова не залежить від апаратних і програмних платформ, що дозволяє пов'язувати різномірні системи;
- виразна потужність XML досить велика для того, щоб описати дані практично будь-якої складності.

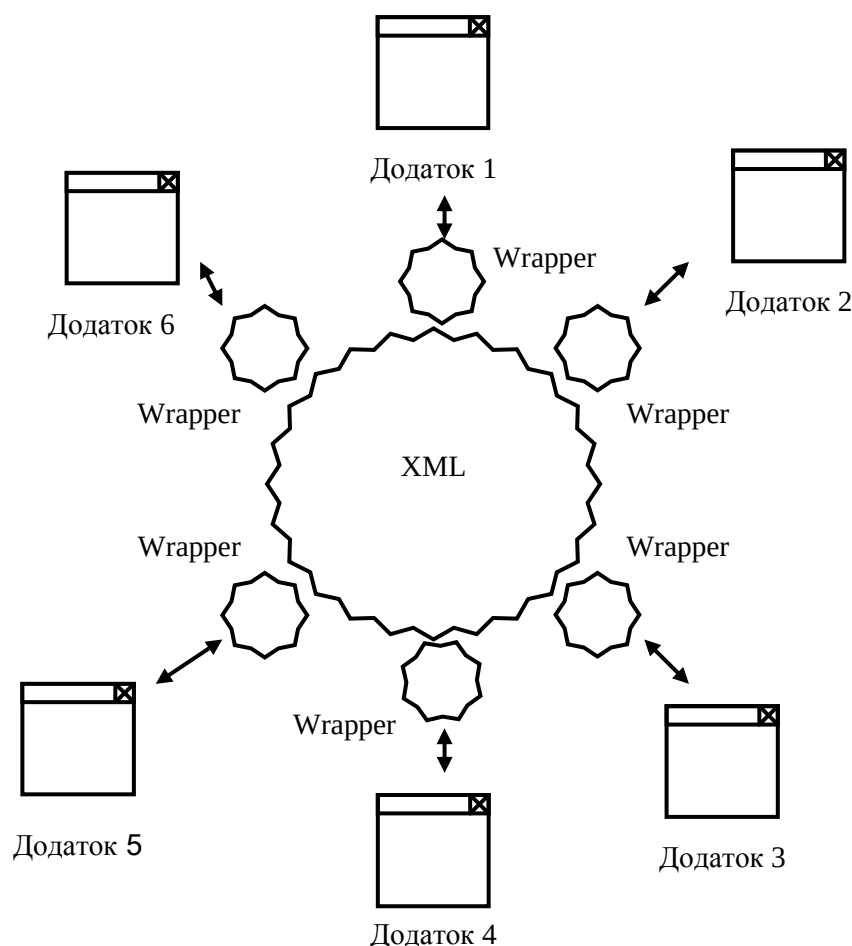


Рис. 12.2. XML-схема додатків.

2. Веб-сервіси (веб-служби).

Веб-сервіс (web service) – програмна система, що має ідентифікатор URI, і загальнодоступні інтерфейси які визначені на мові XML. Опис цієї програмної системи може бути знайдено іншими додатками, які можуть взаємодіяти з нею відповідно до цього опису у вигляді повідомлень, заснованих на XML, і переданих за допомогою інтернет-протоколів. Веб-служба є одиницею модульності при використанні *сервіс-орієнтованої архітектури* додатку.

Сервіс-орієнтована архітектура (SOA, service-oriented architecture) – модульний підхід до розробки програмного забезпечення, заснований на використанні сервісів зі стандартизованими інтерфейсами.

В основі SOA лежать принципи багатократного використання функціональних елементів ІТ, уніфікації типових операційних процесів. Компоненти програми можуть бути розподілені між різними вузлами мережі, і пропонуються як незалежні і слабо пов'язані, замінні сервіси-додатки.

Інтерфейс компонентів SOA-програми здійснює інкапсуляцію деталей реалізації конкретного компонента (ОС, мови програмування і таке інше).

SOA добре зарекомендувала себе при побудові великих корпоративних програмних систем. Цілий ряд розробників та інтеграторів пропонують інструменти і рішення на основі SOA (наприклад, платформи Microsoft .NET, IBM WebSphere, SAP NetWeaver, Diasoft та ін.).

Веб-сервіси .NET мають такі переваги:

1. Відкритість стандартів. У веб-сервісах відсутні будь-які приховані або недоступні елементи. Кожен аспект технології, від способу пошуку веб-сервісу до її опису і організації зв'язку з нею, визначений загальнодоступними стандартами.

2. *Міжплатформеність.* Мова програмування, що дозволяє створювати XML-документи і відправляти інформацію за допомогою HTTP, дозволяє взаємодіяти з будь-яким веб-сервісом. Можна отримувати веб-послугу з системи, відмінної від .NET.
3. *Простота.*
4. *Підтримка повідомлень на зрозумілій людині мовою.* Перехід від двійкових стандартів, застосовуваних у COM і CORBA, до XML-тексту дозволив спростити виправлення помилок і забезпечив можливість здійснювати взаємодію з веб-сервісами по звичайних каналах HTTP.

Реалізація веб-сервісів .NET здійснюється так само просто, як і активізація віддаленого веб-сервісу або виклик методу локального класу. Це досягається за рахунок застосування інструментів, що надаються системою .NET Framework, які дозволяють створити повноцінний веб-сервіс, без необхідності вивчення деталей роботи таких стандартів, як SOAP, WSDL і UDDI. При цьому виконуються наступні дії:

1. Веб-сервіс розробляється як .NET-клас з атрибутами, які ідентифікують його як веб-сервіс з деякими функціями.
2. У середовищі .NET автоматично створюється документ WSDL, де описується, як клієнт повинен взаємодіяти з веб-сервісом.
3. Споживач знаходить створений веб-сервіс і може додати відповідне веб-посилання в проект Visual Studio .NET.
4. У середовищі .NET здійснюється автоматична перевірка документа WSDL і генерується проксі-клас, який дозволяє споживачу взаємодіяти з веб-сервісом.

5. Споживач викликає один з методів вашого класу веб-сервісу. З його точки зору цей виклик зовні нічим не відрізняється від виклику методу будь-якого іншого класу, хоча взаємодія відбувається насправді з проксі-класом, а не з веб-сервісом.
6. Проксі-клас перетворює, передані параметри в повідомлення SOAP і відправляє його веб-сервісу.
7. Потім проксі-клас отримує SOAP-відповідь, перетворює його у відповідний тип даних і повертає його як звичайний тип даних .NET.
8. Споживач використовує отримані дані.

При роботі веб-сервісів .NET використовується технологія ASP .NET, що є частиною системи .NET Framework. Вона також вимагає підтримки з боку сервера Microsoft IIS.

Робота веб-сервісів побудована на використанні декількох відкритих стандартів:

- *XML* – розширювана мова розмітки, призначена для зберігання і передачі структурованих даних;
- *SOAP* – протокол обміну повідомленнями на базі XML;
- *WSDL* – мова опису зовнішніх інтерфейсів веб-сервісів на базі XML;
- *UDDI* – універсальний інтерфейс розпізнавання, опису та інтеграції (Universal Discovery, Description, and Integration). Каталог веб-сервісів і відомостей про компанії, що надають веб-сервіси в загальне користування або конкретним компаніям.

3. Специфікація WSDL.

Кожен веб-сервіс надає документ WSDL (Web Service Description Language – мова опису веб-сервісу), в якому описується все, що клієнту необхідно для роботи з цим

сервісом. WSDL-документ надає простий і послідовний спосіб завдання розробником синтаксису виклику будь-якого веб-методу. Більше того, цей документ дозволяє використовувати інструменти автоматичного генерування проксі-класів, подібні включеним в середовища Visual Studio .NET і .NET Framework. Завдяки зазначеним засобам використання веб-сервісу є таким же простим, як і застосування локального класу.

WSDL-документ має заснований на XML формат, відповідно до якого інформація поділяється на п'ять груп. Перші три групи є абстрактні визначення, які не залежать від особливостей платформи, мережі або мови, а дві групи, що залишилися включають конкретні описи.

4. Протокол SOAP.

Зв'язок між веб-сервісами та їх клієнтами здійснюється за допомогою повідомлень у форматі XML.

SOAP (Simple Object Access Protocol – простий протокол доступу до об'єктів) являє собою протокол повідомлень для вибору веб-сервісів.

Основна ідея стандарту SOAP полягає в тому, що повідомлення повинні бути закодовані в стандартизованому XML-форматі.

Крім повідомлень SOAP, для обміну даними з сервісами .NET можна використовувати методи GET і POST протоколу HTTP.

Переваги застосування формату SOAP перед іншими форматами для передачі даних:

1. Кодувати в XML структури даних та набори DataSet з використанням SOAP так само легко, як і дані простих скалярних типів.
2. При використанні SOAP-повідомлень надаються додаткові інструменти, що дозволяють легко

додавати, наприклад, функції забезпечення безпеки або трасування.

3. Є набори інструментів SOAP для різних мов програмування (і навіть для попередніх версій Microsoft C++ і Visual Basic). Інакше, для того щоб забезпечити зв'язок з сервісом за допомогою методів GET і POST протоколу HTTP, доведеться, очевидно, самотійно конструювати рядок запиту, а потім проводити синтаксичний аналіз відповіді.

5. Стандарт DISCO.

Стандарт DISCO надає найпростіший спосіб отримання доступу до файлів маніфестів, що дозволяє групувати посилання на веб-сервіси.

DISCO-файл може включати файли різних веб-серверів і підтримує «динамічний пошук» – автоматичний пошук каталогу файлів веб-сервісів на сервері.

Файли маніфесту корисні тим, що об'єднують безліч веб-сервісів в єдиному списку, однак вони не дозволяють клієнтам відшукувати веб-сервіси певного типу без вказівки найменування компанії-розробника.

6. Специфікація UDDI.

Специфікація UDDI (Universal Description, Discovery, and Integration – універсальне опис, пошук і інтеграція) дозволяє уникнути зазначених проблем за допомогою використання спеціального сховища (репозиторію), де підприємства та організації можуть розміщувати дані про надані ними сервісах. Ініціаторами створення технології UDDI стали понад 100 компаній (повний список можна знайти за адресом <http://www.uddi.org/community.html>), включаючи Sun і Microsoft. Об'єднавши свої зусилля, ці

компанії розробили проект специфікації UDDI, яка після закінчення 18 місяців була стандартизована.

Інформація в цьому репозиторії повинна оновлюватися вручну. З цією метою деякі «вузлові оператори» зберігають ідентичні копії репозиторію UDDI. Ці компанії забезпечують зберігання зазначеного сховища та безкоштовний доступ до нього для популяризації веб-сервісів. Крім того, Майкрософт включила версію UDDI в програмне забезпечення сервера Windows .NET для використання в корпоративних мережах Інтернету.

У сховищі UDDI містяться відомості про підприємства, що надають веб-сервіси, про тип кожного сервісу і зв'язках з інформацією і специфікаціями, що відносяться до цих сервісів. Інтерфейс UDDI сам по собі представляє собою веб-сервіс. Для реєстрації або пошуку служби слід відправити SOAP-повідомлення.

Головними недоліками веб-сервісів є менша продуктивність і більший розмір мережевого трафіку в порівнянні з такими технологіями як RMI, CORBA, DCOM за рахунок використання текстових XML-повідомлень.

Лекція 13. Організація процесу розробки веб-контенту CMS/CMF.

План:

1. Система керування веб-контентом.
2. Система Drupal.

Література

1. Беллиньясо М. Разработка Web-приложений в среде ASP.NET 2.0: с примерами на С# / М. Беллиньясо. – М. : «Диалектика», 2007. – 640 с.
2. Старыгин А. XML: разработка Web-приложений / А. Старыгин. – СПб. : «БХВ-Петербург», 2003. – 585 с.
3. Шорт С. Разработка XML Web-сервисов средствами Microsoft .NET / С. Шорт. – СПб. : «БХВ-Петербург», 2003. – 480 с.

1. Система керування веб-контентом.

Система управління контентом (Content management system, CMS) – комп'ютерна програма, яка використовується для створення, редагування, управління та публікації контенту деяким систематичним чином.

Зазвичай такі системи використовуються для зберігання і публікації великої кількості документів, зображень, музики або відео.

Система керування веб-контентом (Web content management system, WCMS або Web CMS) – програмне забезпечення CMS класу, реалізоване зазвичай у вигляді веб-додатку, і призначене для створення, і керування HTML вмістом. WCMS зазвичай використовується для управління і контролю великими, динамічно змінюваними колекціями веб-матеріалу (HTML документами і

пов'язаними з ними картинками). Така система спрощує процес створення, управління, редагування контенту і багато інших важливих завдань, які пов'язані з підтримкою цих процесів.

WCMS надає наступні можливості:

1. *Застосування автоматичних шаблонів відображення* (в HTML або XML форматі), автоматично застосовуються до нового або існуючого контенту. Тим самим вигляд всіх документів може задаватися з одного місця.
2. *Простота редагування контенту.* Користувачу досить легко створювати і управляти контентом, оскільки йому або взагалі не потрібно знання мов програмування або мов розмітки, або потрібно мінімальне знання таких.
3. *Масштабованість.* Можливість розширення функціональності існуючого сайту шляхом установки поставляються з дистрибутивом WCMS плагінів і модулів.
4. *Управління документами.* Є засоби управління життєвим циклом документів з моменту створення до видалення.
5. *Візуалізація контенту.* Будь-який користувач може працювати з віртуальною копією всього веб-сайту, безлічі документів або кодами програм, що дозволяє побачити всі зміни безлічі взаємозалежних ресурсів перед їх остаточним застосуванням.

Залежно від способу застосування шаблонів для генерації веб-сторінок прийнято виділяти три основні типи WCMS-систем: з автономною обробкою, он-лайн обробкою і гібридні системи.

Автономні системи обробляють весь зміст шляхом застосуванням шаблонів перед публікацією веб-сторінок.

On-line системи застосовують шаблони в момент відвідування сайту користувачами (або витягають сторінки і кеша).

Гібридні системи комбінують перші два підходи. Деякі з них замість статичних *HTML* сторінок генерують виконувані коди (*JSP, PHP, Perl*), позбавляючи від необхідності установки WCMS-системи на кожному веб-сервері.

Як приклад системи розглянемо WCMS Drupal.

2. Система Drupal.

Drupal – це WCMS система, розроблена на мові PHP і використовує як сховище даних реляційну базу даних (підтримуються MySQL, PostgreSQL та інші). Архітектура *Drupal* дозволяє застосовувати його для побудови різних типів сайтів – від блогів і форумів, до інформаційних архівів або сайтів новин.

Функціональність забезпечується модулями, що звертаються до загального *API Drupal*. Стандартний набір модулів включає, наприклад, такі функції як новинна стрічка, блог, форум, завантаження файлів, збирач новин, голосування, пошук та ін.

Найбільш важливі функції, надані модулями входять в постачання Drupal:

- єдина категоризація всіх видів вмісту (таксономія)
 - від форумних повідомлень до блогів і новинних статей;
- широкий набір властивостей при побудові рубрикаторів: плоскі списки, ієрархії, ієрархії із загальними предками, синоніми, споріднені категорії;
- вкладеність категорій будь-якої глибини;
- пошук за змістом сайту, в тому числі пошук по таксономії і користувачам;

- розмежування доступу користувачів до документів;
- динамічна побудова меню;
- підтримка XML-форматів: виведення документів в RDF / RSS; агрегації матеріалів з інших сайтів; BlogAPI для публікації матеріалів за допомогою зовнішніх програм;
- підтримка змінних тем оформлення сайту з наданням декількох готових варіантів;
- переклади інтерфейс сайту на різні мови, а також підтримка ведення різномовного контенту;
- можливість створення сайтів з пересічним змістом (наприклад загальною базою користувачів або загальними налаштуваннями);
- роздільні конфігурації сайту для різних віртуальних хостів (у тому числі власні набори модулів і тем оформлення для кожного підсайту);
- механізм для обмеження навантаження на сайт (автоматичне відключення при високій відвідуваності частини інформаційних блоків і модулів).

Існує величезна безліч систем як комерційних, так і безкоштовних. Наприклад Майкрософт пропонує реалізацію WCMS системи на базі *Windows SharePoint Services*.

Каркасна система управління змістом (Content Management Framework, CMF) – це інструментарій для створення систем керування змістом, а також окремих веб-додатків. Деякі CMS, що надають API для розширення своєї функціональності, можна розглядати як CMF, наприклад *WCMS Drupal*.

Лекція 14. Синдикація і агрегування веб-контенту.

План:

1. Синдикація веб-контенту.
2. Агрегування веб-контенту.
3. RSS – сімейство XML-форматів.

Література:

1. Беллиньясо М. Разработка Web-приложений в среде ASP.NET 2.0: с примерами на С# / М. Беллиньясо. – М. : «Диалектика», 2007. – 640 с.
2. Старыгин А. XML: разработка Web-приложений / А. Старыгин. – СПб. : «БХВ-Петербург», 2003. – 585 с.
3. Шорт С. Разработка XML Web-сервисов средствами Microsoft .NET / С. Шорт. – СПб. : «БХВ-Петербург», 2003. – 480 с.

1. Синдикація веб-контенту.

Веб-синдикація – форма синдикації при якій зміст веб-сайту надається іншим численним веб-сайтам. Інакше кажучи, *веб-синдикація* означає створення доступних з сайту *веб-потоків* (feed), що надають всім користувачам у формі короткого зведення інформацію про новий зміст, що з'явився на сайті (це можуть бути новини, повідомлення з форуму і ін.).

Веб-потік – формат даних, використовуваний для надання користувачам часто оновлюваного контенту. Розповсюджувачі контенту *об'єднують* (синдикують) веб-потоки, даючи користувачам можливість *підписатися* на них. Інша назва для веб-потіку – *синдикований потік*. Створення набору веб-потоків, які доступні одночасно в

одному місці називається *агрегуванням*. Для цього використовуються спеціальні агрегатори.

Агрегатор потоків (feed aggregator) – клієнтський веб-додаток, що збирає синдигований веб-контент такий як новинні заголовки, *блоги, підкасти* та інші в одному місці для більш зручного перегляду.

Для *приймаючого сайту* веб-синдикація є ефективним способом розміщення більш вичерпної і своєчасної інформації на своїх сторінках.

Для *сайту передавального* синдиговану інформацію вигоди полягає в більшій мірі його представленості серед різних он-лайн платформ. Крім того, породжується додатковий трафік, що, по-суті, є простою і безплатною формою реклами сайту в мережі Інтернет.

Взаємодія веб-потоків і агрегаторів відбувається в наступному порядку:

1. *Провайдер контенту публікує* посилання на потік зі свого сайту.
2. *Користувач* може зареєструвати це посилання за допомогою *програми-агрегатора* на своєму комп'ютері.
3. *Програма-агрегатор* потім опитує всі сервери, що входять до списку зареєстрованих потоків, з метою отримання нового контенту.
4. При наявності нового контенту *програма-агрегатор* або *інформує* користувача про наявність такого або відразу ж *завантажує* його.

2. Агрегування веб-контенту.

Контент веб-потoku звичайно являє собою *веб-сторінки, гіперпосилання або мультимедіа*. Витяг контенту з сайту в формі веб-потoku зазвичай проводиться засобами самого веб-сайту. Однак, не всі веб-сайти можуть мати веб-потік. У цьому випадку можуть бути використані кошти

сторонніх агентів. *Веб-потік* – це веб-документ, зазвичай в XML форматі, що містить тематичні елементи, що містять посилання на більш повну версію матеріалу. Є зручним інструментом для доставки структурованої інформації. Користувачі можуть підписуватися на веб-потоки за допомогою *агрегаторів* або програм для читання потоків, які комбінують зміст декількох веб-потоків для відображення на одній сторінці (або декількох послідовних сторінках).

Деякі з веб-браузерів містять вбудовані можливості для агрегування потоків. Це робиться шляхом простого введення URL веб-потoku або «кліком» на гіперпосиланням в браузері. Формат веб-потоків не призначений для безпосереднього читання користувачем, оскільки дозволяє автоматично переносити контент з сайту на сайт. Для представлення інформації з веб-потoku зазвичай використовуються 2 формати: *RSS* і *Atom*.

Якщо порівнювати веб-потік з більш традиційною поштовою технологією доставки часто оновлюваної інформації, то можна вказати на наступні переваги першого:

1. Оскільки при підписці користувач не вказує свою адресу електронної пошти, ця технологія позбавлена таких потенційних загроз як спам, віруси, фішинг і крадіжка особистої інформації.
2. При відмові від використання веб-потoku немає необхідності відправляти запит на відмову від підписки; користувач просто виключає даний потік зі свого агрегатору.
3. Є широкі можливості для автоматичного сортування повідомлень від веб-потоків аж до використання складних правил і регулярних виразів.
4. Браузери *Internet Explorer 7.0*, *Opera*, *Safari*, *Firefox* і інші можуть працювати з веб-потоками через

інструменти панелі *Закладок, Вибраного* та інших. Є також спеціалізовані програми для читання веб-потоків, наприклад *FeedDemon, Thunderbird, Outlook 2007* та інші.

Агрегатор дозволяє об'єднати інформацію з різних потоків в одному вікні веб-браузера або веб-додатка. Такий додаток називається *RSS-каналом, новинною стрічкою, агрегатором потоків* або *пошуковим агрегатором*. *Підкастинг-агрегатор* можуть автоматично завантажувати медіа-файли. Об'єднаний контент, агрегатор отримує і інтерпретує зазвичай у форматі *RSS* або інших форматах, заснованих на *XML*, наприклад *RDF/XML* або *Atom*. Найбільш розвинені методи агрегування веб-потоків реалізуються на основі технологій *AJAX* і *XML* компонентів – *веб-віджетів (web widgets)*.

Багато мов програмування мають бібліотеки функцій, що дозволяють завантажувати, обробляти, генерувати і виконувати віддалене завантаження каналів. Наприклад в *Perl* є підтримка декількох бібліотек в просторі імен *XML::RSS*

3. RSS – сімейство XML-форматів.

RSS – сімейство *XML*-форматів, призначених для опису новинних стрічок, анонсів статей, змін в блогах і таке інше.

У різних версіях аббревіатура *RSS* мала різні розшифровки:

- Rich Site Summary (*RSS 0.9x*);
- RDF Site Summary (*RSS 0.9* і *1.0*);
- Really Simple Syndication (*RSS 2.x*).

З історії формату.

Першою відкритою офіційною версією *RSS* стала версія *0.90*. Формат був заснований на *RDF* (*Resource Description Framework* – стандарт схеми опису потоків) і

багатом видався надто складним, після чого з'явилася спрощена версія - 0.91.

У 2000 році відбувся поділ формату:

1. Група розробників зі списку розсилки «RSS-DEV» запропонувала формат *RSS 1.0*, який був заснований на стандартах *XML* і *RDF* організації *W3C*. Розширення формату пропонувалося робити через модулі розширень, описувані в своїх просторах імен. Так як проект використовує вже існуючі стандарти, розглядається його використання в рамках технології *Semantic Web*.
2. Специфікація *RSS 0.92* є розвитком версії *0.91* і орієнтується на тих користувачів, яким *RDF-опис* здавався надмірно складним. Подальшим розвитком цієї гілки став формат *RSS 2.0*, який теж підтримує розширення за допомогою модулів, що лежать у своїх просторах імен.
3. У червні 2006 року з'явився конкурент *RSS* – формат *Atom*.

Через існування декількох різних версій формату *RSS-каналів* програми-агрегатори повинні вміти працювати з усіма варіантами, що створює певні труднощі їх розробникам. Проблеми сумісності виникають також при вставці в *RSS-опису* невеликих *HTML-фрагментів*, які в одних випадках оформляються як *CDATA* вузли, а в інших – як *HTML-кодовані PCDATA* вузли. Існують проблеми з різними форматами представлення дат і метаданих.

Лекція 15. Веб-портали. Класифікація веб-порталів.

План:

1. Веб-портали.
2. Класифікація веб-порталів.

Література:

1. Храмцов П. Б. Основы WEB-технологий / П. Б. Храмцов, С. А. Брик, А. М. Русак, А. И. Сурин. – М. : ИТУИТ.РУ, 2003. – 512 с.

1. Веб-портали.

Портал надає єдину точку безпечного доступу, часто у формі *веб-інтерфейсу*, і призначений для *агрегування* і *персоналізації* інформації за допомогою відповідних *портлетів*.

Наприклад, корпоративний портал (корпоративний інформаційний портал) – середовище інтеграції інформації, людей і процесів з різних організаційних підрозділів. Важливою ознакою корпоративних порталів є децентралізований розподіл і управління контентом, що дозволяє інформації постійно оновлюватися.

Портлет – підключаються програмні компоненти користувацького інтерфейсу, якими керують і які відображаються в *веб-порталі*. *Портлети* генерують фрагменти коду розмітки, які впроваджуються на сторінку portalу. Сторінка portalу являє собою набір непересічних вікон портлетів. Наприклад, вікно електронної пошти, вікно погоди, вікно форуму або новин. *Стандартизація* портлетів націлена на надання в розпорядження програміста портлетів, які можуть бути вбудовані в будь-який портал, що підтримує стандарти.

WSRP (Web Services for Remote Portlets) – протокол, який можна розглядати як стандарт для веб-сервісів, що

дозволяє автоматично вбудовувати віддалено запуснені портлети з абсолютно різних джерел.

Специфікація *Java-портлетів* (JSR168, JSR286) дає можливість для *портлетів* з різних веб-порталів взаємодіяти між собою. Ця специфікація визначає безліч API для взаємодії контейнера портлету з портлетом і розглядає такі питання як *персоналізація, подання та безпека*.

Веб-портал – це веб-сайт, що надає користувачеві різні інтерактивні сервіси, що працюють в рамках одного веб-сайту (веб-сторінки). Веб-портали найчастіше виконують роль єдиної точки доступу до інформації в WWW. Портали представляють інформацію з різних джерел однаковою способом. Поряд з стандартною функцією пошуку, веб-портали пропонують і інші сервіси такі як електронна пошта, новини, форуми, голосування та інші.

Як приклад внизу наведено скріншот головної сторінки веб-порталу «MSN в Росії».



Рис. 15.1. Головної сторінки веб-порталу «MSN в Росії»

2. Класифікація веб-порталів.

Первинна класифікація порталів включає два основні класи:

1. Горизонтальні портали.
2. Вертикальні портали.

Крім того, можна розглядати й інші класи порталів:

1. Персональні.
2. Академічні.
3. Урядові.
4. Корпоративні.
5. Тематичні.

Загальнодоступні (або горизонтальні) портали (ще називають мегапорталами), такі як Yahoo!, Lycos, Excite, Rambler і ін. Ці портали призначені для найширшої аудиторії, що відбивається на утриманні наданої ними інформації та послуг – зазвичай вони носять загальний характер (наприклад, новини про політичні події і культурного життя, електронна пошта, новинні розсилки і т. д.). Сфера діяльності таких порталів перетинається з сферою діяльності засобів масової інформації, тому останнім часом спостерігаються процеси злиття загальнодоступних порталів і засобів масової інформації в рамках однієї компанії.

Вертикальні портали призначені для специфічних видів ринку і обслуговують аудиторію, що користується послугами цього ринку або що працює на ньому. Прикладами таких порталів можуть служити додатки B2C (*Business-to-consumer*), наприклад туристичні агентства, що надають послуги з бронювання місць у готелях, замовлення та доставки квитків, доступу до карт і таке інше, або портали типу B2B (*business-to-business*), що дозволяють своїм клієнтам реалізовувати спільні бізнес-операції (наприклад, вибирати постачальників і здійснювати закупівлю товарів, проводити аукціони і таке інше). Число подібних порталів

останнім часом швидко зростає, оскільки все нові ринки товарів і послуг переміщуються в Інтернет.

Корпоративні портали призначені для співробітників, клієнтів і партнерів одного підприємства (іноді вони називаються B2E-портали – *Business-to-employees*). Користувачі такого порталу дістають доступ до призначених їм сервісів і додатків залежно від їх ролі і персонального профілю, і це найцікавіша категорія порталів в плані реалізації корпоративної інфраструктури і інтеграції додатків.

Корпоративний портал призначений для:

- надання зовнішнім і внутрішнім користувачам можливості персоніфікованого доступу до всіх корпоративних даних і додатків (включаючи неструктуровані і різномірні дані),
- об'єднання ізольованих моделей бізнесу, інтеграція різних корпоративних додатків (у тому числі додатків бізнес-партнерів),
- забезпечення повноцінного цілодобового доступу всіх користувачів (включаючи і мобільних) до ресурсів компанії незалежно від їх місця перебування.

Перше покоління корпоративних порталів має наступні характеристики:

- пошук і індексування широкого набору інформаційних репозитаріїв;
- категоризація інформаційного наповнення;
- управління інформаційним наповненням і його агрегація;
- персоналізація;
- високоефективна розробка додатків і можливості інтеграції з іншими додатками.

Для другого покоління корпоративних порталів, які використовуються як складова частина електронного бізнесу, характерним є:

- надійне середовище реалізації додатків;
- потужні й гнучкі інструменти розробки додатків;
- широкі можливості в області інтеграції додатків;
- відповідність вимогам до інформаційних систем масштабу підприємства;
- підтримка інтеграції з іншими додатками і інформаційними системами партнерів;
- підтримка мобільного/бездротового доступу до даних.

У складі типового корпоративного порталу умовно можна виділити три основних функціональних шару:

1. *Шар базової інфраструктури*, що відповідає за базові сервіси, такі як управління транзакціями, система безпеки, управління порталом і ін. Технічно він містить, як правило, сервер додатків, сервер баз даних і веб-сервер, або декілька подібних серверів.
2. *Шар інтеграції додатків*, що відповідає за взаємодію порталу з усіма існуючими в компанії додатками, такими як СУБД, CRM- і ERP-системи, успадковані додатки і ін.
3. *Шар інтерфейсів*, що включає засоби управління інформаційним наповненням (CMS – Content Management System), інтерфейси для обміну даними з інформаційними системами бізнес-партнерів, засоби для роботи з мобільними і бездротовими пристроями та ін. До цього ж шару відносяться візуальні і невізуальні компоненти порталів, які називають зазвичай портлетами, але іноді мають і інші назви (Pagelets, Gadgets, iViews і т. д.).

З числа найбільш поширених засобів створення порталів можна назвати наступні:

- *Microsoft SharePoint Server 2007*.
- *WebSphere Portal Server* – компанії IBM.

- *Oracle 9iAS Portal* – компанії Oracle Corporation.
- *Enterprise Portal* – компанії SAP Portals.
- *iPlanet Portal Server* – компанії Sun Microsystems.
- *Sybase Enterprise Portal* – компанії Sybase.
- *InfoExchange Portal* – компанії BroadVision.

Лекція 16. Введення в технологію AJAX. Розробка мобільних веб-додатків.

План:

1. Введення в технологію AJAX.
2. Розробка мобільних веб-додатків.

Література:

1. Крейн Д., AJAX в действии : технология – Asynchronous JavaScript and XML = AJAX in Action / Д. Крейн, Э. Паскарелло, Д. Джеймс. – М. : «Вильямс», 2006. – 640 с.
2. Салмре И. Программирование мобильных устройств на платформе .NET Compact Framework / И. Салмре. – М. : «Вильямс», 2006. – 736 с.

1. Введення в технологію AJAX.

AJAX (Asynchronous JavaScript and XML) – це концепція використання декількох суміжних технологій, орієнтована на розробку високоінтерактивних додатків, що швидко реагують на дії користувача, що виконують велику частину роботи на стороні клієнта і взаємодіючих з сервером за допомогою *позасмугових звернень*.

Позасмуговими зверненнями називаються запити до сервера, що призводить до оперативного оновлення сторінки замість її заміни. Позасмуговий виклик HTTP – це HTTP запит, який видається за межами вбудованого модуля, що забезпечує відправку форм HTTP. Виклик ініціюється подією, пов'язаною зі сторінкою HTML і обслуговується компонентом-посередником, зазвичай об'єктом *XmlHttpRequest*.

Популярність AJAX пов'язана з появою сервісу *Google Suggest* в 2005 році. Даний сервіс на основі об'єкту

XMLHttpRequest надає в розпорядження користувача досить динамічний веб-інтерфейс. У процесі введення символів користувачем в поле пошукового запиту JavaScript відправляє їх на сервер і отримує від нього список підказок(рис. 16.1).

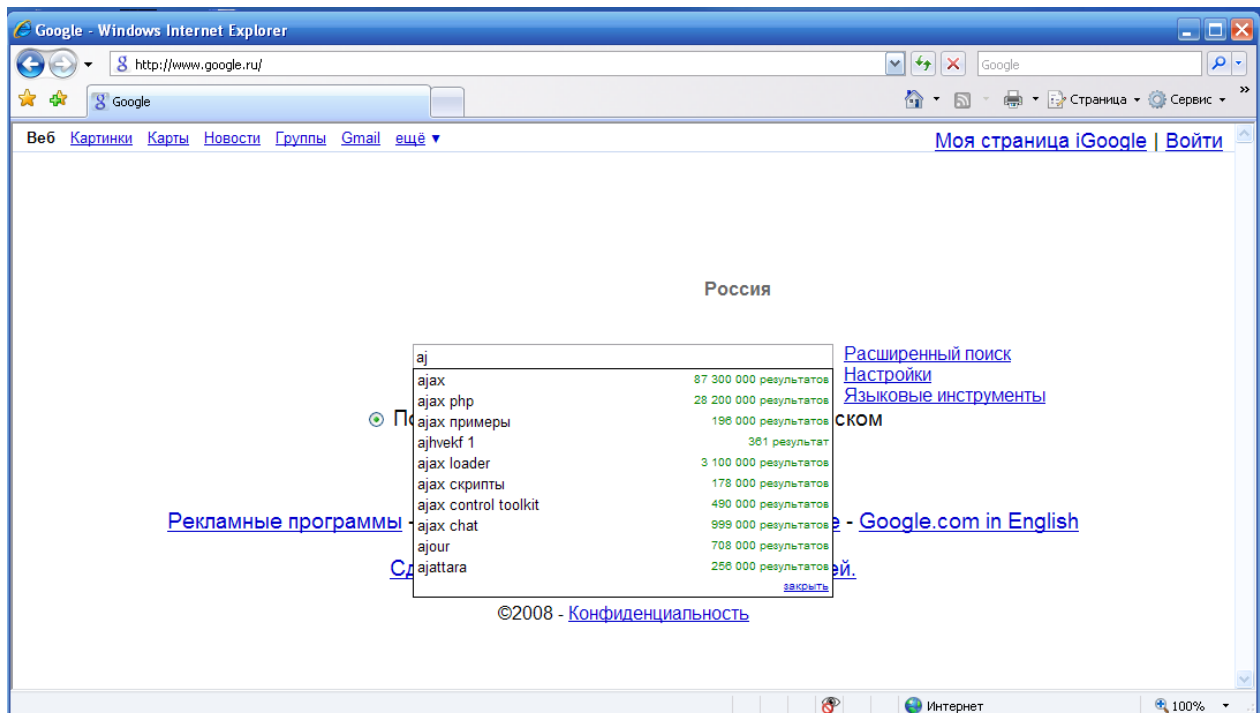


Рис. 16.1. Список підказок в полі пошукового запиту JavaScript.

AJAX застосовується для розробки веб-додатків, до яких висуваються такі вимоги:

1. Додаток повинен передавати користувачам свіжі дані, отримані з серверу.
2. Нові дані повинні інтегруватися в існуючу сторінку без її повного оновлення.

Для роботи з такими додатками в браузері, необхідно, щоб він відповідав вимогам:

1. Підтримка посередників (для позасмугових викликів HTTP). Зазвичай реалізується у формі об'єкту *XmlHttpRequest*.

2. Підтримка оновлюваної моделі DOM.

Об'єкт *XmlHttpRequest* являє собою компактну об'єктну модель для відправки сценарієм звернень HTTP в обхід браузера. Клієнтський код сценарію не може впливати на процес розміщення запиту і результат відправлення запиту. *XmlHttpRequest* дозволяє сценарієм відправляти HTTP запити і обробляти отримані відповіді.

Як формат передачі даних зазвичай використовуються *JSON* або *XML*.

JSON (JavaScript Object Notation) – текстовий формат обміну даними, заснований на *JavaScript* і зазвичай використовуваний саме з цією мовою. Незважаючи на походження від *JavaScript*, формат вважається мовно-незалежним і може використовуватися практично з будь-якою мовою програмування. Для багатьох мов існує готовий код для створення та обробки даних у форматі *JSON*.

JSON будується на двох структурах даних:

1. Набір пар ім'я/значення. У різних мовах це реалізовано як об'єкт, запис, структура, словник, хеш-таблиця, список з ключем або асоціативний масив.
2. Пронумерований набір значень. У багатьох мовах це реалізовано як масив, вектор, список або послідовність.

Наступний приклад показує *JSON*-уявлення об'єкту, що описує викладача.

```
{
  "firstName": "Сергій",
  "lastName": "Петренко",
  "profile": {
    "position": "доцент, к.ф.-м.н.",
    "department": "кафедра інформаційних систем",
    "age": 40
  },
}
```

```
"subjects": [  
    "інтернет-технології",  
    "інформаційні технології"  
]
```

Microsoft .NET надає в розпорядження розробників свою реалізацію AJAX технології – *ASP.NET 2.0 AJAX*.

З архітектурної точки зору, інфраструктура ASP.NET 2.0 *AJAX* складається з двох елементів:

1. **Бібліотека клієнтських сценаріїв** (реалізована на JavaScript). Працює в будь-якому сучасному браузері.
2. **Набір серверних розширень**. Повністю інтегрується з серверними службами і керуючими елементами ASP.NET. Розробники можуть створювати веб-сторінки з розширеною функціональністю, використовуючи практично таку ж методику, яка використовується при розробці серверних сторінок ASP.NET.

2. Розробка мобільних веб-додатків.

Для отримання доступу до ресурсів Інтернет за допомогою тільки мобільного телефону, не вдаючись до допомоги комп'ютеру або модему був розроблений спеціальний стандарт WAP.

WAP (*Wireless Application Protocol*) – протокол доступу до ресурсів Інтернет безпосередньо з мобільного телефону, минаючи комп'ютеру та/або модем.

Для розмітки документів при завантаженні їх в стільникових телефонах і інших мобільних пристроях за стандартом WAP також був розроблена і спеціальна мова – WML (*Wireless Markup Language*).

Спочатку WAP створювався для широкого кола технологій і стандартів бездротового мобільного зв'язку: стільникового, транкового, пейджингового та мікростільникової, а також для підтримки мереж 3G. Даний стандарт інваріантний до операційного ядра, з яким взаємодіє WAP-браузер і розроблявся як відкритий стандарт для бездротової передачі даних, що не залежить від постачальників пристроїв і послуг, оптимізований для мобільних телефонів, що мають дисплей з маленькими розмірами, обмеженою пам'яттю і невисокою продуктивністю.

WAP 2.0 – вдосконалена версія WAP, яка використовує урізаний варіант XHTML і CSS. Це дозволяє працювати з WAP 2.0 сайтами за допомогою звичайного браузера на комп'ютері без установки яких-небудь додаткових *плагінів*.

XHTML MP (XHTML Mobile Profile) – мова розмітки в WAP 2.0, розроблений для мобільних пристроїв.

Архітектура WAP аналогічна WWW. В WAP використовується той же самий спосіб адресації ресурсів і ті ж позначення типів даних. В якості клієнта виступає мобільний пристрій з вбудованим WAP-браузером, запити від якого через WAP-шлюз передаються веб-серверу, і відповідь від останнього через нього ж відправляється клієнтові.

Сервером може виступати самий звичайний веб-сервер. У цьому випадку між WAP-шлюзом і сервером використовується протокол HTTP. З метою зменшення обсягу переданих даних, текстові ресурси, які прийшли від серверу, передаються клієнту в *двійковому вигляді*.

Мова WML нагадує HTML, але орієнтований на пристрої з екраном низького розширення і з невеликим розміром пам'яті. Вся інформація в WML міститься в так званих «*деках*».

Дек – це мінімальний блок даних, який може бути переданий сервером. У деках знаходяться «*карти*» (кожна

карта обмежена тегами `<card>`). На екрані пристрою в кожен момент часу відображається тільки одна *карта*, а користувач може перемикатися між ними переходячи по посиланнях. Розмір *WML-сторінки* повинен знаходитися в межах від 1-4 кілобайт.

Наприклад *WML-сторінка*:

```
<?xml version='1.0'?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
'http://www.wapforum.org/DTD/wml_1.1.xml'>

<wml>
  <head>
    <meta http-equiv="Cache-Control" content="max-age=0" />
  </head>
  <card>
    <p>Hello world!</p>
  </card>
</wml>
```

відобразиться на дисплеї мобільного телефону, де був використаний програмний емулятор телефону (рис. 16.2).



Рис. 16.2. Відображення WML-сторінки на дисплеї мобільного телефону.

Багато мобільних пристроїв можуть показувати документи тільки в *WBXML-форматі*.

WBXML (WAP Binary XML) – формат компактного бінарного уявлення XML. *WBXML* використовується для передачі через бездротові з'єднання з низькою швидкістю. Деякі браузери зі спеціальним плагінами, дозволяють переглядати *WML-сторінки* на звичайному комп'ютері.

В даний час спостерігається тенденція переходу від *WML* до *XHTML*.

Microsoft .NET для створення веб-додатків для мобільних пристроїв надає в розпорядження розробників інструмент *.NET Mobile*.

.NET Mobile являє собою розширення Microsoft ASP.NET і Microsoft .NET Framework. По-суті, *.NET Mobile* являє собою набір серверних керуючих елементів для форм, орієнтованих на використання в бездротових мобільних пристроях. Ці елементи управління генерують різний код для різних пристроїв на мовах *WML*, *HTML* або *Compact HTML* (сHTML).

Керуючі елементи ASP .NET Mobile розширюють функції SDP (Smart Device Programmability) і платформу *.NET Compact Framework*. Вони дозволяють використовувати можливості платформи *.NET Compact Framework* і середовища розробки Visual Studio .NET для створення мобільних веб-додатків завдяки можливості доставки даних на різноманітні мобільні пристрої за допомогою технології ASP.NET. Цей підхід дозволяє створити в середовищі Visual Studio .NET єдиний мобільний веб-додаток, що автоматично виробляє формування даних для відображення на різноманітних пристроях: мобільних телефонах, пейджерів, смартфонах, Pocket PC і ін. При цьому інтегроване середовище розробки дозволяє створювати мобільні веб-додатки, просто перетягуючи керуючі елементи в дизайнер форм.

Система *.NET Mobile* не встановлює жодних компонентів на клієнтський пристрій. Для адаптації форматування під конкретні браузері використовується серверна логіка. Вона генерує дані у форматах WML, HTML, і cHTML. Крім адаптивної генерації веб-сторінок, технологія *.NET Mobile* надає багатий набір засобів індивідуалізації та розширення, забезпечуючи простий спосіб здійснення підтримки нових пристроїв. Крім того, технологія *.NET Mobile* дозволяє розробникам управляти поданням даних для конкретного пристрою або класу пристроїв в одній програмній моделі і забезпечує легку реалізацію підтримки нових пристроїв без переробки існуючих веб-додатків.

Загальна послідовність роботи *.NET Mobile* виглядає наступним чином:

1. Мобільний пристрій. З нього виходить запит на завантаження веб-сторінки.
2. Інтернет. Запит передається через мережу відповідному веб-сервера.
3. Веб-сервер IIS отримує запит від мобільного пристрою і передає його обробнику.
4. *.NET Framework* виконує обробку запиту.
5. *ASP.NET* компілює запитуваний пристроєм документ.
6. *.NET Mobile*. Реалізує елементи веб-сторінки з урахуванням вимог конкретного типу мобільного пристрою.
7. Веб-сторінка повертається назад до клієнтського пристрою.

Лекція 17. Введення в Веб 2.0.

План:

1. Введення в Веб 2.0.
2. Мешапи.

Література:

1. Храмцов П. Б. Основы WEB-технологий / П. Б. Храмцов, С. А. Брик, А. М. Русак, А. И. Сурин. – М. : ИТУИТ.РУ, 2003. – 512 с.

1. Введення в Веб 2.0.

Термін *Веб 2.0* використовується для позначення нових тенденцій у використанні технологій WWW, спрямованих на розширення творчих можливостей користувачів, більш безпечний обмін інформацією та взаємодію між ними.

При цьому більший акцент робиться на формування веб-спільнот та соціально-орієнтованих сайтів таких як, наприклад, *блоги і відеоблоги, фолксономії, вікіпедії* та ін.

Термін набув поширення після конференції з Веб 2.0 Медіа в 2004 році з подачі Тіма О'Рейлі для вираження нового способу взаємодії розробників ПЗ і кінцевих користувачів через Веб. Ключовий принцип ідеології Веб 2.0 був сформульований як: «Інтернет – як платформа».

Правомірність терміну оскаржують багато експертів, у тому числі Тім Бернерс-Лі, оскільки багато компонентів технологій Веб 2.0 існували ще з перших років існування мережі Веб.

Веб 2.0 можна розглядати і як підхід до побудови систем, при якому вони стають *тим краще, чим більше людей* ними користуються в процесі *мережевих взаємодій*. Фактично Веб 2.0 означає перехід веб-сайтів від ізолюваних накопичувачів інформації до взаємопов'язаним

програмним платформам, сприймаються користувачами так як ніби вони виконуються локально на його комп'ютері.

Сайти Веб 2.0 надають для користувачів можливість:

1. Не просто отримувати інформацію, але й виконувати програми виключно через браузер.
2. Розміщувати і керувати своїми даними.

Ще одна важлива концепція Веб 2.0 – «Архітектура участі», яка заохочує користувачів підвищувати цінність ПЗ, шляхом його використання.

В якості ключових для Веб 2.0 розглядаються наступні технології:

1. Веб-сервіси – це програми, доступ до яких здійснюється через протокол *HTTP*, а обмін даними відбувається у форматі *XML* (або похідному від нього). Веб-сервіс реалізується на серверах компанії, що її створила. У будь-який момент користувачеві доступні найсвіжіші дані; Веб-сервіси є платформо-незалежними, оскільки інструменти для роботи з *HTTP* і *XML* є в будь-якій сучасній мові програмування.
2. AJAX. Використання Ajax стало найбільш популярним після того як *Google* почав активно використовувати його при створенні своїх сайтів, таких як *Gmail* і *Google Maps*.
3. Веб-синдикація (на основі технологій *RSS* або *Atom*) – одночасне поширення інформації в тому числі аудіо- та відео- на різні сторінки або web-сайти.
4. Веб *mash-up* – сервіс, який повністю або частково використовує як джерело інформації інші сервіси, надаючи користувачеві нову функціональність для роботи. Веб *mash-up* сервіс може ставати також новим джерелом інформації для інших Веб *mash-up* сервісів. Таким чином, утворюється мережа

залежних один від одного сервісів, інтегрованих один з одним.

5. *Теги* – ключові слова, що описують даний об'єкт, або відносять його до якої-небудь категорії. Теги можна розглядати як мітки, які присвоюються об'єкту, для визначення його місця серед інших об'єктів.
6. *Фолксономія* – популярна класифікація, практика спільної категоризації інформації (посилань, фото, відео кліпів та ін.). За допомогою довільно обраних міток (тегів). Приклади використання *фолксономії*: Flickr, del.icio.us.
7. *Соціальне ПЗ* – широкий діапазон ПЗ, призначеного для обміну і спільного доступу до інформації користувачів мережі Інтернет.

До недоліків Веб 2.0 можна віднести наступні:

1. Залежність від наявності постійного з'єднання.
2. Залежність працездатності сайтів від рішень сторонніх компаній.
3. Залежність якості роботи сервісу від якості роботи багатьох інших компаній.
4. Недостатні можливості існуючої веб-інфраструктури для виконання складних обчислювальних задач в браузері.
5. Уразливість конфіденційних даних, що зберігаються на сторонніх серверах, при несанкціонованому доступі.

2. Мешапи.

Мешап (*Mashup*) – гібридний веб-додаток, що об'єднує дані з декількох джерел в рамках єдиного інтегрованого інструменту. Контент *мешап* зазвичай отримує ззовні за допомогою відкритих інтерфейсів, веб-сервісів, веб-джерел

(наприклад *RSS* або *Atom*) або аналізу документів, що генеруються іншими програмами.

Найбільш часто *мешапи* використовують програмні інтерфейси, які подаються *Amazon, eBay, Flickr, Google, Microsoft, Yahoo* і *YouTube*.

Архітектура *мешапу* включає в себе 3 частини:

1. Провайдер *контенту* – джерело даних.
2. Власне *мешап* – веб-додаток, що пропонує нову функціональність з використанням різних джерел, що не належать йому.
3. *Клієнт* – зазвичай веб-браузер, що відображає веб-сторінку *мешапа*.

Різні типи *мешапів* можуть генерувати *RSS*, веб-сервіси, миттєві і поштові повідомлення.

Слід відрізнити *мешапи* від простого *впровадження даних* з інших сайтів з утворенням складних документів. *Мешап* самостійно отримує зовнішні дані через програмні інтерфейси і, обробляючи їх певним чином, надає їм додаткову цінність.

Як приклад можна привести використання картографічних даних *Google Maps* для додавання до них даних про нерухомість в *Cragislist* (в результаті створюється новий унікальний веб-сервіс, спочатку не пропонований жодним з джерел).

Серед *мешапів* можна виділити наступні групи:

1. *Споживчі мешапи* – комбінуючи дані різних типів з кількох джерел, надають доступ до них за допомогою єдиного графічного інтерфейсу. Найбільш відомий приклад – численні програми *Google Maps*.
2. *Мешапи даних* – змішують дані, що близькі за типом з декількох джерел, надає доступ до них за допомогою єдиного графічного інтерфейсу. Приклад – *Yahoo!Pipes* дозволяє користувачам отримувати потоки інформації з різних джерел і

створювати правила з управління отриманим контентом (наприклад, використовуючи фільтри).

3. *Бізнес-мешапи* – акцент роблять на різних способах агрегування та подання даних, надаючи нові можливості для спільної роботи представників бізнесу та розробників.

Хоча *мешапи* також як і *портали* є технологіями агрегування контенту, є принципова відмінність між ними. *Портали* – більш рання технологія, яка є, по-суті, розширенням традиційних динамічних веб-додатків, в яких процес перетворення даних в гіпертекстові документи розбивається на два етапи: генерація окремих частин розмітки і об'єднання їх на одній сторінці. За генерацію кожної з частин відповідає відповідні портальні додатки, виконувані на портальному сервері або іншому сервері. *Портальна* технологія є серверної технологією, що відповідає за *агрегування* даних тільки на рівні уявлення.

Компанія Майкрософт надає користувачам спеціальний сайт *Microsoft Popfly*, який дозволяє користувачам створювати веб-сторінки, фрагменти програмного коду (для повторного використання) і мешапи за допомогою пакета розробки додатків з підтримкою *Microsoft Silverlight*.

На сайті є чотири інструменти, засновані на *Microsoft Silverlight*:

1. *Розробник ігор*.
2. *Розробник мешапів*.
3. *Розробник веб-сторінок*.
4. *Popfly Space* – простір для розміщення готових мешапів і веб-сторінок, доступних користувачам.

Лекція 18. Додатки для соціальних мереж.

План:

1. «Соціальний Веб».
2. Фолксономія.
3. Семантична веб-мережа.
4. Онтологія.
5. Семантичні веб-сервіси.

Література:

1. Храмцов П. Б. Основы WEB-технологий / П. Б. Храмцов, С. А. Брик, А. М. Русак, А. И. Сурин. – М. : ИТУИТ.РУ, 2003. – 512 с.

1. «Соціальний Веб».

Поняття «Соціальний Веб» (*Social Web*) використовується для опису того, як відбувається соціалізація користувачів і їх взаємодію один з одним за допомогою мережі WWW. Основою для об'єднання користувачів служать найрізноманітніші спільні інтереси.

Термін «Соціальний Веб» може бути використаний для вираження двох понять:

1. Перше поняття пов'язане з описом технологій Веб 2.0, які фокусуються в першу чергу на соціальних взаємодіях і спільнотах.
2. Друге поняття використовується для опису мережі майбутнього, аналогічної WWW.

Можна розглядати як *спільноту* безлічі людей, пов'язаних і взаємодіючих між собою за допомогою контенту в режимі діалогу та спільних дій через Інтернет.

Будучи націленими на стимулювання взаємодії між людьми, соціальне ПЗ для Веб оперує з наступними соціальними атрибутами:

- ідентичність;

- репутація;
- присутність;
- зв'язку;
- групова належність;
- спілкування;
- розділяємий контент.

Соціально програмне забезпечення – це широкий діапазон програмних систем, що дозволяє користувачам взаємодіяти і обмінюватися даними. Цей спосіб комп'ютерно-опосередкованої взаємодії став популярним з появою таких соціальних сайтів як MySpace, Facebook, Однокласники, медіа сайтів Flickr і YouTube, комерційних сайтів eBay.

Багато з цих програм мають такі загальні характеристики як: відкриті API, сервіс-орієнтований дизайн, можливість віддаленого розміщення даних та медіа-файлів. Такі додатки прийнято відносити до систем Веб 2.0.

Усередині соціального ПЗ можна виділити дві групи програмних інструментів: *комунікаційні* та *інтерактивні*.

Комунікаційні інструменти застосовуються для запису, зберігання та подання комунікаційних даних, найчастіше в текстовому вигляді, але все більше в аудіо та відео форматах.

Інтерактивні інструменти застосовуються для підтримки опосередкованого даними (різних медіаформатів) взаємодії між окремими користувачами і їх групами. На відміну від комунікаційних інструментів акцент робиться на підтримці зв'язності користувачів і механізмів спілкування між ними.

На противагу *комунікаційним* інструментам, які є зазвичай асинхронними, *інтерактивні* інструменти переважно синхронні, дозволяючи взаємодіяти користувачам в режимі реального часу (як у випадку

інтернет-телефонії, відеочатів і т. п.) або майже синхронно (служби миттєвих повідомлень, текстові чати і т. п.).

Можна назвати наступні приклади програмних систем, які відносяться до соціального ПЗ:

- Системи обміну миттєвими повідомленнями (*IM – Instant messaging*) дозволяють спілкуватися з іншим користувачем через мережу в режимі реального часу (у відносно захищеному режимі). До найбільш популярних з них можна віднести Skype, ICQ, Yahoo! Messenger, MSN Messenger, AOL Instant Messenger, Miranda IM. До систем, орієнтованих на бізнес, можна віднести IBM Lotus Sametime, Microsoft Messenger і Jabber.
- Інтернет-чати (*IRC – Internet Relay Chat*) дозволяють одночасно декільком користувачам спілкуватися в режимі реального часу.
- Інтернет-форуми прийшли на зміни електронним конференціям (виникли до появи WWW). Користувач форуму може створювати нову «тему», доступну для інших. Інші користувачі можуть переглядати тему і залишати свої коментарі в режимі послідовного запису.
- Веб-блоги (*web logs*), або коротко блоги, можна розглядати як особисті он-лайн журнали окремих користувачів. Власник блогу може розміщувати повідомлення в своєму журналі, в той час як інші користувачі (читачі) можуть залишати до них свої коментарі.
- Вікі-довідники (*wiki*), або просто вікі, – по-суті, веб-сайти, зміст яких може редагуватися відвідувачами сайту. Найбільш відомий приклад – Вікіпедія.
- Сервісні соціальні мережі дозволяють користувачам об'єднуватися в он-лайн режимі навколо спільних для них інтересів, захоплень або

з різних приводів. Наприклад, деякі сайти надають сервіси, за допомогою яких користувачі можуть розміщувати для загального доступу персональну інформацію, необхідну для пошуку партнерів. Приклади: LinkedIn, Вконтакті.

- Соціальні мережі прийняття рішень – веб-мережі для обговорення з метою прийняття рішень. Використовуються для встановлення постійного зв'язку окремих людей з урядом.
- Комерційні соціальні мережі орієнтовані на підтримку бізнес-транзакцій і формування довіри людей до брендів на основі врахування їх думок про продукт, про те як зробити його краще і т. п., тим самим дозволяючи споживачам брати участь у просуванні продукту і розширюючи їх обізнаність.
- Соціальні закладки (*social bookmarking*). Деякі веб-сайти дозволяють користувачам надавати в розпорядження інших список закладок або популярних веб-сайтів. Такі сайти також можуть використовуватися для пошуку користувачів зі спільними інтересами. Приклад: del.icio.us.
- Соціальні каталоги (*social cataloging*) нагадують соціальні закладки, але орієнтовані на використання в академічній сфері, дозволяючи користувачам працювати з базами даних цитат з наукових статей. Приклади: *Academic Search Premier, LexisNexis Academic University, CiteULike, Connotea*.
- Соціальні бібліотеки являють собою додатки, що дозволяють відвідувачам залишати посилання на їхній колекції, книги, аудіозаписи тощо, доступні іншим. Передбачена підтримка системи рекомендацій, рейтингів і т. п. Приклади: discogs.com, imdb.com.

- Багатокористувацькі мережеві ігри (*Massively Multiplayer Online Games*) імітують віртуальні світи з різними системами підрахунку очок, рівнів, змагальності, переможців і переможених. Приклад: World of Warcraft, DOTA2.

2. Фолксономія.

Фолксономія (*folksonomy*) – практика і методика спільної категоризації контенту (посилань, фото, відео кліпів тощо) за допомогою довільно обраних міток (тегів). Вона заснована на спонтанній співпраці групи людей з метою організації контенту і повністю відрізняється від традиційних формальних методів класифікації на основі *індексних термінів*. Як правило, цей феномен виникає тільки в неієрархічних спільнотах, наприклад на загальнодоступних веб-сайтах. Так як учасники *фолксономії* контенту зазвичай є й основними ж її споживачами, використання методики фолксономії призводить до результатів, більш точно відображаючи спільну концептуальну модель контенту всієї групи.

Основними проблемами фолксономії, що приводять до ненадійності і неузгодженості результатів, є:

1. Наявність форм множини.
2. Полісемія.
3. Синонімія.
4. Глибина (специфічність) використання міток.

Фолксономію можна розглядати в якості одного з ключових елементів у розвитку *Семантичної веб-мережі*, в рамках якої всі веб-сторінки містять машинно-орієнтовані метадані, що описують зміст сторінок. Ці метадані повинні значно покращувати точність роботи пошукових систем. Однак, щоб уникнути труднощів, викликаних ненадійністю і неузгодженістю в роботі великих спільнот авторів сторінок, їм рекомендується використовувати

стандарти метаданих, наприклад *Дублінське ядро (Dublin Core)*.

3. Семантична веб-мережа.

Семантична веб-мережа (*Semantic Web*) – частина глобальної концепції розвитку мережі Інтернет, метою якої є реалізація можливості машинної обробки інформації, доступної в мережі WWW. Основний акцент у цій концепції робиться на роботі з *метаданими*, однозначно характеризують властивості і зміст ресурсів WWW, замість використовуваного в даний час *текстового аналізу* документів.

Термін був введений Тімом Бернерсом-Лі в травні 2001 року.

У семантичній веб-мережі передбачається використання

- універсальних ідентифікаторів ресурсів (URI);
- *онтологій* і мов опису метаданих.

Концепція семантичної веб-мережі була прийнята і просувається W3C. Для її впровадження передбачається створення мережі документів, що містять метадані про ресурси WWW, та існуючої паралельно з ними. Тоді як самі ресурси призначені для сприйняття людиною, метадані використовуються машинами (пошуковими роботами та іншими інтелектуальними агентами) для отримання однозначної інформації про властивості цих ресурсів за допомогою механізмів логічного висновку.

Технічну частину семантичної павутини становить сімейство стандартів на мові опису, що включає *XML, XML Schema, RDF, RDF Schema, OWL* та ін. Необхідність опису метаданих так чи інакше призводить до дублювання інформації. Кожен документ повинен бути створений у двох примірниках: розмічених для читання людьми, а також в машинно-орієнтованому форматі.

4. Онтологія.

Онтологія – це спроба всеосяжної і детальної формалізації деякої області знань за допомогою концептуальної схеми. Зазвичай така схема складається з ієрархічної структури даних, що містить всі релевантні класи об'єктів, їхні зв'язки і правила (теореми, обмеження), прийняті в цій галузі.

Сучасні онтології зазвичай складаються з *екземплярів, понять, атрибутів і відносин*.

Для опису онтологій Веб була розроблена спеціальна мова – OWL (Web Ontology Language), побудований на основі XML. Мова OWL може бути використана для опису класів і відносин між ними. В основі мови – представлення дійсності в моделі даних «об'єкт – властивість». Мова застосована не тільки для опису веб-сторінок, але і будь-яких об'єктів дійсності і розглядається як одна з фундаментальних технологій, необхідних для побудови Семантичної веб-мережі.

5. Семантичні веб-сервіси.

У той час як сукупність ресурсів та їх метаданих можна вважати статичною частиною *семантичної павутини*, її динамічну частину представляють семантичні *веб-сервіси* – закінчені елементи програмної логіки з однозначно описаною семантикою, доступні через Веб і придатні для пошуку, композиції і виконання.

Технічно, *семантичний веб-сервіс* відрізняється від звичайного веб-сервісу наявністю не тільки опису інтерфейсу (зазвичай на мові WSDL) в термінах типів даних, переданих сервісу, повертаємих значень і генерованих помилок, але і наявністю семантичного опису всіх його характеристик.

Потенційна вигода від використання семантичних веб-сервісів полягає в можливості автоматичного пошуку (а також композиції) програмними агентами відповідних сервісів для вирішення поставлених завдань.

Проте, складність цього завдання в її загальному формулюванні поки дозволяє домагатися деяких позитивних результатів тільки у вузькоспеціалізованих галузях, явним чином виграють від впровадження сервісно-орієнтованої архітектури, наприклад в інтеграції корпоративних додатків.

Лабораторна робота №1.

Тема «Службові утиліти для роботи в Інтернет. Вивчення протоколу НТТР».

Мета роботи:

- 1) Вивчення структури IP-адреси.
- 2) Ознайомлення з найбільш популярними утилітами для діагностики мережевої конфігурації і мережних з'єднань.
- 3) Ознайомлення з основами протоколу НТТР.

Теоретичний матеріал

1. Структура IP адреси.

IP-адреса складається з двох частин: номера мережі і номера вузла в мережі.

Найпоширенішим є запис IP-адреси у вигляді чотирьох чисел, розділених крапками, кожне з яких представляє значення байту в десятковій формі, наприклад: 213.180.204.11. Запис адреси не передбачає спеціального розмежувального знаку між номером мережі і номером вузла.

Для поділу цих частин зазвичай використовується 2 підходи:

1. За допомогою маски (RFC 950, RFC 1518), що представляє собою число в парі з IP-адресою. За допомогою операції «логічне І» над цими двома числами виділяється номер мережі.
2. За допомогою класів адрес (RFC 791).

Вводиться п'ять класів адрес: A, B, C, D, E.

A, B, C – використовуються для адресації мереж, D і E – мають спеціальне призначення. Ознакою, на підставі

якого IP-адреса відносять до того чи іншого класу, є значення декількох перших бітів адреси.

Таблиця 1.1

Розподіл адрес в IP мережах.

Класс	Первые биты	Наименьший номер сети	Наибольший номер сети	Максимальное число узлов в сети
A	0	1.0.0.0 (0 - не используется)	126.0.0.0 (127 - зарезервирован)	2^{24} (3 байта)
B	10	128.0.0.0	191.255.0.0	2^{16} (2 байта)
C	110	192.0.0.0	223.255.255.0	2^8 (1 байт)
D	1110	224.0.0.0	239.255.255.255	групповые адреса
E	11110	240.0.0.0	247.255.255.255	зарезервировано

В рамках IP протоколу існують обмеження при призначенні IP-адрес, а саме:

- номери мереж і номери вузлів не можуть складатися з довічних нулів або одиниць;
- якщо IP-адреса складається тільки з двійкових нулів, то вона називається невизначеною адресою і позначає адресу того вузла, який згенерував цей пакет;
- якщо в полі номера мережі стоять тільки нулі, то за замовчуванням вважається, що вузол призначення належить тій же самій мережі, що і вузол, який відправив пакет; така адреса може бути використана тільки як адреса відправника;
- якщо всі двійкові розряди IP-адреси дорівнюють 1, то пакет з такою адресою призначення повинен розсилатися всім вузлам, що знаходяться в тій же мережі, що й джерело цього пакету; таку адресу називається обмеженим широкомовним, оскільки пакет не зможе вийти за межі мережі;
- якщо в поле адреси призначення в розрядах, які відповідають номеру вузла, стоять тільки одиниці, то пакет розсилається всім вузлам мережі, номер

- якої зазначений в адресі призначення; такий тип адреси називається *широкомовним*;
- якщо перший октет адреси дорівнює 127, то така адреса називається внутрішньою адресою стека протоколів; вона використовується для тестування програм, організації клієнтської і серверної частин додатків, встановлених на одному комп'ютері;
 - групові адреси, що відносяться до класу D, призначені для економічного розповсюдження в Інтернеті, великій корпоративній мережі аудіо-або відеопрограм.

Стандартним класам мереж можна поставити у відповідність наступні значення маски:

- клас А – 255.0.0.0;
- клас В – 255.255.0.0;
- клас С – 255.255.255.0

Розглянемо наступний приклад:

Вихідні дані	<i>IP адреса</i>	62.76.167.21
	<i>Маска мережі</i>	255.255.255.0
Логічна операція	I	
Результат	<i>Адрес мережі</i>	62.76.167.0
	<i>Номер комп'ютера</i>	21

Для визначення мережевих налаштувань комп'ютеру і мережевого устаткування, діагностики та отримання іншої інформації, що відноситься до інтернет-протоколів, широко використовуються спеціальні утиліти.

2. Утиліта *ipconfig*.

Ipconfig – це утиліта командного рядка для виводу деталей поточного з'єднання комп'ютеру з мережею і контролю над клієнтським сервісом DHCP. DHCP (Dynamic

Host Configuration Protocol) – це мережевий протокол, що дозволяє комп'ютерам автоматично отримувати IP-адресу та інші параметри, необхідні для роботи в мережі TCP / IP.

Синтаксис команди:

ipconfig/ключі

Команда *ipconfig/all* – відображає повну інформацію по всіх мережевих адаптерів.

Приклад виводу для Windows:

```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.

Z:\>ipconfig/all

Настройка протокола IP для Windows

Имя компьютера . . . . . : C1R385N16
Основной DNS-суффикс . . . . . : cs.vsu.ru
Тип узла. . . . . : неизвестный
IP-маршрутизация включена . . . . . : нет
WINS-прокси включен . . . . . : нет
Порядок просмотра суффиксов DNS . . . . . : cs.vsu.ru
                                              cs.vsu.ru
                                              vsu.ru

Подключение по локальной сети 3 - Ethernet адаптер:

VMnet8
DNS-суффикс этого подключения . . . :
Описание . . . . . : VMware Virtual Ethernet Adapter for
Физический адрес. . . . . : 00-50-56-C0-00-08
DHCP включен. . . . . : нет
IP-адрес . . . . . : 192.168.111.1
Маска подсети . . . . . : 255.255.255.0
Основной шлюз . . . . . :

Подключение по локальной сети 2 - Ethernet адаптер:

VMnet1
DNS-суффикс этого подключения . . . :
Описание . . . . . : VMware Virtual Ethernet Adapter for
Физический адрес. . . . . : 00-50-56-C0-00-01
DHCP включен. . . . . : нет
IP-адрес . . . . . : 192.168.61.1
Маска подсети . . . . . : 255.255.255.0
Основной шлюз . . . . . :

Подключение по локальной сети - Ethernet адаптер:

on #2
DNS-суффикс этого подключения . . . : cs.vsu.ru
Описание . . . . . : Intel(R) PRO/100 VE Network Connecti
Физический адрес. . . . . : 00-14-85-17-CD-6C
DHCP включен. . . . . : да
Автонастройка включена . . . . . : да
IP-адрес . . . . . : 10.16.1.146
Маска подсети . . . . . : 255.255.255.0
Основной шлюз . . . . . : 10.16.1.2
DHCP-сервер . . . . . : 62.76.220.205
DNS-серверы . . . . . : 62.76.220.205
                        62.76.220.204
Аренда получена . . . . . : 12 ноября 2008 г. 8:08:31
Аренда истекает . . . . . : 12 ноября 2008 г. 23:08:31

Z:\>_
  
```

3. Утиліта *ping*.

Ping (Packet InterNet Grouper) – це системна програма, призначена для перевірки з'єднань в мережах на основі TCP/IP. Вона відправляє *Echo-Request* запити протоколу ICMP зазначеному вузлу мережі й фіксує відповіді які надходять (ICMP Echo-Reply). Час між відправленням запиту й одержанням відповіді (RTT, Round Trip Time) дозволяє визначати двосторонні затримки (RTT) за маршрутом і частоту втрати пакетів. Що дозволяє побічно визначати завантаженість каналів передачі даних і проміжних пристроїв. Повна відсутність ICMP-відповідей може також означати, що віддалений вузол (або який-небудь з проміжних маршрутизаторів) блокує ICMP Echo-Reply або ігнорує ICMP Echo-Request.

Синтаксис:

ping - параметри кінцеве_ім'я

Кінцеве ім'я – це доменне ім'я або IP-адреса хосту.

Приклад:

```
C:\>ping www.mail.ru

Обмен пакетами с www.mail.ru [194.67.57.26] по 32 байт:

Ответ от 194.67.57.26: число байт=32 время=11мс TTL=118
Ответ от 194.67.57.26: число байт=32 время=11мс TTL=118
Ответ от 194.67.57.26: число байт=32 время=11мс TTL=118
Ответ от 194.67.57.26: число байт=32 время=11мс TTL=118

Статистика Ping для 194.67.57.26:
    Пакетов: отправлено = 4, получено = 4, потеряно = 0 (0% потерь),
Приблизительное время приема-передачи в мс:
    Минимальное = 11мсек, Максимальное = 11 мсек, Среднее = 11 мсек
```

4. Утиліта *tracert*.

Traceroute (скорочено *tracert*) – це службова програма, призначена для визначення маршрутів прямування пакетів

в мережах *TCP/IP*. Робота *tracert* заснована на протоколі *ICMP*.

Traceroute виконує відправку пакетів вказаною вузла мережі, відображаючи при цьому відомості про всі проміжні маршрутизатори, через які пройшли пакети на шляху до цільового вузла. У разі проблем при доставці пакетів до якого-небудь вузла програма *tracert* дозволяє визначити, на якій саме ділянці мережі виникли неполадки.

Синтаксис:

tracert - параметри кінцеве_ім'я

Кінцеве ім'я – це доменне ім'я або IP-адреса хосту.

Приклад:

```
Z:\>tracert www.mail.ru

Трассировка маршрута к www.mail.ru [194.67.57.26]
с максимальным числом прыжков 30:

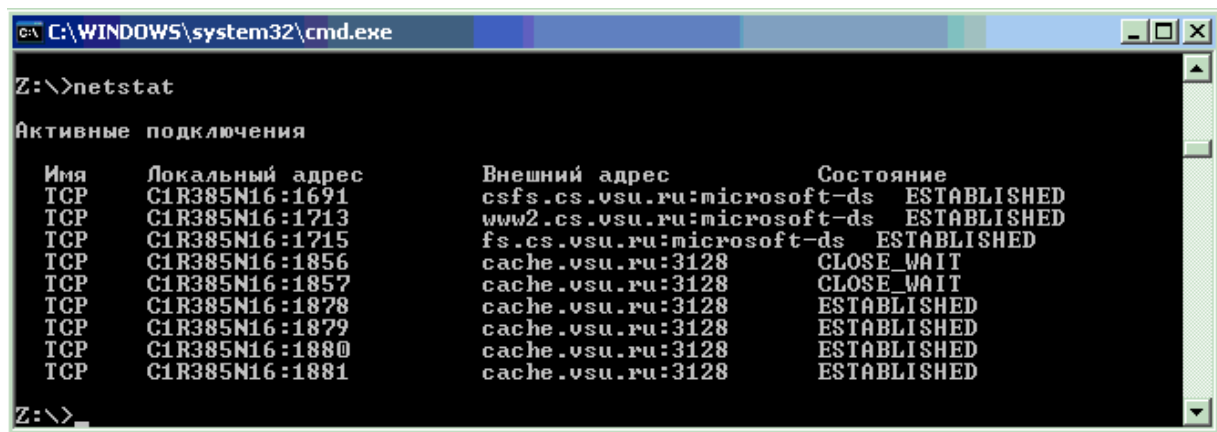
  1  <1 ms    <1 ms    <1 ms    c1swr1.cs.vsu.ru [10.16.1.2]
  2  <1 ms    <1 ms    <1 ms    c2swr1-e1000-4-1-ve100.vsu.ru [62.76.168.42]
  3  <1 ms    <1 ms    <1 ms    c2r1-ve5.vsu.ru [62.76.168.50]
  4  27 ms    22 ms     27 ms    RBNNet-VSU.vsu.ru [62.76.168.1]
  5  28 ms    *          30 ms    cisco13.Moscow.gldn.net [193.232.244.43]
  6  *        *          *        Превышен интервал ожидания для запроса.
  7  36 ms    45 ms     29 ms    cat07.Moscow.gldn.net [194.186.157.82]
  8  24 ms    46 ms     29 ms    cat01.Moscow.gldn.net [194.186.158.110]
  9  44 ms    27 ms     23 ms    mailru-KK12-1-gw.Moscow.gldn.net [195.239.8.10]

 10  36 ms    47 ms     49 ms    mail.ru [194.67.57.26]

Трассировка завершена.
Z:\>_
```

5. Утиліта *netstat*.

Netstat – службова програма, що відображає статистику протоколу і поточних мережевих підключень *TCP/IP*:



6. Утиліта *telnet*.

Telnet – мережевий протокол для реалізації текстового інтерфейсу по мережі. Назва «*telnet*» має також утиліта, що реалізує клієнтську частину протоколу. Історично *telnet* служив для віддаленого доступу до інтерфейсу командного рядка операційних систем. Протокол *telnet* може використовуватися для виконання налагодження інших протоколів на основі транспорту TCP.

Утиліта *telnet* підтримує наступні команди:

- *Close* – закриття поточного з'єднання.
- *Display* – відображення параметрів операції.
- *Open* – підключення до сайту.
- *Quit* – вихід з *telnet*.
- *Set* – встановлення параметрів.
- *Send* – відправлення рядка на сервер.
- *Status* – виведення відомостей про поточний стан.
- *Unset* – скидання параметрів.

Використовуючи утиліту *telnet* можна, наприклад, вручну відправити запит клієнта і отримати відповідь сервера за протоколом HTTP.

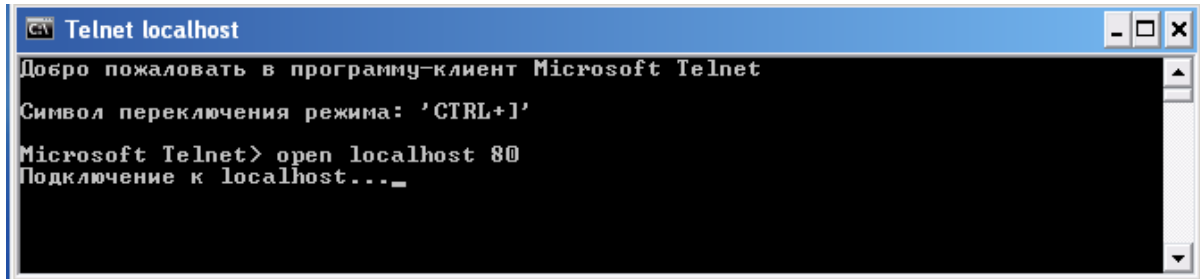
Для цього виконаємо наступну послідовність дій:

1. Запуск утиліти *telnet*.
2. Встановлення з'єднання з веб-сервером за допомогою команди: *open ім'я_хоста 80*.
3. Формування запиту клієнта.

4. Отримання відповіді сервера.

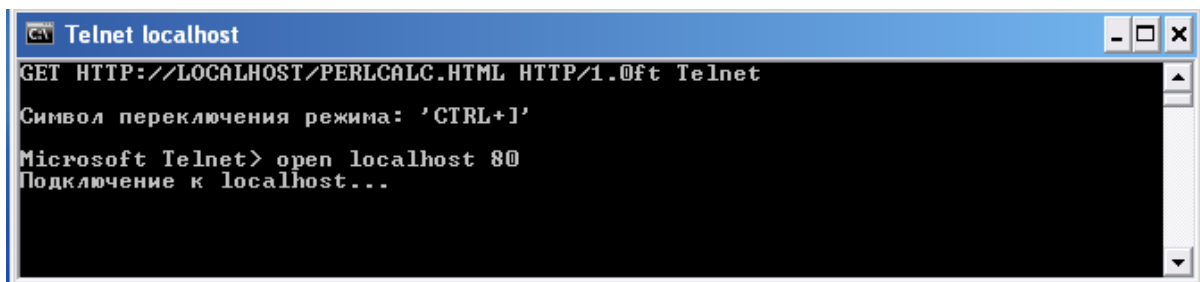
Приклад:

1. Встановлюємо з'єднання: *open localhost 80*

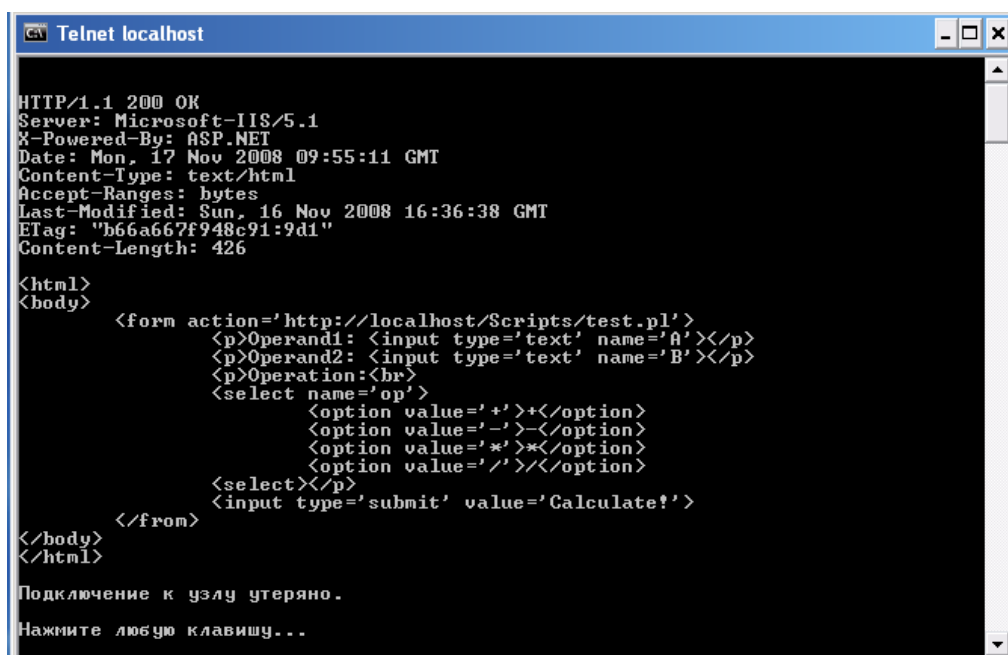


2. Формуємо рядок стану запиту клієнта:

**GET HTTP: //LOCALHOST/PERLCALC.HTML
HTTP/1.0 <ENTER> <ENTER>**



3. Отримуємо відповідь сервера:



Видно, що відповідь веб-сервера *localhost* містить рядок стану (з кодом успішного завершення 200), поля заголовка (*Server, Date, Content-type* і ін.) і тіло, що містить HTML код запитаного клієнтом документа *http://localhost/perlcalc.html*.

Порядок виконання роботи

Завдання 1.

Переглянути параметри:

1. За допомогою утиліти *ipconfig* (запускається в командному рядку командою *ipconfig*) визначте IP-адресу і маску підмережі для свого комп'ютеру.
2. Визначте клас підмережі, в якій знаходиться ваш комп'ютер без використання маски підмережі і по масці підмережі.
3. Визначте адресу підмережі, в якій знаходиться ваш комп'ютер, з використанням функції «Логічне І» над IP-адресою і маскою підмережі. Слід мати на увазі, що операція «Логічне І» повинна проводитися з двійковим поданням операндів.

Завдання 2.

За допомогою утиліти *ping* (запускається в командному рядку командою *ping*) перевірте доступність хостів, мінімальний, середній і максимальний час прийому-передачі ICMP пакетів до них. Можна розглянути хости, наприклад в наступній послідовності:

1. Сервер вашого безпосереднього провайдера або сервера вашої підмережі.
2. Який-небудь сервер вашого регіону.

3. Веб-сервер Інтернет-Університету Інформаційних Технологій: www.intuit.ru.
4. Веб-сервер Університету в Кембриджі: www.cam.ac.uk.
5. Веб-сервер Університету в Каліфорнії: www.ucla.edu.
6. Веб-сервер Університету в Токіо: www.u-tokio.ac.jp.
7. Веб-сервер компанії Microsoft: www.microsoft.com.

Зверніть увагу, що в останньому випадку ICMP-пакети блокуються веб-сервером.

Завдання 3.

За допомогою утиліти *tracert* (запускається в командному рядку командою *tracert*) визначте маршрути проходження і час проходження пакетів до хостів, наведених у завданні 2.

Завдання 4.

1. За допомогою утиліти *netstat* (запускається в командному рядку командою *netstat*) подивіться активні поточні з'єднання і їх стан на вашому комп'ютері.
2. Запустіть кілька екземплярів веб-браузера, завантаживши в них веб-сторінки з різних веб-серверів. Подивіться з допомогою *netstat*, які нові мережеві підключення з'явилися в списку.
3. Закривайте браузери і за допомогою *netstat* перевіряйте зміну списку мережевих підключень.

Завдання 5.

Ознайомлення з протоколом HTTP за допомогою утиліти *telnet*.

1. Запустіть сеанс *telnet* (запускається в командному рядку командою *telnet*). При цьому з'явиться підказка *Microsoft Telnet>*. З повним списком команд можна ознайомитися за допомогою команди *help*.
2. Дозвольте режим відображення введених з клавіатури символів за допомогою команди *set localecho*.
3. Відповідно до протоколу HTTP необхідно встановити з'єднання з веб-сервером. Для цього за допомогою команди *open* встановлюється з'єднання, наприклад: *open www.yandex.ru 80*.
4. Сформууйте клієнтський запит. Як мінімум він повинен містити рядок стану, наприклад:

GET HTTP://WWW.YANDEX.RU/INDEX.HTML
HTTP/1.0

Якщо поля запиту відсутні, то введення закінчується двома натисканнями клавіші <ENTER> для вставки порожнього рядка після заголовка.

Слід звернути увагу на те, що при введенні не можна допускати помилок, оскільки при спробі їх виправити за допомогою клавіші <BACKSPACE>, її натискання інтерпретується як частина запиту.

5. Вивчіть отриману відповідь сервера. Зверніть увагу на код відповіді в рядку стану відповіді веб-сервера в рядку стану і поля заголовка відповіді.

Якщо відповідь сервера дуже велика (в першу чергу через розмір документа в тілі відповіді), то зміст відповіді сервера у вікні інтерпретатора командного рядка обрізається з початку. У цьому випадку рекомендується для перегляду заголовка замість методу *GET* використовувати метод *HEAD*.

Контрольні завдання

1 За допомогою якої утиліти за заданим доменному імені хосту можна визначити його IP адресу? Визначте IP адресу хоста *www.mail.ru*.

2 За допомогою утиліти *telnet* визначте який веб-сервер встановлений на хості *www.rbc.ru*.

3 Визначте маршрут проходження ICMP пакетів до хосту *www.ttt.com*. Визначте приблизну географічну локалізацію хосту.

Лабораторна робота №2.

Тема «Принципи веб-дизайну. Знайомство з Microsoft Expression Web.»

Мета роботи:

1. Ознайомлення з основними поняттями та принципами веб-дизайну.
2. Знайомство з основними можливостями з розробки веб-сторінок і веб-сайтів за допомогою **Microsoft Expression Web 2**.

Теоретичний матеріал

Веб-дизайн (web page design) – це процес проектування, планування, моделювання та реалізації доставки електронного змісту через мережу Інтернет з використанням технологій (на основі мов розмітки), придатних для інтерпретації та візуалізації веб-браузером або іншим графічним веб-інтерфейсом користувача.

Дуже важливим є приведення веб-ресурсу у відповідність стандартам W3C, що забезпечує доступність змісту для людей з обмеженими фізичними можливостями та користувачів портативних пристроїв, а також кросплатформеність коду розмітки ресурсу.

Основними аспектами веб-дизайну (в першу чергу для комерційних веб-сайтів) є наступні:

- Зміст. Інформаційне наповнення має бути релевантним тематиці сайту і орієнтованим на цільову аудиторію сайту.
- Зручність у використанні (*usability*). Сайт повинен мати дружній користувальницький інтерфейс з простою і надійною навігацією.

- Зовнішній вигляд. Графіка і текст повинні бути виконані в одному стилі і узгоджені на всіх сторінках сайту. Стиль оформлення повинен демонструвати професіоналізм, привабливість і релевантність.
- Видимість. Сайт повинен легко знаходитись за допомогою більшості пошукових систем і рекламних майданчиків.

По-суті, веб-сайт – це інформаційна система, яка містить два основних компоненти:

- Компоненту подання (*front-end*). Видимий зміст (розмітка сторінок, графіка, аудіо та текст).
- Компоненту реалізації (*back-end*). Пов'язана з організацією та ефективною реалізацією вихідних кодів. Включає в себе не відображаємі сценарії, серверні компоненти, що є основою для компонентів подання.

Складанням *технічного завдання* на розробку веб-сайту для фахівців займається *менеджер проекту*. Робота з *замовником* починається зі складання *короткого опису*, в якому викладаються побажання замовника по *візуальному поданні* і структурі сайту. З урахуванням можливостей програмних і дизайнерських засобів і виходячи з короткого опису, *менеджер* складає *технічне завдання*, яке має бути *затверджене замовником*.

Етапи проектування веб-сайту залежать від обсягу сайту, його функціональності і багато чого іншого.

Розробка веб-сайту включає в себе наступні етапи:

- Дизайн головної і типових сторінок сайту. (Виконується зазвичай в графічному редакторі).
- HTML-кодування, в результаті якого створюється код, який можна переглядати за допомогою браузера.
- Програмування сайту. Може здійснюватися як «з чистого аркуша», так і за допомогою спеціального

високорівневого пакету – системи управління сайтом (CMS).

- Розміщення сайту в Веб, наповнення контентом і публікація.
- Оптимізація веб-сайту з метою підвищення його видимості в Веб.
- Здача сайту замовнику.

Для дизайну веб-сайту і розробки веб-додатків для нього має широкий спектр інструментальних засобів. Компанія Майкрософт має вже давню традицію створення WYSIWYG-інструментів такого роду. Серед таких інструментів можна назвати, зокрема *Microsoft Visual Studio .NET* і *Microsoft Expression Web*.

Microsoft Expression Web.

Microsoft Expression Web є складовою частиною пакету *Microsoft Expression Studio*.

Microsoft Expression Studio – пакет графічних програм компанії Майкрософт, призначених для розробників і дизайнерів. Використовує програмний інтерфейс *WinFX*.

WinFX (тепер *NET Framework 3.0*) – прикладний програмний інтерфейс, що входить до складу операційних систем *Windows Vista* і *Windows Server 2008*.

До складу *Microsoft Expression Studio* входять наступні інструменти:

- *Microsoft Expression Blend* – WYSIWYG-інструмент для проектування користувальницького інтерфейсу додатків, створюваних на основі *Windows Presentation Foundation*.
- *Microsoft Expression Web* – WYSIWYG-інструмент для дизайну веб-сайтів і редактор HTML.
- *Microsoft Expression Media* – векторно/растровий графічний редактор.

- *Microsoft Expression Design* – професійний інструмент для графічного дизайну та підготовки ілюстрацій для настільних і web-додатків.
- *Microsoft Expression Encoder* – професійний мультимедіа-кодек для вмісту у форматі VC-1 для *Microsoft Silverlight*.

Microsoft Expression Web 2 дозволяє розробляти веб-сторінки, що інтегрують в рамках веб-сайту такі технології як *XML*, *CSS 2.1*, *ASP.NET 2.0*, *XHTML*, *XSLT* і *JavaScript*., А також впроваджувати в сторінки керуючі елементи *ASP.NET*, *Microsoft Silverlight* і *AJAX*. Для роботи з цим інструментом потрібна наявність встановленої компоненти *.NET Framework 2.0*.

На сторінці

<http://www.microsoft.com/expression/products/overview.aspx?key=web> є посилання на навчальне керівництво, яке демонструє основи роботи з цим середовищем розробки. Зокрема там демонструється створення порожнього веб-сайту, майстер-сторінки, стильового оформлення, створення *.asp сторінки. Також показується яким чином можна додати підтримку RSS на сайті і як створити сторінку, що містить елемент управління *Silverlight* для перегляду відео. Одна із вправ присвячена впровадженню елементу управління *.NET* для відображення даних з джерела на основі *XML*. Показано також, як вставити *AJAX* елемент на сторінку для асинхронного виклику функції та оновлення індикатора часу.

Необхідно приділити особливу увагу створенню майстер-сторінок.

Оскільки сторінки веб-додатки становлять якесь єдине ціле, об'єднане єдиними вимогами до оформлення, змісту, розташуванню елементів управління, то створювані веб-сторінки сьогодні дуже рідко розробляються «з нуля». Зазвичай при створенні сторінок використовуються готові шаблони, або вони створюються в процесі розробки веб-

сторінок додатку. Одним із засобів вирішення подібних завдань є *майстер-сторінки* (master pages). Вони реалізують просту модель створення шаблонів форм з можливістю їх повторного використання.

Для реалізації даного механізму існують такі типи сторінок, як майстер-сторінки (master pages) і *сторінки змісту* (content pages).

Майстер-сторінка являє собою шаблон сторінки, який може містити будь-які елементи, допустимі для звичайної сторінки, а також програмний код. Зазвичай майстер-сторінка містить фіксовані елементи, однакові для всіх сторінок, і заповнювач вмісту для іншої частини сторінки. Найбільш типовими фіксованими елементами є верхній і нижній колонтитули, панель навігації, панель меню та інші.

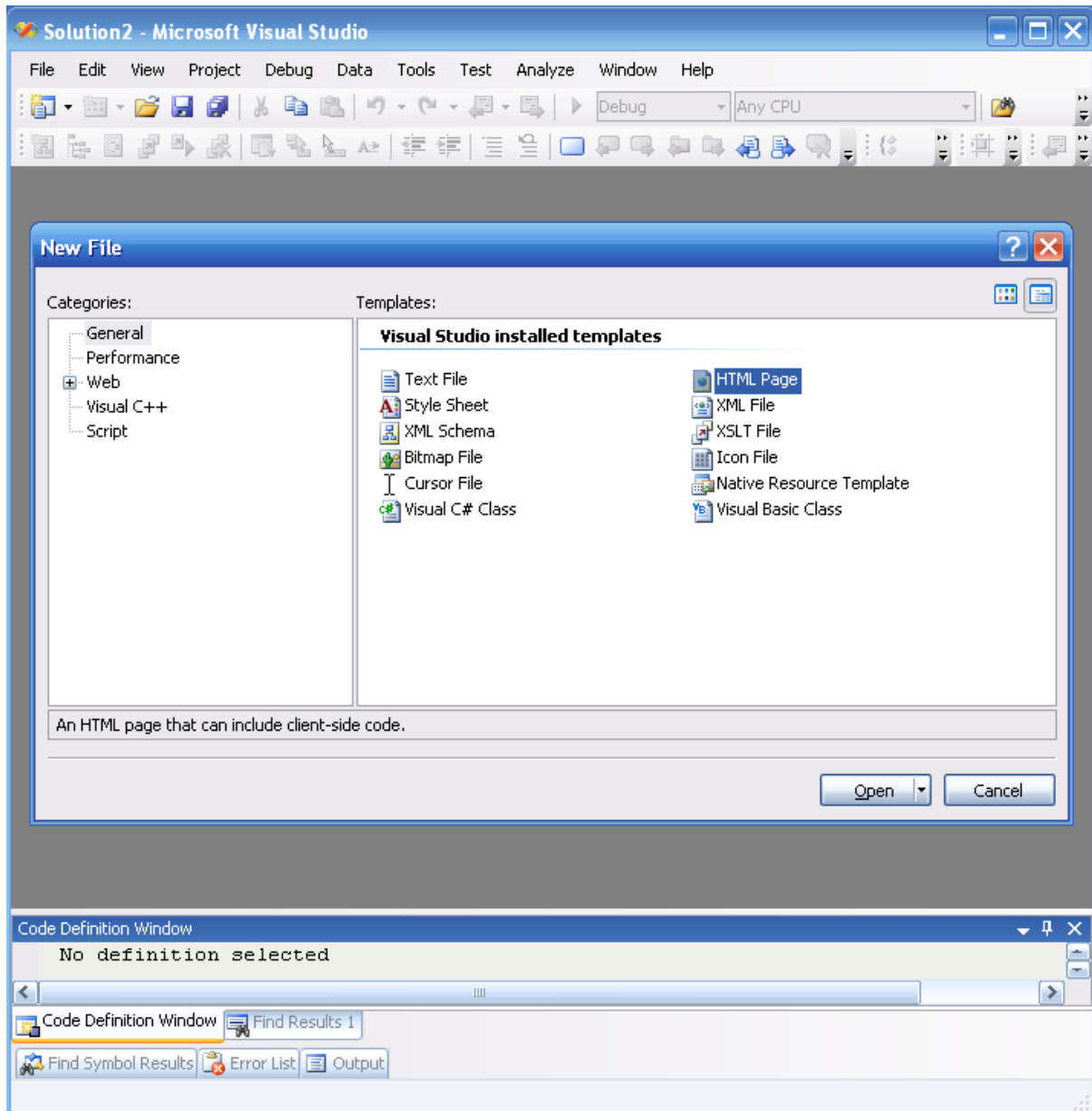
Сторінка змісту включає в себе допустимі елементи управління і за допомогою їх визначає зміст, яким заповнюються спеціальні області майстер-сторінок. Кожна *сторінка змісту* посилається лише на одну майстер-сторінку з якої вона отримує елементи. Крім фіксованих елементів майстер-сторінки вона також надає і додатковий зміст.

Майстер-сторінка повинна містити елемент управління *ContentPlaceHolder*, який призначений для визначення області, куди сторінка змісту може вставляти зміст. При створенні нової майстер-сторінки елемент *ContentPlaceHolder* створюється за замовчуванням. Для того щоб перетворити звичайну сторінку в сторінку змісту, необхідно як значення властивості *MasterPageFile* сторінки вказати ім'я майстер-сторінки, а також додати на сторінку потрібний елемент керування.

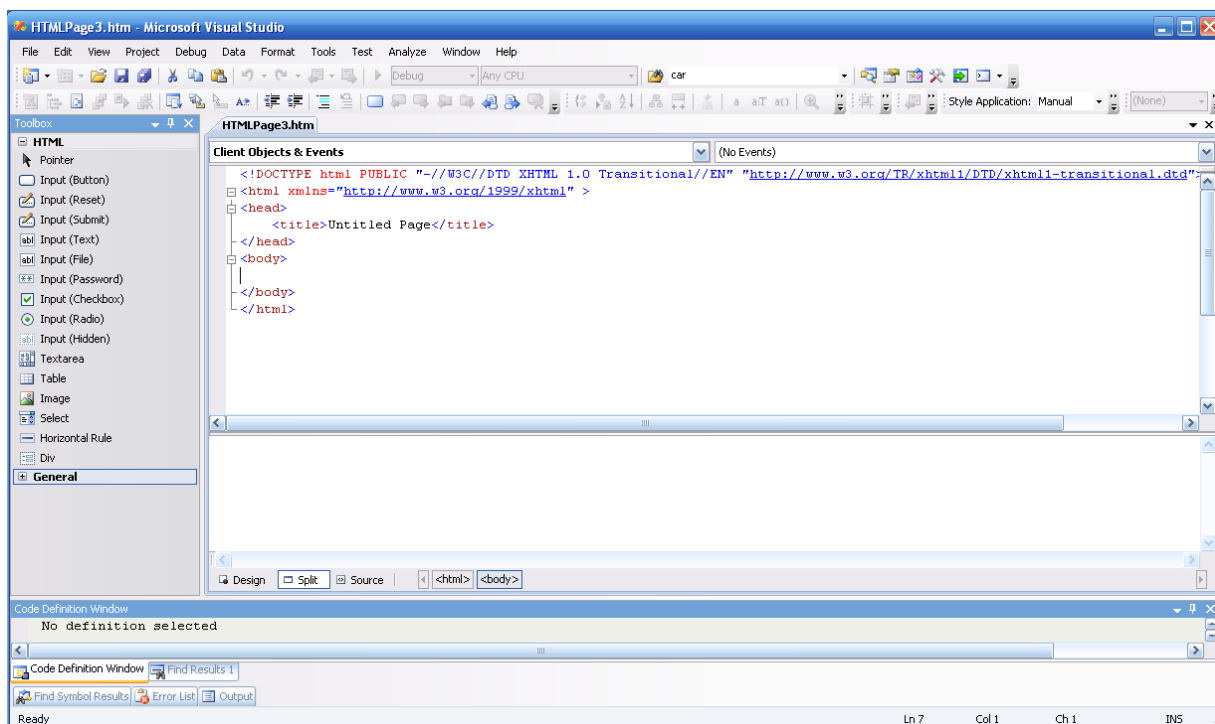
Порядок виконання роботи

1. Створення HTML сторінки в Microsoft Visual Studio. NET.

Для дизайну веб-сторінок в Visual Studio. NET необхідно вибрати розділ меню *File => New* і у вікні вибрати тип файлу *HTML Page*:

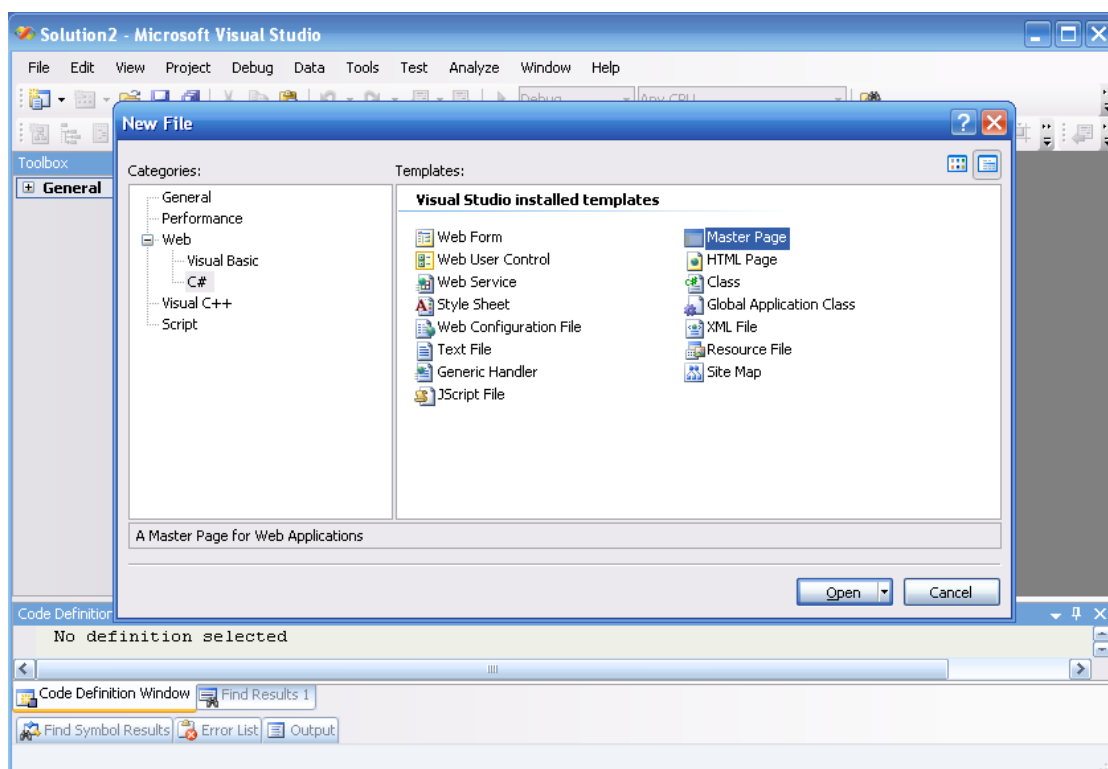


Для дизайну є панель *Toolbox*, що містить набір основних елементів html сторінки (підключається через меню: *View> Toolbox*):

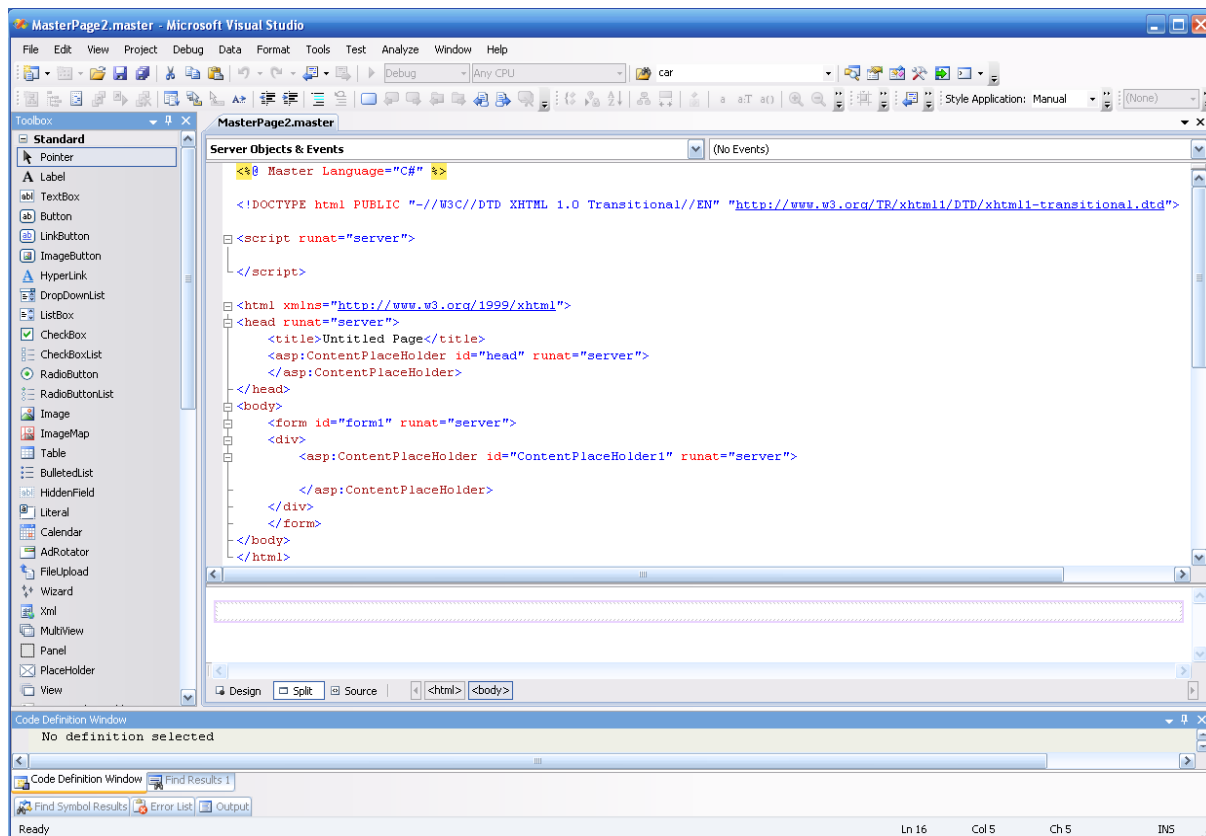


2. Створення майстер-сторінки в *Microsoft Visual Studio.NET*.

Для створення майстер-сторінки необхідно вибрати розділ меню *File => New* і у вікні вибрати тип файлу *Web> C#> Master Page*:



Після цього з'явиться вікно майстер-сторінки, що містить порожню форму і два контейнери змісту *ContentPlaceHolder* (в заголовку і в тілі документа):

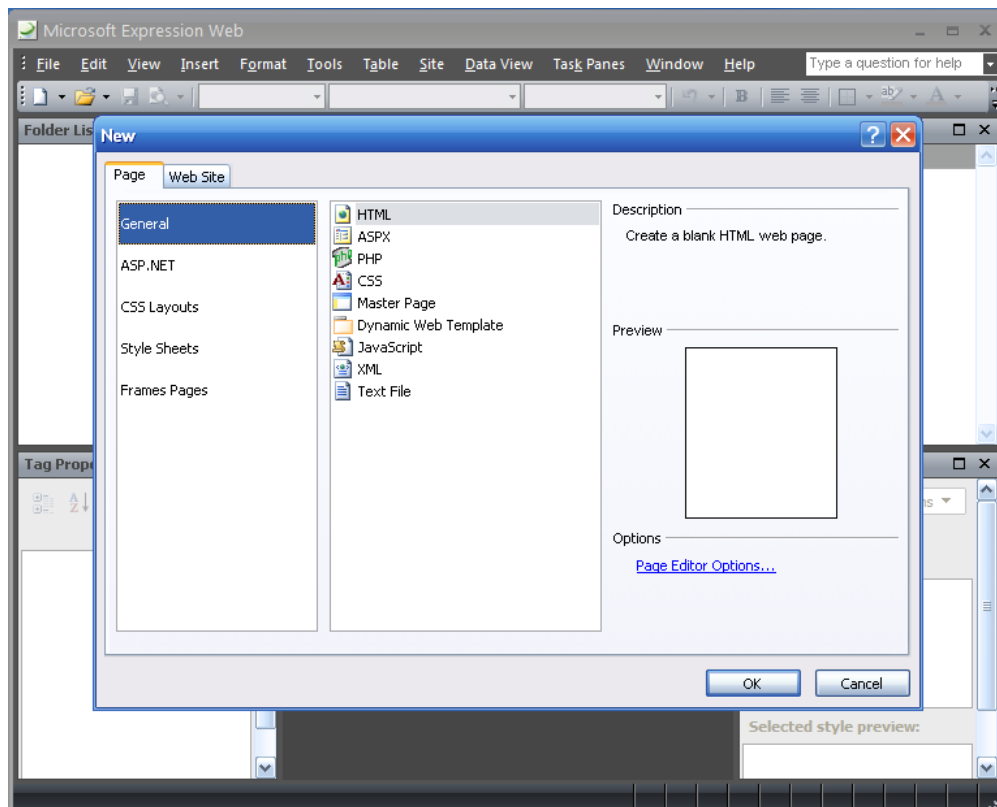


3. Знайомство з *Microsoft Expression Web*

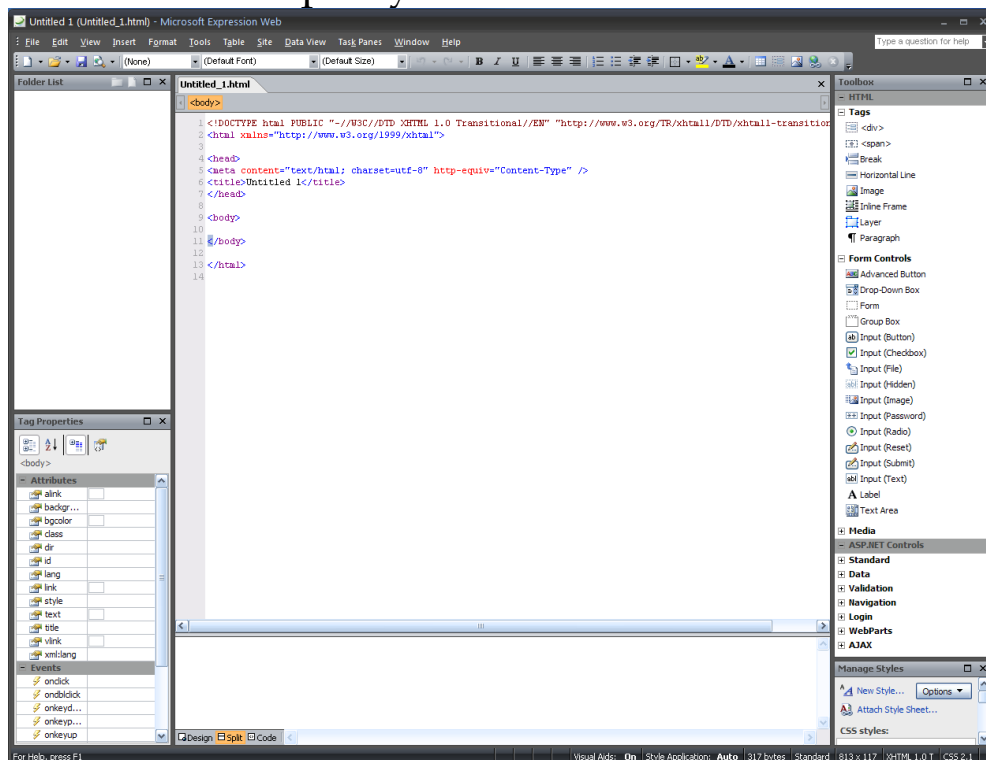
1) Створення нового документа.

Microsoft Expression Web можна використовувати в якості візуального редактора документів різноманітних форматів: HTML, ASP.NET, PHP, CSS, JavaScript, XML. Підтримується робота з шаблонами, майстер-сторінками та веб-сайтами в цілому.

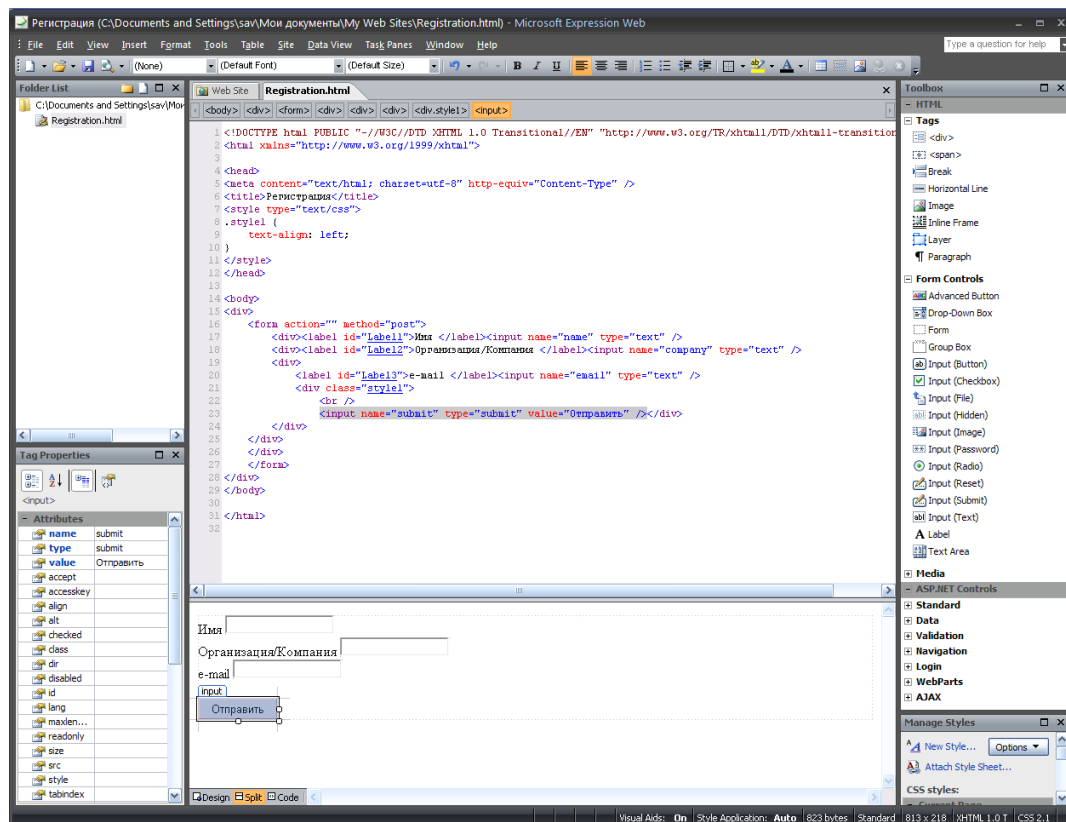
Вибравши розділ меню *File > New*, викличте наступне діалогове вікно:



Виберіть формат документа HTML.
В результаті з'явиться наступне вікно, що містить порожню HTML сторінку:

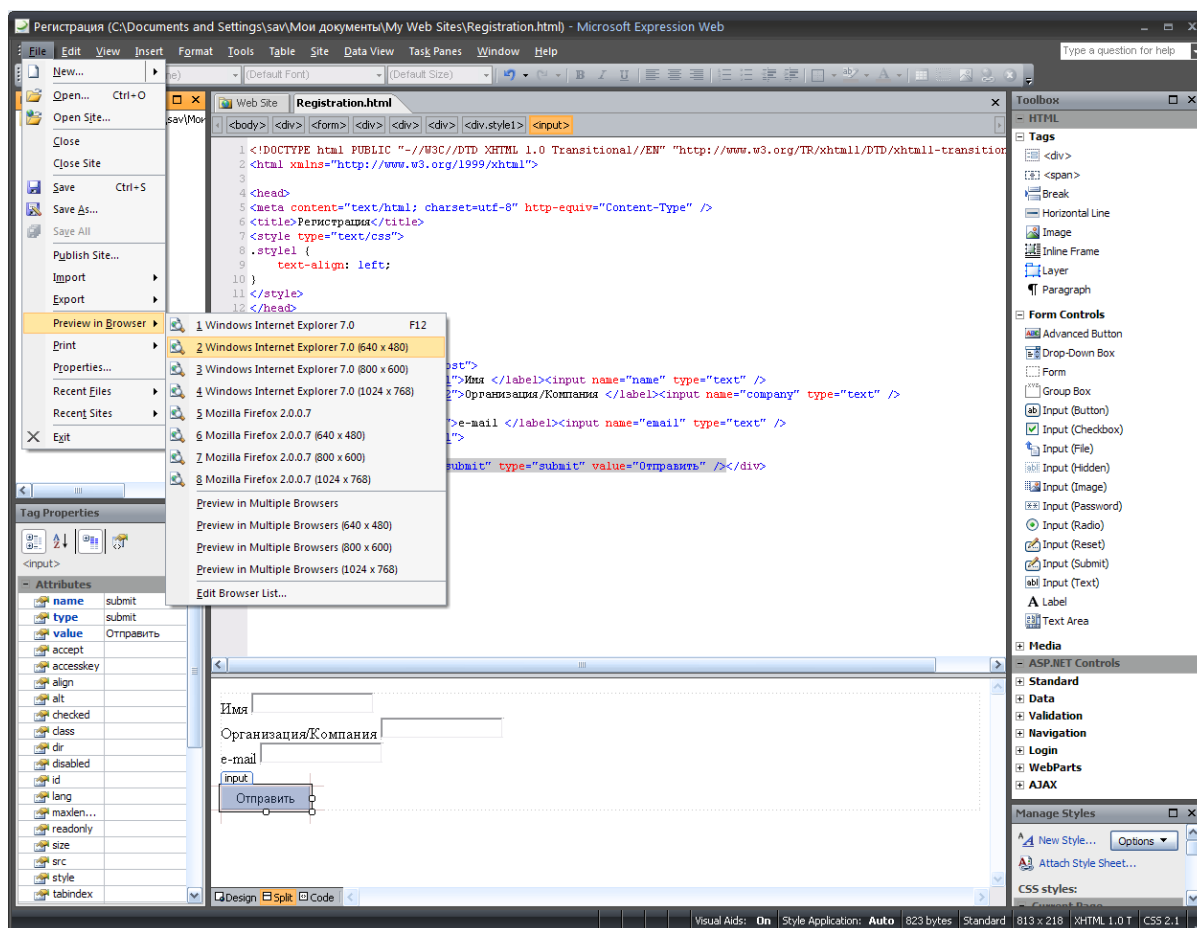


Використовуючи панель інструментів (Toolbox), створіть веб-сторінку для реєстрації користувача наступного виду:



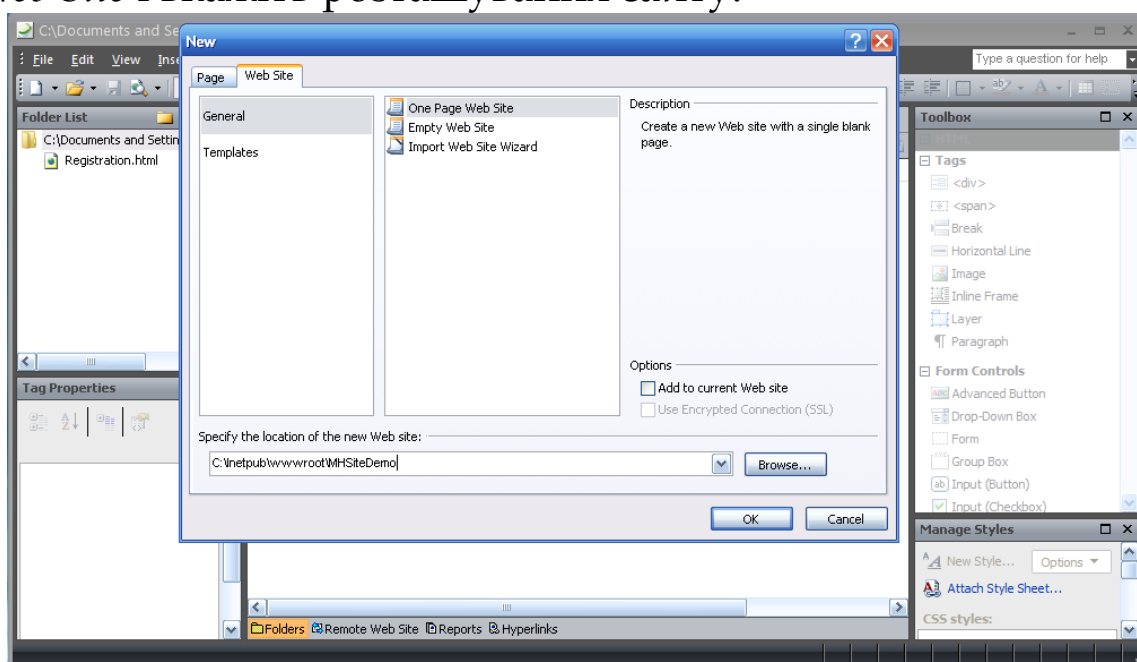
Для зручності вставки елементів форми рекомендується використовувати в якості контейнера елемент `<div>`.

Після збереження документа (*File* > *Save As*) можна попередньо подивитися сторінку в одному з веб-браузерів:

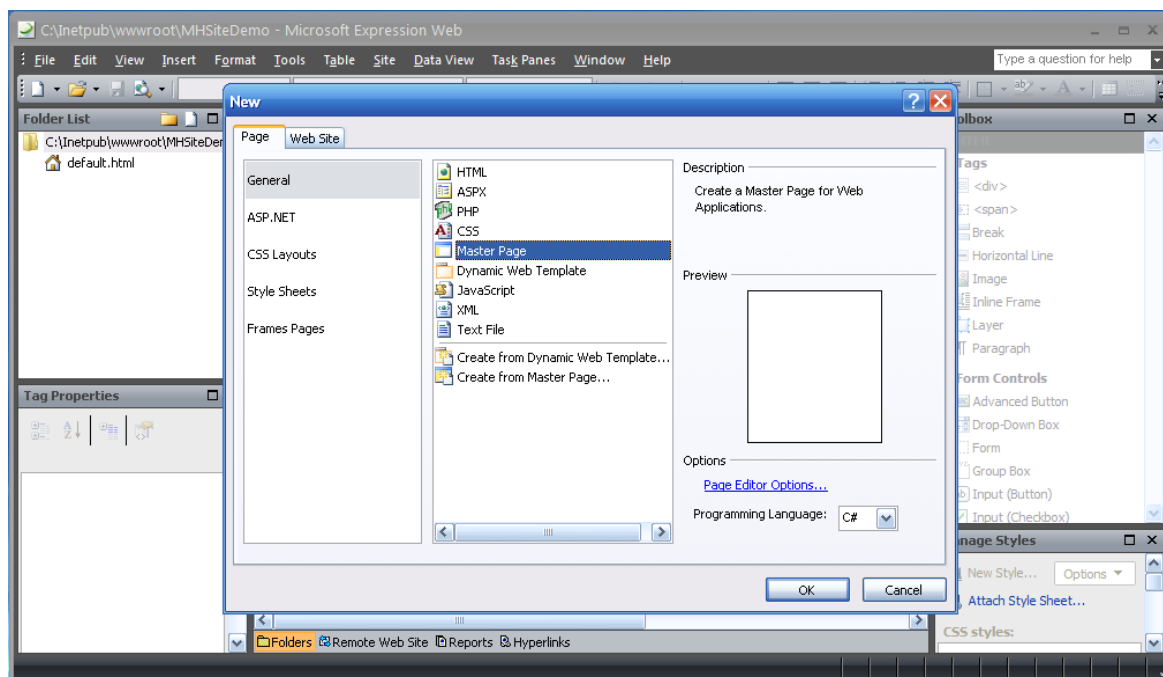


2) Створення веб-сайту.

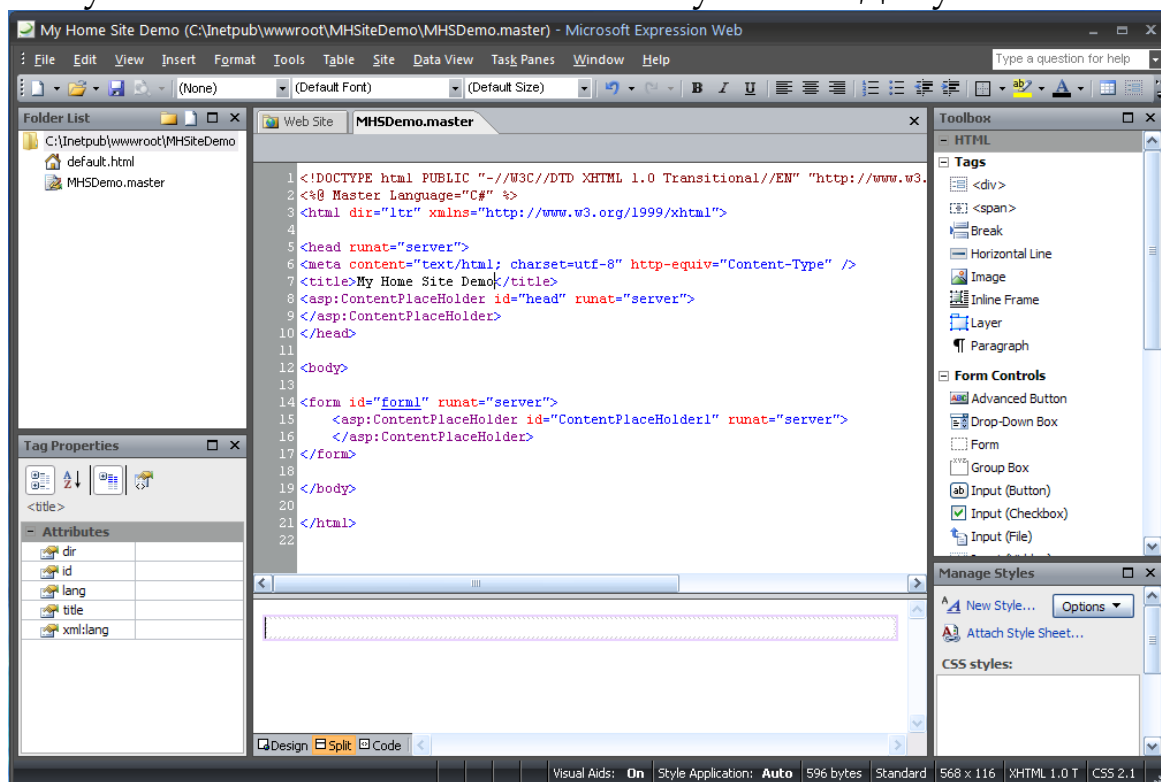
Для створення пустого веб-сайту виберіть розділ меню *File > New*, потім на закладці *Web Site* виберіть *Empty Web Site* і вкажіть розташування сайту:



Як шаблон для створення сторінок веб-сайту можна використовувати майстер-сторінку:



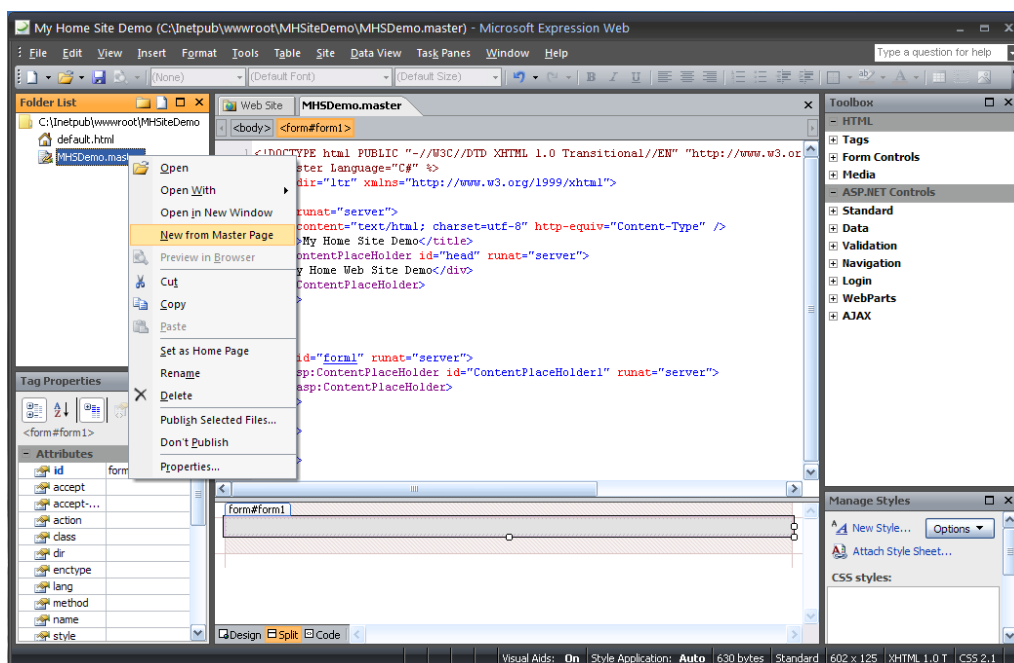
Створена майстер-сторінка містить два контейнери змісту *ContentPlaceHolder*: в заголовку і в тілі документа:



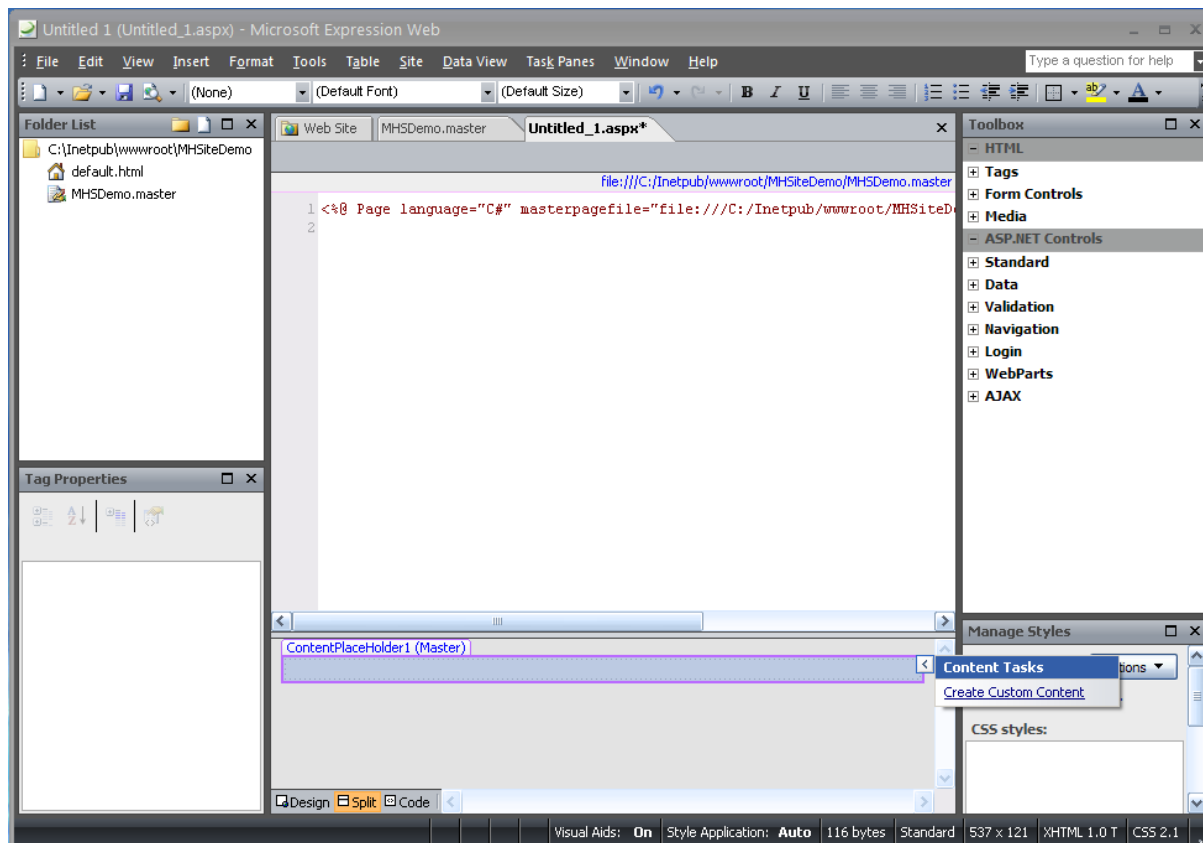
Всі зміни, внесені в цю *майстер-сторінку* будуть автоматично репліцируватися у всіх документах веб-сайту, створених на її основі. Вся специфіка новостворюваних документів повинна бути локалізована в межах контейнера змісту (*ContentPlaceHolder*). Наприклад, у заголовок створимо загальний заголовок «*My Home Web Site Demo*» для всіх сторінок веб-сайту:

```
<head runat="server">
<meta content="text/html; charset=utf-8" http-equiv="Content-Type"
/>
<title>My Home Site Demo</title>
    <asp:ContentPlaceHolder id="head" runat="server">
        <div>My Home Web Site Demo</div>
    </asp:ContentPlaceHolder>
</head>
```

Для створення нової веб-сторінки на основі майстер-сторінки необхідно вибрати в панелі списку файлів майстер-сторінку і потім в контекстному меню (викликається кліком правої кнопки миші) вибрати розділ «*New from Master Page*»:

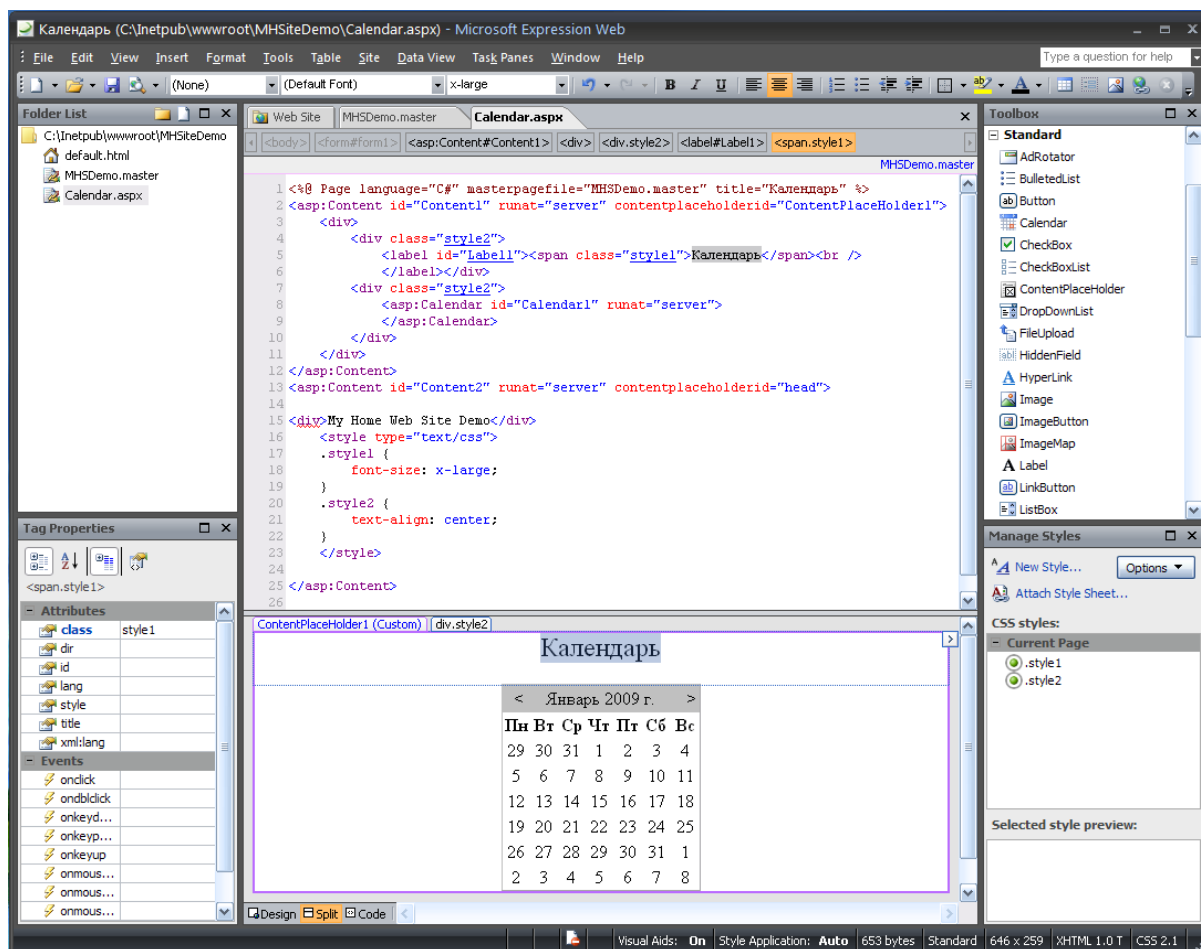


Далі, в режимі відображення «Split» для елемента «ContentPlaceHolder1» по стрілці праворуч від нього слід вибрати з контекстного меню розділ «Create Custom Content»:

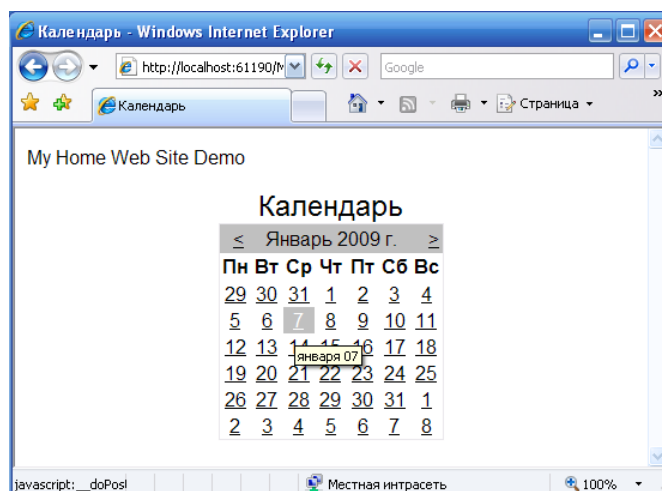


В результаті в коді документа з'являться пара тегів `</asp: Content>` `</ asp: Content>`, між якими може бути додано нове зміст сторінки.

Як приклад розглянемо наступний зміст:



Після збереження документа його можна переглянути у веб-браузері:



Контрольні завдання

1. Вставте логотип (можна використовувати будь-яку картинку) на сторінках веб-сайту, створеного в *Microsoft Expression Web*, шляхом винесення необхідних змін у *майстер-сторінки* сайту.

2. Створіть сторінку реєстрації користувача веб-сайту, створеного в *Microsoft Expression Web*, на основі *майстер-сторінки*.

Лабораторна робота №3.

Тема «Введення в JavaScript. Програмна взаємодія з HTML документами на основі DOM API.»

Мета роботи:

1. Ознайомлення з мовою розробки клієнтських веб-сценаріїв JavaScript.
2. Вивчення основ мови та її застосування для автоматизації процесу розмітки і додавання інтерактивних можливостей веб-сторінок.

Теоретичний матеріал

1. Елементи мови JavaScript.

JavaScript дозволяє «оживити» веб-сторінку. Це реалізується шляхом додавання до статичного опису фрагмент виконуваного коду. JavaScript-сценарій може взаємодіяти з будь-якими компонентами HTML-документа і реагувати на зміну їх стану.

JavaScript не є строго типізованою мовою, в змінних можуть зберігатися практично будь-які типи даних.

Як і програма на мові Java, сценарій JavaScript виконується під управлінням інтерпретатора. Однак якщо Java-додаток або Java-апплет компілюється в байтовий код, то сценарій JavaScript інтерпретується на рівні початкового тексту.

Слід зазначити, що мовні конструкції JavaScript збігаються з відповідними засобами C++ і Java.

1.1. Структура сценарію.

Сценарієм JavaScript вважається фрагмент коду, розташований між дескрипторами `<SCRIPT>` і `</SCRIPT>`:

Текст HTML-документа

<SCRIPT>

Код сценарію

</SCRIPT>

Текст HTML-документа

1.2. Змінні.

У сценаріях JavaScript змінні можуть зберігати дані будь-яких типів: числа, рядки тексту, логічні значення, посилання на об'єкти, а також спеціальні величини, наприклад «нульове» значення *null* або значення *NaN*, яке повідомляє про неприпустимість операції.

Змінна в мові JavaScript оголошується за допомогою ключового слова **var**. Так, наприклад, вираз

```
var selected = "first item";
```

створює змінну з ім'ям *selected* і присвоює їй як значення рядок символів «*first item*». Змінні можуть оголошуватися також автоматично. Це відбувається при присвоєнні значення змінної, яка не зустрічалася раніше в даному сценарії. Так, у наступному прикладі створюється змінна з ім'ям *rating*, якій присвоюється числове значення, рівне 512.5:

```
rating = 512.5;
```

1.3. Об'єкти.

У мові JavaScript не передбачені засоби для роботи з класами в тому вигляді, в якому вони реалізовані в C++ або Java. Розробник сценарію не може створити підклас на основі існуючого класу, перевизначити метод або виконати якусь іншу операцію з класом. Сценарію, написаним на

мові JavaScript, в основному доступні лише готові об'єкти. Побудову нового об'єкту доводиться виконувати лише в окремих випадках.

Об'єкти містять *властивості* (властивості об'єктів можна порівняти з змінними) і *методи*. Об'єкти, а також їх властивості та методи ідентифікуються *іменами*. Об'єктами є форми, зображення, гіпертекстові посилання та інші компоненти веб-сторінки, HTML-документ, який відображається у вікні браузера, вікно браузера і навіть сам браузер. У процесі роботи JavaScript сценарій звертається до цих об'єктів, отримує інформацію і керує ними.

Крім того, розробнику сценарію на мові JavaScript доступні об'єкти, не пов'язані безпосередньо з HTML-документом. Їх називають *зумовленими*, або *незалежними* об'єктами. За допомогою цих об'єктів можна реалізувати масив, описати дату і час, виконати математичні обчислення і вирішити деякі інші завдання.

Перший об'єкт, з яким необхідно познайомитися, щоб написати найпростіший сценарій, – це об'єкт *document*, який описує HTML документ, я що відображається у вікні браузера. Нижче наведено початковий текст веб-сторінки, що містить сценарій, дії якого зводяться до виведення рядка тексту у вікні браузера.

```
<HTML>
<HEAD>
<TITLE>Перший сценарій JavaScript</TITLE></HEAD>
<BODY>
  <SCRIPT LANGUAGE="JavaScript">
    document.write("Перевірка сценарію JavaScript");
  </SCRIPT>
</BODY>
</HTML>
```

Імена чутливі до регістрів символів, і якщо ви спробуєте звернутися до поточного документа на ім'я

Document, інтерпретатор JavaScript відобразить повідомлення про помилку.

Основне призначення сценаріїв JavaScript – створювати динамічно змінювані об'єкти, коректувати зміст HTML-документів в залежності від особливостей оточення, здійснювати взаємодію з користувачем і т.д.

1.4. Операції.

Набір операторів в JavaScript, їх призначення та правила використання в основному збігаються з прийнятими в мові C ++. Винятком є операція, що задається символом «+».

В JavaScript символ «+» визначає як підсумовування числових значень, так і конкатенацію рядків.

Так, наприклад, в результаті обчислення виразу

$$sum = 47 + 21;$$

змінної *sum* буде присвоєно значення 68, а після виконання операції

$$sum = \text{"рядок 1"} + \text{"рядок 2"};$$

в змінну *sum* буде записана послідовність символів "рядок 1 рядок 2".

Розглянемо ще один приклад:

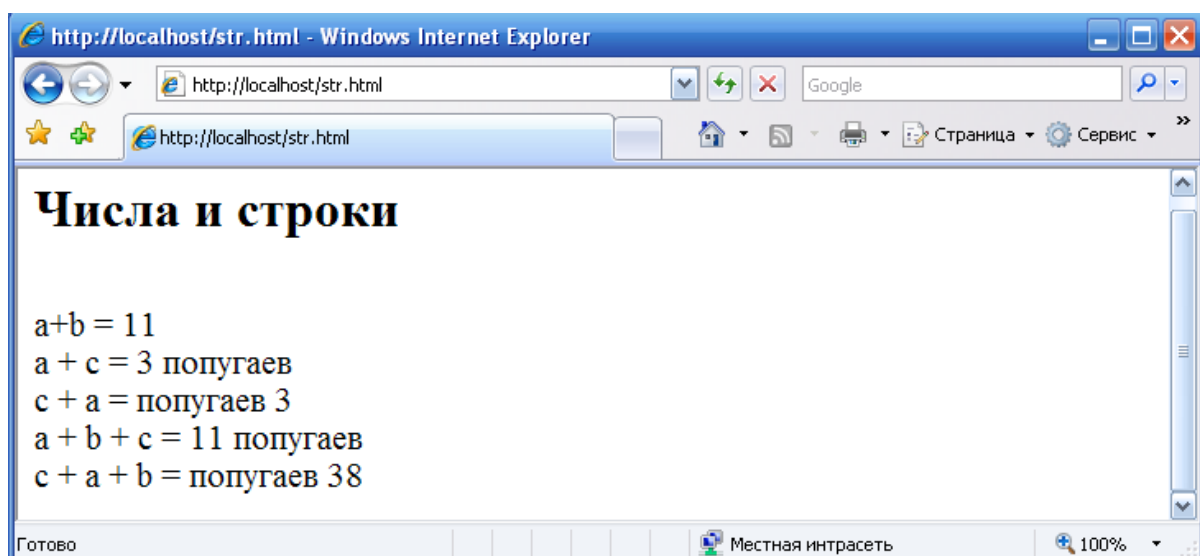
```
<HTML>
<BODY>
<H2>Числа и строки</H2><BR>
  <SCRIPT LANGUAGE="JavaScript">

  var a = 3;
  var b = 8;
  var c = " попугаїв ";

  document.write("a+b="); document.write(a + b);
  document.write("<BR>");
  document.write( "a + c = "); document.write(a+c);
```

```
document.write("<BR>");  
document.write("c + a = "); document.write (c + a);  
document.write ("<BR>");  
document.write ("a + b + c = "); document.write(a + b +  
c);  
document.write("<BR>");  
document.write("c + a + b = "); document.write(c + a +  
b);  
document.write("<BR>");  
  
</SCRIPT>  
</BODY>  
</HTML>
```

У вікні браузера, наведений вище HTML-код виглядає так, як показано на скріншоті:



Перший рядок відображає результат підсумовування двох числових значень, друга і третя - результат конкатенації рядка і символьного представлення числа. Якщо операція підсумовування чисел передує конкатенації, JavaScript обчислює суму чисел, представляє її у символьному вигляді, потім виробляє конкатенацію двох рядків. Якщо ж першою в вираженні вказана операція конкатенації, то JavaScript спочатку перетворює числові

значення в символічний вигляд, а потім виконує конкатенацію рядків.

1.5. Керуючі конструкції.

Керуючі конструкції, що використовуються в мові C++, в основному застосовні і в сценаріях JavaScript.

В JavaScript додатково визначені мовні конструкції, відсутні в C++, а саме: оператори *for ... in* і *with*.

У прикладі 1 за допомогою оператора циклу на веб-сторінці формується таблиця множення чисел.

Приклад 1.

```
<html>
<body>
  <table>
    <script language="JavaScript">
      document.write("<tr><td>&nbsp;</td>");
      for (i = 1; i < 10; i++) document.write("<td>" + i + "&nbsp;</td>");
      document.write("</tr>");
      for (i = 1; i < 10; i++)
      {
        document.write("<tr><td>" + i + "&nbsp;</td>");
        for (j = 1; j < 10; j++)
        {
          document.write("<td    bgcolor='#00ffa0'>" + (i*j) +
            "&nbsp;</td>");
        }
        document.write("</tr>");
      }
    </script>
  </table>
</body>
</html>
```

Окремої уваги заслуговує оператор *new*. Незважаючи на те, що більшість об'єктів уже створені браузером і

доступні сценарієм, в деяких випадках доводиться створювати об'єкти в процесі роботи. Це відноситься до зумовленим об'єктам та об'єктам, які визначаються розробником сценарію. Для створення об'єкта використовується оператор *new*, який викликається наступним чином:

змінна = new тип_об'єкта (параметри)

1.6. Функції.

Формат оголошення функції виглядає наступним чином:

function ім'я функції ([параметри]) тіло функції

Оголошення функції починається з ключового слова **function**. Так само, як і в мові С для ідентифікації функції використовується ім'я, при виклику функції можуть передаватися параметри, а після закінчення виконання повертатися значення. Однак, на відміну від С, тип значення і типи установки не задаються. Нижче показані два способи виклику функції:

- *ім'я_функції ([параметри]);*
- *змінна = ім'я функції ([параметри]);*

У другому випадку значення, що повертається функцією, присвоюється зазначеній змінній.

1.7. Область видимості змінних.

Робота зі змінними в тілі функції підпорядковується наступним правилам:

1. Якщо змінна оголошена за допомогою ключового слова *var*, доступ до неї здійснюється за

- правилами, подібним тим, які використовуються в мові С.
2. Змінна, оголошена всередині функції, вважається *локальною*. Область видимості такої змінної обмежується тілом функції, в якій вона оголошена.
 3. Змінна, оголошена поза функцією, вважається *глобальною*. До неї можна звертатися з будь-якої точки сценарію.
 4. Якщо локальна і глобальна змінні мають однакові імена, то в тілі функції локальна змінна «маскує» глобальну.
 5. Якщо змінна створюється автоматично, тобто якщо вона не оголошена за допомогою ключового слова *var*, але присутня в лівій частині оператора прямого присвоювання, то вона вважається глобальною і стає доступною з будь-якої точки сценарію.

2. HTML DOM.

DOM (Document Object Model) – являє собою стандарт консорціуму W3C для програмного доступу до документів HTML або XML. Фактично це платформо- та мовно-нейтральний інтерфейс, що дозволяє програмам і сценаріям динамічно звертатися і оновлювати вміст, структуру і стиль документа.

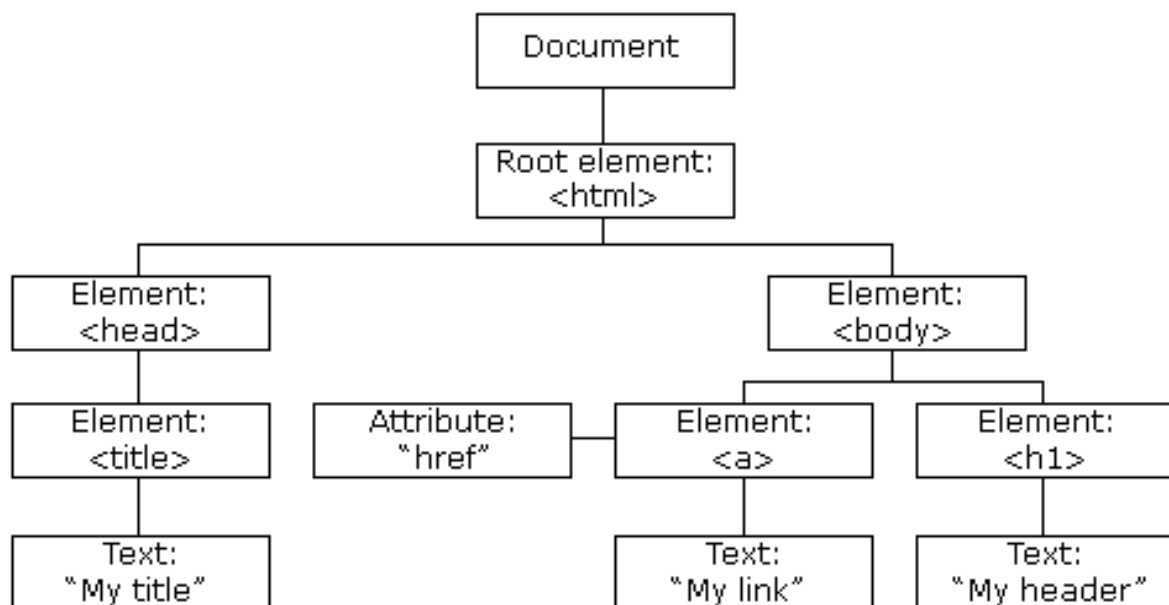
В рамках даного стандарту можна виділити 3 частини:

- Core DOM – стандартна модель будь-якого структурованого документа.
- XML DOM – стандартна модель XML документа.
- HTML DOM – стандартна модель HTML документа.

DOM визначає об'єкти і властивості всіх елементів документа і методи (інтерфейс) для доступу до них.

HTML DOM визначає об'єкти і властивості всіх HTML елементів і методи (інтерфейс) для доступу до них. Інакше кажучи, HTML DOM описує яким чином необхідно отримувати, змінювати, додавати і видаляти HTML елементи.

У відповідності з моделлю DOM все, що міститься всередині HTML документа - є вузлом. Тобто HTML документ представляється у вигляді дерева вузлів, якими є елементи, атрибути та текст.



2.1. Вузли дерева HTML документа.

Згідно моделі DOM:

- Весь документ представляється вузлом документа.
- Кожен HTML тег є вузлом елементу.
- Текст всередині HTML елементів представляється текстовими вузлами.
- Кожному HTML атрибуту відповідає вузол атрибуту.
- Коментарі є вузлами коментарів.

Приклад 2.

```
<html>
<head>
  <title>HTML документ</title>
</head>
<body>
  <h1>Заголовок </h1>
  <p>Просто текст</p>
</body>
</html>
```

У цьому прикладі кореневим вузлом є тег `<html>`. Всі інші вузли утримуються всередині `<html>`. У цього вузла є два дочірніх вузла: `<head>` і `<body>`. Вузол `<head>` містить вузол `<title>`, а вузол `<body>` містить вузли `<h1>` і `<p>`.

Слід звернути особливу увагу на те, що текст, розташований у вузлі елемента відповідає текстовому вузлу. У прикладі `<title>HTML документ</title>` вузол елемента `<title>` містить текстовий вузол «HTML документ», тобто «HTML документ» не є значенням елемента `<title>`. Тим не менш, в рамках HTML DOM значення текстового вузла може бути доступно допомогою властивості `innerHTML`.

Всі вузли HTML документа можуть бути доступні за допомогою дерева, при цьому їх вміст може бути змінено або видалено, а також можна додати нові елементи.

Всі вузли дерева знаходяться в ієрархічних відносинах між собою. Для опису цих відносин використовуються терміни *батько*, *дочірній елемент* і *нащадок*. Батьківські вузли мають дочірні вузли, а дочірні елементи одного рівня називаються нащадками (братами або сестрами).

Відносно вузлів дерева дотримуються наступні принципи:

- Самий верхній вузол дерева називається кореневим.

- Кожен вузол, за винятком кореневого, має рівно один батьківський вузол.
- Вузол може мати будь-яке число дочірніх вузлів.
- Кінцевий вузол дерева не має дочірніх вузлів.
- Нащадки мають загального батька.

2.2. Програмний інтерфейс HTML DOM.

В рамках DOM моделі HTML можна розглядати як безліч вузлових *об'єктів*. Доступ до них здійснюється за допомогою *JavaScript* або інших мов програмування. Програмний інтерфейс DOM включає в себе набір стандартних *властивостей* і *методів*.

Властивості представляють деякі сутності (наприклад, `<h1>`), а *методи* – дії над ними (наприклад, *додати* `<a>`).

До типових властивостей DOM відносяться наступні:

- *x.innerHTML* – внутрішнє текстове значення HTML елемента *x*;
- *x.nodeName* – ім'я *x*;
- *x.nodeValue* – значення *x*;
- *x.parentNode* – батьківський вузол для *x*;
- *x.childNodes* – дочірній вузол для *x*;
- *x.attributes* – вузли атрибутів *x*.

Вузловий об'єкт, який відповідає HTML елементу підтримує такі методи:

- *x.getElementById* (*id*) – отримати елемент з вказаним **id**;
- *x.getElementsByTagName* (*name*) – отримати всі елементи з вказаним ім'ям тега (**name**);
- *x.appendChild* (*node*) – вставити дочірній вузол для *x*;
- *x.removeChild* (*node*) – видалити дочірній вузол для *x*.

Приклад 3.

Для отримання тексту з елемента `<p>` зі значенням атрибута `id «demo»` в HTML документі можна використовувати наступний код:

```
txt = document.getElementById("demo").innerHTML
```

Такий самий результат може бути отриманий по-іншому:

```
txt=document.getElementById("demo").childNodes[0].nodeValue
```

В рамках DOM можливі 3 способи доступу до вузлів:

1. За допомогою методу `getElementById(ID)`. При цьому повертається елемент із зазначеним **ID**.
2. За допомогою методу `getElementsByTagName(name)`. При цьому повертаються всі вузли з вказаним іменем тега (у вигляді індексованого списку). Перший елемент в списку має нульовий індекс.
3. Шляхом переміщення по дереву з використанням відносин між вузлами.

Для визначення довжини списку вузлів використовується властивість `length`.

Приклад 4.

```
x = document.getElementsByTagName("p");
for (i = 0; i < x.length; i++)
{
    document.write(x[i].innerHTML);
    document.write("<br/>");
}
```

У даному прикладі всередину HTML документа вставляється у вигляді списку текстовий вміст всіх елементів відповідних тегом `<p>`.

Для навігації по дереву навколо поточного вузла можна використовувати такі властивості:

- *parentNode*;
- *firstChild*;
- *lastChild*.

Для безпосереднього доступу до тегів можна використовувати 2 спеціальні властивості:

- *document.documentElement* – для доступу до кореневого вузла документа;
- *document.body* – для доступу до тегу `<body>`.

2.3. Властивості вузлів.

В HTML DOM кожен вузол є об'єктом, який може мати методи (функції) і властивості. Найбільш важливими є такі властивості:

- *nodeName*;
- *nodeValue*;
- *nodeType*.

Властивість *nodeName* вказує на ім'я вузла. Ця властивість має такі особливості:

- властивість *nodeName* призначена тільки для читання;
- властивість *nodeName* вузла елемента точно відповідає імені тега;
- властивість *nodeName* вузла атрибуту відповідає імені атрибута;
- властивість *nodeName* текстового вузла завжди дорівнює **#text**;
- властивість *nodeName* вузла документа завжди одно **#document**.

Зауваження: *nodeName* завжди містить ім'я тегу HTML елемента у верхньому регістрі.

Властивість *nodeValue* вказує на значення вузла. Ця властивість має такі особливості:

- властивість *nodeValue* вузла елемента не визначено;

- властивість *nodeValue* текстового вузла вказує на сам текст;
- властивість *nodeValue* вузла атрибуту вказує на значення атрибуту.

Властивість *nodeType* повертає тип вузла. Ця властивість призначена тільки для читання.

Найбільш важливими типами вузлів є наступні:

Тип елементу	Тип вузла
Element	1
Attribute	2
Text	3
Comment	8
Document	9

2.4. Зміна HTML елементів.

HTML елементи можуть бути змінені за допомогою використання JavaScript, HTML DOM і подій.

У прикладі 5 показано, як можна динамічно змінювати текстовий зміст тегу `<p>`:

Приклад 5.

```
<html>
  <body>
    <p id="p1">Hello World!</p>
    <script type="text/javascript">
      document.getElementById("p1").innerHTML="New text!";
    </script>
  </body>
</html>
```

3. Діалогові елементи.

В JavaScript підтримується робота з наступними діалоговими елементами інтерфейсу:

1) **Alert.** Застосовується для повідомлення користувача, що працює з веб-браузером.

Синтаксис:

alert ("повідомлення");

2) **Confirm.** Застосовується для вибору користувачем одного з двох варіантів відповіді «Так / Ні». Відповідно *Confirm* повертає значення *true* / *false*.

Синтаксис:

confirm ("питання");

3) **Prompt.** Застосовується для введення користувачем значення. При натисканні «ОК» повертається введене значення, у випадку «Cancel» повертається значення *null*.

Синтаксис:

prompt ("питання / запит", "значення за замовчуванням");

Нижче наводиться код веб-сторінки, в якій користувач має можливість вибрати колір тексту за допомогою діалогового елемента.

Приклад 6.

```
<html>
<body>
// здесь будет отображаться текст
<div id="c" style="color:blue">Вы выбрали цвет текста:
черный</div>

<script language="JavaScript">
// пользователь выбирает цвет текста
var tcolor = prompt("Выберите цвет текста: red, blue, green,
yellow, black","black");
// задается текст
```



```
document.getElementById("c").innerHTML = "Ви вибрали цвет  
текста: " + tcolor;  
// задається цвет текста  
document.getElementById("c").style.color = tcolor;  
  
</script>  
</body>  
</html>
```

Порядок виконання роботи

Використовуючи *Microsoft Expression Studio 2* (або інший редактор), підготуйте на основі прикладів 1-2, 4-6 відповідні веб-сторінки і перегляньте їх за допомогою веб-браузера.

Контрольне завдання

Створення таблиці випадково вибраних кольорів.

Взявши за основу сценарій побудови таблиці множення, побудуйте таблицю випадково вибраних кольорів. Колір осередки таблиці задається за допомогою атрибута *bgcolor*. Колір комірки описується в рамках трикомпонентної моделі RGB, наприклад: `<td bgcolor = "#c0a145">`. Для генерації кожної компоненти можна використовувати генератор випадкових чисел за допомогою методів об'єкту *Math* і перетворення в шістнадцятковий формат:

```
color = Math.round (255.0 * Math.random ());  
r = color.toString (16);
```

Результуючий колір утворюється шляхом конкатенації компонентів:

```
color = r + g + b;
```

Зразковий вид результату роботи сценарію:

6852a0	1e6eac	2fa1a8	a378b5	34b1e0	238a8b	378c56	619f8d	581d8
11392a	59d966	ba33aa	631033	a33c65	636319	2fae43	4611f8	7f7794
ba7a67	cacb60	6d7160	3b1cb3	265979	b6bc2e	ff26ce	2d59d	6e4e1
7b677a	c4a6bc	bec14f	85437d	1a106f	583c37	4ea14	852213	59f120
909eb6	f395f	3b130	6c083	33310	41a531	d93bf1	1a3ec9	aab213
325f4	8fed	3693	d054dc	f2c484	ac8ef	3aea6	80afbe	e4bcd6
b03872	b13fe	5d7966	71585a	9c845e	233be1	8fded	ab4870	18ccf4
2e4287	cdf15f	927c81	8b7d6e	33c468	eec091	feba0	7b739c	2df9df
90a82d	a66dcf	a7a5f0	5a77f7	b1e64a	9658c7	d5cb45	ea2e82	b1b02d

Лабораторна робота №4.

Тема «Клієнтські сценарії. Використання регулярних виразів.»

Мета роботи:

1. Отримати уявлення про загальні принципи обробки в JavaScript подій, пов'язаних з вікном веб-браузера, веб-сторінкою, що міститься в браузері і елементами документа.
2. Навчитися використовувати найпростіші елементи регулярних виразів для пошуку підрядків, структура і зміст яких описується нетривіальним шаблоном.

Теоретичний матеріал

1. Обробка подій в JavaScript.

Популярність JavaScript багато в чому обумовлена саме тим, що написаний на ньому сценарій може реагувати на дії користувача і інші зовнішні події. Кожна з подій пов'язана з тим чи іншим об'єктом: формою, гіпертекстовим посиланням або навіть з вікном, що містить поточний документ.

Як приклади зовнішніх подій, на які можуть реагувати об'єкти JavaScript, можна навести наступні.

- закінчення завантаження документа у вікно (або закінчення завантаження документів у всі фрейми вікна). Ця подія пов'язана з об'єктом *window*;
- клацання мишею на об'єкт. Ця подія може бути пов'язана з інтерактивним елементом форми або з гіпертекстовим посиланням;

- отримання об'єктом фокусу введення. Ця подія може бути пов'язана з об'єктами типу Text, Password і з іншими інтерактивними елементами;
- передача на сервер даних, введених користувачем за допомогою інтерактивних елементів. Зв'язується з формою.

Обробка події проводиться за допомогою спеціально призначеного для цього фрагмента коду, званого *обробником події*. Для кожної події JavaScript надає свій обробник. Однак при побудові сценарію можна створювати власний обробник події і використовувати його замість обробника, заданого за замовчуванням.

Ім'я обробника визначає, яку подію він повинен обробляти. Так, для того щоб сценарій потрібним чином відреагував на клацання мишею, використовується обробник з ім'ям *onClick*, для обробки події, що полягає в отриманні фокусу введення, – обробник *onFocus*.

Для того щоб вказати інтерпретатору JavaScript на те, що обробкою події повинен займатися обробник, необхідно включити в HTML-дескриптор такий вираз:

Ім'я_обробника="команди_обробника"

Цей вираз включається в тег, який описує об'єкт, з яким пов'язана подія.

Наприклад, якщо необхідно обробити подію, яка полягає в отриманні фокусу полем введення, дескриптор, що описує цей інтерактивний елемент, повинен мати приблизно такий вигляд:

<Input type = "text" name = "Inform" onFocus = "handleFocus();" >

Ім'я обробника є одним з атрибутів HTML-дескриптора, а команди, призначені для обробки події, виступають в ролі значення цього атрибута. В даному

випадку обробка події проводиться в тілі функції *handleFocus()*. В принципі, оброблювачем може бути не тільки функція, але і будь-яка послідовність команд JavaScript у вигляді складеного оператора.

Наступний приклад демонструє обробку події, яка пов'язана з наведенням курсору миші на гіперпосилання:

```
<a href = "http://www.myhp.edu" onmouseover="alert('An  
onMouseOver event'); return false">  
      
</a>
```

Нижче наводиться повний текст HTML документа с JavaScript сценарієм, в якому обробляється подія натиснення кнопки миші, і визначається, яка саме з них була натиснута:

Приклад 1.

```
<html>  
  <head>  
    <script language = "javascript">  
      function whichButton(event)  
      {  
        if (event.button == 2)  
        {  
          alert("Ви клацнули правою кнопкою миші!");  
        }  
        else  
        {  
          alert("Ви клацнули лівою кнопкою миші!");  
        }  
      }  
    </script>  
  </head>  
  
  <body onmousedown="whichButton(event)">  
    <p> Клацніть клавішею миші в будь-якому місці документа  
  </p>
```

```
</body>
</html>
```

Таким чином, для того щоб обробити якусь стандартну подію в браузері, необхідно додати у відповідний HTML-тег атрибут, який відповідає цій події, вказавши як значення атрибуту ім'я JavaScript функції. Список атрибутів, які визначені для HTML-тегів наводиться нижче:

IE: *Internet Explorer*, **F:** *Firefox*, **O:** *Opera*, **W3C:** *стандарт*

Атрибут	Опис	Номер версії браузера			W3C
		IE	F	O	
onabort	Перерване завантаження зображення	4	1	9	Так
onblur	втрата фокусу елементом	3	1	9	Так
onchange	Зміна вмісту в поле вводу	3	1	9	Так
onclick	Клацання миші на об'єкті	3	1	9	Так
ondblclick	Подвійне клацання миші на об'єкті	4	1	9	Так
onerror	Помилка при завантаженні зображення або документа	4	1	9	Так
onfocus	Отримання фокуса елементом	3	1	9	Так
onkeydown	Натискання клавіші	3	1	Hi	Так
onkeypress	Клавіша натиснута	3	1	9	Так
onkeyup	Відтискання клавіші	3	1	9	Так
onload	Завершення завантаження сторінки або зображення	3	1	9	Так
onmousedown	Натискання кнопки миші	4	1	9	Так
onmousemove	Переміщення курсору миші	3	1	9	Так
onmouseout	Зсув курсору миші з об'єкта	4	1	9	Так
onmouseover	Наведення курсору миші на об'єкт	3	1	9	Так

onmouseup	Відтискання кнопки миші	4	1	9	Так
onreset	Кнопка «Reset» натиснута	4	1	9	Так
onresize	Зміна розміру вікна	4	1	9	Так
onselect	Виділення тексту	3	1	9	Так
onsubmit	Кнопка «Submit» натиснута	3	1	9	Так
onunload	Вихід з веб-сторінки	3	1	9	Так

2. Регулярні вирази

Регулярні вирази – система пошуку текстових фрагментів в електронних документах, заснована на спеціальній системі запису зразків для пошуку.

Зразок, що задає правило пошуку, називається «шаблоном». Застосування регулярних виразів принципово змінило технології електронної обробки текстів.

За допомогою регулярних виразів можна задавати структуру шуканого шаблону і його позицію всередині рядка (наприклад, на початку або в кінці рядка, на кордоні або не на межі слова).

При описі структури шаблону використовуються:

- гнучка система квантифікаторів (операторів повторення);
- оператори опису наборів символів та їх типу (числові, нечислові, спеціальні).

Для того, щоб задати положення шуканого фрагменту усередині рядка, можна використовувати один з наступних операторів:

Подання	Позиція
^	Початок рядка
\$	Кінець рядка
\b	Границя слова
\B	Не границя слова
(?=шаблон)	Шуканий рядок впливає після зазначеного рядка (з переглядом вперед)
(?!шаблон)	Шуканий рядок не впливає після зазначеного

	рядка (з переглядом вперед)
(?<=шаблон)	Шуканий рядок впливає після зазначеного рядка (з переглядом назад)
(?<!шаблон)	Шуканий рядок не впливає після зазначеного рядка (з переглядом назад)

Крім того, мова регулярних виразів надає набір квантифікаторів, що дозволяють вказати кількість повторень шаблону:

Подання	Число повторень
{ <i>n</i> }	Рівне <i>n</i>
{ <i>m</i> , <i>n</i> }	Від <i>m</i> до <i>n</i> включно
{ <i>m</i> , }	Не менше <i>m</i>
{, <i>n</i> }	Не більше <i>n</i>

Є і більш прості квантифікатори:

Подання	Число повторень	Еквівалент
*	Нуль або більше	{0,}
+	Одне або більше	{1,}
?	Нуль або одне	{0,1}

Для того, щоб задати всередині шаблону групи символів можна використовувати наступні оператори:

Оператор	Опис
[xyz]	Будь-який символ із зазначеної множини
[^xyz]	Будь-який символ що не входить у вказану множину
[x-z]	Будь-який символ з зазначеного діапазону
[^x-z]	Будь-який символ що не входить у вказаний діапазон
. (крапка)	Будь-який символ, окрім символів розриву або переносу рядка
\w	Будь-який буквено-цифровий символ, включаючи символ підкреслення
\W	Будь-який не буквений символ
\d	Будь-яка цифра

\D	Будь-який нецифровий символ
\s	Будь-який не відображаємі символ
\S	Будь-який символ (крім не відображаємих символів)

Для угруповання окремих частин шаблону можна використовувати наступні оператори:

Оператор	Опис
()	Пошук групи символів всередині дужок і збереження знайденої відповідності
(?:)	Пошук групи символів всередині дужок без збереження знайденої відповідності
	Комбінування частин в одне вираження з подальшим пошуком кожної із частин окремо. Аналогічно оператору «АБО».

Якщо шаблон пошуку включає спеціальні (як правило не відображаємі) символи, для їх опису можна використовувати такі позначення:

Позначення	Опис
\0	Символ з нульовим кодом
\n	Символ нового рядка
\r	Символ початку рядка
\t	Символ табуляції
\v	Символ вертикальної табуляції
\xxx	Символ, що має заданий восьмеричний ASCII код <i>xxx</i>
\xdd	Символ, що має заданий шістнадцятковий ASCII код <i>dd</i>
\uxxxx	Символ, що має ASCII код виражений Юнікодi <i>xxxx</i>

Квантифікаторам в регулярних виразах відповідає максимально довгий рядок з можливих (тобто квантифікатори є «жадібними»). Це може призводити до

деяких проблем. Наприклад, шаблон (<. *>), що описує на перший погляд HTML-теги, насправді буде виділяти більші фрагменти в документі.

Наприклад, рядок виду

<P> <i> Регулярні вирази <i> -

зручний інструмент для пошуку в рядках </ p>, який формально відповідає зазначеному вище шаблону

Для вирішення даної проблеми можна використовувати два підходи:

1. У регулярному виразі враховуються символи, що не відповідають бажаному зразком (наприклад, <[^>] *> для вищеописаного випадку).
2. Визначення квантифікатору як нежадібного (ледачого) – більшість реалізацій дозволяють це зробити, додавши після нього знак питання.

Наприклад, за шаблоном (<. *?>) будуть знайдені всі теги з розглянутого рядка.

Таким чином, виходять такі «нежадібні» модифікації квантифікаторів:

Квантифікатор	Опис
*?	«Не жадібний» еквівалент *
+?	«Не жадібний» еквівалент +
{n,}?	«Не жадібний» еквівалент {n,}

Слід, однак, мати на увазі, що використання «ледачих» квантифікаторів може призвести до ситуації, коли виразу відповідає занадто короткий, зокрема, порожній рядок.

3. Використання регулярних виразів в JavaScript

При пошуку по тексту можна використовувати шаблон, що описує підрядок. В JavaScript такий шаблон може бути описаний за допомогою об'єкту *RegExp*. У найпростішому випадку такий шаблон описує окремий символ, однак має сенс його використовувати для регулярних виразів.

Наступний нижче код описує *RegExp* об'єкт з ім'ям *pattn*, що містить регулярний вираз, який описує ціле десяткове число:

```
var pattn = new RegExp("/[0-9]+/");
```

Об'єкт *RegExp* має 3 вбудованих методи: *test()*, *exec()* і *compile()*.

Метод *test()* виконує пошук за шаблоном:

```
var pattn = new RegExp("[0-9]+");  
document.write(pattn.test("38 попугаїв"));
```

Результат:

```
true
```

Метод *exec()* виконує пошук підрядка за шаблоном і повертає знайдені відповідності; якщо відповідностей немає, повертається значення *null*:

```
var pattn=new RegExp("[0-9]+");  
document.write(pattn.exec("38 попугаїв"));
```

Результат:

38

Якщо необхідно знайти всі відповідності, то при виклику конструктора *RegExp* слід вказати додатковий параметр "g", який вказує на необхідність глобального пошуку.

Приклад 2.

```
var pattn = new RegExp("[0-9]+", "g");
do
{
    result = pattn.exec("1 попугай, 2 попугая,..., 38 попугаїв");
    document.write(" " + result);
}
while (result != null)
```

Результат:

1 2 38 null

Метод *compile()* застосовується для зміни раніше створеного шаблону.

Приклад 3.

```
var pattn = new RegExp("[0-5]+");
document.write(pattn.exec("38 попугаїв"));

pattn.compile("[6-9]+");
document.write("; " + pattn.exec("38 попугаїв"));
```

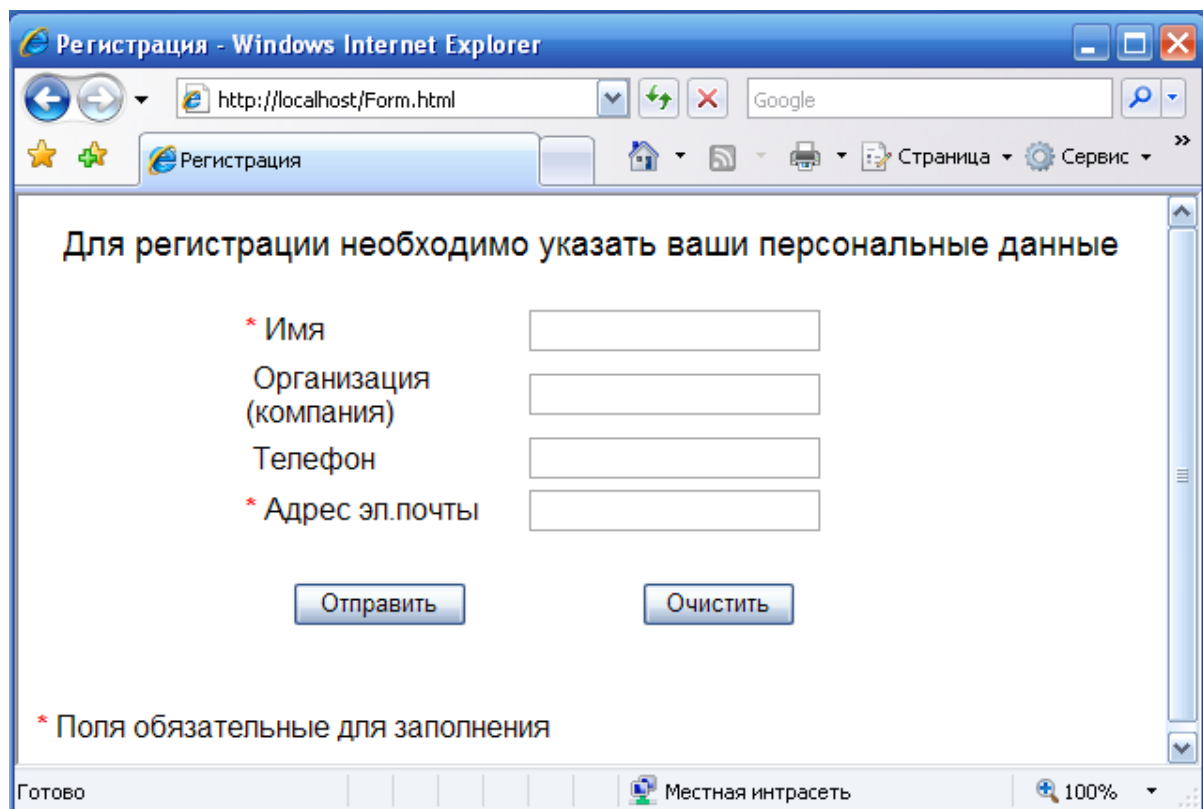
Результат:

3;8

Порядок виконання роботи

Перевірка значень, введених користувачем в поля форми для реєстрації.

1) Для виконання лабораторної роботи необхідно створити веб-сторінку, що містить форму з полями, наступного виду:



2) У тегу `<form>` додайте обробник події відправки даних виду:

```
onSubmit = "CheckData(); return false;"
```

У даному випадку вказана функція обробник `CheckData()`. Оператор `return false;` запобігає автоматичне відправлення даних після виконання функції-обробника. Відправка даних буде виконуватися з обробника.

3) Додайте на сторінці секцію JavaScript коду, що описує функцію-обробника:

```
function CheckData()
{
    var ans;

    ans = confirm("Вы уверены, что хотите отправить введенные
данные ?");
    if (ans) submit();
}
```

Як це видно з коду, функція *CheckData()* у разі підтвердження з боку користувача самостійно викликає метод *submit()* для передачі даних з форми.

4) Тепер необхідно додати перевірку значень, введених в поля форми користувачем.

Перш за все, необхідно переконатися в тому, що заповнені всі поля, обов'язкові для введення. Для цього можна використовувати перевірку на рівність нулю довжини рядка (властивість *length*), що є значенням вузла дерева документу, що відповідає полю введення, наприклад: *document.getElementById("uname").Value.length*.

Наступна перевірка повинна контролювати структуру і зміст полів. Для цього можна використовувати об'єкт *RegExp*, наприклад:

```
var validEMail, pattn;

pattn = new RegExp("^[\.\_A-Za-z0-9]+?@[.\_A-Za-z0-9]+?\. [A-Za-z0-9]{2,6}$");

validEMail = pattn.test(document.getElementById("email").value));
```

У цьому фрагменті описана перевірка структури електронної адреси з поля форми з ідентифікатором *"email"*. Для перевірки був використаний шаблон на основі регулярного виразу.

Контрольне завдання

Самостійно побудуйте регулярний вираз, що описує шаблон для перевірки номера телефону, і внесіть всі необхідні зміни і доповнення в функцію *CheckData()*.

Лабораторна робота №5.

Тема «Розробка CGI-додатків на Perl і PHP.»

Мета роботи:

Ознайомлення з:

1. основами мов розробки веб-сценаріїв на мовах Perl і PHP;
2. синтаксисом мов Perl і PHP;
3. реалізацією обробки даних, отриманих від клієнтського додатка, на стороні веб-сервера.

Теоретичний матеріал

1. Основи розробки сценаріїв на мові Perl

Програма на мові Perl складається з *декларацій* і *операторів*. Будь-який текст, починаючи з символу «#» і до кінця рядка, вважається коментарем і ігнорується.

Для *змінних* декларацій не потрібно. До тих пір доки їм не буде присвоєно яке-небудь конкретне значення, вони просто містять невизначене значення *undef*. Декларації можуть розташовуватися в будь-якому місці програми, так як обробляються на етапі компіляції, що передує етапу виконання програми.

Оператори мови Perl підрозділяються на *прості* і *складові*. *Складові* оператори складаються з блоків, укладених у фігурні дужки. На відміну від мови C, фігурні дужки в складових операторах обов'язкові, навіть якщо в них укладений тільки один оператор. Оператори розділяються крапкою з комою.

Знайомство з Perl можна починати зі сценарію 1.

Сценарій 1. Виведення рядка вітання на Perl.

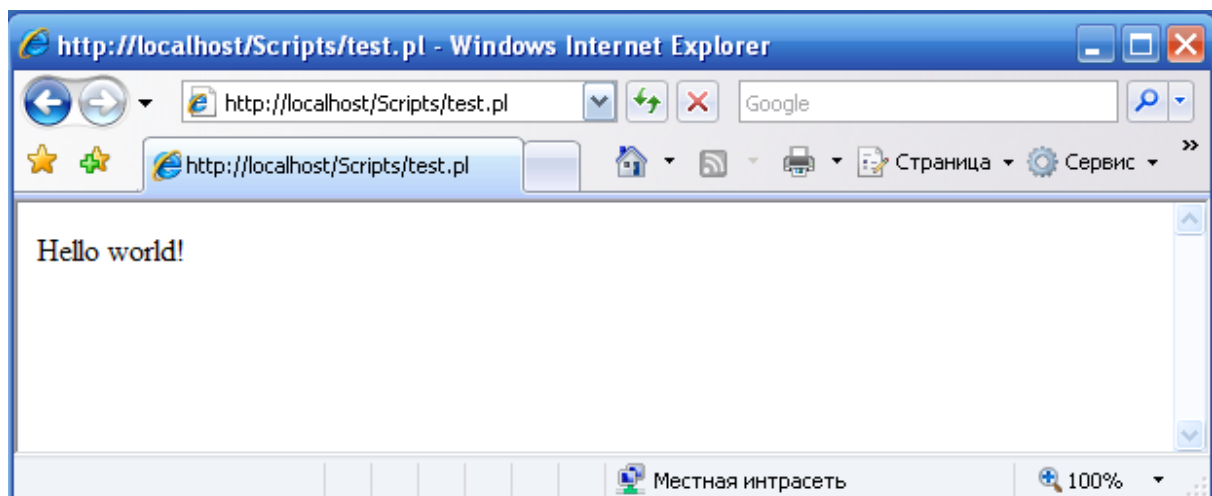

```
#!/c:/perl/bin/perl  
print "Content-type:text/html\n\n";  
print "Hello world!";
```

Перший рядок сценарію, оформлений у вигляді коментаря, що вказує на фізичне розміщення інтерпретатора мови Perl. Решта рядків фактично формують відповідь веб-сервера.

У другому рядку у вихідний потік передається поле *Content-type* заголовка відповіді сервера, і врешті вставляється порожній рядок, що відокремлює заголовок від тіла відповіді сервера.

В останньому рядку поміщається зміст тіла відповіді сервера.

У результат виконання сценарію даного сценарію отримаємо наступну сторінку:



У наступному прикладі буде розглянуто сценарій, що зчитує і обробляє дані, отримані веб-сервером із запиту клієнта. Вихідні дані повинні вводитися користувачем в поля форми веб-сторінки, завантаженої в веб-браузері.

Сценарій 2. Виведення списку параметрів, отриманих сервером в запиті від клієнта.

```
#!/c:/perl/bin/perl

print "Content-type: text/html\n\n";
print "<HTML><BODY>\n";

$method = $ENV{'REQUEST_METHOD'};

if ($method eq 'POST')
{
    $length = $ENV{'CONTENT_LENGTH'};
    read(STDIN, $qstr, $length);
}
else
{
    if ($method eq 'GET') { $qstr = $ENV{'QUERY_STRING'}; }
    else
    {
        print "Method      ".$method."      is      not      supported
</BODY></HTML>";
        #exit(0);
    }
}

print "<P>Метод = ", $method;
print "<p>Строка параметров: <p>\n";
print $qstr;

# зворотнє перекодування

$qstr =~ tr/+//;
$qstr =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;

# виділення списку параметрів

print "<p>А теперь отдельные поля:<p>";
@pars = split(/&/, $qstr);

# визначення розміру масиву

$n_pars = @pars;
```

```
# виділення імені і значення для кожного параметра

for ($i=0; $i<$n_pars; $i++)
{

    # виділення списку з двох змінних $name та $value

    ($name, $value) = split(/=/, $pars[$i]);
    print "Параметр <B>", $name, "</B> дорівнює <I>", $value,
    "</I><br>";
}
print "</HTML></BODY>\n";
```

Даний сценарій орієнтований на передачу даних з веб-форми одним з основних методів: *GET* або *POST*. Оскільки передача даних у цих методах відрізняється, то в сценарії спочатку визначається метод передачі даних шляхом звернення до змінної оточення *REQUEST_METHOD*, значення якої є сценарієм через однойменний елемент асоціативного масиву (хешу) *ENV*, що містить значення всіх змінних оточення.

Після визначення, що використовувався в запиті клієнта методу, вибирається адекватний спосіб читання параметрів, отриманих з веб-форми:

- У випадку методу *POST* визначається загальний розмір переданих даних (в байтах) із змінної оточення *CONTENT_LENGTH*, а потім блок даного розміру зчитується з вхідного потоку *STDIN* за допомогою функції *read*.
- У випадку методу *GET* дані доступні в змінній оточення *QUERY_STRING*.
- Якщо метод запиту не збігається ні з одним з розглянутих вище або його значення не визначено, то відбувається примусове завершення сценарію з видачею відповідного повідомлення.

Оскільки в рамках протоколу HTTP символи, відмінні від латинських букв і цифр передаються у вигляді шістнадцяткових кодів (прогалини замінюються на «+»), потрібно попереднє зворотнє перетворення отриманих даних за допомогою операторів заміни *s///* і *tr//* з використанням шаблонів у вигляді регулярних виразів.

Далі за допомогою функції *split* відбувається поділ блоку символів на *підрядки* по заданому *роздільнику*:

- *Пари*, що описують ім'я параметра і його значення поділяються за допомогою символу «&». Результат поділу поміщається в скалярний масив *@pars*.
- Всередині пари *ім'я* і *значення* розділяються символом «=».

Розглянутий приклад може бути взятий за основу для будь-якого сценарію, що обробляє дані з веб-форми, оскільки дозволяє отримати вихідні дані, отримані від клієнта. Розширення сценарію зводиться до додавання коду обробки отриманих даних і формування підсумкового документа, що повертається веб-сервером клієнтові.

Розглянемо для прикладу сценарій, що виконує чотири арифметичні операції над цілими числами, які користувач вводить через поля веб-форми в браузері.

Сценарій 3. *Калькулятор арифметичних операцій для цілих операндів.*

```
#!/c:/perl/bin/perl

print "Content-type: text/html\n\n";
print "<HTML><BODY>\n";

$method = $ENV{'REQUEST_METHOD'};

if ($method eq 'POST')
{
    $length = $ENV{'CONTENT_LENGTH'};
```

```
        read(STDIN, $qstr, $length);
    }
    else
    {
        if ($method eq 'GET') { $qstr = $ENV{'QUERY_STRING'}; }
        else
        {
            print "Method      ".$method."      is      not      supported
</BODY></HTML>";
            exit(0);
        }
    }

    $qstr =~ tr/+/ /;
    $qstr =~ s/%([a-zA-Z0-9][a-zA-Z0-9])/pack("C", hex($1))/eg;

    @pars = split(/&/, $qstr);
    $n_pars = @pars;

    foreach $par (@pars)
    {
        ($name, $value) = split(/=/, $par);
        $opers{$name} = $value;
    }

    $op1 = int($opers{'A'});
    $op2 = int($opers{'B'});
    $op = $opers{'op'};

    switch:
    {
        if ($op eq '+') { $res = $op1 + $op2; last switch; }
        if ($op eq '-') { $res = $op1 - $op2; last switch; }
        if ($op eq '*') { $res = $op1 * $op2; last switch; }
        if ($op eq '/') {
            if ($op2 == 0)
            {
                print " Divide by zero! </BODY></HTML>";
                exit(0);
            }
            else

```

```

        {
            $res = $op1 / $op2; last switch;
        }
    }

    {
        print "Operator ".$op." is not supported
</BODY></HTML>";
        exit(0);
    }
}

print "Result: ".$op1.$op.$op2.' = '.$res;
print "<p><a href= \"'".$ENV{'HTTP_REFERER'}.\"'>Back</a></p>";

print "</HTML></BODY>\n";

```

У даному прикладі з рядка запиту витягуються 3 параметри з іменами «А», «В» і «ор», після чого в залежності від значення третього параметра виконується відповідна арифметична операція над першими двома параметрами.

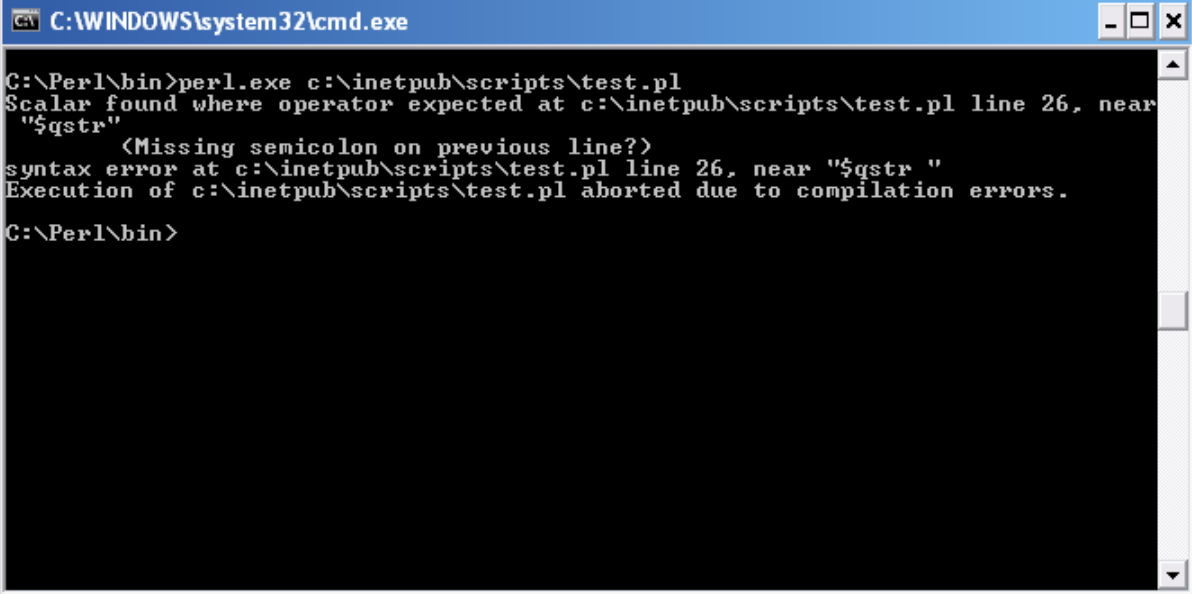
Operand1:	34
Operand2:	12
Operation:	+
Calculate!	

У результаті формується *html* сторінка з результатом обчислення і зворотнім посиланням на вихідну сторінку з формою. Для визначення адреси вихідної сторінки використовується значення змінної оточення *HTTP_REFERER*:

Result: 34*12 = 408

[Back](#)

Для пошуку синтаксичних помилок в сценаріях на Perl можна використовувати безпосередній запуск інтерпретатора в командному рядку, який знаходиться в установчому каталозі Perl:



```
C:\WINDOWS\system32\cmd.exe

C:\Perl\bin>perl.exe c:\inetpub\scripts\test.pl
Scalar found where operator expected at c:\inetpub\scripts\test.pl line 26, near
"$qstr"
    <Missing semicolon on previous line?>
syntax error at c:\inetpub\scripts\test.pl line 26, near "$qstr "
Execution of c:\inetpub\scripts\test.pl aborted due to compilation errors.

C:\Perl\bin>
```

2. Основи розробки сценаріїв на мові PHP

PHP – сценарії можуть розміщуватися в окремому файлі (з розширенням *.php*) або вбудовуються безпосередньо в HTML документ.

Існує кілька способів впровадження коду PHP в HTML документи:

- За допомогою відкриваючого тегу `<? Php` і закриваючого тегу `?>`.
- За допомогою коротких тегів `<? та ?>`. Дана можливість доступна тільки при спеціальному налаштуванні.
- За допомогою тегів `<script language = "php">` та `</script>`.

- Шляхом використання *echo* тегів в стилі ASP: `<% та %>`. Така можливість доступна при відповідному конфігураційному налаштуванні.

Надалі в прикладах буде використовуватися перший з варіантів впровадження PHP коду.

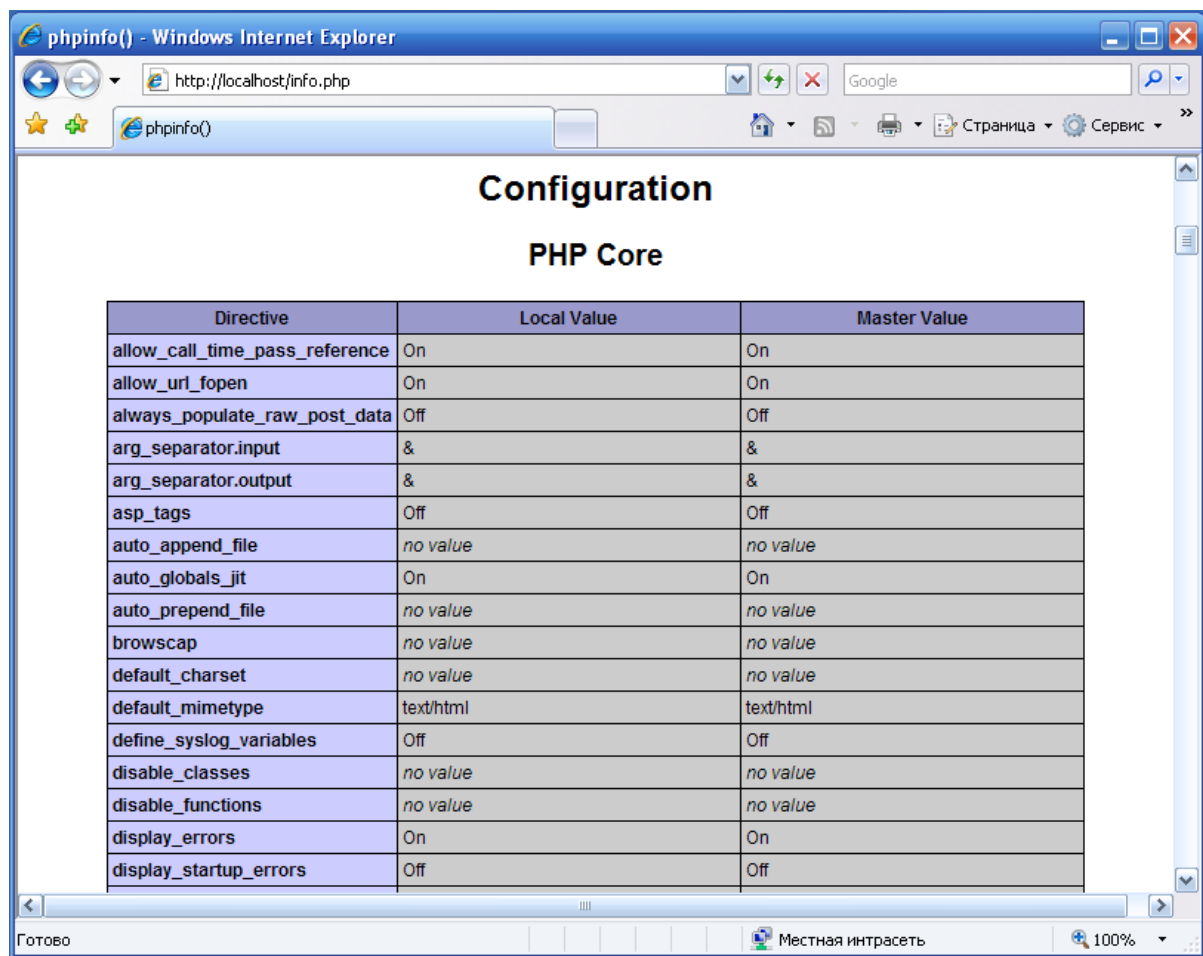
Код, який знаходиться всередині зазначених тегів, обробляється інтерпретатором PHP, весь інший код залишається незмінним.

Для того, щоб побачити поточні налаштування PHP, і для перевірки його працездатності корисно використовувати спеціальну функцію *phpinfo()*:

Сценарій 4. Використання *php*-функції *phpinfo()*.

```
<?php  
    phpinfo();  
?>
```

Після виконання цього коду, в веб-браузері можна буде побачити приблизно наступне (показана невелика частина):



Directive	Local Value	Master Value
allow_call_time_pass_reference	On	On
allow_url_fopen	On	On
always_populate_raw_post_data	Off	Off
arg_separator.input	&	&
arg_separator.output	&	&
asp_tags	Off	Off
auto_append_file	no value	no value
auto_globals_jit	On	On
auto_prepend_file	no value	no value
browscap	no value	no value
default_charset	no value	no value
default_mimetype	text/html	text/html
define_syslog_variables	Off	Off
disable_classes	no value	no value
disable_functions	no value	no value
display_errors	On	On
display_startup_errors	Off	Off

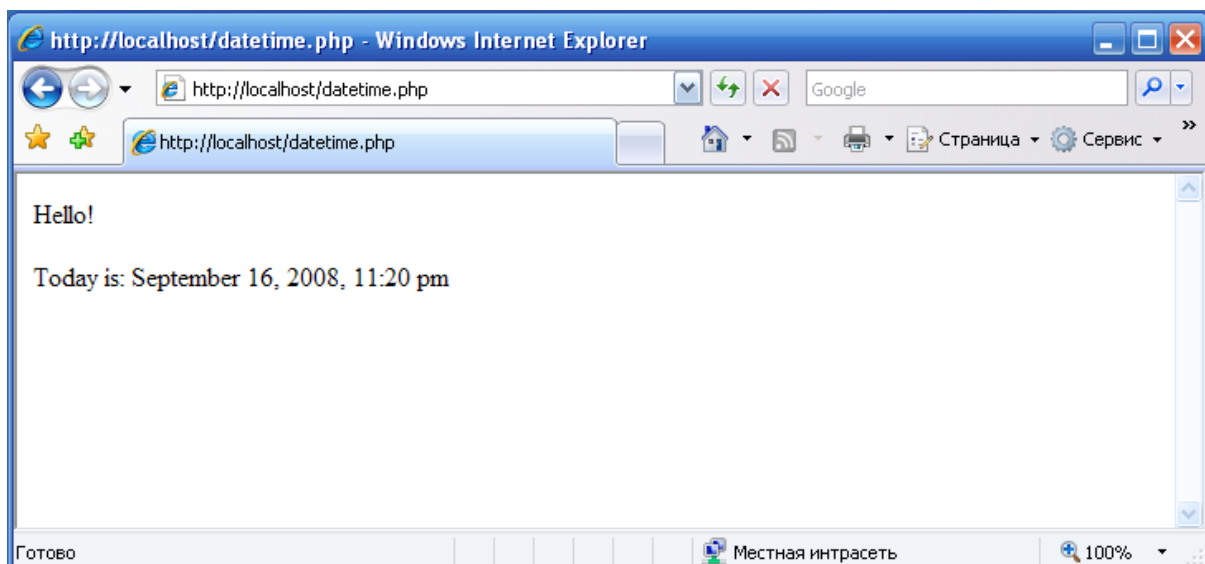
В цілому ж лістинг містить інформацію про встановлені опції і розширення PHP, версії PHP, інформацію про веб-сервер і змінних оточення, інформацію про версію ОС, шляхах, налаштуваннях конфігураційних змінних, полях заголовка HTTP і PHP ліцензії.

Наступний приклад демонструє варіант з впровадженням PHP коду в HTML:

Сценарій 5. Впровадження PHP коду в HTML документ.

```
<html>
<body>
<p>Hello! </p>
<p>Today is:
<?php
    $today = date("F j, Y, g:i a");
    echo($today);
?>
</p>
</body>
</html>
```

Результат обробки цього документа представлений нижче:



Одним з головних завдань, що вирішуються за допомогою PHP, є обробка даних, одержуваних від користувача через веб-форми. Розглянемо, яким чином в PHP реалізується така обробка.

Сценарій 6. Виведення списку параметрів, отриманих сервером в запиті від клієнта.

```
<?php
    $method = $_SERVER["REQUEST_METHOD"];

    if ($method == "GET") $query = "_GET";
    elseif ($method == "PUT") $query = "_PUT";
    else die("$method is not supported!");

    print "<p><b>Method</b>: $method </p>";
    print "<p><u>Params:</u></p>";

    foreach ($$query as $name => $value)
    {
        print "<b>$name</b> = <i>$value</i> <br>";
    }
?>
```

У першому рядку сценарію за допомогою змінної оточення `REQUEST_METHOD` з глобальної асоціативного масиву `$_SERVER` визначається метод передачі даних в запиті клієнта. Залежно від обраного методу передані дані будуть вилучатись або з глобальної змінної `$_GET` або з `$_PUT`. Якщо метод відрізняється від `GET` або `PUT`, або невизначений, то відбувається примусове завершення роботи сценарію з видачею повідомлення через виклик функції `die()`. Загалом, можна також використовувати глобальний масив `$_REQUEST`, що містить усередині себе масиви `$_GET`, `$_POST` і `$_COOKIE`, що дозволяє позбавитися від перевірки методу передачі.

Конструкція виду `$$query` демонструє непряме звернення до змінної, тобто змінна `$query` містить ідентифікатор іншої змінної, і для звернення до неї необхідно додати ще один знак `$`.

Обидві змінні `$_GET` і `$_POST` є асоціативними масивами, тому для перебору елементів був використаний спеціальний оператор:

foreach (ім'я_масиву **as** ключ => значення)

Слід звернути увагу на те, що всередину строкових констант, обмежених символами “ ” можна вставляти змінні. Після обробки такого рядка інтерпретатором замість змінної вставляється її фактичне значення. Також для конкатенації рядків можна використовувати оператор ‘.’ В цілому можна сказати, що в PHP можна використовувати оператори розгалуження, вибору і циклів аналогічні тим, що використовуються в мові C.

Код сценарію, що реалізовує чотири арифметичні операції над цілими числами представлений в лістингу нижче:

Сценарій 7. *Калькулятор арифметичних операцій для цілих операндів.*

```
<?php
    $method = $_SERVER["REQUEST_METHOD"];

    if ($method == "GET") $query = "_GET";
    elseif ($method == "PUT") $query = "_PUT";
    else die("$method is not supported!");

    $q = $$query;

    $a = $q["A"];
    $b = $q["B"];
    $op = $q["op"];

    switch ($op)
    {
        case '+': $result = (int)$a + (int)$b; break;
        case '-': $result = (int)$a - (int)$b; break;
        case '*': $result = (int)$a * (int)$b; break;
        case '/': {
            if ($b == '0') die("divide by zero!");
        }
        default: die("operator $op is not defined");
    }
}
```

```
print "<p>Result: $a $op $b = $result </p>";  
print  
href="$_SERVER['HTTP_REFERER'].">Back</a></p>";  
?>
```

У PHP є широкий діапазон функцій для роботи з файлами. Наприклад, наступний приклад демонструє читання файлу, в якому знаходиться виконуваний PHP код:

Сценарій 8. Читання файлу, що містить PHP код, виконуваного сценарію.

```
<?php  
$fh = fopen("read.php", "r");  
if (!$fh) die("Cannot open file");  
  
while (!feof($fh))  
{  
    $line = fgets($fh);  
    echo $line, "<br>";  
}  
fclose($fh);  
?>
```

У даному прикладі використовуються практичні ті ж самі функції, що і в мові C:

- **fopen** (шлях_до_файлу, тип_доступу) – відкриття файлу;
- **feof** (указатель_на_файл) – перевірка на наявність ознаки кінця файлу;
- **fgets** (указатель_на_файл) – читання рядка з файлу;
- **fclose** (указатель_на_файл) – закриття файлу.

Для запису даних у файл можна використовувати функцію **fputs** (указатель_на_файл, рядок). У наступному прикладі згенеровані функцією *rand()* псевдовипадкові числа зберігаються у файлі *rand.dat*:

Сценарій 9. Збереження у файлі послідовності псевдовипадкових чисел.

```
<?php
    $n = 10;

    $fh = fopen("rand.dat", "w");

    if (!$fh) die("Cannot open file");

    srand();
    for ($i=0; $i < $n; $i++)
    {
        $d = rand(0,100);
        fputs($fh, "$d\n");
    }
    fclose($fh);
?>
```

Перед виконанням даного сценарію слід правильно налаштувати права доступу для веб-сервера до директорії, в якій зберігається файл *rand.dat*.

Порядок виконання роботи

Для практичного вивчення прикладів в даній лабораторній роботі необхідна наявність встановлених і правильно сконфігурованих інтерпретаторів мов *Perl* і *PHP*.

Частина 1. Мова Perl.

1. Створіть файл з текстом *сценарію 1*. Файл повинен мати розширення *pl*, і розміщуватися в директорії *Scripts* (або *cgi-bin*). Перевірте налаштування доступу до папки для веб-сервера, який повинен мати право на виконання сценаріїв в цій папці.

При збереженні файлу в редакторі також слід також вибрати правильне кодування символу переносу рядка. Крім того, перший рядок сценарію повинен містити правильний шлях до директорії, в якій встановлено інтерпретатор мови Perl (зазвичай це файл *perl.exe* або подібний йому).

Якщо текст сценарію містить синтаксичні помилки, то після спроби його запустити на виконання веб-сервером, останній поверне клієнтові відповідь, що містить код внутрішньої помилки сервера. Тому перед запуском сценарію рекомендується виконати його перевірку. Для перевірки сценарію на наявність синтаксичних помилок зручно використовувати безпосередній запуск інтерпретатора вручну. Для цього в командному рядку з директорії, зазначеної в першому рядку сценарію, необхідно запустити виконуваний модуль (зазвичай *perl.exe*) з аргументом, що є ім'ям файлу (із зазначенням шляху), що містить текст сценарію. При наявності синтаксичних помилок в сценарії інтерпретатор видасть повідомлення із зазначенням відповідних номерів рядків у файлі, в яких ці помилки виявлені. Виправляйте помилки доти, поки інтерпретатор не перестане видавати повідомлення про помилки.

Для того, щоб подивитися роботу сценарію, необхідно в браузері набрати його URL по HTTP-протоколу.

2. Підготуйте файл з текстом сценарію 2.

Для перевірки працездатності даного сценарію можна виконати його безпосередній запуск у веб-браузері через його URL з додаванням рядка параметрів, наприклад:

<http://localhost/Scripts/test.pl?a=2&b=14>

Підготуйте HTML сторінку, що містить форму з полями для введення даних. Вставте в тегу `<FORM>` атрибут `ACTION` зі значенням, рівним URL сценарію, як метод виконання запиту вкажіть в атрибуті `METHOD` значення `GET`. Перевірте роботу форми. Зробіть те ж саме, але для методу `POST`.

3. Підготуйте файл з текстом сценарію 3 і HTML сторінку, що містить форму з полями для введення операндів (прості поля для введення тексту з іменами «A» і «B») і вибору арифметичної операції (поле типу «*select*» з ім'ям «*op*»). Додайте кнопку типу «*submit*» і атрибут `ACTION` зі значенням, рівним URL сценарію, в тегу `<FORM>`:

```
<html>
<body>
  <form action='http://localhost/Scripts/test.pl'>
    <p>Operand1: <input type='text' name='A'></p>
    <p>Operand2: <input type='text' name='B'></p>
    <p>Operation:<br>
    <select name='op'>
      <option value='+'>+</option>
      <option value='-'>-</option>
      <option value='*'>*</option>
      <option value='/'>/</option>
    </select></p>
    <input type='submit' value='Calculate!'>
  </form>
</body>
</html>
```

Перевірте роботу сценарію.

Частина 2. Мова PHP.

1. Підготуйте текстовий файл з розширенням *RHP* і розмістіть його в директорії відповідно до конфігураційними налаштуванням інтерпретатора *RHP* (параметр *doc_root*).

Після запуску сценарію ви побачите сторінку конфігурації, в якій ви побачите значення змінних ядра *RHP*, встановлені бібліотеки функцій, значення змінних оточення і глобальних змінних *RHP*.

Даний сценарій можна використовувати для перевірки списку переданих даних від клієнта, якщо URL сценарію використовувати у формі (атрибут *ACTION*) або викликати його безпосередньо у веб-браузері.

2. Підготуйте файл з текстом сценарію 6 і HTML сторінку з формою для перевірки його роботи.

Перевірте роботу сценарію для методів *GET* і *POST*.

3. Підготуйте файл з текстом сценарію 7 і відповідну HTML сторінку з формою, як в аналогічному прикладі мовою *Perl*.

4. Підготуйте файл з текстом сценарію 8. Для першого аргументу функції *forep* вкажіть ім'я цього файлу.

Перевірте роботу сценарію.

5. Підготуйте файл з текстом сценарію 9. Для виконання сценарію необхідно правильно налаштувати права доступу для веб-сервера до директорії, в якій буде зберігатися файл (має бути дозволений доступ на запис). З міркувань безпеки рекомендується для запису створювати окрему директорію.

Запустіть сценарій. Переконайтеся, що був створений файл *rand.dat* і перегляньте його вміст.

Контрольні завдання

1. Витяг списку слів з тексту (сценарій на мові *Perl*).

1) Підготуйте веб-сторінку з формою, що містить поле для введення тексту і кнопку типу *submit*. Для атрибута

Action у формі вкажіть як значення URL perl-сценарію, наприклад *http://localhost/Scripts/wcount.pl*.

2) Підготуйте сценарій на мові Perl, який витягує з тексту, отриманого від клієнтського додатка, список слів за допомогою функції *split*. Як перший аргумент при виклику функції необхідно вказати регулярний вираз, що задає список роздільників слів. Також сценарій повинен показати загальне число знайдених слів.

2. Витяг списку слів з тексту (сценарій на мові PHP).

1) Змініть веб-сторінку з формою, розроблену в попередньому завданні: для атрибута *Action* у формі вкажіть як значення URL PHP-сценарію, наприклад *http://localhost/wcount.php*.

2) Підготуйте сценарій на мові PHP, який витягує з тексту, отриманого від клієнтського додатка, список слів за допомогою функції *preg_split*, який має наступний синтаксис:

```
array preg_split(string pattern, string subject)
```

тобто повертає масив, що складається з підрядків заданого рядка *subject*, який розбитий на межі, відповідним шаблоном *pattern*. Шаблон описується за допомогою відповідного регулярного виразу. Також сценарій повинен показати загальне число знайдених слів.

Лабораторна робота №6.

Тема «Знайомство з середовищем розробки Microsoft Visual Studio.NET. Структура програми на C #. Основи мови C #.»

Мета роботи:

1. Знайомство з середовищем розробки додатків Microsoft Visual Studio.NET.
2. Знайомство зі структурою програми мовою C # (для консольного застосування).

Теоретичний матеріал

1. Загальні відомості.

Microsoft Visual Studio.NET – це інтегроване середовище розробки для створення, документування, запуску та налагодження програм, написаних на мовах .NET.

Це середовище розробки є відкритим мовним середовищем. Поряд з мовами програмування, що з самого спочатку включені в середовище – C++, C #, J#, *Visual Basic*, – в неї можуть додаватися будь-які мови програмування, компілятори яких створюються сторонніми розробниками. Необхідною умовою для включення мов в середовище **Visual Studio.NET** є використання єдиного каркаса - платформи *Framework.Net*.

Платформа *Framework.NET* дозволяє:

- легко використовувати компоненти, які були розроблені на різних мовах;
- розробляти єдиний додаток з декількох частин на різних мовах.

Для розробки додатків в Visual Studio.NET використовуються *проекти*.

Проект (Project) – це основна одиниця, з якою має справу розробник. Спочатку він повинен вибрати тип проекту, після чого Visual Studio створює каркас проекту відповідно до обраного типу.

Проект складається з класів, зібраних в одному або декількох *просторах імен*. *Простори імен* (Namespaces) дозволяють структурувати проекти, що містять велику кількість класів, об'єднуючи в одну групу близькі класи.

Кілька проектів можуть об'єднуватися в *рішення* (Solution), яке також може включати ресурси, необхідні цим проектам.

З точки зору розробника кінцевим результатом його роботи, одержуваних після компіляції вихідного програмного коду, є *рішення*, а з точки зору CLR (Common Language Runtime – загальномовного середовища виконання) – *збірка* (assembly), що містить PE файл, тобто модуль у форматі виконуваного файлу PE (Portable Executable) для 32-розрядної ОС Windows або DLL (Dynamic Link Library) файл.

Visual Studio.Net пропонує велику різноманітність можливих типів проектів. Для ознайомлення з основами мови C# в основному будуть використані консольні додатки без залучення звичних в Windows можливостей для користувача інтерфейсу.

У C# немає глобальних функцій, тому за замовчуванням при створенні консольного застосування оголошується клас *Program*, який містить функцію *static Main()*, що служить початковою точкою виконання програми. Функція *Main* може бути оголошена без параметрів або з параметром, що представляє собою масив рядків.

Також програма на C# може містити *коментарі* і *атрибути*.

У C# використовуються однорядкові і багаторядкові коментарі. Однорядковий коментар починається з пари

символів «//», і весь текст до кінця рядка є коментарем. Багаторядковий коментар починається з пари символів /* і закінчується */.

Атрибут – засіб додавання декларативною інформації до елементів програмного коду. Призначення атрибутів: організація взаємодії між програмними модулями, додаткова інформація про умови виконання коду, управління серіалізацією (тобто правила збереження інформації), налагодження, та ін. Атрибути можуть бути як стандартними, так і заданими користувачем. Стандартні атрибути використовуються CLR і впливають на те, як виконуватиметься проект. Атрибут розміщується в дужках «[]».

2. Типи даних C#.

Стандарт мови C++ включає наступний набір фундаментальних типів.

- Логічний тип (*bool*).
- Символьний тип (*char*).
- Цілі типи. Вони можуть відрізнятися розміром: *short*, *int*, *long*, а також можуть бути знаковими (*signed*) або без знаковими (*unsigned*).
- Типи з плаваючою крапкою. Вони також можуть відрізнятися розмірами: *float*, *double* і *long double*.

Тип *void* вказує на відсутність інформації.

До конструйованих типів відносяться наступні:

- Показчики (наприклад, *char**).
- Посилання (наприклад, *char&*).
- Масиви (наприклад, *char[]*).

Також мова дозволяє розробнику конструювати власні типи:

- Перелічуваних типи (*enum*).
- Структури (*struct*).
- Класи.

У мові C# всі типи можна розглядати і під іншим ракурсом, розділивши їх на чотири категорії:

- Типи-значення (*value*).
- Посилальні (*reference*).
- Показчики (*pointer*).
- Тип *void*.

Для посилального типу значення задає посилання на область пам'яті в «купі» (*heap*), де розташований відповідний об'єкт. Для типу-значення значенням є власне дані, а пам'ять для них виділяється в стек.

Логічний, арифметичний, структури, перерахування відносяться *типам-значень*. Масиви, рядки і класи відносяться до *посилальним* типам.

І посилальні, і звичайні типи є похідними від базового класу *object*. У тих випадках, коли звичайний тип повинен вести себе як об'єкт, створюється оболонка (*wrapper*), яку можна розглядати як контрольний об'єкт, поміщений в купу, і в неї копіюється значення змінної звичайного типу. Оболонка автоматично позначається таким чином, що система знає, яке значення вона містить. Цей процес називається *упаковкою* (*boxing*), а зворотний процес – *розпакуванням* (*unboxing*).

Упаковка відбувається автоматично, для цього потрібно лише привласнити значення звичайного типу змінної типу *object*. Упаковка та розпакування дозволяють обробляти будь-який тип як об'єкт. Наприклад, у виразі

7.ToString();

ціле число 7 упаковується шляхом виклику функції *Int32.ToString()*.

Масиви в C# можуть бути *багатомірними* (*multidimensional*) або *невирівняними* (*jagged*). Більш складні структури даних такі, як *стек* і *хеш-таблиця* визначені в просторі імен *System.Collections*.

3. Вирази та оператори C#.

Вираження будуються з операндів – констант, змінних, функцій, – об'єднаних знаками операцій і дужками. При обчисленні виразу визначається його значення і тип.

У таблиці нижче наведений список операцій C#.

Категорія операції	Операції
Арифметичні	+ - * / %
Логичні (<i>boolean</i> і побітові)	& ^ ! ~ &&
Рядкові	+
Інкремент і декремент	++ --
Здвиг	>> <<
Порівняння	== != < > <= >=
Присвоєння	= += -= *= /= %= &= = ^= <<= >>=
Звернення до члену класу	.
Індексація	[]
Приведення типу (Cast)	()
Умова	?:
Створення об'єкту	new()
Інформація про тип	is sizeof typeof
Управління винятками	checked unchecked
Непрямі і адресації	* -> [] &

Ім'я та *тип* змінної задаються при її оголошенні і залишаються незмінними протягом усього часу її життя. Особливістю мови C# є вимога *обов'язкової ініціалізації* змінної до початку її використання. Спроба використати неініціалізовану змінну призводить до помилок, що виявляється ще на етапі компіляції.

По використовуваних виразах і операторам, C# схожий на C++. Так в програмах на C# використовуються такі оператори як:

- Оператор присвоювання (=).
- Складовий оператор ({}).

- Оператори вибору: *if-else* і *switch*.
- Оператори циклу: *for*, *while*, *оператор*.
- Оператори *break* і *continue*.
- Оператор *return*.
- Оператор переходу *goto*

Крім того, введені кілька нових інструкцій.

Наприклад, оператор *foreach* дозволяє отримати доступ до всіх елементів масиву чи колекції по черзі, в порядку зростання індексів. Його синтаксис:

foreach (*тип ідентифікатор in контейнер*) оператор

У С# процедури і функції існують тільки як методи деякого класу, вони не визначені поза класом. Роль бібліотек процедур і функцій виконують бібліотеки класів. Бібліотека класів *Framework Class Library* (FCL), доступна в мові С#, істотно розширює можливості мови.

4. Класи і методи в С#.

У Visual Studio.Net, і в С# зокрема, будь-яка програмна система розглядається як сукупність класів, об'єднаних в проекти, простору імен, рішення.

Опис класу має наступний синтаксис:

[Атрибути] [модифікатори] **class** ім'я_класу
[:список_батьків] {тіло_класа}

У тілі класу можуть бути оголошені:

- константи;
- поля;
- конструктори і деструктори;
- методи;
- події;
- делегати;
- класи (структури, інтерфейси, перерахування).

Поля класу синтаксично є звичайними змінними (об'єктами) мови. Їх опис задовольняє звичайним правилам оголошення змінних. Поля характеризують *властивості об'єктів класу*.

Методи класу синтаксично є звичайними процедурами і функціями мови. Методи містять описи операцій, доступних над об'єктами класу. Методи, звані *властивостями* являються спеціальною синтаксичною конструкцією, призначеної для забезпечення ефективної роботи з класами.

Конструктор являє собою спеціальний метод класу, що дозволяє створювати об'єкти класу. Його ім'я повинне збігатися з ім'ям класу. Якщо розробник не визначає конструктор класу, то до класу автоматично додається конструктор за замовчуванням – конструктор без аргументів.

Делегат в С# є опис спеціального випадку класу і задає визначення функціонального типу (класу) даних. Екземплярами класу є функції. Кожен делегат описує безліч функцій із заданою сигнатурою. Кожна функція (метод), сигнатура якого збігається з сигнатурою делегата, може розглядатися як екземпляр класу, заданого делегатом. Синтаксис оголошення делегата має наступний вигляд:

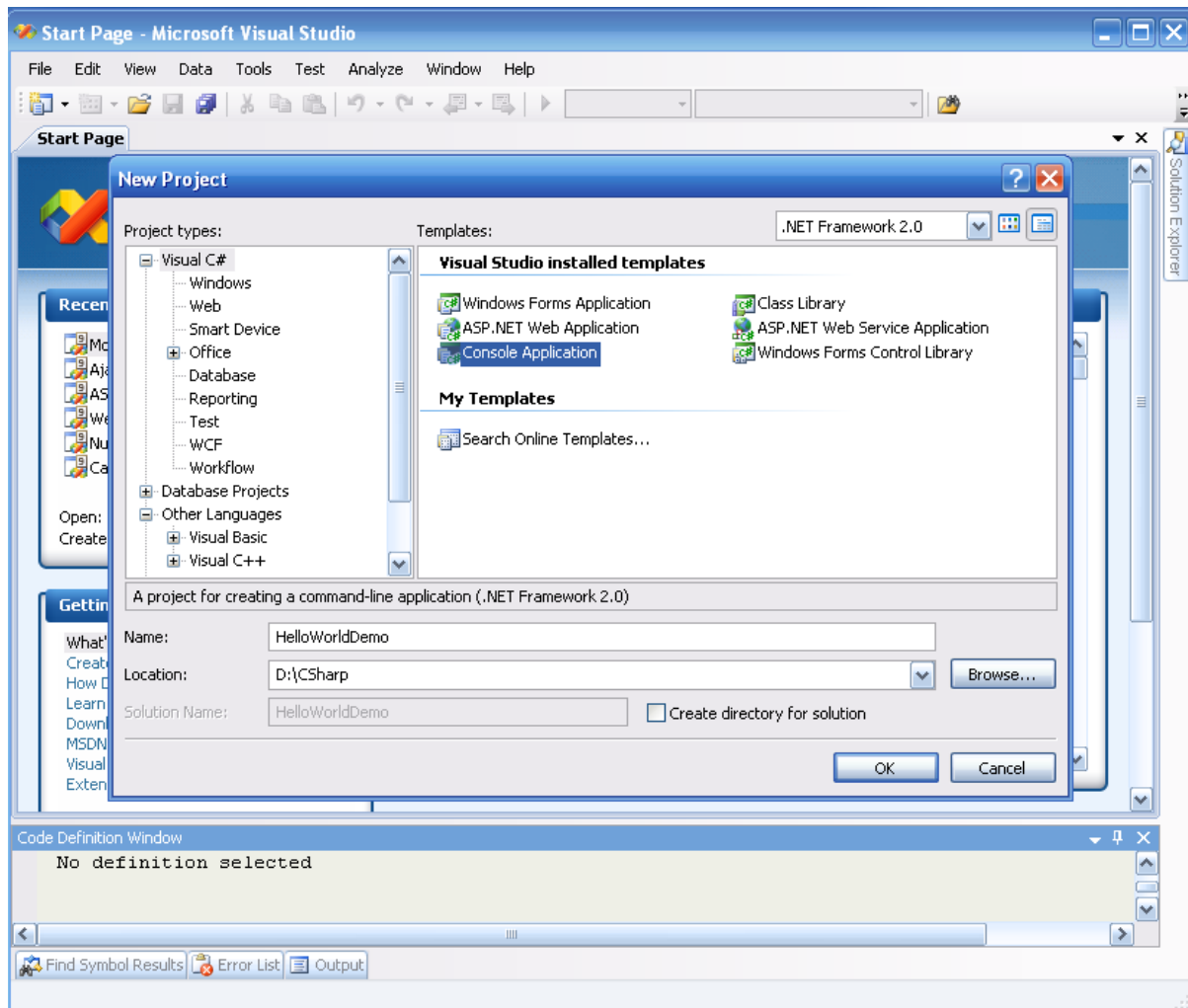
[<Специфікатор доступу>] *delegate* <тип результату>
<ім'я класу> (<список аргументів>);

Порядок виконання роботи

Для практичного вивчення прикладів в даній лабораторній роботі необхідно наявність встановленого пакета *Microsoft Visual Studio* (версії не раніше 2003). Залежно від встановленої версії можливі несуттєві відмінності в інтерфейсі середовища при створенні і роботі з проектом консольного застосування.

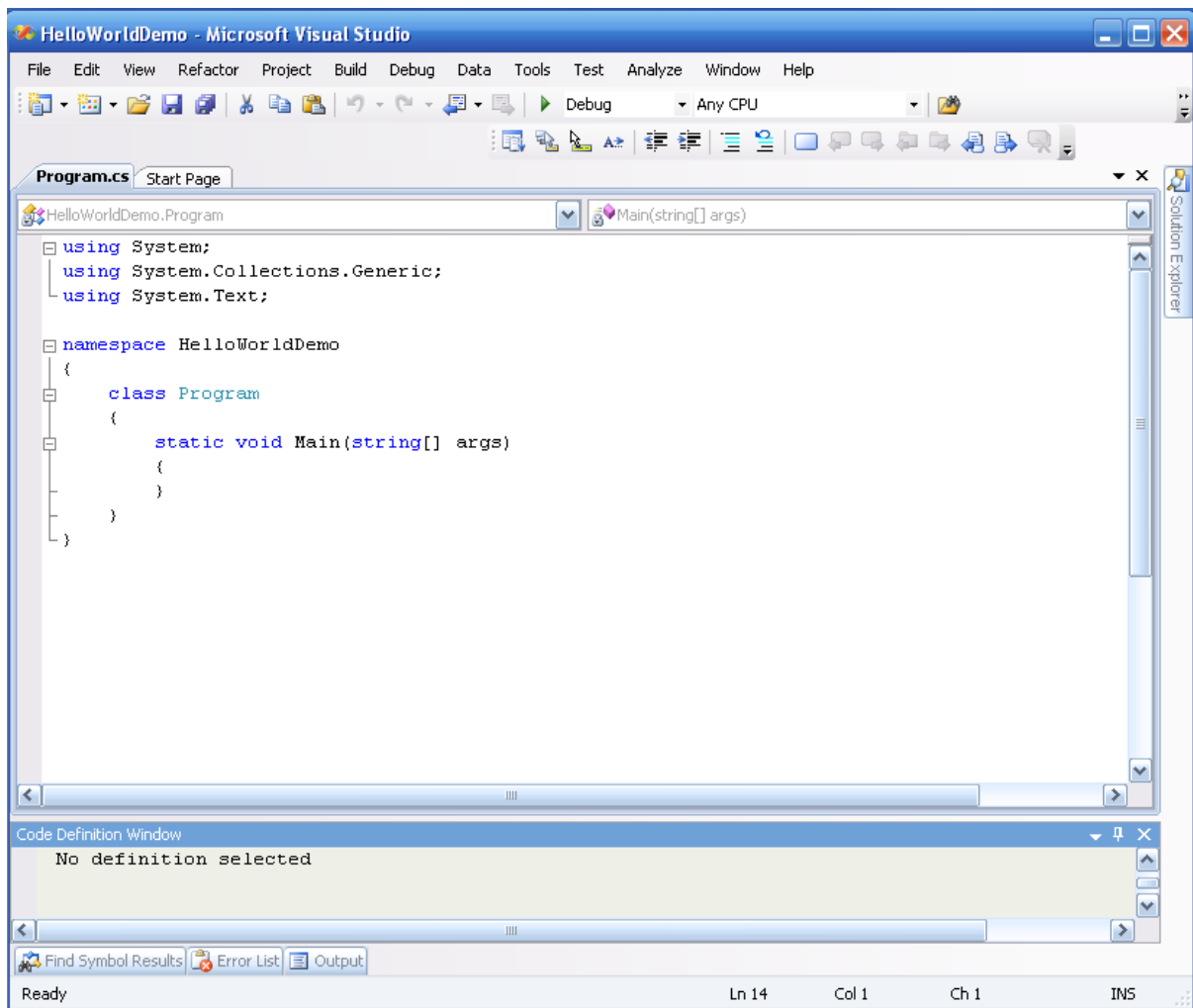
1. Створення проекту консольного застосування.

1) Перший крок при розробці будь-якої програми - створення проекту. Для цього через розділи меню: File> New> Project запускається панель створення нового проекту такого вигляду:



2) Зі списку пропонованих шаблонів необхідно вибрати проект консольного застосування (*Console Application*).

3) Після створення проекту має з'явитися таке вікно, що містить текст програми *Program.cs* мовою C#:



У даній програмі оголошення *using System* дає можливість посилатися на класи, які знаходяться в просторі імен *System*, так що їх можна використовувати, не додаючи «*System.*» Перед ім'ям типу.

Простір імен *System* містить багато корисних класів. Одним з них є і клас *Console*, який використовується при створенні консольних додатків.

Клас проекту – *Program* розміщений в просторі імен, що має за замовчуванням те ж саме ім'я (*HelloWorldDemo*), що і рішення, що містить в даному прикладі єдиний проект.

Оскільки в C# немає глобальних функцій, тому в даному прикладі оголошується клас *Program*, який містить функцію *static Main()*, що служить початковою точкою виконання програми. Функція *Main* може бути оголошена без параметрів або з параметром, що представляє собою

масив рядків. Так як функція *Main* є точкою входу, вона повинна бути *статичною* (static) функцією, тобто вона не пов'язана з конкретним об'єктом класу, в якому оголошена.

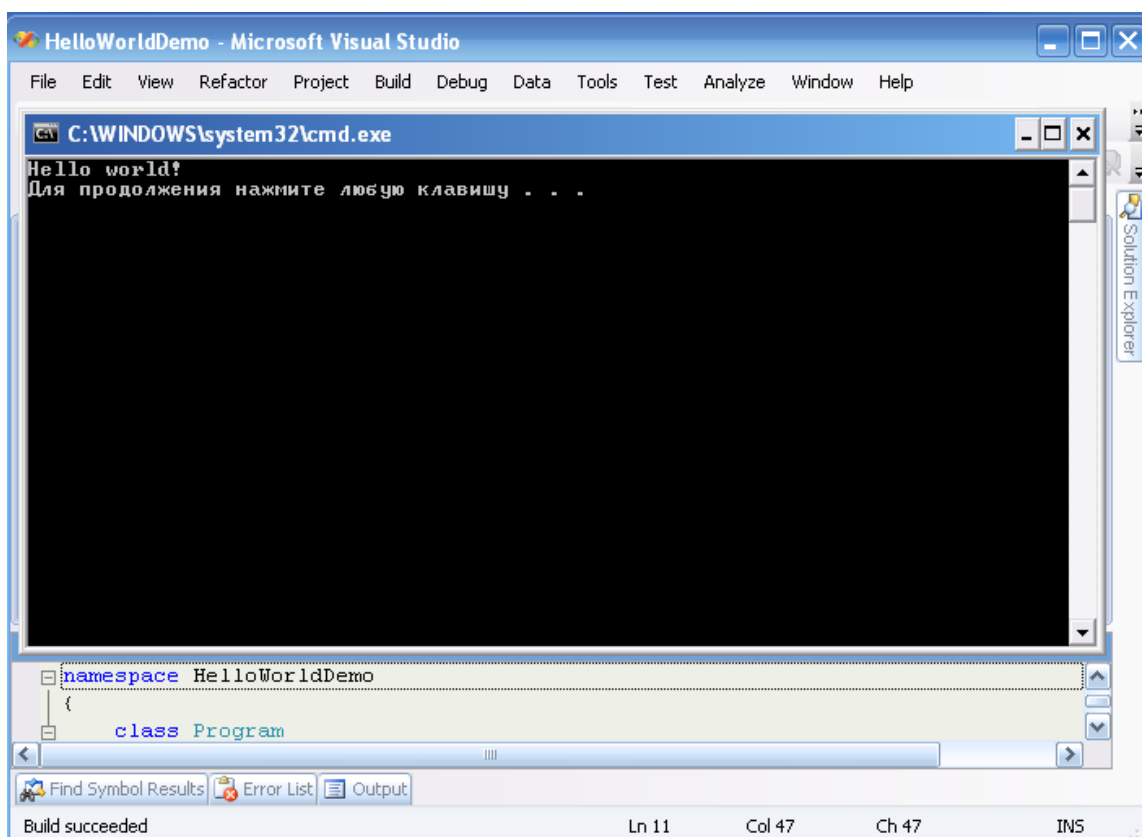
2. Додамо всередині функції *Main* наступний рядок:

```
Console.WriteLine("Hello world!");
```

який використовує метод *WriteLine* класу *Console* для виведення рядка «*Hello world!*».

3. Компіляція проекту. Можна використовувати або меню (Build> Build *Ім'я проекту*) або поєднання клавіш <Shift + F6>. При наявності повідомлень про помилки необхідно їх виправити і скомпілювати проект знову.

4. Запуск програми. Можна використовувати або меню (Debug> Start Without Debugging) або поєднання клавіш <Ctrl + F5>. В результаті отримаємо наступне консольне вікно:

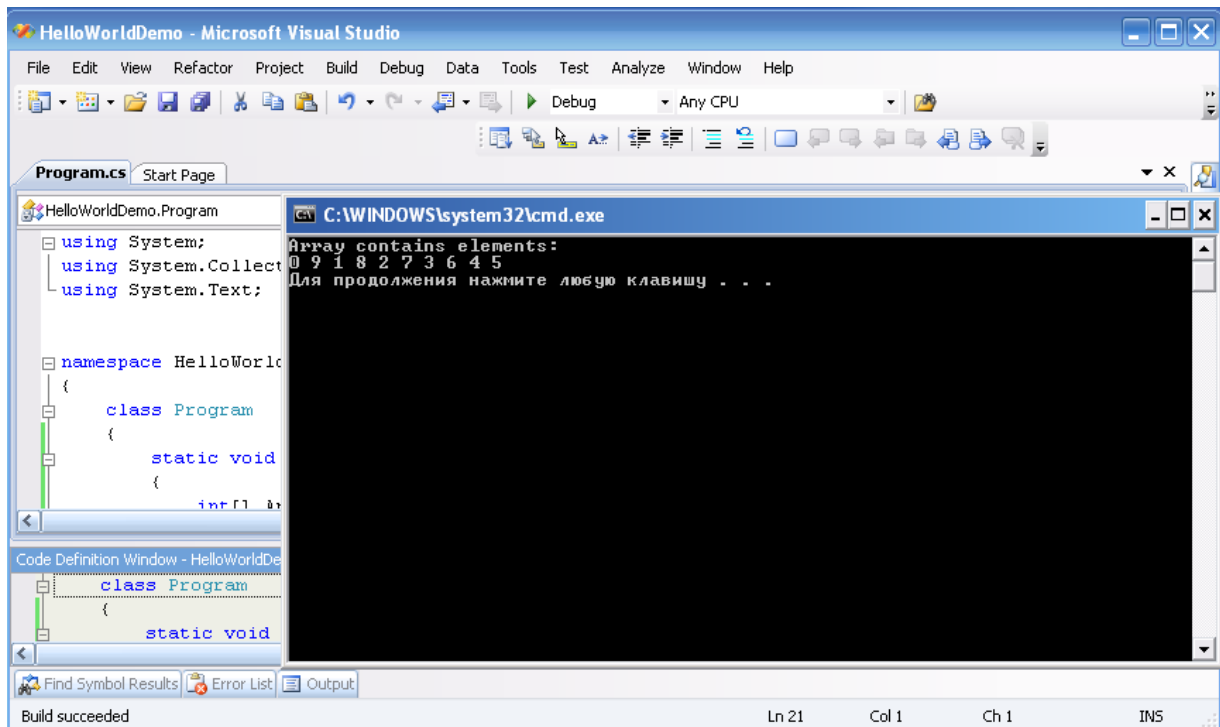


5. Ускладнимо додаток. Для цього додамо у функцію *Main* наступний код для виведення в консольне вікно змісту масиву цілих чисел:

```
static void Main(string[] args)
{
    int[] ArInt = { 0, 9, 1, 8, 2, 7, 3, 6, 4, 5 };

    Console.WriteLine("Array contains elements:");
    foreach (int item in ArInt)
    {
        Console.Write("{0} ", item);
    }
    Console.WriteLine("");
}
```

Після повторення кроків 3-4 отримаємо наступне вікно:



Контрольне завдання

Розробка консольного застосування для обчислення коренів квадратного рівняння.

Потрібно розробити програму, що за заданим значенням коефіцієнтів a , b і c квадратного рівняння (значення вводяться з клавіатури користувачем) обчислює і відображає на екрані корені рівняння.

Для цього додатка потрібно наступні методи:

- `string Console.ReadLine()` – читання рядка символів з вхідного потоку.
- `double Convert.ToDouble(string)` – перетворення рядка символів в число з плаваючою комою подвійної точності.
- `double Math.Sqrt(double)` – витяг квадратного кореня числа.

Лабораторна робота №7.

Тема «Основи мови С#. Робота з масивами і рядками. Інтерфейси і колекції.»

Мета роботи:

1. Продовження знайомства із середовищем розробки додатків Microsoft Visual Studio.NET.
2. Вивчення особливостей роботи з масивами і рядками в С#.
3. Ознайомлення з реалізацією інтерфейсами в С#.

Теоретичний матеріал

1. Масиви

Мова С# надає засоби для роботи з одновимірними масивами, масивами масивів і багатовимірними масивами.

Нижче представлені способи оголошення посилань на масиви різної розмірності і конфігурації:

```
// Оголошення масиву розмірності 3 з елементів типу int.  
// Фактично це масив складових, що представляють собою масиви  
// розмірності 2 одновимірних масивів елементів типу int.  
// Розміри всіх складових масивів одного рівня рівні  
// між собою ("прямокутний" масив).  
  
int[, ,] ar1;  
  
// Оголошення одновимірного масиву з  
// одновимірних масивів елементів, кожен з яких є  
// одновимірним масивом елементів типу int.  
  
int[][][] ar2;  
  
// Оголошення одновимірного масиву складових, кожен з яких є
```

```
// двовимірним масивом масивів елементів типу int.  
// При цьому у всіх складових можуть бути різні розміри  
  
int[,] ar3;  
  
// Оголошення двовимірного масиву складових, кожна з яких є  
// одновимірним масивом елементів типу int.  
// При цьому у всіх складових можуть бути різні розміри.  
  
int[,][] ar4;
```

Розглянуті приклади демонструють визначення двох різних категорії масивів:

- прямокутні масиви;
- ламані (*jagged*) масиви.

Оголошення масиву може бути поєднане з ініціалізацією. При оголошенні з відкладеним ініціалізацією сам *масив* не формується, а створюється тільки посилання на *масив*, що має невизначене значення *Null*. Тому поки *масив* не буде реально створений і його елементи ініціалізовані, використовувати його не можна.

Можливі два варіанти оголошення *масиву* поєднаного з ініціалізацією.

У першому випадку ініціалізація є явною і задається *списком констант*, наприклад:

```
int[] odd = {1, 3, 5, 7, 9};
```

У другому випадку створення і ініціалізація масиву реалізується разом з викликом конструктора *масиву*. Наприклад:

```
int[] even = new int[5];
```

Всі масиви є нащадками класу *System.Array*, який у свою чергу успадковує ряд інтерфейсів: *ICloneable*, *ICollection*,

ICollection, *IEnumerable*, а, отже, реалізує всі їхні методи і властивості. Крім того, клас *Array* має велике число власних методів і властивостей.

Завдяки спадкоємства від класу *System.Array*, для масивів визначені такі вбудовані методи копіювання, пошуку, звернення, сортування. Масиви можна розглядати також як колекції і використовувати цикл *foreach* для перебору його елементів.

Нижче наведені основні властивості і методи класу *System.Array*:

Властивість	Опис
IsFixedSize	Повертає <i>true</i> , якщо масив є статичним
IsReadOnly	Для всіх масивів повертає значення <i>false</i>
IsSynchronized	Повертає <i>true</i> або <i>false</i> , залежно від того, чи встановлена синхронізація доступу для масиву
Length	Число елементів у масиві
Rank	Розмірність масиву

Метод	Опис
BinarySearch	Бінарний пошук елементів
Clear	Виконує ініціалізацію елементів. Числові елементи стають нульовими, а рядкові приймають значення <i>Null</i>
Clone	Клонування масиву
Copy	Копіювання частини або всього масиву в інший масив
CopyTo	Копіюються всі елементи одновимірного масиву в інший одновимірний масив, починаючи з заданого індексу
CreateInstance	Створення примірника масиву
GetEnumerator	Повертає інтерфейс <i>IEnumerator</i> . Необхідний для використання в циклі foreach
GetLength	Повертає число елементів масиву за вказаною вимірюванням
GetLowerBound, GetUpperBound	Повертає нижню і верхню межу за вказаною виміру. Для масивів нижня межа завжди дорівнює нулю.

GetValue etValue	Повертає або встановлює значення елемента масиву із зазначеними індексами.
IndexOf	Індекс першого входження зразка в масив
LastIndexOf	Індекс останнього входження зразка в масив
Reverse	Сортування одновимірному масиву в зворотному порядку
Sort	Сортування масиву

2. Рядки

У мові C # визначено клас *char[]*, і його можна використовувати для представлення рядків постійної довжини. Однак масив *char[]* – це звичайний масив, тому його не можна ініціалізувати рядком символів. У C# не визначене перетворення з класу *char[]* в клас *String*. У *String* є динамічний метод *ToCharArray*, що задає подібне перетворення в *char[]*.

Об'єкти класу *String* оголошуються з явною або відкладеною ініціалізацією, з явним або неявним викликом конструктора класу. Найчастіше, при оголошенні строкової змінної конструктор явно не викликається, а ініціалізація задається строковою константою. У класі *String* досить багато конструкторів, які дозволяють сконструювати рядок з:

- символу, повторюваного вказане число разів;
- масиву символів *char[]*;
- частини масиву символів.

Методи класу *String* дозволяють виконувати вставку, видалення, заміну, пошук входження підрядка в рядок. Клас *String* успадковує методи класу *Object*, а також методи чотирьох інтерфейсів: *Comparable*, *Cloneable*, *Convertible*, *Enumerable*.

Деякі з членів класу наведені в таблиці:

Метод	Опис
Length	Ця властивість повертає довжину зазначеного

	рядка
Insert	Вставляє підрядок в задану позицію
Remove	Видаляє підстроку в заданій позиції
Replace	Замінює підрядок в заданій позиції на нову підрядок
Substring	Виділяє підстроку в заданій позиції
IndexOf, IndexOfAny, LastIndexOf, LastIndexOfAny	Визначаються індекси першого і останнього входження заданого підрядка або будь-якого символу із заданого набору
StartsWith, EndsWith	Повертається <i>true</i> або <i>false</i> , залежно від того, починається або закінчується рядок заданої підрядком
PadLeft, PadRight	Виконує заповнення потрібним числом прогалів на початку і в кінці рядка
Trim, TrimStart, TrimEnd	Видаляються пробіли на початку і в кінці рядка, або тільки з одного кінця
ToCharArray	Перетворення рядка в масив символів

Клас `String` не дозволяє змінювати існуючі об'єкти. Для цієї мети є інший клас – клас `StringBuilder`, який дозволяє виправити даний недолік. Цей клас належить до змінюваних класів і його можна знайти в просторі імен `System.Text`.

Об'єкти цього класу оголошуються з явним викликом конструктора класу. Конструктор класу перевантажений, і поряд з конструктором без параметрів, що створює порожній рядок, є набір конструкторів, яким можна передати дві групи параметрів:

- Перша група дозволяє задати *рядок* або *підрядок*, значенням якого буде ініціалізування створювання об'єкту класу `StringBuilder`.
- Друга група параметрів дозволяє задати розмір об'єкту, тобто обсяг пам'яті, що відводиться для екземпляру класу `StringBuilder`.

Приклади конструкторів:

- *public StringBuilder (string str, int size)*. Параметр *str* задає рядок для ініціалізації, *size* - розмір об'єкту;
- *public StringBuilder (int cursize, int maxsize)*. Параметри *cursize* і *maxsize* задають початковий і максимальний розмір об'єкту;
- *public StringBuilder (string str, int start, int len, int size)*. Параметри *str*, *start*, *len* задають рядок ініціалізації, *size* – розмір об'єкту.

Над рядками цього класу визначені (також як і для рядків класу *String*) операції:

- присвоювання (=);
- дві операції перевірки еквівалентності (==) і (!=);
- взяття індексу ([]).

Операція конкатенації (+) не визначена над рядками класу *StringBuilder*, її роль грає метод *Append*, який додає новий рядок в кінець існуючого.

3. Інтерфейси

Інтерфейс являє собою повністю абстрактний клас, всі методи якого абстрактні. Проте методи *інтерфейсу* оголошуються без вказівки модифікатора доступу, і клас, що успадковує *інтерфейс*, зобов'язаний повністю реалізувати всі методи *інтерфейсу*. У цьому – відмінність від класу, що посяде абстрактний клас, де нащадок може реалізувати лише деякі методи батьківського абстрактного класу, залишаючись абстрактним класом.

Інтерфейс дозволяє описувати деякі бажані властивості, якими можуть володіти об'єкти різних класів.

Серед інтерфейсів, вбудованих в бібліотеку базових класів .NET, можна особливо виділити такі як:

- *IEnumerable* (для роботи з наборами об'єктів, в т. ч. з використанням оператора *foreach*).
- *IClonable* (копіювання об'єктів).
- *IComparable* (для порівняння і сортування об'єктів).

Простір імен `System.Collections`, призначений для роботи з наборами об'єктів, підтримує інтерфейси:

- *ICollection* (визначає загальні характеристики класу набору елементів).
- *IComparer, IDictionary* (дозволяє представляти зміст об'єкту у вигляді пар ім'я-значення).
- *IDictionaryEnumerator* (нумерація змісту об'єкту, що підтримує *IDictionary*).
- *IEnumerable, IEnumerator*.
- *HashCodeProvider* (повертає хеш-код за допомогою обраного алгоритму хешування).
- *IList* (забезпечує методи додавання, видалення та індексування елементів у списку об'єктів).

Порядок виконання лабораторної роботи

Для вивчення приводяться в лабораторній роботі приклади, які необхідно створити в *Microsoft Visual Studio* проект консольного застосування і описати клас *Program*, використовуючи код, що приводиться в прикладах.

1. Робота з масивами.

Наступні дві програми демонструють, яким чином виконується ініціалізація і робота з прямокутними і ламаними масивами в C#.

1) Прямокутний масив.

```
using System;
using System.Collections.Generic;
using System.Text;

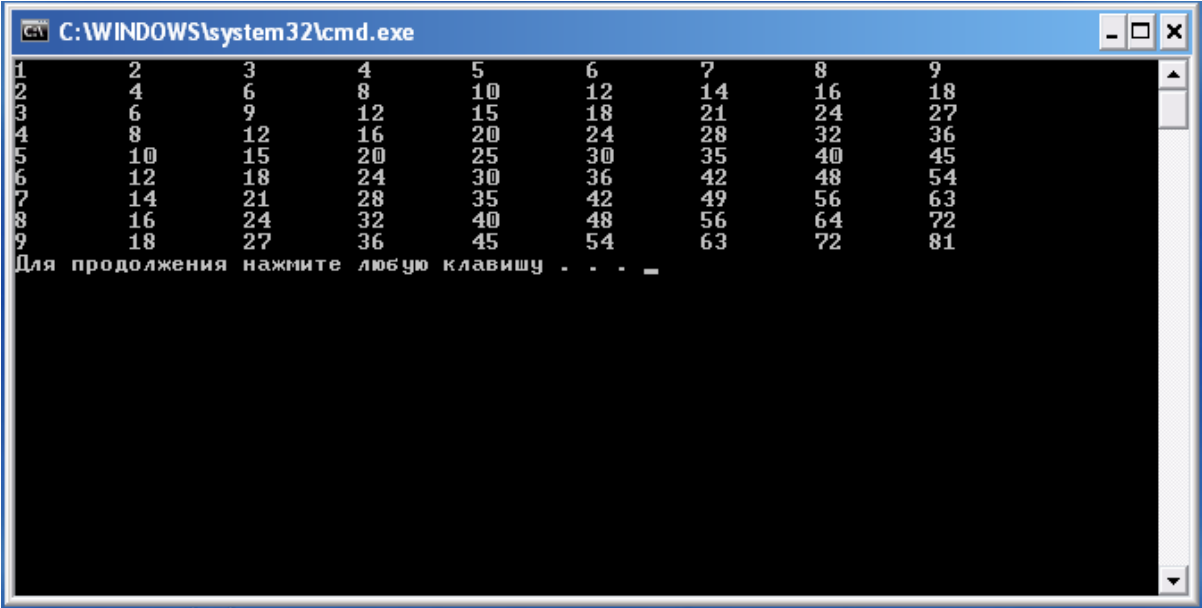
namespace HelloWorldDemo
{
    class Program
    {
        static void Main(string[] args)
        {
```

```
// Прямокутний багатовимірний масив
int[,]multMatr;
multMatr = new int[10,10];

// заповнення масиву 9 на 9:
for (int i = 1; i <= 9; i++)
    for (int j = 1; j < 9; j++)
        multMatr[i, j] = i * j;

// Виведення елементів багатовимірного масиву
for (int i = 1; i <= 9; i++)
{
    for (int j = 1; j <= 9; j++)
    {
        Console.Write(multMatr [i, j] + "\t");
    }
    Console.WriteLine();
}
}
```

В результаті виконання програми отримаємо таблицю множення такого вигляду:



The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The output displays a 9x9 multiplication table. The first 9 rows contain the products of integers from 1 to 9. The first column contains the integers 1 through 9. Below the table, the text "Для продолжения нажмите любую клавишу . . ." is displayed.

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

Для продолжения нажмите любую клавишу . . .

2) Ламаний масив.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace HelloWorldDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            // Ламаний багатовимірний масив з п'яти внутрішніх
            // масивів різного розміру
            int[][] JagArr = new int[10][];

            // Ініціалізація генератора випадкових чисел
            Random rand = new Random();

            // Динамічне створення ламаного масиву
            for (int i = 0; i < JagArr.Length; i++)
            {
                JagArr[i] = new int[i + rand.Next(10)];
            }

            // Виведення рядків на консоль
            // Кожен елемент за замовчуванням має значення,
            // рівне
            for (int i = 0; i < 10; i++)
            {
                // Властивість Length масиву повертає його розмір
                Console.WriteLine("Length of row {0} is {1}:\\t", i,
                    JagArr[i].Length);
                for (int j = 0; j < JagArr[i].Length; j++)
                {
                    Console.Write(JagArr[i][j] + " ");
                }
                Console.WriteLine();
            }
        }
    }
}
```

```

    }
}
}

```

В результаті виконання програми отримаємо наступну таблицю:

```

C:\WINDOWS\system32\cmd.exe
Length of row 0 is 0:
Length of row 1 is 9:  0 0 0 0 0 0 0 0 0
Length of row 2 is 11: 0 0 0 0 0 0 0 0 0 0 0
Length of row 3 is 12: 0 0 0 0 0 0 0 0 0 0 0 0
Length of row 4 is 7:  0 0 0 0 0 0 0
Length of row 5 is 7:  0 0 0 0 0 0 0
Length of row 6 is 10: 0 0 0 0 0 0 0 0 0 0
Length of row 7 is 16: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Length of row 8 is 12: 0 0 0 0 0 0 0 0 0 0 0 0
Length of row 9 is 15: 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Для продолжения нажмите любую клавишу . . . -

```

2. Властивості і методи класу *Array*.

Наступний приклад демонструє використання деяких з властивостей і методів класу *Array*:

```

using System;
using System.Collections.Generic;
using System.Text;

namespace HelloWorldDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            // Масив символічних рядків
            string[] Brands = new string[]

```



```
        {"Audi",  
        "BMW",  
        "Buick",  
        "Chevrolet",  
        "Citroen",  
        "Dodge",  
        "Ferrari",  
        "Fiat",  
        "Ford",  
        "Honda",  
        "Hyundai",  
        "Cherokee",  
        "Cherry",  
        "Lada",  
        "Lamborghini",  
        "Lincoln",  
        "Mazda",  
        "Mercedes",  
        "Mitsubishi",  
        "Nissan",  
        "Opel",  
        "Peugeot",  
        "Plymoth",  
        "Pontiac",  
        "Renault",  
        "Rover",  
        "Saab",  
        "Subaru",  
        "Suzuki",  
        "Toyota",  
        "Volkswagen",  
        "Volvo"};  
  
    // Виведення марок автомобілів відповідно до порядку  
    елементів у масиві  
  
    Console.WriteLine("Here is the array of car brands:");  
  
    for (int i = 0; i < Brands.Length; i++)  
        Console.Write(Brands[i] + "\t");
```

```
        Console.WriteLine("\n\n");

        // Сортуння елементів у зворотному порядку

        Array.Reverse(Brands);

        // Виведення автомарок
        Console.WriteLine("Here is the list once reversed:");
        for (int i = 0; i < Brands.Length; i++)
            Console.Write(Brands[i] + "\t");

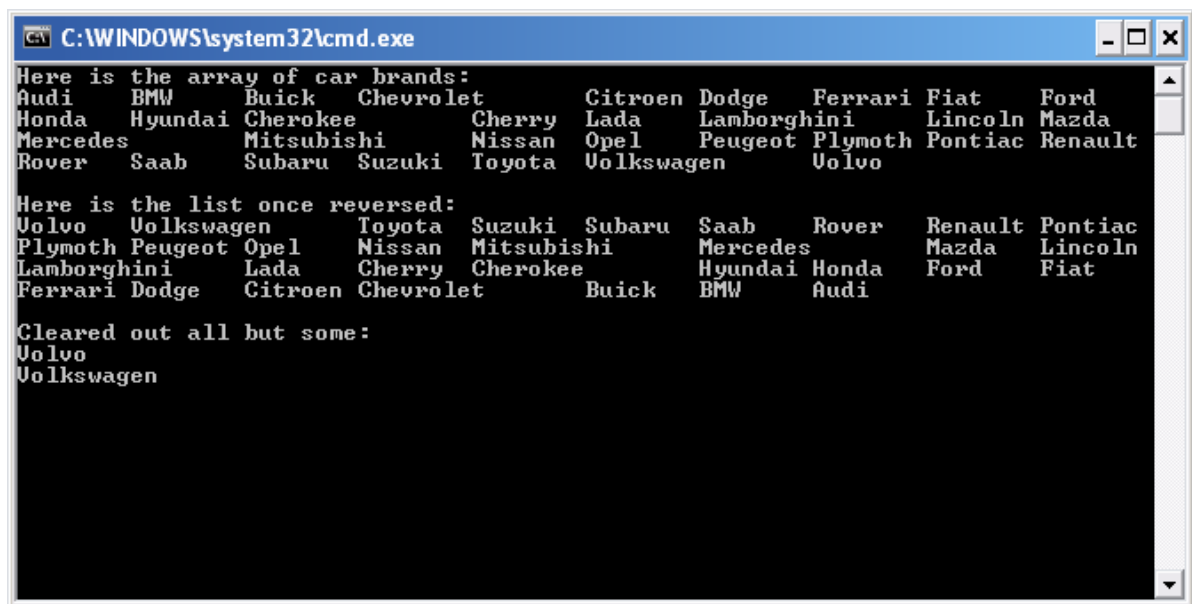
        Console.WriteLine("\n\n");

        // Залишаються тільки перший і останній

        Console.WriteLine("Only two remain: ");
        Array.Clear(Brands, 2, Brands.Length-2);

        for (int i = 0; i < Brands.Length; i++)
        {
            Console.Write(Brands[i] + "\t\n");
        }
    }
}
```

Результат роботи програми:



```
C:\WINDOWS\system32\cmd.exe
Here is the array of car brands:
Audi BMW Buick Chevrolet Citroen Dodge Ferrari Fiat Ford
Honda Hyundai Cherokee Cherry Lada Lamborghini Lincoln Mazda
Mercedes Mitsubishi Nissan Opel Peugeot Plymouth Pontiac Renault
Rover Saab Subaru Suzuki Toyota Volkswagen Volvo

Here is the list once reversed:
Volvo Volkswagen Toyota Suzuki Subaru Saab Rover Renault Pontiac
Plymouth Peugeot Opel Nissan Mitsubishi Mercedes Mazda Lincoln
Lamborghini Lada Cherry Cherokee Hyundai Honda Ford Fiat
Ferrari Dodge Citroen Chevrolet Buick BMW Audi

Cleared out all but some:
Volvo
Volkswagen
```

3. Використання класу *StringBuilder*.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace HelloWorldDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            StringBuilder WordList = new
            StringBuilder("Каждый Охотник ");

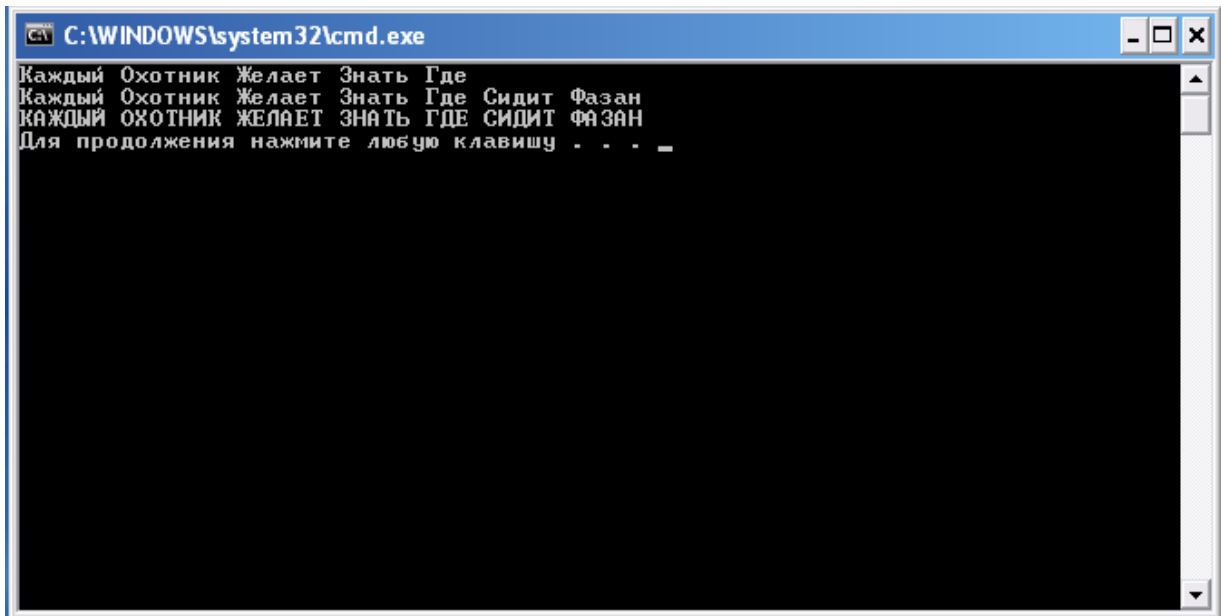
            WordList.Append("Желает Знать Где ");
            Console.WriteLine(WordList);

            WordList.Append("Сидит Фазан");
            Console.WriteLine(WordList);

            // Зробити всі букви прописними
            string Spectrum = WordList.ToString().ToUpper();
            Console.WriteLine(Spectrum);
        }
    }
}
```

```
}  
}  
}
```

Результат роботи програми:



```
C:\WINDOWS\system32\cmd.exe  
Каждый Охотник Желает Знать Где  
Каждый Охотник Желает Знать Где Сидит Фазан  
КАЖДЫЙ ОХОТНИК ЖЕЛАЕТ ЗНАТЬ ГДЕ СИДИТ ФАЗАН  
Для продолжения нажмите любую клавишу . . .
```

4. Використання класу *ArrayList*.

Використання класу *ArrayList* з простору імен *System.Collections* дозволяє ефективно реалізувати роботу з масивами об'єктів, оскільки багато можливостей, які необхідні для цього, реалізовані спочатку, зокрема методи вставки, видалення і нумерації елементів.

Для використання можливостей *ArrayList* використовується не звичайне спадкування, а модель включення у вигляді делегування викликів на виконання різних дій класу похідному від *ArrayList*:

```
public class NBooks : IEnumerable  
{  
    // nbList – внутрішній клас, який буде робити всю роботу  
    private ArrayList nbList;  
  
    // Створюємо об'єкт класу nbList за допомогою конструктора
```

NBooks

```
public NBooks() {nbList = new ArrayList();}

// Реалізуємо потрібні нам методи для прийому викликів ззовні і передачі їх nbList

// Метод для вставки об'єкта NBook
public void AddNBook(NBook nb)
{ nbList.Add(nb); }

// Метод для видалення об'єкту NBook
public void RemoveNBook(int nbToRemove)
{ nbList.RemoveAt(nbToRemove); }

// Властивість, що повертає кількість об'єктів NBook
public int NBookCount
{ get { return nbList.Count; } }

// Метод для очищення об'єкту - видалення всіх об'єктів NBook
public void ClearAllNBooks()
{ nbList.Clear(); }

// Метод, який відповідає на питання - чи є вже в наборі такий об'єкт NBook
public bool NBookIsPresent(NBook c)
{ return nbList.Contains(c); }

// Все, що пов'язано з реалізацією IEnumerator, перенаправляється в nbList
public IEnumerator GetEnumerator()
{ return nbList.GetEnumerator(); }
}
```

Реалізація класу *NBook* і код програми, що використовує клас *NBooks* представлені нижче:

```
using System;
using System.Collections.Generic;
using System.Collections;
```

```
using System.Text;

namespace NBooks
{
    public class NBook
    {

        // конструктор класу NBook

        public string Model;
        public string CPU_model;
        public int CPU_clock;
        public int RAM_size;

        public NBook(string mname, string CPU, int Clock, int RAM)
        {
            Model = mname;
            CPU_model = CPU;
            CPU_clock = Clock;
            RAM_size = RAM;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            NBooks nbLot = new NBooks();

            // Створення списку об'єктів NBook

            nbLot.AddNBook( new NBook("ASUS A7Sn","Intel Core 2 Duo
T8300", 2400, 2048));
            nbLot.AddNBook( new NBook("Acer Aspire 5530G-
803G25Mi","AMD Turion X2 Ultra ZM80", 2100, 3072));
            nbLot.AddNBook( new NBook("Fujitsu Amilo Si 2636", "Intel
Core 2 Duo T8300", 2400, 2048));
            nbLot.AddNBook( new NBook("HP Pavilion tx2650er", "AMD
Turion X2 Ultra ZM82", 2200, 4096));
```

```
// Виводимо інформацію про кожен об'єкт за допомогою
конструкції foreach

    Console.WriteLine("You have {0} in the lot: \n",
nbLot.NBookCount);
    foreach (NBook nb in nbLot)
    {
        Console.WriteLine("Model: {0}", nb.Model);
        Console.WriteLine("CPU: {0}", nb.CPU_model);
        Console.WriteLine(" {0} GHz", nb.CPU_clock);
        Console.WriteLine("RAM: {0} GB\n", nb.RAM_size);
    }

// Видаляємо один з ноутбуків

    nbLot.RemoveNBook(3);
    Console.WriteLine("You have {0} in the lot.\n",
nbLot.NBookCount);

// Додаємо ще один ноутбук і перевіряємо його наявність в
наборі

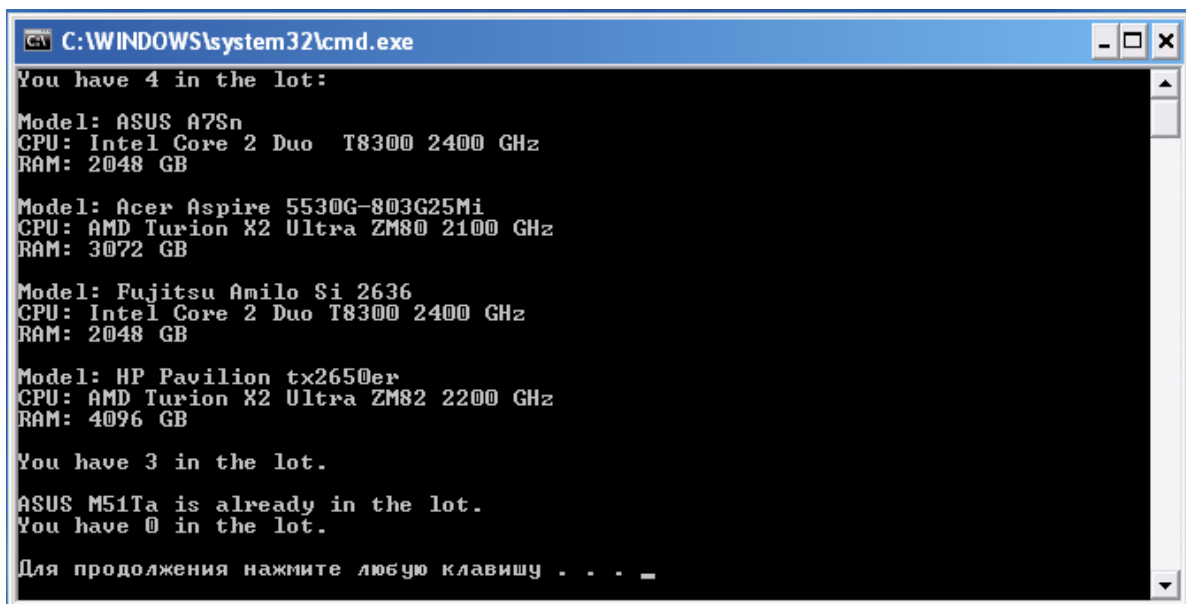
    NBook temp = new NBook("ASUS M51Ta", "AMD Turion™ X2
Ultra ZM84", 2300, 4096);
    nbLot.AddNBook(temp);

    if(nbLot.NBookIsPresent(temp))
        Console.WriteLine(temp.Model + " is already in the lot.");

// Видалити все

    nbLot.ClearAllNBooks();
    Console.WriteLine("You have {0} in the lot.\n",
nbLot.NBookCount);
    }
    }
}
```

Результат виконання програми:



```
C:\WINDOWS\system32\cmd.exe
You have 4 in the lot:
Model: ASUS A7Sn
CPU: Intel Core 2 Duo T8300 2400 GHz
RAM: 2048 GB

Model: Acer Aspire 5530G-803G25Mi
CPU: AMD Turion X2 Ultra ZM80 2100 GHz
RAM: 3072 GB

Model: Fujitsu Amilo Si 2636
CPU: Intel Core 2 Duo T8300 2400 GHz
RAM: 2048 GB

Model: HP Pavilion tx2650er
CPU: AMD Turion X2 Ultra ZM82 2200 GHz
RAM: 4096 GB

You have 3 in the lot.
ASUS M51Ta is already in the lot.
You have 0 in the lot.

Для продолжения нажмите любую клавишу . . . _
```

Контрольні завдання

Необхідно розробити консольний додаток для введення з клавіатури масиву рядків і пошуку серед них рядків, що містять заданий строковий фрагмент.

Для пошуку потрібно використання методу `IndexOf` (`string findThisString`) для строкових елементів масиву. Метод повертає позицію початку пошукового підрядка від початку рядка, або значення `-1` при відсутності відповідності.

Лабораторна робота №8.

Тема «Основи розробки веб-додатків за допомогою ASP.NET.»

Мета роботи:

1. Ознайомлення з основними етапами розробки веб-додатків на основі ASP.NET в середовищі *Microsoft Visual Studio.NET*.
2. Вивчення структури проекту *ASP.NET Web Application*.

Теоретичний матеріал

ASP.NET файл є текстовим файлом і може містити коди HTML, XML і мов сценаріїв. Коди останніх виконуються на веб-сервері. Файл ASP.NET має спеціальне розширення «*.aspx*».

Порядок роботи ASP.NET виглядає наступним чином:

- Коли веб-браузер запрошує файл ASP.NET, веб-сервер IIS перенаправляє запит модулю ASP.NET на сервері.
- Модуль ASP.NET читає файл построчно і виконує, коди сценаріїв, що містяться у файлі.
- Веб-браузеру повертається назад файл ASP.NET, але вже у вигляді звичайного HTML документа.

Будь-яка сторінка ASP.NET представлена класом, похідним від класу *System.Web.UI*, який визначає властивості, методи і події, загальні для всіх сторінок, призначених для обробки середовищем ASP.NET.

Найбільш важливі властивості цього об'єкту наведені в таблиці нижче:

Властивість	Опис
<i>Application</i>	Повертає об'єкт <i>HttpApplicationState</i>

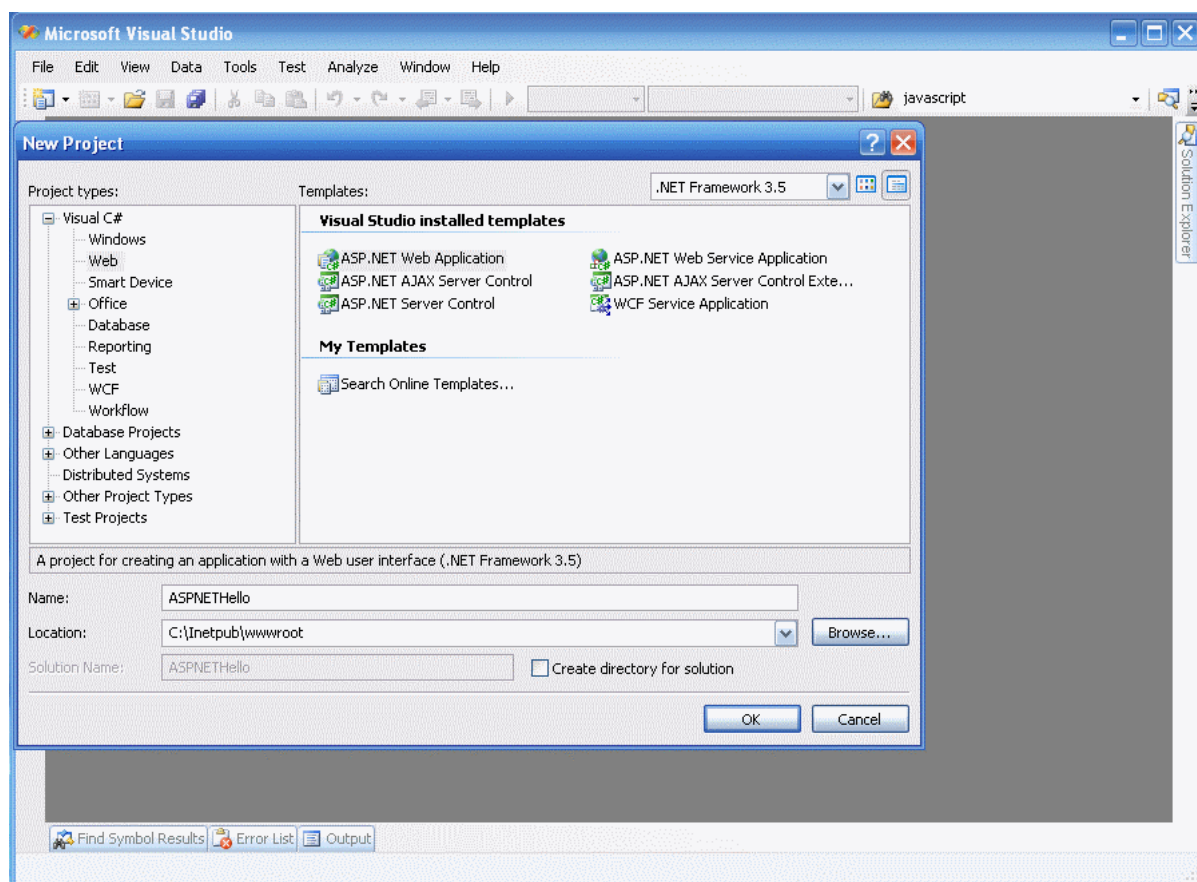
<i>Cache</i>	Повертає об'єкт <i>Cache</i> , в якому зберігаються дані програми, в т. ч. і самої сторінки
<i>IsPostBack</i>	Повертає значення, що визначає, чи була сторінка завантажена клієнтом вперше раз, або завантажена повторно у відповідь на запит клієнта
<i>Request</i>	Повертає об'єкт <i>HttpRequest</i> , використовуваний для отримання інформації про вхідному запиті HTTP
<i>Response</i>	Повертає об'єкт <i>HttpResponse</i> , використовуваний для формування відповіді сервера клієнту
<i>Server</i>	Повертає об'єкт <i>HttpServerUtility</i>
<i>Session</i>	Повертає об'єкт <i>System.Web.SessionState.HttpSessionState</i> , за допомогою якого виходить інформація про поточний час HTTP

Така побудова проекту дозволяє зберігати окремо код подання для генерації HTML коду (у файлі * *.aspx*) від програмної логіки (у файлі * *.aspx.cs*), що в багатьох випадках істотно спрощує розробку складних веб-додатків.

Порядок виконання лабораторної роботи

Для роботи з прикладами, що приводяться в даній лабораторній роботі, потрібно установка середовища розробки Microsoft Visual Studio 2005+ і веб-сервера IIS 5+ (Internet Information Server).

1. Створення нового проекту в середовищі Microsoft Visual Studio з використанням шаблону *ASP.NET Web Application*:

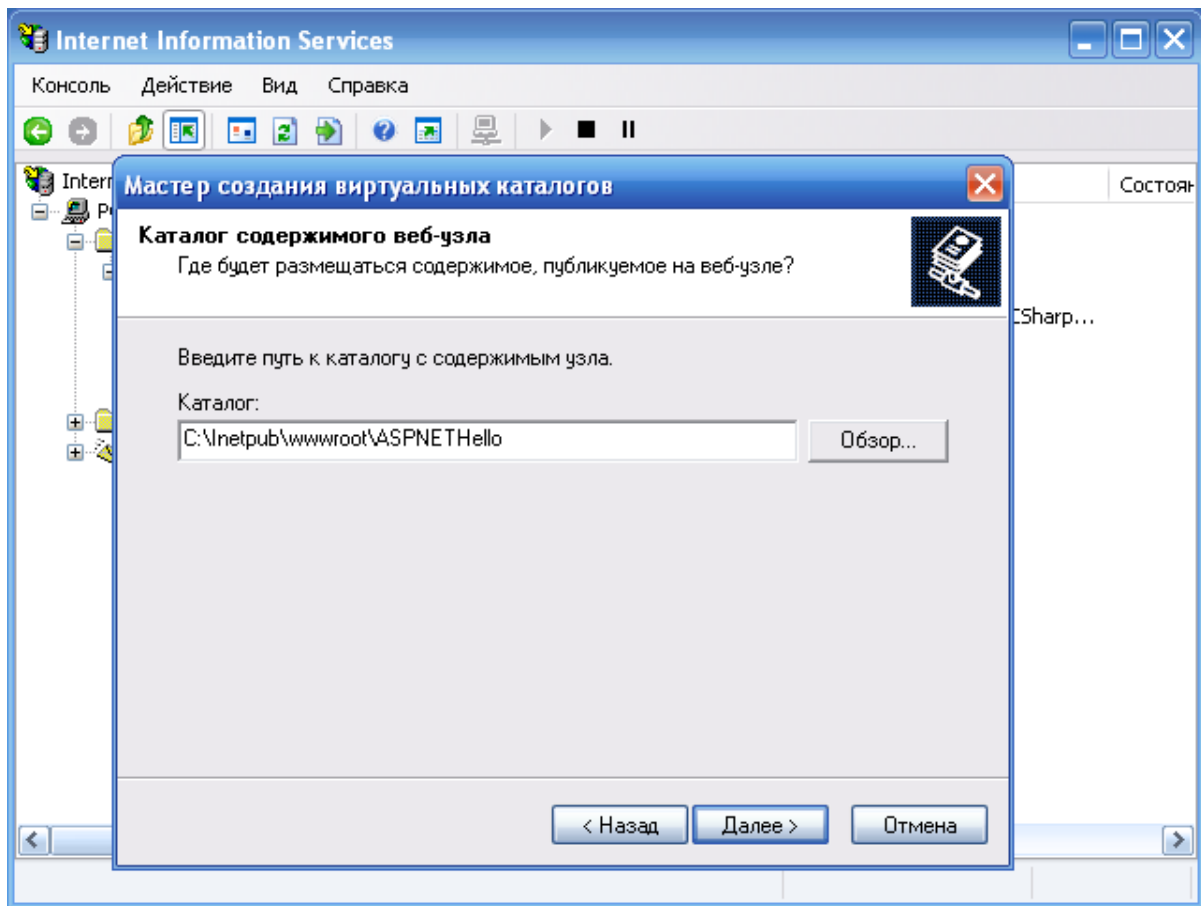


2. Кожне з веб-додатків для ІІС повинна розміщуватись у своєму віртуальному каталозі, якому відповідає фізичний каталог на диску.

Для створення віртуального каталогу необхідно:

1) відкрити оснастку Microsoft Internet Information Services (Пуск> Панель управління> Адміністрування> Internet Information Services);

2) розкрити гілку «Веб-сайти» і, перейшовши в «Веб-вузол за замовчуванням», створити новий віртуальний каталог:



3) у властивостях створеного віртуального каталогу вибрати закладку «Документи» і додати в список документів, що використовуються за замовчуванням, документ *Default.aspx*.

Після завершення створення проекту, він міститиме файли *Default.aspx*, *Default.aspx.cs* і *Default.asp.designer.cs*.

Перший з них буде містити приблизно наступний код:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="ASPNETHello._Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
```

```
<head runat="server">
  <title>Untitled Page</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
    </div>
  </form>
</body>
</html>
```

З цього коду видно, що, по-перше, для створення коду HTML, що повертається браузеру, буде використаний мову С# (див. Атрибут *Language*). По-друге, код С# міститься в окремому файлі, який буде виконуватися на веб-сервері (див. Атрибут *CodeBehind*). І, нарешті, атрибут *Inherits* вказує на ім'я класу, визначеного в *CodeBehind*.

Важливим нововведенням в ASP.NET є атрибут *runat="server"*, розміщений в тегах `<form>`. Він означає, що даний елемент повинен бути оброблений середовищем виконання ASP.NET.

3. Тепер між тегами `<div>` і `</div>` у файлі *Default.aspx* можна вставити код:

```
<h1>Hello!</h1>
<h1>I am</h1>
<%=Request.ServerVariables["HTTP_USER_AGENT"] %>
```

У даному прикладі було використано властивість *Request* об'єкту, похідного від *Page*, для отримання значення змінної оточення *HTTP_USER_AGENT*.

Код С#, який міститься у файлі *Default.aspx.cs* може виглядати приблизно так:

```
using System;
```

```
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;

namespace ASPNETHello
{
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }
    }
}
```

Тут міститься опис методу *Page_Load*, який викликається при завантаженні веб-сторінки *Default.aspx* з віртуального каталогу для даного проекту. У даному прикладі метод містить пусте тіло.

4. Після компіляції проекту (опція меню «*Build*» або <Shift + F6>) і його виконання (<Ctrl + F5>) можна буде побачити в браузері сторінку приблизно наступного вигляду:

```

Hello!
I am
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR
1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30; InfoPath.2; .NET
CLR 3.0.04506.648; .NET CLR 3.5.21022)>
```

Контрольне завдання

Створіть ASP.NET додаток, що виводить список всіх змінних оточення веб-серверу.

Для цього буде потрібно вставка наступного коду:

```
<%  
// метод AllKeys формує список всіх ключів масиву  
string[] rkeys = Request.ServerVariables.AllKeys;  
  
// створення змінної типу StringBuilder (тобто змінною рядка)  
StringBuilder output = new StringBuilder();  
  
// перебір всіх елементів масиву rkeys і формування списку  
значень  
foreach (string rkey in rkeys)  
{  
    output.Append(rkey + "=" + Request.ServerVariables[rkey] +  
"<br>");  
}  
// формування відповіді сервера  
Response.Write(output);  
%>
```

Лабораторна робота №9.

Тема «Серверні елементи управління ASP.NET.»

Мета роботи:

1. Знайомство із засобами автоматизації розробки веб-додатків в *ASP.NET* у вигляді серверних елементів управління *WebForm*.

Теоретичний матеріал

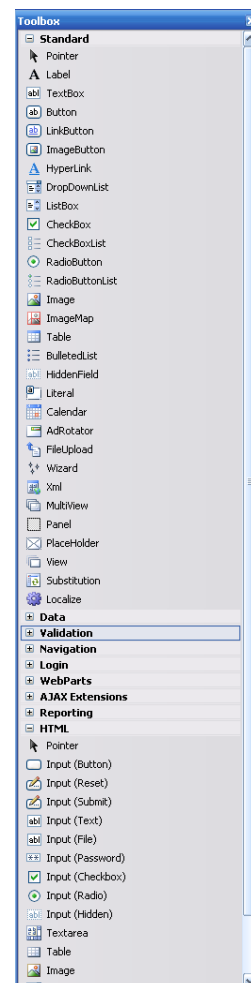
Важливою особливістю *ASP.NET* є використання серверних елементів управління на веб-сторінці (елементи *WebForm*), які фактично є тегами, які є зрозумілими веб-серверу. Ці елементи визначені в просторі імен *System.Web.UI.WebControls*.

Прийнято виділяти три типи серверних елементів управління:

- Серверні елементи управління *HTML* – звичайні *HTML* теги.
- Елементи управління веб-сервера – нові теги *ASP.NET*.
- Серверні елементи управління для перевірки даних (валідації) – застосовуються для валідації вхідних даних від клієнтського додатку (зазвичай веб-браузера).

Переваги від використання таких елементів при розробці веб-додатків:

- Скорочується кількість коду, написаного вручну (що особливо помітно для складних елементів документа). Елемент просто



«перетягується» з панелі інструментів, після чого виконується налаштування його параметрів у спеціальному вікні. При цьому всі зміни автоматично заносяться безпосередньо в *.aspx файл.

- З програмної точки зору кожного з цих елементів управління відповідає певний клас в бібліотеці базових класів .NET, що дозволяє писати для них такий же код як і для будь-яких інших класів.
- Для будь-якого елемента управління WebForm визначений набір подій, оброблюваних на веб-сервері.
- Для будь-якого елемента управління WebForm надається можливість для перевірки введення даних користувачем.

1. Серверні елементи управління HTML

За замовчуванням такі елементи управління в ASP.NET файлах розглядаються як текст. Для їх програмування потрібно додавання атрибуту *runat="server"* у відповідний HTML елемент. Крім того, всі серверні елементи управління HTML повинні бути розміщені всередині області дії тегу *<form>*, який також має атрибут *runat="server"*.

Серверний елемент управління HTML	Опис
HtmlAnchor	Управління HTML елементом <i><a></i>
HtmlButton	Управління HTML елементом <i><button></i>
HtmlForm	Управління HTML елементом <i><form></i>
HtmlGeneric	Управляє HTML елементами, що не описуються як елементи управління HTML, наприклад, <i><body></i> , <i><div></i> , <i></i> та ін.

HtmlImage	Управління HTML елементом <image>
HtmlInputButton	Управління HTML елементами <input type="button">, <input type="submit"> та <input type="reset">
HtmlInputCheckBox	Управління HTML елементом <input type="checkbox">
HtmlInputFile	Управління HTML елементом <input type="file">
HtmlInputHidden	Управління HTML елементом <input type="hidden">
HtmlInputImage	Управління HTML елементом <input type="image">
HtmlInputRadioButton	Управління HTML елементом <input type="radio">
HtmlInputText	Управління HTML елементами <input type="text"> та <input type="password">
HtmlSelect	Управління HTML елементом <select>
HtmlTable	Управління HTML елементом <table>
HtmlTableCell	Управління HTML елементами <td> и <th>
HtmlTableRow	Управління HTML елементом <tr>
HtmlTextArea	Управління HTML елементом <textarea>

2. Елементи керування веб-серверу

Подібно серверним елементам керування, HTML елементи даного типу також створюються на веб-сервері і передбачають додавання атрибуту *runat="server"*. Однак, вони можуть не відповідати конкретним елементам HTML, але представляти більш складні елементи.

Загальний синтаксис для опису таких елементів:

```
<asp:тип_елемента id="ідентифікатор" runat="server"/>
```

Перелік доступних елементів такого типу наведено в таблиці.

Елемент керування веб-серверу	Опис
AdRotator	Банерна рулетка
Button	Відображення кнопки
Calendar	Відображення календаря
CalendarDay	Елемент вибору дня календаря
CheckBox	Відображення прапорця
CheckBoxList	Група прапорців
DataGrid	Відображення полів джерела даних
DataList	Відображення елементів з джерела даних за допомогою шаблонів
DropDownList	Випадаючий список
HyperLink	Гіперпосилання
Image	Зображення
ImageButton	Кнопка у вигляді зображення
Label	Відображення статичного змісту, доступного для програмування, і з можливістю задання стилю
LinkButton	Кнопка з гіперпосиланням
ListBox	Випадаючий список з одиничним або множинним виділенням
ListItem	Елемент списку
Literal	Відображення статичного вмісту, доступного для програмування, але без можливості задання стилю
Panel	Контейнер для інших елементів керування
Placeholder	Місце для додавання елементів керування програмним способом
RadioButton	Радіо-кнопка
RadioButtonList	Група радіо-кнопок
BulletedList	Маркерний список
Repeater	Повторюючий список елементів
Style	Задання стилю елемента керування
Table	Таблиця
TableCell	Комірка таблиці
TableRow	Рядок таблиці
TextBox	Поле для введення тексту
Xml	Відображення XML файлу або результату XSLT перетворення

3. Серверні елементи управління для перевірки даних (валідації)

Елементи управління даного типу застосовуються для перевірки введених даних.

Мають наступний синтаксис:

```
<asp:тип_елемента id="ідентифікатор" runat="server" />
```

Найбільш важливі елементи наводяться в наступній таблиці.

Елемент керування для перевірки даних	Опис
CompareValidator	Порівнює значення, введене в один елемент керування зі значенням, введеним в інший елемент, або з фіксованим значенням
CustomValidator	Дозволяє задавати користувацький метод перевірки введених значень
RangeValidator	Перевіряє, що значення, введене користувачем, знаходиться між двома величинами
RegularExpressionValidator	Перевіряє введене значення на відповідність вказаному шаблону
RequiredFieldValidator	Перевіряє обов'язкову наявність введеного значення
ValidationSummary	Відображає звіт по всім помилкам перевірки значень, які виникли на веб-сторінці

Порядок виконання лабораторної роботи

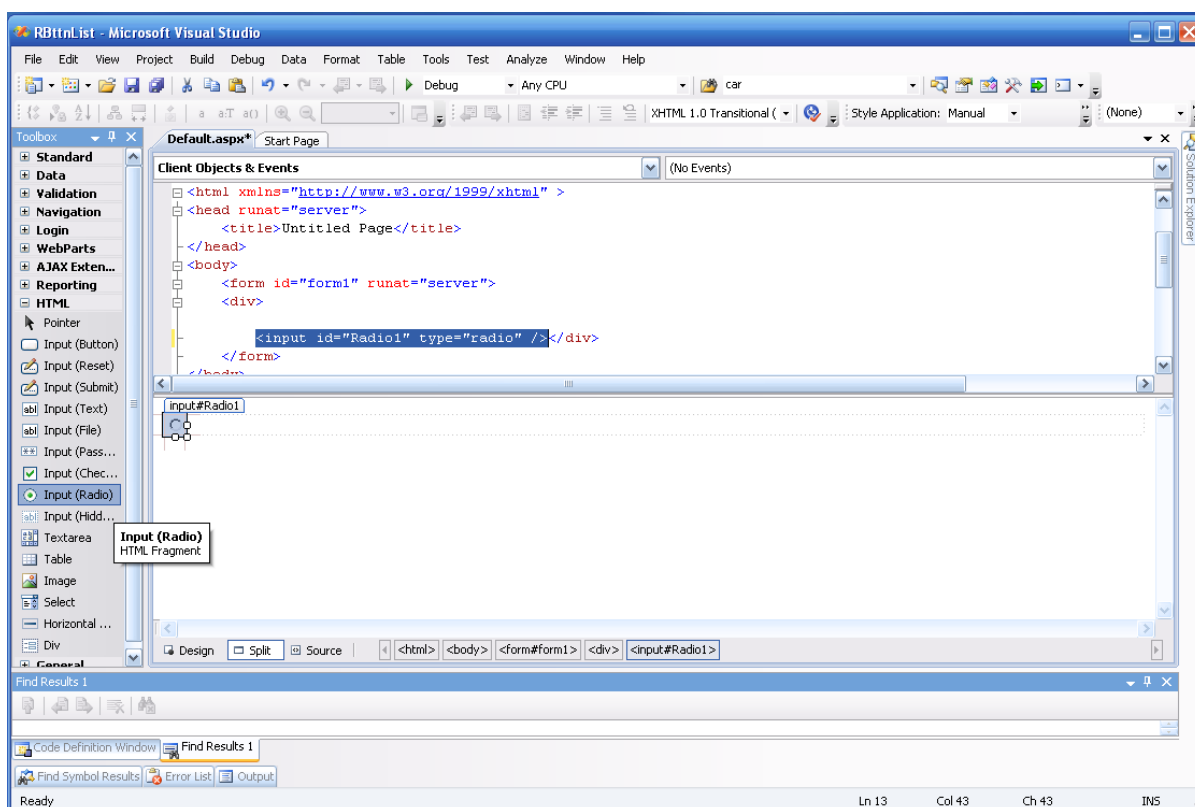
Для роботи з прикладами, що приводяться в даній лабораторній роботі, потрібно установка середовища

розробки Microsoft Visual Studio 2005+ і веб-сервера IIS 5+ (Internet Information Server).

1. Серверні елементи керування HTML.

1) Створіть новий проект по шаблону *ASP.NET Web Application*.

2) На сторінці *Default.aspx* додайте в форму стандартні HTML елементи введення даних. Це можна зробити, використовуючи панель елементів управління, як це показано на скріншоті нижче.



Код форми повинен виглядати наступним чином:

```
<form id="form1" runat="server">
  <div>
    Ваша оцінка:<br><br>
    <input id="r1" type="radio" title="Отлично" value="Отлично"
    runat="server"/>Відмінно <br />
```

```
<input id="r2" type="radio" value="Хорошо" runat="server"
/>Добре<br />
<input id="r3" type="radio" value="Удовлетворительно"
runat="server" />Задовільно<br />
<input id="r4" type="radio" value="Неудовлетворительно"
runat="server" />Незадовільно<br />
<p id="ans" runat="server"></p>
<br>
<input id="Submit1" type="submit" value="Отправить"
OnServerClick="submit" runat="server" /><br />
</div>
</form>
```

Обробник події *OnServerClick* необхідно описати в файлі *Default.aspx.cs*. Код методу *submit*:

```
protected void submit(object sender, System.EventArgs e)
{
    if (r1.Checked == true)
        ans.InnerHtml = "Вы оценили на отлично";
    else if (r2.Checked == true)
        ans.InnerHtml = "Вы оценили на хорошо";
    else if (r3.Checked == true)
        ans.InnerHtml = "Вы оценили на удовлетворительно";
    else if (r4.Checked == true)
        ans.InnerHtml = "Вы оценили на неудовлетворительно";
    else
        ans.InnerHtml = "Вы не выбрали оценку";
}
```

3) Виконайте компіляцію проекту і запустіть його на виконання (або вручну завантажте документ в браузері, вказавши URL відповідного віртуального каталогу).

2. Елементи управління веб-серверу.

Якщо для новоствореного проекту типу *ASP.NET Web Application* у файлі *Default.aspx* всередині форми (між тегам *<form>* і *</ form>*) вставити наступний код (або зробити це

перетягуванням потрібних елементів з панелі інструментів, розділ «Standard»):

```
<asp:ListBox id="LstBx" runat="server" width="100" height="80"
AutoPostBack="True" OnSelectedIndexChanged="LBSelChanged" >
    <asp:ListItem value="Sunday">Sunday</asp:ListItem>
    <asp:ListItem value="Monday">Monday</asp:ListItem>
    <asp:ListItem value="Tuesday">Tuesday</asp:ListItem>
    <asp:ListItem value="Wednesday">Wednesday</asp:ListItem>
    <asp:ListItem value="Thursday">Thursday</asp:ListItem>
    <asp:ListItem value="Friday">Friday</asp:ListItem>
    <asp:ListItem value="Saturday">Saturday</asp:ListItem>
</asp:ListBox>
<br/><br/>
<asp:Label id="Label1" runat="server"/>
```

і описати обробку подій *Page_Load* і *OnSelectedIndexChanged* у відповідному *.aspx.cs файлі:

```
protected void Page_Load(object sender, EventArgs e)
{
    Label1.Text = "You selected: " + Label1.Text;
}

protected void LBSelChanged(object sender, System.EventArgs e)
{
    if (LstBx.SelectedItem != null)
        Label1.Text = "You selected: " + LstBx.SelectedItem.Value;
    else
        Label1.Text = "You selected: ";
}
```

то при запуску проекту після вибору користувачем елемента зі списку можна буде побачити у вікні браузера:

Sunday	▲
Monday	
Tuesday	
Wednesday	▼

You selected: Wednesday

У даному прикладі атрибут *AutoPostBack="True"* вказує на необхідність негайної обробки події на сервері.

3. Серверні елементи управління для перевірки даних (валідації)

Наступний код показує, яким чином виконується перевірка змісту поля введення форми:

```
<form id="form1" runat="server">
  <asp:TextBox ID="TextBox1" runat="server"
Width="123px"></asp:TextBox>
  <br />
  <asp:RegularExpressionValidator
    ID="RegularExpressionValidator1"
    runat="server"
    ControlToValidate="TextBox1"
    ErrorMessage="Not valid e-mail"
    ValidationExpression="\w+([-+!]\w+)*@\w+([-.\w+)*\.\w+([-
.\w+)*" >
  </asp:RegularExpressionValidator>
</form>
```

В даному випадку атрибут *ControlToValidate* вказує на те, що контролюється зміст елемента з ідентифікатором *TextBox1* (поля введення тексту), у разі невідповідності змісту контрольованого поля з шаблоном, описаному у вигляді регулярного виразу в атрибуті *ValidationExpression*, видається повідомлення, вказане в атрибуті *ErrorMessage*.

Шаблон описує допустимий формат адреси електронної пошти. Додайте код в проект і перевірте роботу програми.

Контрольне завдання

Створіть веб-додаток *ASP.NET*, що підтримує введення на сторінці міжнародного телефонного номера користувачем і використовує серверний елемент управління *<asp:RegularExpressionValidator>*.

Лабораторна робота №10.

Тема «Робота з джерелами даних в ASP.NET.»

Мета роботи:

1. Вивчення елементів *WebForm*, призначених для відображення на веб-сторінці даних, одержуваних з джерел даних.

Теоретичний матеріал

У ASP.NET використовуються два елементи управління *WebForm* для управління відображенням даних, одержуваних з джерела даних:

- *DataGrid* – Елемент керування, що відображає зміст об'єкту ADO.NET *DataSet* у вигляді таблиці.
- *DataList* – Елемент керування для вибору значень, що заповнюються з джерела даних.

Якщо необхідно відобразити дані, отримані за запитом користувача з джерела даних, у вигляді таблиці на веб-сторінці, то ASP.NET надає в розпорядження веб-програміста зручний елемент керування *DataGrid*, який був введений в ASP.NET 1.x. але тепер його функції перекриваються *GridView*. Елемент управління *GridView* може не тільки показувати дані, але і сортувати, вибирати, редагувати їх. Якщо цієї функціональності недостатньо, її можна розширити, написавши власні обробники подій.

Елементи, які можуть бути пов'язані з джерелами даних, різноманітні, наприклад, *DropDownList*, *ListBox*, *CheckBoxList*, *RadioButtonList*, *BulletedList*.

1. Елемент *GridView* (*DataGrid*)

Елемент управління *GridView* – головний елемент керування для подання інформації з баз даних в ASP.NET.

Він є «спадкоємцем» елемента управління *DataGrid*, який використовувався в попередніх версіях ASP.NET. Основне призначення цього елемента керування – представлення користувачу інформації в табличному вигляді з джерела даних з можливістю фільтрації, сортування та редагування.

Найпростіше налаштувати цей елемент керування за допомогою вбудованого майстра. При використанні майстра необхідно:

1. Вибрати тип джерела даних.
2. У властивостях джерела даних (елементу управління *DataSource*) налаштувати параметри підключення до бази даних.
3. Згенерувати або ввести код команд SELECT, INSERT, UPDATE і DELETE. При цьому необов'язково заповнювати дані для всіх команд.
4. Налаштувати додаткові параметри об'єкту *GridView*, наприклад, дозволити розбиття на сторінки, фільтрацію, сортування, вибрати оформлення і т. п.

2. *DataList*

У порівнянні з елементом керування *GridView* елемент управління *DataList* надає більше можливостей по налаштуванню зовнішнього вигляду даних, які беруться з джерела, однак вимагає значно більше часу і зусиль по налаштуванню. Додавати нові записи, також як і при використанні елемента керування *GridView*, засобами *DataList* неможливо.

Робота з цим елементом керування *DataList* (на закладці *Default.aspx* в режимі *Design*) виглядає наступним чином:

1. Після перетягування елемента керування *DataList* в форму, необхідно налаштувати для нього джерело даних (або використовуючи візуальний інтерфейс – за допомогою

об'єкту *DataSource*, або програмно через властивість *DataSource - DataSet*);

2. Далі потрібно перейти в режим редагування шаблонів для записів. Для цього в контекстному меню вибрати *EditItemTemplate*. Для *Item Templates* з'являться чотири області:

- *ItemTemplate* – шаблон для відображення звичайних елементів на формі;
- *AlternatingItemTemplate* – необов'язковий шаблон для відображення кожного другого запису. Зазвичай для нього тільки трохи змінюється фон у порівнянні з *ItemTemplate* – щоб зручніше було сприймати довгі сторінки;
- *SelectedItemTemplate* – шаблон для відображення обраного в даний момент запису;
- *EditItemTemplate* – шаблон для відображення редагованого в даний момент запису.

Ці властивості можна також визначити і через розділ *Properties* в контекстному меню елемента керування *DataList*. Можна обмежитися також простим вибором одного з шаблонів автоформатування.

3. Потім потрібно заповнити необхідними елементами управління (і просто кодом HTML) кожен з шаблонів. Для кожного поля, інформацію про яке потрібно надавати користувачам, необхідно створити свій елемент управління. Для всіх шаблонів, крім *EditItemTemplate*, для більшості полів будуть використовуватися елементи управління *Label* (хоча можливі й інші варіанти). Для *EditItemTemplate* найчастіше використовуються елементи управління *TextBox*.

Щоб зв'язати елементи управління з полями в джерелі даних, найпростіше використати посилання *EditDataBindings* в майстрові, який з'являється для кожного з доданих елементів управління.

Порядок виконання лабораторної роботи

1. Використання елементу *GridView* (*DataGrid*).

1) У наступному прикладі в якості джерела даних використовується база даних в форматі MS ACCESS, що містить таблицю *lesson* з полями:

- id (тип – лічильник);
- Title (тип – текст);
- Teacher (тип – текст);
- Type (тип – текст).

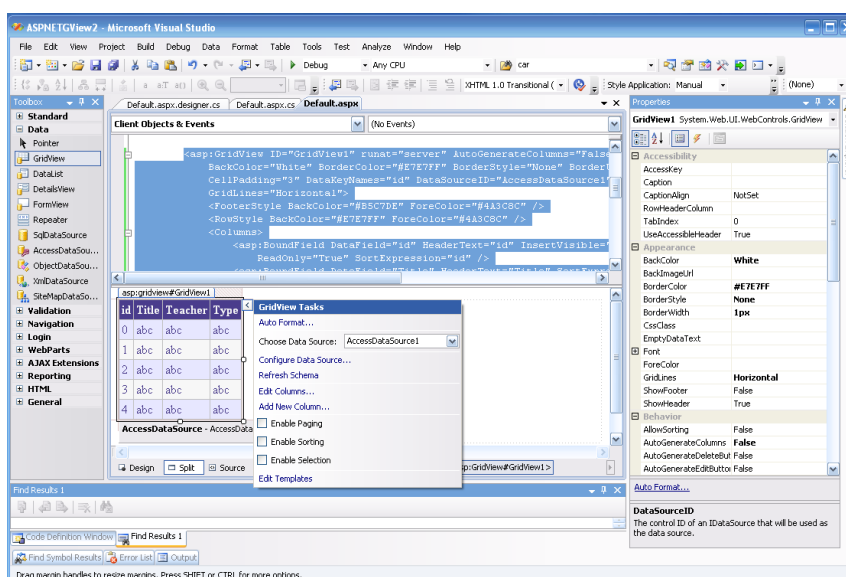
Створіть файл у форматі MS ACCESS, що містить цю таблицю.

2) Створіть новий проект по шаблону *ASP.NET Web Application*.

3) Для сторінки *Default.aspx* виберіть режим відображення *Split*

4) З панелі елементів управління *Toolbox* (відображається за допомогою меню *View> Toolbox*) перетягніть у форму елемент *GridView* (з групи *Data*).

5) Для елементу *GridView* налаштуйте джерело даних (*Choose Data Source, Configure Data Source*) і форматування (можна вибрати один із шаблонів автоформатування – *Auto Format*).



6) Автоматично згенерований код подання (файл *Default.aspx*) може виглядати наступним чином:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="ASPNETGVView2._Default"
%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Untitled Page</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>

      <asp:GridView ID="GridView1" runat="server"
AutoGenerateColumns="False"
  BackColor="White" BorderColor="#E7E7FF"
BorderStyle="None" BorderWidth="1px"
  CellPadding="3" DataKeyNames="id"
DataSourceID="AccessDataSource1"
  GridLines="Horizontal">
    <FooterStyle BackColor="#B5C7DE" ForeColor="#4A3C8C" />
    <RowStyle BackColor="#E7E7FF" ForeColor="#4A3C8C" />
    <Columns>
      <asp:BoundField DataField="id" HeaderText="id"
InsertVisible="False"
  ReadOnly="True" SortExpression="id" />
      <asp:BoundField DataField="Title" HeaderText="Title"
SortExpression="Title" />
      <asp:BoundField DataField="Teacher" HeaderText="Teacher"
  SortExpression="Teacher" />
      <asp:BoundField DataField="Type" HeaderText="Type"
SortExpression="Type" />
    </Columns>
```

```

    <PagerStyle BackColor="#E7E7FF" ForeColor="#4A3C8C"
HorizontalAlign="Right" />
    <SelectedRowStyle BackColor="#738A9C" Font-Bold="True"
ForeColor="#F7F7F7" />
    <HeaderStyle BackColor="#4A3C8C" Font-Bold="True"
ForeColor="#F7F7F7" />
    <AlternatingRowStyle BackColor="#F7F7F7" />
</asp:GridView>
<asp:AccessDataSource ID="AccessDataSource1" runat="server"
    DataFile="D:\CSharp\ASPNETDataGrid\lessons.mdb"
    SelectCommand="SELECT * FROM
[lesson]"></asp:AccessDataSource>

</div>
</form>
</body>
</html>

```

У даному прикладі використаний елемент управління `<asp: GridView>`, що має ідентифікатор «*GridView1*».

7) Якщо джерело даних не налаштований автоматично (при описі коду подання):

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>

            <asp:DataGrid ID="GridView1" runat="server"
BackColor="White"
            BorderColor="White" BorderStyle="Ridge" BorderWidth="2px"

```

```
CellPadding="3"
    CellSpacing="1" GridLines="None" >

    <FooterStyle BackColor="#C6C3C6" ForeColor="Black" />
    <SelectedItemStyle BackColor="#9471DE" Font-Bold="True"
ForeColor="White" />
    <PagerStyle BackColor="#C6C3C6" ForeColor="Black"
HorizontalAlign="Right" />
    <ItemStyle BackColor="#DEDFDE" ForeColor="Black" />
    <HeaderStyle BackColor="#4A3C8C" Font-Bold="True"
ForeColor="#E7E7FF" />

    </asp:DataGrid>

</div>
</form>
</body>
</html>
```

то це можна зробити програмно у відповідному програмному коді, наприклад:

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
using System.Data.OleDb;

namespace ASPNETDataGrid
{
    public partial class _Default : System.Web.UI.Page
```



```
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {

            // Підключення до джерела даних

            OleDbConnection cn = new OleDbConnection();
            cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" +
@"data source = D:\CSharp\ASPNETDataGrid\lessons.mdb";
            cn.Open();

            // Формується рядок SQL запиту даних з джерела

            string str = "SELECT * from lesson";

            // Відбувається з'єднання з базою даних
            // За допомогою керованого провайдера OLE DB

            OleDbDataAdapter dAdapt = new OleDbDataAdapter(str,
cn);

            // Отримання даних з джерела

            DataSet myDS = new DataSet("lessons");

            dAdapt.Fill(myDS, "lesson");

            // Заповнення таблиці даними

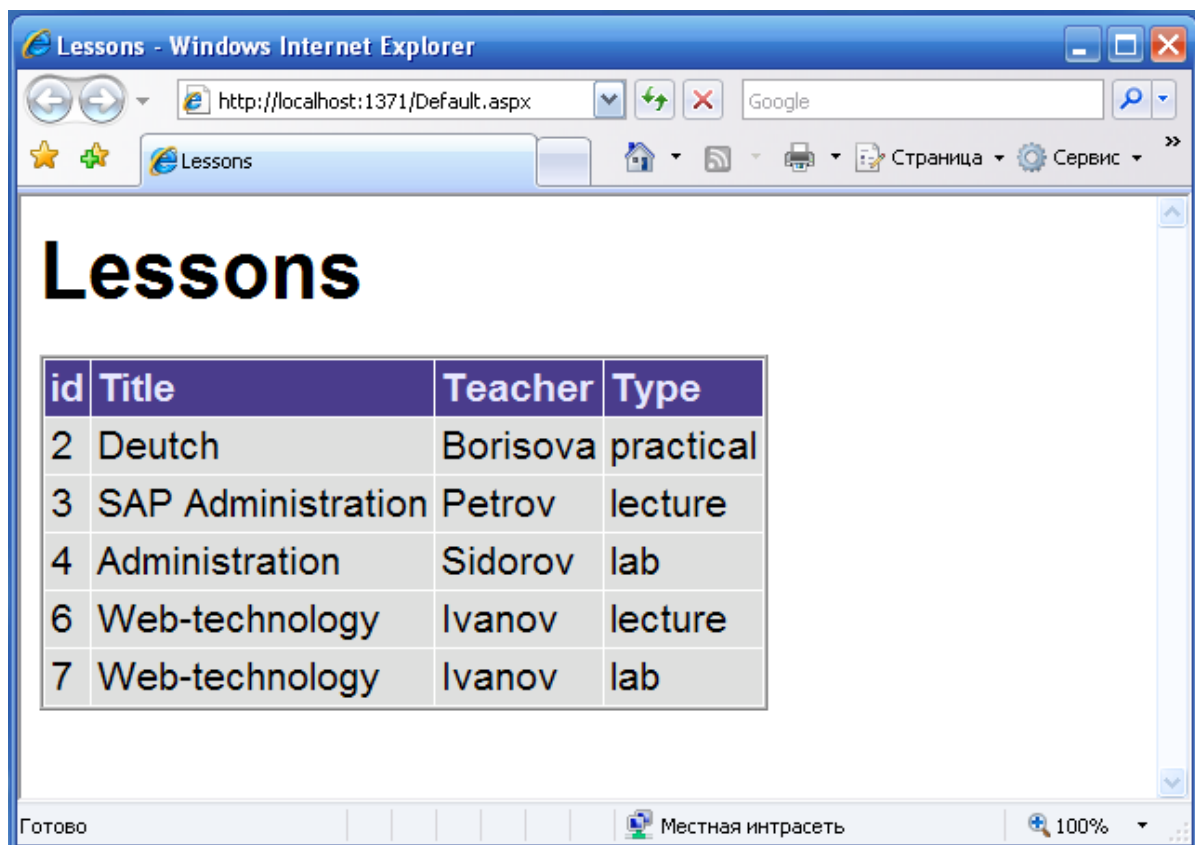
            GridView1.DataSource =
myDS.Tables["lesson"].DefaultView;
            GridView1.DataBind();

            cn.Close();
        }
    }
}
```

Слід звернути увагу на те, що в програмі додано простір імен:

- `using System.Data.OleDb`. Властивість об'єкту **Page** - `IsPostBack` показує, чи була сторінка клієнта завантажена в перший раз або повторно у відповідь на передані клієнтом дані.

8) Результат виконання виглядає наступним чином:



6. Використання елемента *DataList*.

1) У наступному нижче прикладі в якості джерела даних використовується база даних в форматі MS ACCESS, що містить таблицю *lesson* (як у попередньому прикладі).

2) Створіть новий проект по шаблону *ASP.NET Web Application*.

3) Для сторінки *Default.aspx* виберіть режим відображення *Split*.

4) З панелі елементів управління *Toolbox* (відображається за допомогою меню *View> Toolbox*) перетягніть у форму елемент *DataList* (з групи *Data*).

5) Для елементу *DataList* налаштуйте джерело даних (*Choose Data Source, Configure Data Source*) і форматування (можна вибрати один із шаблонів автоформатування – *Auto Format*).

6) Вийде наступний код подання:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="ASPNETGView2._Default"
%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Untitled Page</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>

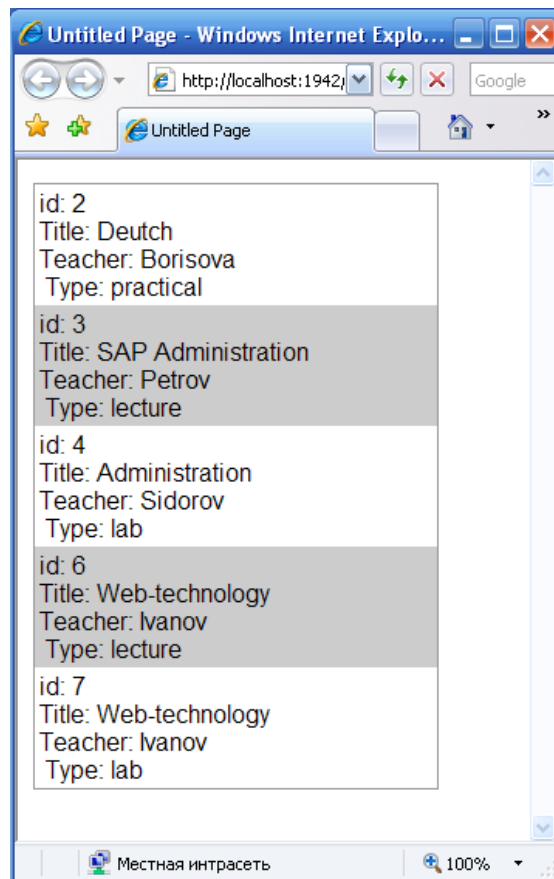
      <asp:DataList ID="DataList1" runat="server" BackColor="White"
        BorderColor="#999999" BorderStyle="Solid"
BorderWidth="1px" CellPadding="3"
        DataKeyField="id" DataMember="DefaultView"
DataSourceID="AccessDataSource1"
        ForeColor="Black" GridLines="Vertical" style="margin-right:
0px" Width="262px" >
        <FooterStyle BackColor="#CCCCCC" />
        <AlternatingItemStyle BackColor="#CCCCCC" />
        <SelectedItemStyle BackColor="#000099" Font-Bold="True"
ForeColor="White" />
        <HeaderStyle BackColor="Black" Font-Bold="True"
ForeColor="White" />
```

```
<ItemTemplate>
    id:
    <asp:Label ID="idLabel" runat="server" Text='<%# Eval("id")
%>' />
    <br />
    Title:
    <asp:Label ID="TitleLabel" runat="server" Text='<%#
Eval("Title") %>' />
    <br />
    Teacher:
    <asp:Label ID="TeacherLabel" runat="server" Text='<%#
Eval("Teacher") %>' />
    &nbsp;<br />
    &nbsp;Type:
    <asp:Label ID="TypeLabel" runat="server" Text='<%#
Eval("Type") %>' />
    <br />
</ItemTemplate>
</asp:DataList>

<asp:AccessDataSource ID="AccessDataSource1" runat="server"
    DataFile="D:\CSharp\ASPNETDataGrid\lessons.mdb"
    SelectCommand="SELECT * FROM [lesson]">
</asp:AccessDataSource>

</div>
</form>
</body>
</html>
```

Результат роботи в браузері:



7) Альтернативний варіант – використання методу *GetDataItem* об'єкту *DataBinder* для заповнення елементів замість *<asp: Label>*.

Нижче представлений відповідний код подання:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="ASPNETDataGrid._Default"
%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head2" runat="server">
  <title>Lessons</title>
</head>
<body>
  <form id="form2" runat="server">
```

```
<div>
  <h1>Lessons</h1>
</div>
<asp:DataList ID="DataList1" runat="server" BackColor="White"
  BorderColor="White" BorderStyle="Ridge" BorderWidth="2px"
  CellPadding="3"
  CellSpacing="1">
  <FooterStyle BackColor="#C6C3C6" ForeColor="Black" />
  <ItemStyle BackColor="#DEDFDE" ForeColor="Black" />
  <SelectedItemStyle BackColor="#9471DE" Font-Bold="True"
  ForeColor="White" />
  <HeaderStyle BackColor="#4A3C8C" Font-Bold="True"
  ForeColor="#E7E7FF" />
  <ItemTemplate>
    <%# DataBinder.GetDataItem(Container)%>
  </ItemTemplate>
</asp:DataList>

</form>
</body>
</html>
```

Програмний код:

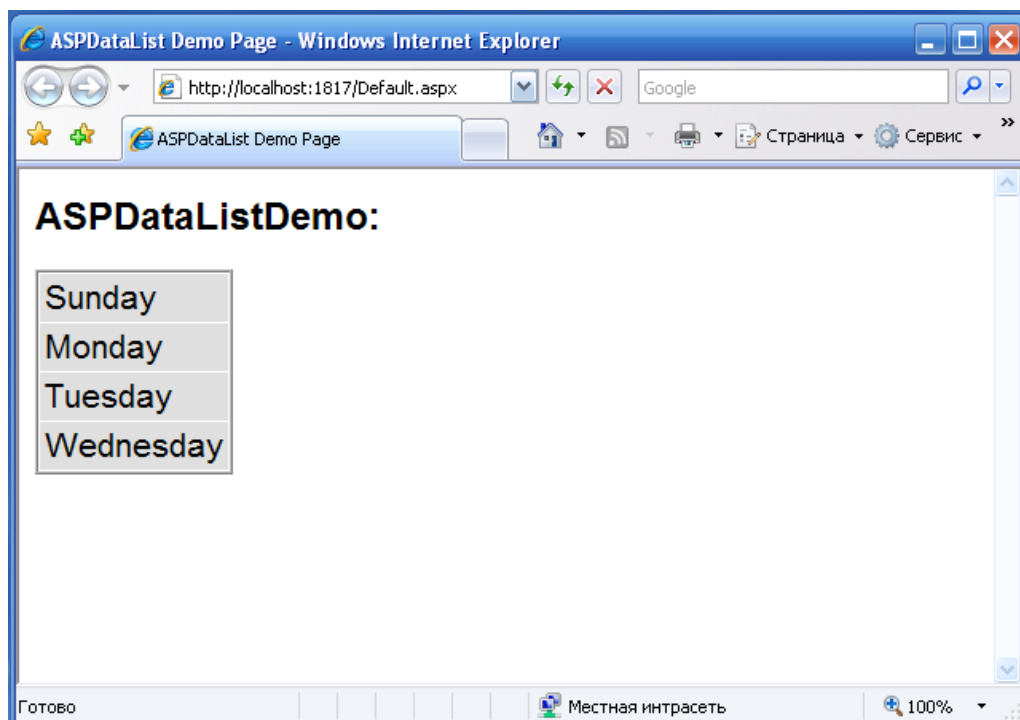
```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;

namespace ASPNETDataGrid
{
```

```
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            ArrayList DWeek = new ArrayList();
            DWeek.Add("Sunday");
            DWeek.Add("Monday");
            DWeek.Add("Tuesday");
            DWeek.Add("Wednesday");
            DWeek.Add("Thursday");
            DWeek.Add("Friday");
            DWeek.Add("Saturday");

            DataList1.DataSource = DWeek;
            DataList1.DataBind();
        }
    }
}
```

Результат, який відображається в веб-браузері:



3. Використання елемента *ListBox* (із заповненням даними із звичайного масиву).

1) Створіть новий проект по шаблону *ASP.NET Web Application*.

2) Для сторінки *Default.aspx* виберіть режим відображення *Split*

3) З панелі елементів управління *Toolbox* (відображається за допомогою меню *View> Toolbox*) перетягніть у форму елемент *ListBox* (з групи *Data*).

4) Вийде наступний код подання:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="ASPNETListBox._Default"
%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Untitled Page</title>
</head>
<body>

  <form id="Form1" runat="server">

    <asp:ListBox ID="ListBox1" runat="server"></asp:ListBox>

  </form>
</body>
</html>
```

і відповідний програмний код:

```
using System;
using System.Collections;
using System.Configuration;
```



```
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;

namespace ASPNETListBox
{
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

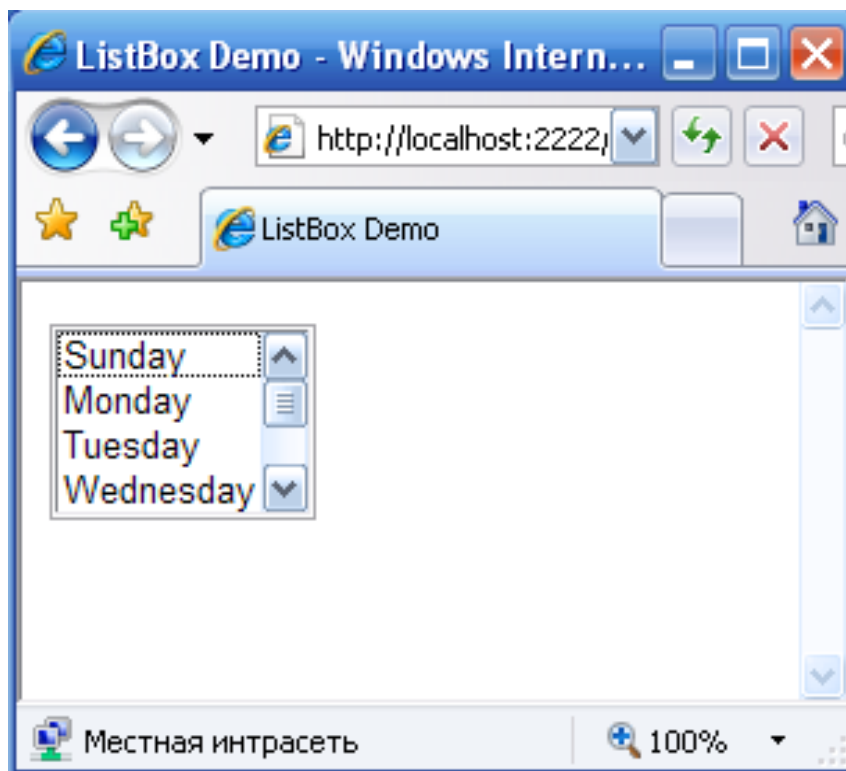
            // Масив рядків, який потрібно вставити

            ArrayList DWeek = new ArrayList();
            DWeek.Add("Sunday");
            DWeek.Add("Monday");
            DWeek.Add("Tuesday");
            DWeek.Add("Wednesday");
            DWeek.Add("Thursday");
            DWeek.Add("Friday");
            DWeek.Add("Saturday");

            // Зв'язування елемента управління з об'єктом DWeek

            ListBox1.DataSource = DWeek;
            ListBox1.DataBind();
        }
    }
}
```

Результат виконання:



Контрольні завдання.

Створіть веб-додаток ASP.NET, що відображає на веб-сторінці зміст таблиці *Exam*, що містить поля: ідентифікатор запису, прізвище студента, назва дисципліни, оцінка. При розробці використовуйте елемент `<asp: GridView>` або `<asp: DataList>`.

Лабораторна робота №11.

Тема «Структура XML документа. XML-схеми.»

Мета роботи:

1. Ознайомлення зі структурою XML-документа, основами синтаксису мови та методами контролю вмісту документа за допомогою схем.

Теоретичний матеріал

XML-документ можна представити у вигляді дерева, яке починається з «кореня» і закінчується «листям».

Нижче наведено приклад XML-документа:

```
<?xml version="1.0" encoding="Windows-1251"?>
<mail>
  <to>user1@domain.ru</to>
  <from>user2@domain.ru </from>
  <subject>Зустріч</heading>
  <body>Зателефонуй мені завтра вранці</body>
</mail>
```

У першому рядку поміщена XML декларація, що описує версію XML і використання кодування.

У наступному рядку описаний кореневий елемент документа.

У наступних чотирьох рядках описано 4 дочірніх елемента відносно кореневого (*to*, *from*, *subject* і *body*).

І в останньому рядку вказується кінець кореневого елемента.

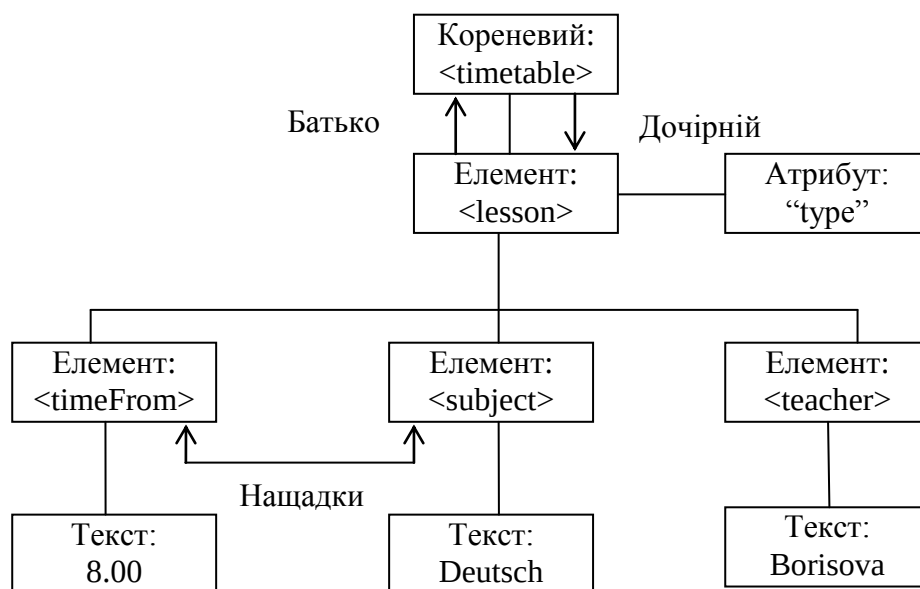
XML-документ повинен містити *кореневий* елемент, який є *батьківським* для всіх інших елементів. Будь-який елемент (крім тих, що знаходяться на самому нижньому

рівні дерева) може мати вкладені елементи (дочірні елементи).

За допомогою термінів *батько*, *дочірній* і *нащадок* описуються відносини між елементами в дереві XML-документа. *Батьки* містять *дочірні елементи*, а дочірні елементи одного рівня називаються *нащадками* (братами або сестрами).

Елементи можуть мати *текстовий вміст* і *атрибути*.

Нижче наведена схема, що показує відношення між елементами і атрибутами цього документа:



Крім того в XML-документі слід виконувати ще кілька нескладних правил:

- Кожен XML-елемент повинен мати закриваючий тег.
- XML-теги чутливі до регістру.
- Повинні дотримуватися правила вкладеності елементів.
- XML-документи повинні мати кореневий елемент, який є батьківським для всіх інших елементів.
- Значення XML-атрибутів повинні братися в подвійні лапки.

Деякі із символів (їх всього 5) мають спеціальне значення в форматі XML. Для їх позначення використовуються альтернативні поєднання:

заміна	СИМВОЛ
<	<
>	>
&	&
'	'
"	"

Коментарі в XML-виділяються також як і в HTML:

`<!-- Це коментар -->`

На відміну від HTML, який ігнорує повтори пробілів, XML зберігає повну довжину рядка з прогалин.

XML-документи, оформлені відповідно до наведених вище правил, називаються «правильно оформленими». Якщо при цьому вони ще й відповідають правилам DTD (Document Type Definition), то вони є і «дійсними».

Дійсні XML-документи містять посилання на DTD файл, наприклад:

`<!DOCTYPE note SYSTEM "mail.dtd">`

DTD схема призначена для визначення структури XML-документа шляхом задання списку допустимих елементів, наприклад:

```
<!DOCTYPE mail [  
  <!ELEMENT note (to,from,subject,body)>  
  <!ELEMENT to    (#PCDATA)>  
  <!ELEMENT from  (#PCDATA)>  
  <!ELEMENT subject (#PCDATA)>  
  <!ELEMENT body  (#PCDATA)>  
>]
```

Консорціум W3C підтримує альтернативну до DTD, засновану на XML-схемі, звану як «*XML Schema*»:

```
<xs:element name="mail">  
  <xs:complexType>  
    <xs:sequence>  
      <xs:element name="to"    type="xs:string"/>  
      <xs:element name="from"  type="xs:string"/>  
      <xs:element name="subject" type="xs:string"/>  
      <xs:element name="body"  type="xs:string"/>  
    </xs:sequence>  
  </xs:complexType>  
</xs:element>
```

Для перевірки дійсності XML-документа можна використовувати спеціальні валідатори, наприклад *W3C валідатор* (<http://validator.w3.org/>). Для перевірки схем також існують спеціальні валідатори, наприклад *валідатор XML Schema* (<http://www.w3.org/2001/03/webdata/xsv>).

Згідно специфікації W3C XML програма повинна припинити обробку XML-документа, як тільки буде виявлена помилка в цьому документі.

1. DTD схема

DTD (Document Type Definition) визначає допустимі будівельні блоки XML-документа, шляхом зазначення списку допустимих елементів і атрибутів.

DTD може описуватися як усередині XML-документа, так і за допомогою зовнішнього посилання.

Приклад внутрішнього опису:

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT mail (to,from,subject,body)>
  <!ELEMENT to   (#PCDATA)>
  <!ELEMENT from  (#PCDATA)>
  <!ELEMENT subject (#PCDATA)>
  <!ELEMENT body  (#PCDATA)>
]>
<note>
  <to>user1@domain.ru</to>
  <from>user2@domain.ru</from>
  <heading>Зустріч</heading>
  <body> Зателефонуй мені завтра вранці </body>
</note>
```

У даному прикладі:

- **!DOCTYPE mail** визначає кореневий елемент документа **mail**.
- **!ELEMENT note** визначає елемент **note**, який містить чотири елементи: "*to, from, subject, body*".
- **!ELEMENT to** визначає елемент **to** типу "**#PCDATA**".
- **!ELEMENT from** визначає елемент **from** типу "**#PCDATA**".
- **!ELEMENT subject** визначає елемент **subject** типу "**#PCDATA**".
- **!ELEMENT body** визначає елемент **body** типу "**#PCDATA**".

Приклад зовнішнього опису:

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "mail.dtd">
```

```
<note>
  <to>user1@domain.ru</to>
  <from>user2@domain.ru</from>
  <heading>Встреча</heading>
  <body> Зателефонуй мені завтра вранці </body>
</note>
```

Для чого необхідне використання DTD?

- За допомогою DTD XML-файли можуть містити опис власного формату.
- Незалежні групи людей можуть обмінюватися даними.
- DTD-схема може бути використана для перевірки дійсності, як документів одержуваних ззовні, так і власних документів.

З точки зору DTD XML-документи все (HTML документи) будуються з наступних будівельних блоків:

- *Елементи (Elements)*. Елементи можуть містити текст, інші елементи або бути порожніми.
- *Атрибути (Attributes)*. Атрибути завжди розміщуються всередині відкриваючого тега елемента. Атрибути завжди записуються у вигляді пари *ім'я / значення*.
- *Entities*.
- *PCDATA*. Означає структуровані символічні дані. Підлягає аналізу за допомогою парсера на наявність спеціальних символів (*entities*) та елементів розмітки.
- *CDATA*. Означає текст, зміст якого не розглядається аналізатором.

При описі структури елементів, що містять вкладені елементи можна використовувати спеціальні оператори, що задають множинність їх входження:

- *'+'*: один і більше разів, наприклад, *<!ELEMENT note (message+)>*;

- ‘*’: нуль і більше разів, наприклад, *<!ELEMENT note (message*)>*;
- ‘?’: нуль і один раз, наприклад, *<!ELEMENT note (message?)>*

Оголошення виду *<!ELEMENT note (message|body)>* вказує на можливість входження в елемент *note* іншого елемента: *message* АБО *body*.

Атрибути елементів описуються у формі:

*<!ATTLIST element-name attribute-name attribute-type
default-value>*

наприклад:

<!ATTLIST payment type CDATA "check">

Що відповідає XML-коду:

<payment type="check" />.

Тип атрибута може приймати одне із значень:

Тип	Опис
CDATA	Символьні данні
(<i>en1</i> <i>en2</i> ..)	Значення зі списку
ID	Унікальний <i>id</i>
IDREF	<i>id</i> іншого елемента
IDREFS	Список інших <i>id</i>
NMTOKEN	Допустиме XML ім'я
NMTOKENS	Список допустимих XML імен
ENTITY	Спеціальні символи
ENTITIES	Список спеціальних символів
NOTATION	Ім'я нотації
xml:	Зумовлене XML ім'я

Значення за замовчуванням може бути одним з наступних:

Значення	Інтерпретація
#REQUIRED	Обов'язково
#IMPLIED	Не обов'язково
#FIXED <i>value</i>	Значення фіксовано

2. XML Schema

Так само як DTD схема XML Schema визначає допустимі будівельні блоки XML-документа.

XML Schema:

- елементи, що входять в документ;
- атрибути, допустимі в документі;
- дочірні елементи;
- порядок дочірніх елементів;
- кількість дочірніх елементів;
- чи може елемент бути порожнім або може містити текст;
- типи елементів і атрибутів;
- фіксовані і значення за замовчуванням елементів і атрибутів.

Передбачається, що в перспективі DTD схеми будуть замінені XML Schema в більшості веб-додатків оскільки XML-схема:

- Розширювана для майбутніх доповнень.
- Більш багаті і могутні виразні можливості.
- Є реалізацією XML.
- Підтримує типи даних.
- Підтримують простори імен.

XML Schema стала W3C рекомендацією з 2001 року.

Розглянемо як приклад XML-документ:

```
<?xml version="1.0" encoding="Windows-1251"?>
```

```
<mail>
  <to>user1@domain.ru</to>
  <from>user2@domain.ru </from>
  <subject>Зустріч</heading>
  <body>Зателефонуй мені завтра вранці</body>
</mail>
```

Структура даного документа може бути розглянута за допомогою наступної *XML Schema*:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.myhp.edu"
xmlns="http://www.myhp.edu"
elementFormDefault="qualified">
  <xs:element name="mail">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="subject" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

У даному прикладі елемент *mail* має тип **ComplexType**, оскільки містить інші елементи. Інші елементи документа мають простий тип, оскільки не включають інших елементів.

Посилання на схему в XML-документі виглядає наступним чином:

```
<?xml version="1.0"?>
<mail
xmlns="http://www.myhp.edu "
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://www.myhp.edu mail.xsd">
  <to>user1@domain.ru</to>
  <from>user2@domain.ru </from>
  <subject>Зустріч</heading>
  <body> Зателефонуй мені завтра вранці body>
</mail>
```

Елемент `<schema>` є кореневим елементом будь-якої схеми XML Schema.

Даний елемент може містити кілька атрибутів, наприклад:

```
<?xml version="1.0"?>
  <xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.myhp.edu"
    xmlns="http://www.myhp.edu"
    elementFormDefault="qualified">
    ...
    ...
  </xs:schema>
```

Наступний фрагмент:

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

Вказує на те, що елементи і типи даних, що використовуються в схемі входять в простір *http://www.w3.org/2001/XMLSchema*. Причому, всі елементи і типи даних з цього простору імен повинні мати префікс **xs**.

Фрагмент:

```
targetNamespace="http://www.myhp.edu"
```

вказує на те, що елементи, які визначаються в схемі входять в простір «*http://www.myhp.edu*».

У фрагменті:

```
xmlns="http://www.myhp.edu"
```

вказується, що простором імен за замовчуванням є «http://www.myhp.edu».

Наступний фрагмент:

```
elementFormDefault="qualified"
```

вказує на те, що будь-які елементи, оголошені в схемі, повинні належати простору імен.

Посилання на зовнішню схему може виглядати наступним чином:

```
<?xml version="1.0"?>
<mail xmlns="http://www.myhp.edu"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.myhp.edu mail.xsd">
  <to>user1@domain.ru</to>
  <from>user2@domain.ru </from>
  <subject>Зустріч</heading>
  <body> Зателефонуй мені завтра вранці </body>
</mail>
```

Тут рядок:

```
xsi:schemaLocation="http://www.myhp.edu mail.xsd">
```

вказує на місце розташування файлу схеми.

3. Визначення простих елементів

Синтаксис для визначення простого елементу:

```
<xs:element name="xxx" type="yyy"/>
```

де *xxx* - ім'я елемента і *ууу* тип даних елемента.

Вбудованими типами даних елементів є наступні:

- `xs:string`;
- `xs:decimal`;
- `xs:integer`;
- `xs:boolean`;
- `xs:date`;
- `xs:time`.

Наприклад, фрагмент XML-документа:

```
<lastname>Refsnes</lastname>  
<age>36</age>  
<dateborn>1970-03-27</dateborn>
```

описується в схемі наступним чином:

```
<xs:element name="lastname" type="xs:string"/>  
<xs:element name="age" type="xs:integer"/>  
<xs:element name="dateborn" type="xs:date"/>
```

Наступні фрагменти:

```
<xs:element name="color" type="xs:string" default="red"/>
```

i

```
<xs:element name="color" type="xs:string" fixed="red"/>
```

описують значення елемента за замовчуванням і фіксоване значення відповідно.

Усі атрибути описуються простими типами даних.

Прості елементи не можуть мати атрибутів. Якщо елемент має атрибути, він розглядається як той, що має складний тип. Сам атрибут розглядається завжди як має простий тип.

Опис атрибуту зазвичай дається наступним чином:

```
<xs:attribute name="xxx" type="yyy"/>
```

де *xxx* – ім'я атрибута, а *yyy* – визначає тип даних атрибута.

Вбудованими типами даних для атрибутів є наступні:

- xs:string;
- xs:decimal;
- xs:integer;
- xs:boolean;
- xs:date;
- xs:time.

Наприклад, XML-елемент з атрибутом:

```
<lastname lang="EN">Smith</lastname>
```

описується відповідною схемою:

```
<xs:attribute name="lang" type="xs:string"/>
```

Значення за замовчуванням і фіксоване значення атрибуту описуються наступним чином:

```
<xs:attribute name="lang" type="xs:string" default="EN"/>
```

і

```
<xs:attribute name="lang" type="xs:string" fixed="EN"/>
```

відповідно.

Для опису обов'язкових атрибутів (за замовчуванням всі атрибути є необов'язковими) використовується наступна нотація:

```
<xs:attribute name="lang" type="xs:string" use="required"/>
```

4. Обмеження на зміст

Для завдання допустимих значень величин XML-елементів та атрибутів можна використовувати обмеження. Обмеження на значення XML-елементів називаються *фасетами*.

Список можливих обмежень наводиться нижче:

Обмеження	Опис
enumeration	Визначає список допустимих значень
fractionDigits	Вказує максимальну кількість десяткових позицій. Має бути невід'ємним.
length	Вказує точне число символів або елементів у списку. Має бути невід'ємним
maxExclusive	Вказує верхню межу числових значень
maxInclusive	Вказує верхню межу числових значень
maxLength	Вказує на максимальну кількість символів або елементів у списку. Має бути невід'ємним
minExclusive	Визначає нижню межу числових значень, що не включається
minInclusive	Визначає нижню межу числових значень, що включається
minLength	Визначає мінімальне число символів або елементів у списку. Повинно бути невід'ємним
pattern	Визначає точно послідовність символів (шаблон), яка є допустимою
totalDigits	Визначає точне число цифр. Повинно бути невід'ємним
whiteSpace	Визначає як обробляти не відображаються символи (пробіл, табуляція і ін.)

Приклади обмеження:

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```



```
</xs:restriction>
</xs:simpleType>
</xs:element>
```

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9]{8}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

5. Опис складних елементів

При описі документів, що мають складну ієрархічну структуру, можна спочатку визначити всі елементи і атрибути, а потім посилатися на них.

Розглянемо наступний приклад:

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

Складний елемент у схемі можна визначити наступним чином:

```
<xs:element name="firstname" type="xs:string"/>
<xs:element name="lastname" type="xs:string"/>

<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="firstname"/>
      <xs:element ref="lastname"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Інший підхід полягає в повторному використанні іменованих елементів.

1. Елемент «*employee*» element може бути оголошений безпосередньо на ім'я:

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Оскільки дочірні елементи «*firstname*» і «*lastname*» оточені індикатором `<sequence>` то вони повинні в документі з'являтися саме в цій послідовності.

2. Елемент «*employee*» може мати атрибут типу, який посилається на ім'я складного типу:

```
<xs:element name="employee" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Кілька елементів можуть посилатися на той же самий складний тип:

```
<xs:element name="employee" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
<xs:element name="member" type="personinfo"/>
```

```
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Також можна будувати складний елемент на базі існуючого складного елемента з додаванням кількох елементів:

```
<xs:element name="employee" type="fullpersoninfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Порожній складний елемент не може мати змісту, тільки атрибути.

За допомогою спеціальних індикаторів можна вказувати, як елементи можуть використовуватися в документах.

Всього використовується сім індикаторів:

- Індикатор порядку (*Order*):
 - All;
 - Choice;
 - Sequence.
- Індикатор входження (*Occurrence*):
 - maxOccurs;
 - minOccurs.
- Групові (*Group*) індикатори:
 - Group name;
 - attributeGroup name.

Приклад:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Порядок виконання лабораторної роботи

1. Опис структури XML, документа за допомогою DTD-схеми.

1) Створення XML-документа *timetable.xml*. Зміст документа наведено нижче:

```
<?xml version="1.0"?>
<timetable>
  <day dayOfWeek="Monday">
    <lesson type="practical">
      <timeFrom>08.00</timeFrom>
      <timeTo>09.30</timeTo>
      <subject>Deutsch</subject>
```

```
        <teacher>Borisova</teacher>
        <room>216</room>
    </lesson>
    <lesson type="lecture">
        <timeFrom>09.40</timeFrom>
        <timeTo>11.10</timeTo>
        <subject>SAP Administration</subject>
        <teacher>Egorov</teacher>
        <room>384</room>
    </lesson>
    <lesson type="practical">
        <timeFrom>11.20</timeFrom>
        <timeTo>12.50</timeTo>
        <subject>SAP Administration</subject>
        <teacher>Petrov</teacher>
        <room>384</room>
    </lesson>
</day>
</timetable>
```

Завантажте цей файл в браузері. Зверніть увагу на ієрархічну структуру документа.

2) Додайте в початок файлу внутрішню DTD-схему:

```
<?xml version="1.0"?>

<!DOCTYPE timetable[
<!ELEMENT timetable (day*)>
<!ELEMENT day (lesson+)>
<!ELEMENT lesson (timeFrom, timeTo, subject, teacher, room?)>
<!ELEMENT timeFrom (#PCDATA)>
<!ELEMENT timeTo (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT teacher (#PCDATA)>
<!ELEMENT room (#PCDATA)>
<!ATTLIST day
    dayOfWeek CDATA #REQUIRED>
<!ATTLIST lesson
    type CDATA #REQUIRED>
```

] >

3) Збережіть файл і перевірте його на правильність за допомогою програми-валідатора, наприклад, за адресою: <http://validator.w3.org>. Спробуйте видалити який-небудь елемент або атрибут, перевірте документ знову за допомогою валідатора.

4) Винесіть схему в зовнішній файл з розміщенням на ньому посилання всередині XML-документа. Перевірте, як буде завантажуватися XML-документ.

2. Опис структури XML-документа за допомогою XML Schema.

1) Створення XML-документа *timetable2.xml*. Зміст документа наведено нижче.

```
<?xml version="1.0"?>

<timetable      xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="timetable.xsd">

<day dayOfWeek="Monday">
  <lesson type="practical">
    <timeFrom>08.00</timeFrom>
    <timeTo>09.30</timeTo>
    <subject>Deutsch</subject>
    <teacher>Borisova</teacher>
    <room>216</room>
  </lesson>
  <lesson type="lecture">
    <timeFrom>09.40</timeFrom>
    <timeTo>11.10</timeTo>
    <subject>SAP Administration</subject>
    <teacher>Egorov</teacher>
    <room>384</room>
  </lesson>
  <lesson type="practical">
    <timeFrom>11.20</timeFrom>
```

```
<timeTo>12.50</timeTo>
<subject>SAP Administration</subject>
<teacher>Petrov</teacher>
<room>384</room>
</lesson>
</day>
</timetable>
```

2) Створення файлу *timetable.xsd*, що містить XML Schema:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name = "timetable">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref = "day" minOccurs = "0"
maxOccurs = "unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name = "day">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref = "lesson" maxOccurs =
"unbounded"/>
      </xs:sequence>
      <xs:attribute name = "dayofWeek" use = "required"
type = "xs:string"/>
    </xs:complexType>
  </xs:element>
  <xs:element name = "lesson">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref = "timeFrom"/>
        <xs:element ref = "timeTo"/>
        <xs:element ref = "subject"/>
        <xs:element ref = "teacher"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
        <xs:element ref = "room" minOccurs = "0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name = "timeFrom" type = "xs:string"/>
  <xs:element name = "timeTo" type = "xs:string"/>
  <xs:element name = "subject" type = "xs:string"/>
  <xs:element name = "teacher" type = "xs:string"/>
  <xs:element name = "room" type = "xs:string"/>
</xs:schema>
```

3) Перевірте схему на правильність за допомогою програми-валідатора, наприклад, за адресою: <http://www.w3.org/2001/03/webdata/xsv>.

Контрольне завдання

У наведеному нижче XML-документі описана екзаменаційна відомість:

```
<gradeReport id="120851">
  <date>10-06-2008</date>
  <subject>Computer Science Fundamentals</subject>
  <examiner>prof.Litvinov</examiner>
  <gradeList>
    <gradeRecord id="1">
      <student>Ivanov</student>
      <grade>4</grade>
    </gradeRecord>
    <gradeRecord id="2">
      <student>Petrov</student>
      <grade>3</grade>
    </gradeRecord>
    <gradeRecord id="3">
      <student>Sidorov</student>
      <grade>5</grade>
    </gradeRecord>
  </gradeList>
</gradeReport>
```



```
</gradeList>  
</gradeReport>
```

Опишіть структуру даного документа за допомогою *DTD* схеми і *XML Schema*. Перевірте обидві схеми за допомогою відповідних програм-валідаторів.

Лабораторна робота №12.

Тема «Програмна обробка XML документів за допомогою XML DOM.»

Мета роботи:

1. Ознайомлення з основними принципами XML DOM.
2. Ознайомлення з методами програмної обробки XML документів шляхом маніпулювання вузлами дерева документа.

Теоретичний матеріал

XML DOM визначає об'єкти і властивості всіх XML елементів і методи (інтерфейс) для доступу до них. Інакше кажучи, XML DOM описує яким чином необхідно отримувати, змінювати, додавати і видаляти XML елементи.

У відповідності з моделлю DOM все, що міститься всередині XML документа – є вузлом. Тобто XML документ представляється у вигляді дерева вузлів, якими є елементи, атрибути та текст.

Оскільки структури HTML і XML документів дуже схожі, а HTML DOM і XML DOM є частинами більш загального стандарту DOM, то і багато аспектів HTML DOM легко переносяться в XML DOM. Тому основну увагу буде приділено саме специфічним особливостям саме XML DOM. Рекомендується попереднє ознайомлення з лабораторною роботою 3.

1. Структурний аналіз (парсинг) XML

Всі сучасні браузері мають вбудовані XML аналізатори (парсери) для читання і обробки XML.

Аналізатор зчитує XML документ, розміщує його в пам'яті і перетворює в XML DOM об'єкт, доступний для мов програмування. Всі приклади тут наведені на JavaScript.

Є деякі відмінності між аналізаторами в Microsoft і в інших браузерах. Перший підтримує як завантаження XML файлів, так і текстових рядків, що містять XML код, в той час як в інших браузерах використовуються роздільні аналізатори. При цьому всі аналізатори мають функції для переміщення по дереву XML документа, доступу, вставки і видалення вузлів в дереві.

Розглянемо приклад завантаження XML об'єктів (файлів і рядків) за допомогою XML аналізатора Microsoft.

```
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");  
xmlDoc.async="false";  
xmlDoc.load("timetable.xml");
```

У першому рядку програми створюється порожній об'єкт XML документа Microsoft. Далі для запобігання роботи сценарію до повного завантаження документа прапор асинхронності встановлюється в «false». У третьому рядку міститься інструкція завантажити XML файл «*timetable.xml*».

У наступному приклад відбувається завантаження рядки з XML кодом для подальшого аналізу.

```
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");  
xmlDoc.async="false";  
xmlDoc.loadXML(txt);
```

Слід звернути на різницю між методами *load()* і *loadXML()* за їх призначенням.

Зауваження. Сучасні браузери не допускають міждоменні звернення до файлів з міркувань безпеки, тобто сама веб-сторінка (з програмним кодом) і XML файл

повинні фізично знаходитися на одному сервері. В іншому випадку браузер видасть повідомлення про помилку доступу.

Нижче наведені також кросплатформені реалізації завантаження XML *файлу* і XML *рядка* відповідно.

```
<html>
<body>
<script type="text/javascript">
try //Internet Explorer
{
    xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
}
catch(e)
{
    try //Firefox, Mozilla, Opera, etc.
    {

        xmlDoc=document.implementation.createDocument("", "", null);
    }
    catch(e) {alert(e.message)}
}
try
{
    xmlDoc.async=false;
    xmlDoc.load("timetable.xml");
    document.write("xmlDoc is loaded, ready for use");
}
catch(e) {alert(e.message)}
</script>
</body>
</html>
```

```
<html>
<body>
<script type="text/javascript">
text="<timetable>";
text=text+"<lesson>";
text=text+"<timeFrom>08.00</timeFrom>";
```

```
text=text+"<subject>Deutsch</subject>";
text=text+"<teacher>Borisova</teacher>";
text=text+"</lesson>";
text=text+"/<timetable>";

try //Internet Explorer
{
    xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
    xmlDoc.async="false";
    xmlDoc.loadXML(text);
}
catch(e)
{
    try //Firefox, Mozilla, Opera, etc.
    {
        parser=new DOMParser();
        xmlDoc=parser.parseFromString(text,"text/xml");
    }
    catch(e) {alert(e.message)}
}
document.write("xmlDoc is loaded, ready for use");
</script>
</body>
</html>
```

2. Програмний інтерфейс XML DOM

В рамках DOM моделі **XML** можна розглядати як безліч вузлових об'єктів. Доступ до них здійснюється за допомогою *JavaScript* або інших мов програмування. Програмний інтерфейс DOM включає в себе набір стандартних *властивостей* і *методів*.

Властивості представляють деякі сутності (наприклад, <day>), а *методи* – дії над ними (наприклад, додати <lesson>).

У XML DOM використовуються практично ті ж властивості і методи, що і в HTML DOM.

Наприклад, результатом виконання наступного нижче JavaScript коду буде текстовий зміст елемента `<subject>` у файлі *timetable.xml*.

```
txt = xmlDoc.getElementsByTagName ("subject") [0]  
      .childNodes [0] .nodeValue;
```

Результат: «*Deutsch*».

В рамках DOM XML можливі 3 способи доступу до вузлів:

1. За допомогою методу *getElementsByTagName* (name). При цьому повертаються всі вузли з вказаним ім'ям тегу (у вигляді індексованого списку). Перший елемент в списку має нульовий індекс.
2. Шляхом обходу вузлів дерева з використанням циклічних конструкцій.
3. Шляхом переміщення по дереву з використанням відносин між вузлами.

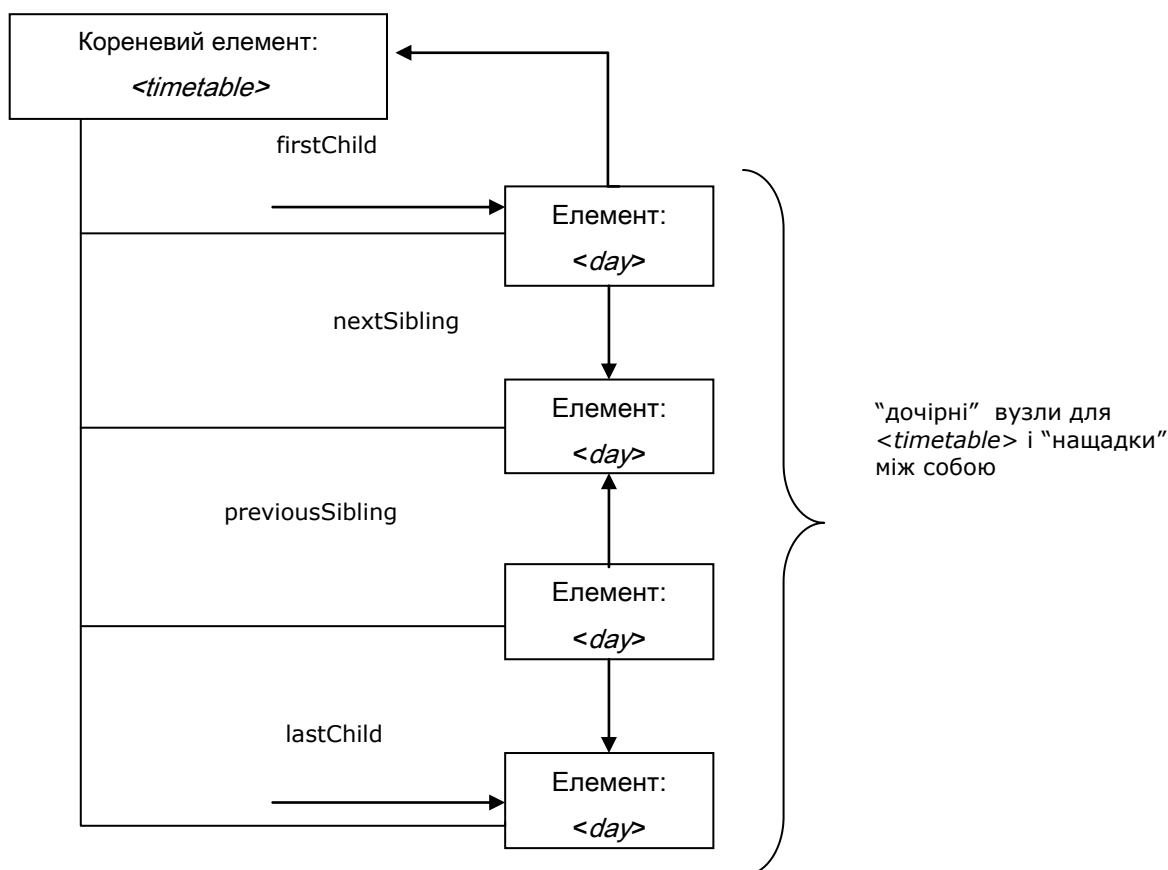
Для визначення довжини списку вузлів використовується властивість *length*.

3. Переміщення між вузлами дерева

У XML DOM відносини між вузлами визначені у вигляді наступних властивостей вузлів:

- *parentNode*;
- *childNodes*;
- *firstChild*;
- *lastChild*;
- *nextSibling*;
- *previousSibling*.

Характер відносин між вузлами представлений на наступному малюнку:



4. Ігнорування порожніх текстових вузлів

Firefox і деякі інші браузері сприймають невідображаємі символи як текстові вузли (на відміну від *Internet Explorer*). Така ситуація призводить до проблем при використанні властивостей *firstChild*, *lastChild*, *nextSibling*, *previousSibling*. Для того, щоб ігнорувати такі порожні текстові вузли можна використовувати наступний прийом:

```

function get_nextSibling(n)
{
    y = n.nextSibling;
    while (y.nodeType!=1)
    {
        y = y.nextSibling;
    }
    return y;
}
  
```

Оскільки вузли *елементів* мають тип 1, то в тому випадку, коли вузол-нащадок не є вузлом *елемента*, відбуватиметься переміщення до наступного вузла доти, поки не буде знайдений вузол *елементу*.

5. Зміна значення атрибута

Вузли атрибутів можуть брати текстові значення. Зміна цього значення реалізується або через метод *setAttribute ()*, або через властивість вузла атрибуту *nodeValue*

Метод *setAttribute ()* змінює значення існуючого атрибуту або створює новий атрибут.

Наприклад:

```
xmlDoc = loadXMLDoc("timetable.xml");  
x = xmlDoc.getElementsByTagName('lesson');  
x[0].setAttribute("type","lab");
```

Властивість *nodeValue* можна використовувати для зміни значення атрибуту вузла:

```
xmlDoc = loadXMLDoc("timetable.xml");  
x = xmlDoc.getElementsByTagName("lesson")[0];  
y = x.getAttributeNode("type");  
y.nodeValue = "lab";
```

Видалення вузла з дерева реалізується за допомогою методу *removeChild()*:

```
xmlDoc=loadXMLDoc("timetable.xml ");  
y = xmlDoc.getElementsByTagName("lesson")[0];  
xmlDoc.documentElement.removeChild(y);
```


6. Властивості об'єкту Node

IE: Internet Explorer, *F*: Firefox, *O*: Opera, *W3C*: (Стандарт).

Властивість	Опис	IE	F	O	W3C
		Версія			
baseURI	Повертає абсолютний URI вузла	Hi	1	Hi	Так
childNodes	Повертає властивість NodeList (список дод.вузлів)	5	1	9	Так
firstChild	Повертає перший дочірній вузол	5	1	9	Так
lastChild	Повертає останній дочірній вузол	5	1	9	Так
localName	Повертає локальну частину імені вузла	Hi	1	9	Так
namespaceURI	Повертає URI вузла в просторі імен	Hi	1	9	Так
nextSibling	Повертає наступний дочірній вузол	5	1	9	Так
nodeName	Повертає ім'я вузла в залежності від типу	5	1	9	Так
nodeType	Повертає тип вузла	5	1	9	Так
nodeValue	Встановлює або повертає значення вузла в залежності від типу	5	1	9	Так
ownerDocument	Повертає кореневий елемент (об'єкт document) для вузла	5	1	9	Так
parentNode	Повертає батьківський вузол	5	1	9	Так
prefix	Встановлює або повертає префікс простору імен вузла	Hi	1	9	Так
previousSibling	Повертає безпосередньо передуючий вузол	5	1	9	Так
textContent	Встановлює або повертає текстовий зміст сайту	Hi	1	Hi	Так
xml	Повертає XML код вузла	5	Hi	Hi	Hi

7. Методи об'єкту Node

Метод	Опис	IE	F	O	W3C
appendChild()	Додати новий вузол в кінець списку дочірніх вузлів	5	1	9	Так
cloneNode()	Клонування вузла	5	1	9	Так
compareDocumentPosition()	Порівняння позицій двох вузлів	Hi	1	Hi	Так
getFeature(feature, version)	Повертає об'єкт DOM, який реалізує спеціалізований API			Hi	Так
getUserData(key)	Повертає об'єкт, що асоціюється з ключем поточного вузла. Перед цим об'єкт повинен бути асоційований з поточним вузлом шляхом виклику <i>setUserData</i> з тим же ключем			Hi	Так
hasAttributes()	Повертає істинне значення, якщо вузол має атрибути	Hi	1	9	Так
hasChildNodes()	Повертає істинне значення, якщо вузол має дочірні вузли	5	1	9	Так
insertBefore ()	Вставляє новий вузол перед існуючим вузлом	5	1	9	Так
isDefaultNamespace (URI)	Визначає, чи є зазначений <i>namespaceURI</i> значенням за замовчуванням			Hi	Так
isEqualNode()	Перевіряє рівність двох вузлів	Hi	Hi	Hi	Так
isSameNode()	Перевіряє ідентичність двох вузлів	Hi	1	Hi	Так
isSupported(featur	Визначає – чи			9	Так

e,version)	підтримується зазначена характеристика вузлом				
removeChild()	Видаляє дочірній вузол	5	1	9	Так
replaceChild()	Замінює дочірній вузол	5	1	9	Так
setUserData(key,data,handler)	Асоціює об'єкт з ключем у вузлі			Ні	Так

Порядок виконання лабораторної роботи

При виконанні даної лабораторної роботи потрібно XML-документ *timetable.xml*, який використовувався в лабораторній роботі 11:

```
<?xml version="1.0"?>
<timetable>
  <day dayOfWeek="Monday">
    <lesson type="practical">
      <timeFrom>08.00</timeFrom>
      <timeTo>09.30</timeTo>
      <subject>Deutsch</subject>
      <teacher>Borisova</teacher>
      <room>216</room>
    </lesson>
    <lesson type="lecture">
      <timeFrom>09.40</timeFrom>
      <timeTo>11.10</timeTo>
      <subject>SAP Administration</subject>
      <teacher>Egorov</teacher>
      <room>384</room>
    </lesson>
    <lesson type="practical">
      <timeFrom>11.20</timeFrom>
      <timeTo>12.50</timeTo>
      <subject>SAP Administration</subject>
      <teacher>Petrov</teacher>
      <room>384</room>
    </lesson>
  </day>
</timetable>
```

1. Створення JavaScript сценарію завантаження XML-документу.

Створіть текстовий файл *loadxmldoc.js*, що містить опис функції завантаження XML-документу:

```
function loadXMLDoc(dname)
{
  try //Internet Explorer
  {
    xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
  }
  catch(e)
  {
    try //Firefox, Mozilla, Opera, etc.
    {
      xmlDoc=document.implementation.createDocument("", "", null);
    }
    catch(e) {alert(e.message)}
  }
  try
  {
    xmlDoc.async=false;
    xmlDoc.load(dname);
    return(xmlDoc);
  }
  catch(e) {alert(e.message)}
  return(null);
}
```

і збережіть його в тій же папці, де знаходиться файл *timetable.xml*.

Код виклику цієї функції може виглядати наступним чином:

```
<html>
<head>
<script type="text/javascript" src="loadxmldoc.js">
```

```
</script>
</head>

<body>
<script type="text/javascript">
    xmlDoc=loadXMLDoc("timetable.xml");
    document.write("xmlDoc is loaded, ready for use");
</script>
</body>
</html>
```

2. Переміщення по дереву вузлів.

Підготуйте наступну HTML сторінку:

```
<html>
<head>
<script type="text/javascript" src="loadxmldoc.js">
</script>
</head>
<body>
<script type="text/javascript">
    xmlDoc = loadXMLDoc("timetable.xml");
    x = xmlDoc.getElementsByTagName("subject");
    for (i=0; i<x.length; i++)
    {
        document.write(x[i].childNodes[0].nodeValue);
        document.write("<br />");
    }
</script>
</body>
</html>
```

Після завантаження сторінки в браузері можна буде побачити наступний результат:

```
Deutsch
SAP Administration
SAP Administration
```

3. Зміна значення елемента.

Наступний приклад демонструє зміну значення елемента *<subject>*:

```
xmlDoc=loadXMLDoc("timetable.xml");
x=xmlDoc.getElementsByTagName("subject")[0].childNodes[0];
x.nodeValue="Java programming";

x = xmlDoc.getElementsByTagName("subject");
for (i=0; i<x.length; i++)
{
    document.write(x[i].childNodes[0].nodeValue);
    document.write("<br />");
}
```

Внесіть відповідні зміни в попередню сторінку і завантажте її в браузері.

4. Переміщення по вузлах дерева з використанням відносин між ними.

Наступний код показує, як використовуючи відносини *firstChild* і *nextSibling* можна отримати для поточного вузла список його дочірніх вузлів:

```
x = xmlDoc.getElementsByTagName("lesson")[0].childNodes;
y = xmlDoc.getElementsByTagName("lesson")[0].firstChild;

for (i = 0; i < x.length; i++)
{
    if (y.nodeType == 1)
    {
        document.write(y.nodeName + "<br />");
    }
    y=y.nextSibling;
}
```

Внесіть необхідні зміни в *html* сторінку і завантажте її в браузері.

Контрольне завдання

У наведеному нижче XML документі описана екзаменаційна відомість:

```
<gradeReport id="120851">
<date>10-06-2008</date>
<subject>Computer Science Fundamentals</subject>
<examiner>prof.Litvinov</examiner>
  <gradeList>
    <gradeRecord id="1">
      <student>Ivanov</student>
      <grade>4</grade>
    </gradeRecord>
    <gradeRecord id="2">
      <student>Petrov</student>
      <grade>3</grade>
    </gradeRecord>
    <gradeRecord id="3">
      <student>Sidorov</student>
      <grade>5</grade>
    </gradeRecord>
  </gradeList>
</gradeReport>
```

1. Використовуючи методи DOM XML, сформууйте HTML сторінку, що містить таблицю з трьох стовпців: *номер, студент, оцінка*.

2. Використовуючи методи DOM XML, замініть цифрові значення оцінок їх словесними еквівалентами, наприклад «4» на «good».

Лабораторна робота №13.

Тема «Форматування і перетворення XML-документу за допомогою XSL. XSLT перетворення XML-документу.»

Мета роботи:

1. Знайомство з методами форматування і перетворення XML документів на основі XSLT перетворень.

Теоретичний матеріал

XSLT можна визначити наступним чином:

- XSLT позначає *XSL Transformations*.
- XSLT є найважливішою частиною XSL перетворення.
- XSLT дозволяє перетворювати один XML в інший XML документ.
- XSLT використовує XPath для переміщення по структурі XML документа.
- XSLT є W3C рекомендацією.

XSLT використовується для перетворення XML-документу в інший XML-документ або в документ іншого розпізнаваного браузером типу, наприклад HTML або XHTML. Зазвичай XSLT робить це, перетворюючи кожен XML елемент у відповідний йому (X) HTML елемент.

За допомогою XSLT можна додавати або видаляти елементи і атрибути в результуючому документі. Також можливе перегрупування і сортування елементів, фільтрація елементів при відображенні і багато іншого.

Фактично XSLT перетворює вихідне XML дерево в результуюче XML дерево.

XSLT використовує XPath для пошуку інформації в XML документі, тобто XPath є інструментів навігації по елементах і атрибутах XML документів.

У процесі перетворення XSLT використовує XPath для пошуку частин вихідного документа, що відповідають одному або більше заданих шаблонів. Коли відповідність знайдено, XSLT перетворює знайдену частину вихідного документа у відповідну частину результуючого документа.

XSLT є W3C рекомендацією з листопада 1999 року.

Переважає більшість браузерів мають підтримку XML і XSLT.

Internet Explorer

Починаючи з 6 версії, *Internet Explorer* підтримує XML, простори імен, CSS, XSLT і XPath. Версія 5 не сумісна з офіційною W3C XSL Рекомендацією.

Mozilla Firefox

Починаючи з версії 1.0.2, Firefox підтримує XML і XSLT (CSS).

Mozilla

Mozilla містить *Expat for XML* парсер підтримує відображення XML + CSS. Також має підтримку простору імен. Реалізує XSLT перетворення.

Netscape

Починаючи з версії 8, *Netscape* використовує ядро на основі *Mozilla*, і тому має таку ж підтримку XML / XSLT.

Opera

Починаючи з версії 9, *Opera* підтримує XML і XSLT (CSS). Версія 8 підтримує тільки XML + CSS.

1. Оголошення XSL

Кореневим елементом, що вказує на те, чи документ є XSL таблицею стилів є:

`<Xsl: stylesheet>`

або повністю рівноцінний йому

<Xsl: transform>

Відповідно до W3C XSLT Рекомендацією, коректний спосіб оголошення таблиці стилів XSL виглядає наступним чином:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

або

```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Для того щоб XSLT елементи, атрибути та характеристики були доступні на початку документа необхідно оголосити простір імен XSLT:

xmlns: xsl = "http://www.w3.org/1999/XSL/Transform"

що вказує на офіційний простір імен W3C XSLT. При цьому також слід вказати атрибут *version = "1.0"*.

2. Реалізація перетворення за допомогою сценарію.

XSLT перетворення з XML в XHTML, що виконується самими браузерами на основі таблиці стилів XSL не завжди є бажаним, оскільки може підтримуватися не усіма браузерами.

Використання як альтернативи JavaScript дозволяє:

- Виконувати перевірку типу браузера.
- Використовувати відповідні таблиці стилів залежно від типу браузера і потреб користувачів.

Іншим рішенням для браузерів, що не підтримують XSLT є перетворення XML в XHTML на веб-сервері.

Порядок виконання лабораторної роботи

1. Реалізація XSLT перетворення XML документа в XHTML засобами браузера.

1) Створіть файл *ttable.xml* такого змісту:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="ttable.xsl"?>
<timetable>
  <lesson>
    <timeFrom>09.30</timeFrom>
    <timeTo>11.00</timeTo>
    <subject>Deutsch</subject>
    <teacher>Borisova</teacher>
  </lesson>
  <lesson>
    <timeFrom>11.10</timeFrom>
    <timeTo>12.20</timeTo>
    <subject>SAP Administration</subject>
    <teacher>Petrov</teacher>
  </lesson>
  <lesson>
    <timeFrom>12.40</timeFrom>
    <timeTo>14.00</timeTo>
    <subject>SAP Administration</subject>
    <teacher>Ivanov</teacher>
  </lesson>
  <lesson>
    <timeFrom>14.00</timeFrom>
    <timeTo>15.20</timeTo>
    <subject>Wen-technology</subject>
    <teacher>Loktev</teacher>
  </lesson>
</timetable>
```

2) Підготуйте для нього відповідний файл таблиці стилів (*ttable.xsl*):

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet                                version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
    <h2>My academical timetable</h2>
    <table border="1">
      <tr bgcolor="#a5abfa">
        <th align="left">Time From</th>
        <th align="left">Subject</th>
        <th align="left">Teacher</th>
      </tr>
      <xsl:for-each select="timetable/lesson">
        <tr>
          <td><xsl:value-of select="timeFrom"/></td>
          <td><xsl:value-of select="subject"/></td>
          <td><xsl:value-of select="teacher"/></td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Перший рядок у файлі *ttable.xml* виду:

```
<? Xml-stylesheet type = "text / xsl" href = "ttable.xsl" ?>
```

являє собою посилення на відповідну таблицю стилів.

У даному прикладі для опису шаблону був використаний елемент `<xsl: template>`.

Атрибут *match* застосовується для зв'язування XML елемента з шаблоном. Значення атрибута *match* є вираз

XPath. В даному випадку *match* = "/" вказує на весь документ.

Зміст елемента *<xsl:template>* описує фрагмент HTML коду у вихідному документі. Елемент *<xsl:value-of>* може бути використаний для вилучення значення XML елементу і додавання його у вихідний потік при перетворенні. Значення атрибута *select* є вираженням *XPath*, яке нагадує нотацію, що використовується в файловій системі; знак (/) вказує на звернення до підкаталогу.

Елемент *<xsl:for-each>* дозволяє обирати кожен елемент XML з великої кількості вузлів.

3) При наявності відповідного браузера можна буде побачити наступний результат після завантаження файлу *ttable.xml*:

My academical timetable		
Time From	Subject	Teacher
09.30	Deutsch	Borisova
11.10	SAP Administration	Petrov
12.40	SAP Administration	Ivanov
14.00	Wen-technology	Loktev

Для фільтрації виведення елементів можна додати атрибут елемента *<xsl: for-each>*, що задає критерій відбору елементів. Наприклад:

```
<Xsl: for-each select = "timetable / lesson [subject = 'SAP
Administration']">
```

Перевірте роботу даного фільтра.

Допустиме використання наступних операторів для опису фільтру:

- = (дорівнює);
- != (не дорівнює);
- & lt; (менше ніж);
- & gt; (більше ніж).

2. Реалізація перетворення за допомогою *JavaScript*.

1) Підготуйте *html* файл, що містить код:

```
<html>
  <head>
    <script>
      function loadXMLDoc(fname)
      {
        var xmlDoc;
// Код для IE
        if (window.ActiveXObject)
        {
          xmlDoc=new
            ActiveXObject("Microsoft.XMLDOM");
        }
// Код для Mozilla, Firefox, Opera и др.
        else if (document.implementation
          && document.implementation.createDocument)
        {
          xmlDoc=document.implementation.createDocument(
            "", "", null);
        }
        else
        {
          alert('Your browser cannot handle this script');
        }
        xmlDoc.async=false;
        xmlDoc.load(fname);
        return(xmlDoc);
      }

      function displayResult()
      {
        xml=loadXMLDoc("ttable.xml");
```

```
        xsl=loadXMLDoc("ttable.xsl");
// Код для IE
        if (window.ActiveXObject)
        {
            ex=xml.transformNode(xsl);

            document.getElementById("example").innerHTML=
            ex;
        }
// Код для Mozilla, Firefox, Opera и др.
        else if (document.implementation
        && document.implementation.createDocument)
        {
            xsltProcessor=new XSLTProcessor();
            xsltProcessor.importStylesheet(xsl);
            resultDocument =
            xsltProcessor.transformToFragment(xml,document);

            document.getElementById("example").appendChild(
            resultDocument);
        }
    }
</script>
</head>
<body id="example" onLoad="displayResult()">
</body>
</html>
```

Функція *loadXMLDoc()* завантажує XML і XSL файли залежно від типу браузера.

Функція *displayResult()* використовується для відображення XML файлу в стилі, що задається XSL файлом. Вона виконує наступні дії:

- завантажує XML і XSL;
- визначає тип браузеру;
- якщо браузер підтримує *ActiveX* об'єкти:
 - за допомогою методу *transformNode()* таблиця стилів XSL застосовується до XML документу;

- формується тіло поточного документу.
 - якщо браузер клієнта не підтримує *ActiveX* об'єкти:
 - створюється новий об'єкт *XSLTProcessor* і в нього імпортується XSL файл;
 - за допомогою методу *transformToFragment()* таблиця стилів XSL застосовується до XML документу;
 - формується тіло поточного документа.
- 2) Завантажте цей документ у веб-браузері.

3. Реалізація перетворення за допомогою JavaScript

1) Підготуйте файл, що містить наступний код сценарію на мові ASP:

```
<%  
'Load XML  
set xml = Server.CreateObject("Microsoft.XMLDOM")  
xml.async = false  
xml.load(Server.MapPath("ttable.xml"))  
  
'Load XSL  
set xsl = Server.CreateObject("Microsoft.XMLDOM")  
xsl.async = false  
xsl.load(Server.MapPath("ttable.xsl"))  
  
'Transform file  
Response.Write(xml.transformNode(xsl))  
>
```

На початку сценарію створюється екземпляр парсеру *Microsoft XML parser* (XMLDOM), і XML файл завантажується в пам'ять. Далі створюється ще один примірник парсеру, і XSL файл завантажується в пам'ять. В останньому рядку виконується еретворення XML файлу з використанням XSL файлу в XHTML, який відправляється назад браузеру.

2) Перевірте роботу сценарію в веб-браузері.

Контрольне завдання

У наведеному нижче XML документі описана екзаменаційна відомість:

```
<gradeReport id="120851">
<date>10-06-2008</date>
<subject>Computer Science Fundamentals</subject>
<examiner>prof.Litvinov</examiner>
  <gradeList>
    <gradeRecord id="1">
      <student>Ivanov</student>
      <grade>4</grade>
    </gradeRecord>
    <gradeRecord id="2">
      <student>Petrov</student>
      <grade>3</grade>
    </gradeRecord>
    <gradeRecord id="3">
      <student>Sidorov</student>
      <grade>5</grade>
    </gradeRecord>
  </gradeList>
</gradeReport>
```

Побудуйте для нього XSL файл, необхідний для XSLT перетворення початкового XML документа в HTML сторінку, що містить заголовок відомості і таблицю оцінок зі стовпцями: *номер, студент, оцінка*.

Лабораторна робота №14.

Тема «Розробка веб-служби в ASP.NET.»

Мета роботи:

1. Знайомство з інструментом розробки ASP.NET веб-служб в середовищі Microsoft Visual Studio.

Порядок виконання лабораторної роботи

Розглянемо як приклад створення за допомогою ASP.NET веб-служби, яка переводить будь-яке ціле десяткове число в один з форматів за вибором: двійковий, вісімковий, десятковий.

1. Створіть новий проект (тип: *ASP Web Service*), наприклад, під іменем *ASPNETCalcWebService*. У результаті буде автоматично згенеровано файл *Service1.aspx.cs*.

Програмна логіка веб-служби буде реалізована на мові C # в CodeBehind файлі *Service1.aspx.cs*:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

namespace ASPNETCalcWebService
{
    /// <summary>
    /// Summary description for Service1
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [ToolboxItem(false)]
    public class Service1 : System.Web.Services.WebService
```

```
{  
    [WebMethod]  
    public string HelloWorld()  
    {  
        return "Hello World";  
    }  
}
```

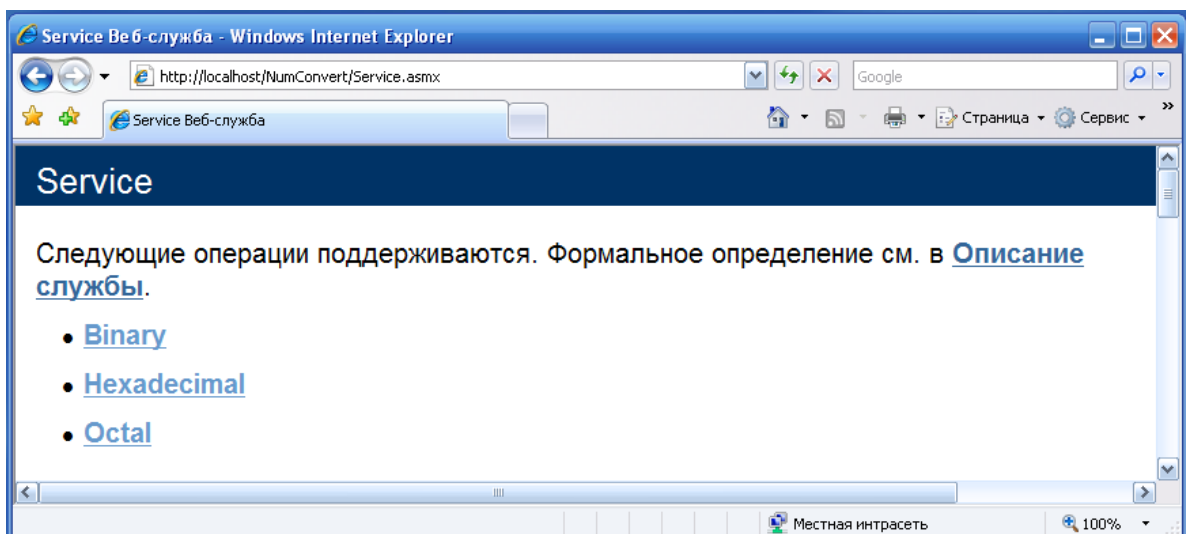
2. Для реалізації логіки веб-служби в цьому файлі, замініть у файлі *Service1.asmx.cs* метод *HelloWorld ()* на 3 нових методи, за допомогою яких будуть виконуватися всі перетворення:

```
using System;  
using System.Collections;  
using System.ComponentModel;  
using System.Data;  
using System.Web;  
using System.Web.Services;  
using System.Web.Services.Protocols;  
  
namespace ASPNETCalcWebService  
{  
    /// <summary>  
    /// Summary description for Service1  
    /// </summary>  
    [WebService(Namespace = "http://tempuri.org/")]  
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]  
    [ToolboxItem(false)]  
    public class Service1 : System.Web.Services.WebService  
    {  
        // Uncomment the following line if using designed components  
        // InitializeComponent();  
    }  
  
    // Перетворення в двійкову систему числення  
  
    [WebMethod]
```

```
public string Binary(int x) {  
    return Convert.ToString(x, 2);  
}  
  
// Перетворення в вісімкову систему числення  
  
[WebMethod]  
public string Octal(int x)  
{  
    return Convert.ToString(x, 8);  
}  
  
// Перетворення в шістнадцяткову систему числення  
  
[WebMethod]  
public string Hexadecimal(int x)  
{  
    return Convert.ToString(x, 16);  
}  
}
```

Атрибут *WebMethod* у цьому файлі вказує на те, що описуваний метод повинен бути доступний за протоколом HTTP для користувачів.

3. Відкомпілюйте і запустіть проект. У результаті в браузері повинна з'явитися наступна сторінка:



4. Активуйте гіперпосилання «Опис служби». У вікні браузера має з'явитися опис веб-служби у форматі WSDL:

```
<?xml version="1.0" encoding="utf-8" ?>
<wsdl:definitions
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:tns="http://tempuri.org/"
  xmlns:s="http://www.w3.org/2001/XMLSchema"
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  targetNamespace="http://tempuri.org/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
      targetNamespace="http://tempuri.org/">
      <s:element name="Binary">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1"
              maxOccurs="1" name="x" type="s:int" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="BinaryResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0"
              maxOccurs="1" name="BinaryResult"
              type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="Octal">
        <s:complexType>
          <s:sequence>
```

```

        <s:element          minOccurs="1"
            maxOccurs="1" name="x" type="s:int"
        />
    </s:sequence>
</s:complexType>
</s:element>
<s:element name="OctalResponse">
    <s:complexType>
        <s:sequence>
            <s:element          minOccurs="0"
                maxOccurs="1" name="OctalResult"
                type="s:string" />
        </s:sequence>
    </s:complexType>
</s:element>
<s:element name="Hexadecimal">
    <s:complexType>
        <s:sequence>
            <s:element          minOccurs="1"
                maxOccurs="1" name="x" type="s:int"
            />
        </s:sequence>
    </s:complexType>
</s:element>
<s:element name="HexadecimalResponse">
    <s:complexType>
        <s:sequence>
            <s:element          minOccurs="0"
                maxOccurs="1"
                name="HexadecimalResult"
                type="s:string" />
        </s:sequence>
    </s:complexType>
</s:element>
</s:schema>
</wsdl:types>
<wsdl:message name="BinarySoapIn">
    <wsdl:part name="parameters" element="tns:Binary" />
</wsdl:message>
<wsdl:message name="BinarySoapOut">

```

```

        <wsdl:part                                name="parameters"
            element="tns:BinaryResponse" />
    </wsdl:message>
    <wsdl:message name="OctalSoapIn">
        <wsdl:part name="parameters" element="tns:Octal" />
    </wsdl:message>
    <wsdl:message name="OctalSoapOut">
        <wsdl:part                                name="parameters"
            element="tns:OctalResponse" />
    </wsdl:message>
    <wsdl:message name="HexadecimalSoapIn">
        <wsdl:part                                name="parameters"
            element="tns:Hexadecimal" />
    </wsdl:message>
    <wsdl:message name="HexadecimalSoapOut">
        <wsdl:part                                name="parameters"
            element="tns:HexadecimalResponse" />
    </wsdl:message>
    <wsdl:portType name="ServiceSoap">
        <wsdl:operation name="Binary">
            <wsdl:input message="tns:BinarySoapIn" />
            <wsdl:output message="tns:BinarySoapOut" />
        </wsdl:operation>
        <wsdl:operation name="Octal">
            <wsdl:input message="tns:OctalSoapIn" />
            <wsdl:output message="tns:OctalSoapOut" />
        </wsdl:operation>
        <wsdl:operation name="Hexadecimal">
            <wsdl:input message="tns:HexadecimalSoapIn" />
            <wsdl:output message="tns:HexadecimalSoapOut"
                />
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding                                name="ServiceSoap"
        type="tns:ServiceSoap">
        <soap:binding
            transport="http://schemas.xmlsoap.org/soap/http" />
        <wsdl:operation name="Binary">
            <soap:operation
                soapAction="http://tempuri.org/Binary"

```

```

        style="document" />
    <wsdl:input>
        <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
        <soap:body use="literal" />
    </wsdl:output>
</wsdl:operation>
<wsdl:operation name="Octal">
    <soap:operation
        soapAction="http://tempuri.org/Octal"
        style="document" />
    <wsdl:input>
        <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
        <soap:body use="literal" />
    </wsdl:output>
</wsdl:operation>
<wsdl:operation name="Hexadecimal">
    <soap:operation
        soapAction="http://tempuri.org/Hexadecimal"
        style="document" />
    <wsdl:input>
        <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
        <soap:body use="literal" />
    </wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:binding
    name="ServiceSoap12"
    type="tns:ServiceSoap">
    <soap12:binding
        transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="Binary">
        <soap12:operation
            soapAction="http://tempuri.org/Binary"
            style="document" />
        <wsdl:input>

```



```

        <soap12:body use="literal" />
    </wsdl:input>
    <wsdl:output>
        <soap12:body use="literal" />
    </wsdl:output>
</wsdl:operation>
<wsdl:operation name="Octal">
    <soap12:operation
        soapAction="http://tempuri.org/Octal"
        style="document" />
    <wsdl:input>
        <soap12:body use="literal" />
    </wsdl:input>
    <wsdl:output>
        <soap12:body use="literal" />
    </wsdl:output>
</wsdl:operation>
<wsdl:operation name="Hexadecimal">
    <soap12:operation
        soapAction="http://tempuri.org/Hexadecimal"
        style="document" />
    <wsdl:input>
        <soap12:body use="literal" />
    </wsdl:input>
    <wsdl:output>
        <soap12:body use="literal" />
    </wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="Service">
    <wsdl:port
        binding="tns:ServiceSoap"
        name="ServiceSoap"
        <soap:address
            location="http://localhost/NumConvert/Service.as
            mx" />
    </wsdl:port>
    <wsdl:port
        binding="tns:ServiceSoap12"
        name="ServiceSoap12"
        <soap12:address
            location="http://localhost/NumConvert/Service.as

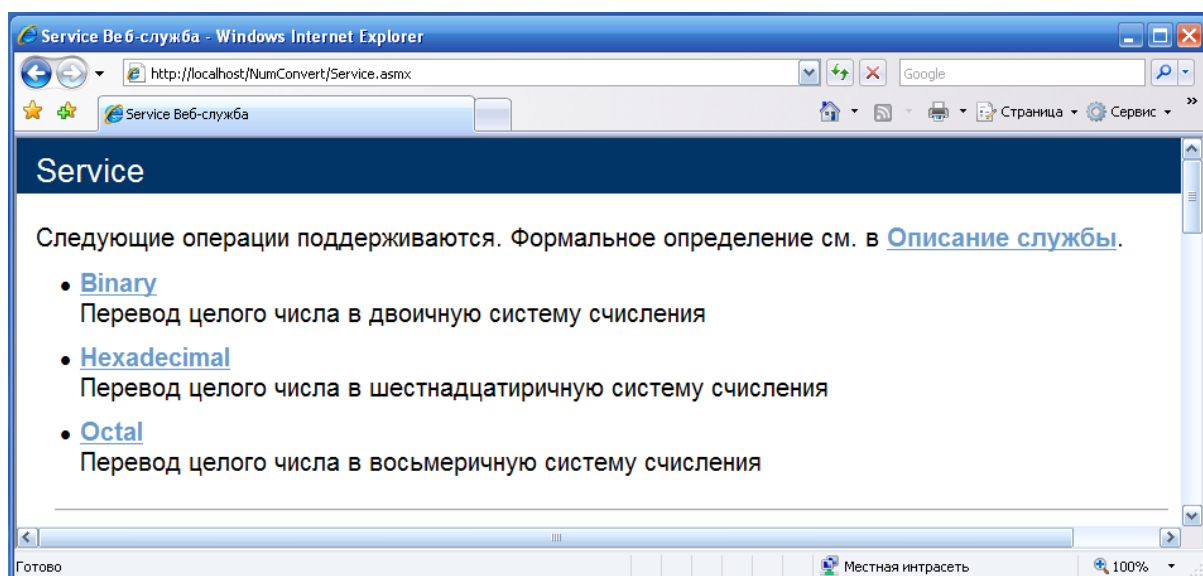
```

```
mx" />  
</wsdl:port>  
</wsdl:service>  
</wsdl:definitions>
```

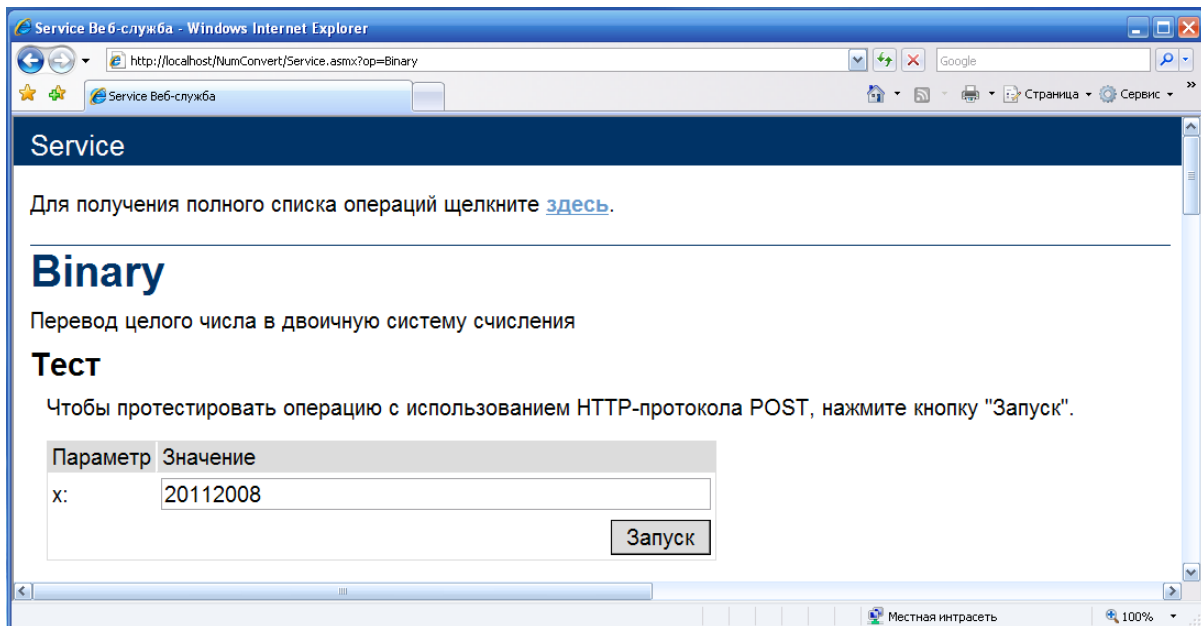
5. Якщо необхідно повідомити більш докладну інформацію для користувачів по кожному з доступних в цій веб-службі методів, достатньо буде додати параметр *Description* в атрибуті *WebMethod*, наприклад:

```
[WebMethod (Description = "Переведення цілого числа в двійкову  
систему числення")]  
public string Binary(int x)  
{  
    return Convert.ToString(x, 2);  
}
```

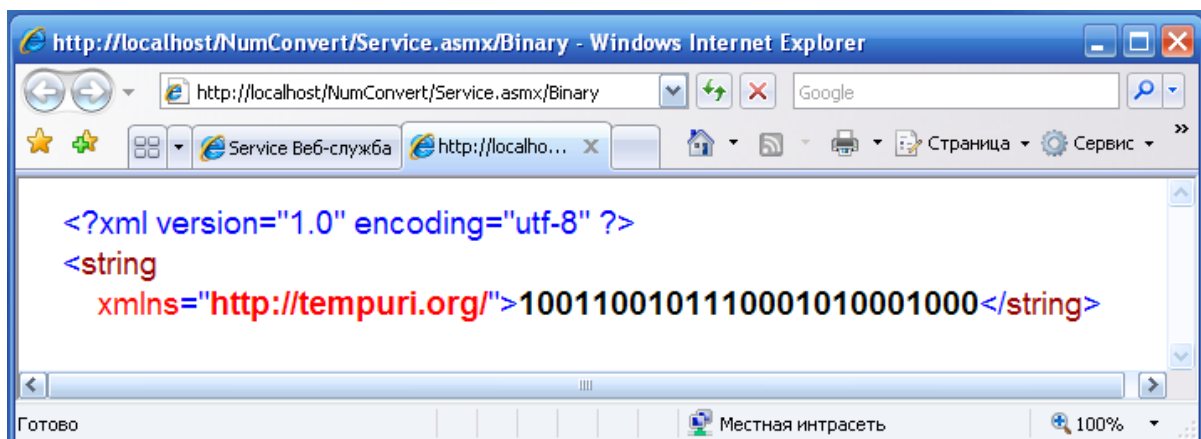
В результаті в браузері буде отримана наступна сторінка:



6. Виберіть метод *Binary*. При цьому повинна завантажитися веб-сторінка виду:



У результаті роботи веб-служби (після введення числа і натиснення кнопки «Запуск») буде створена наступна сторінка:



Контрольне завдання

Створіть ASP.NET веб-службу, яка буде підтримувати три методи, які повертають відповідно *назву* поточного дня тижня, *номер* поточного дня в *місяці*, *номер* поточного дня в *році*.

Для отримання необхідних даних на стороні сервера можна використовувати властивості і методи:

```
Label1.Text = DateTime.Now.DayOfWeek.ToString();  
Label1.Text = DateTime.Now.Day.ToString();  
Label1.Text = DateTime.Now.DayOfYear.ToString();
```

Лабораторна робота №15.

Тема «Розробка веб-служби в ASP.NET. Створення проксі-збірки для веб-служби.»

Мета роботи:

1. Продовження знайомства з інструментами розробки ASP.NET веб-служб і їх використання за допомогою проксі-збірки в середовищі Microsoft Visual Studio.

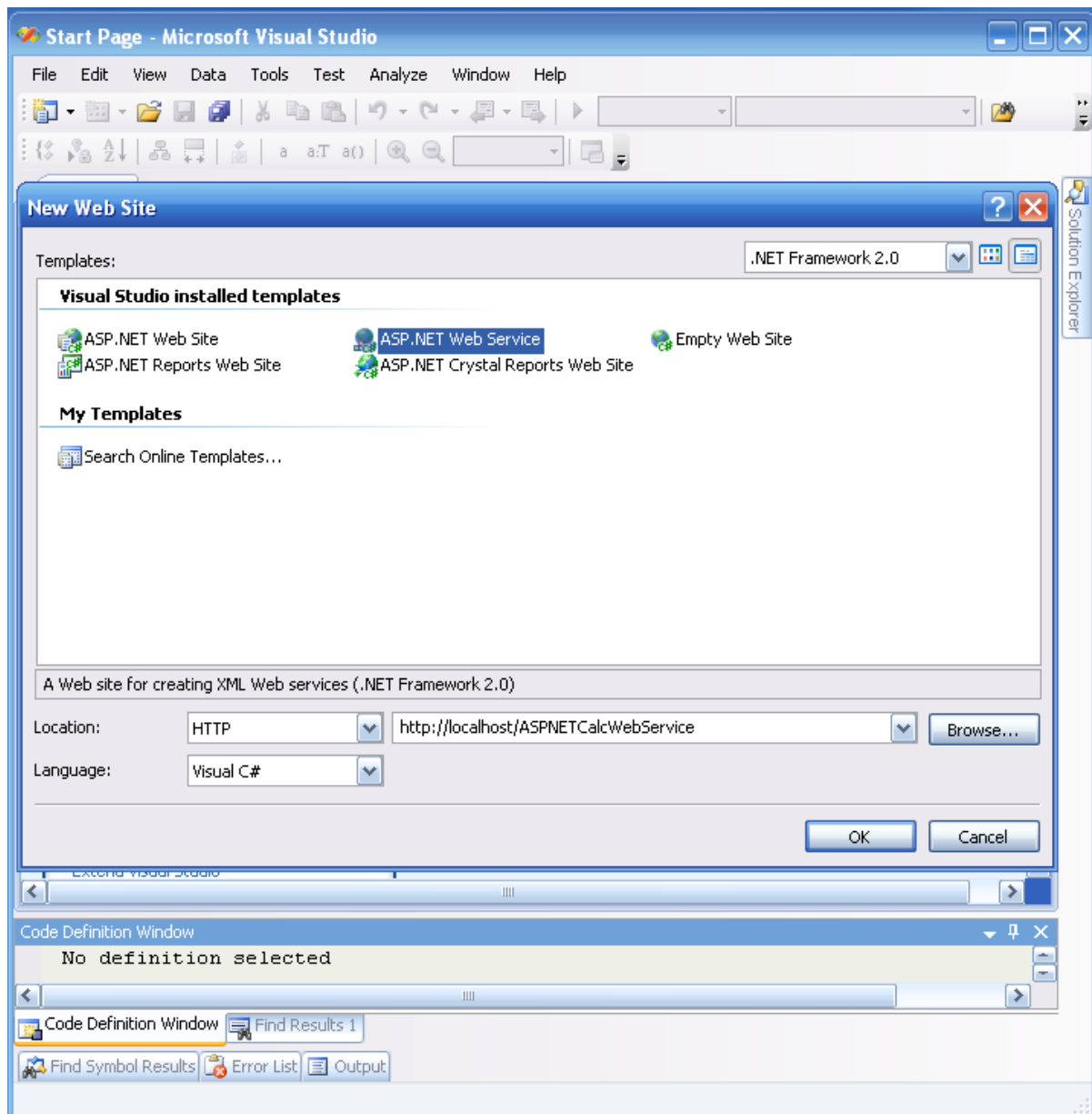
Теоретичний матеріал

У попередній лабораторній роботі розглядалося створення веб-служби *ASPNETCalcWebService*, що виконувала переклад десяткового цілого числа в системі числення з основою 2, 8, 16. При цьому передача даних клієнтом (по протоколу SOAP) і отримання їх вироблялося в XML форматі. Це може бути незручним для клієнта і зажадати розробку додаткових компонентів для конвертації даних в звичний для кінцевого користувача формат.

Уникнути цього можна, якщо звертатися до веб-службі не безпосередньо, а за допомогою проксі-збірки на C# або іншою мовою програмування. Така проксі-збірка створюється автоматично в Visual Studio.NET, наприклад, у вигляді клієнта *Windows Forms*, і бере на себе всю роботу по взаємодії з веб-службою.

Порядок виконання лабораторної роботи

1. Створення веб-служби.
 - 1) Створіть веб-сайт для проекту типу "*ASP.NET Web Service*":



У складі нового проекту будуть автоматично створені файли *Service.asmx* (містить код подання) і *Service.cs* (містить програмний код).

2) Замініть у файлі *Service.cs* метод *HelloWorld()* на методи, що реалізують всі перетворення (див. *Лабораторну роботу 14*):

```
// Перетворення в двійкову систему числення
```

```
[WebMethod]  
public string Binary(int x) {
```

```
        return Convert.ToString(x, 2);
    }

    // Перетворення в вісімкову систему числення

    [WebMethod]
    public string Octal(int x)
    {
        return Convert.ToString(x, 8);
    }

    // Перетворення в шістнадцяткову систему числення
    [WebMethod]
    public string Hexadecimal(int x)
    {
        return Convert.ToString(x, 16);
    }
}
```

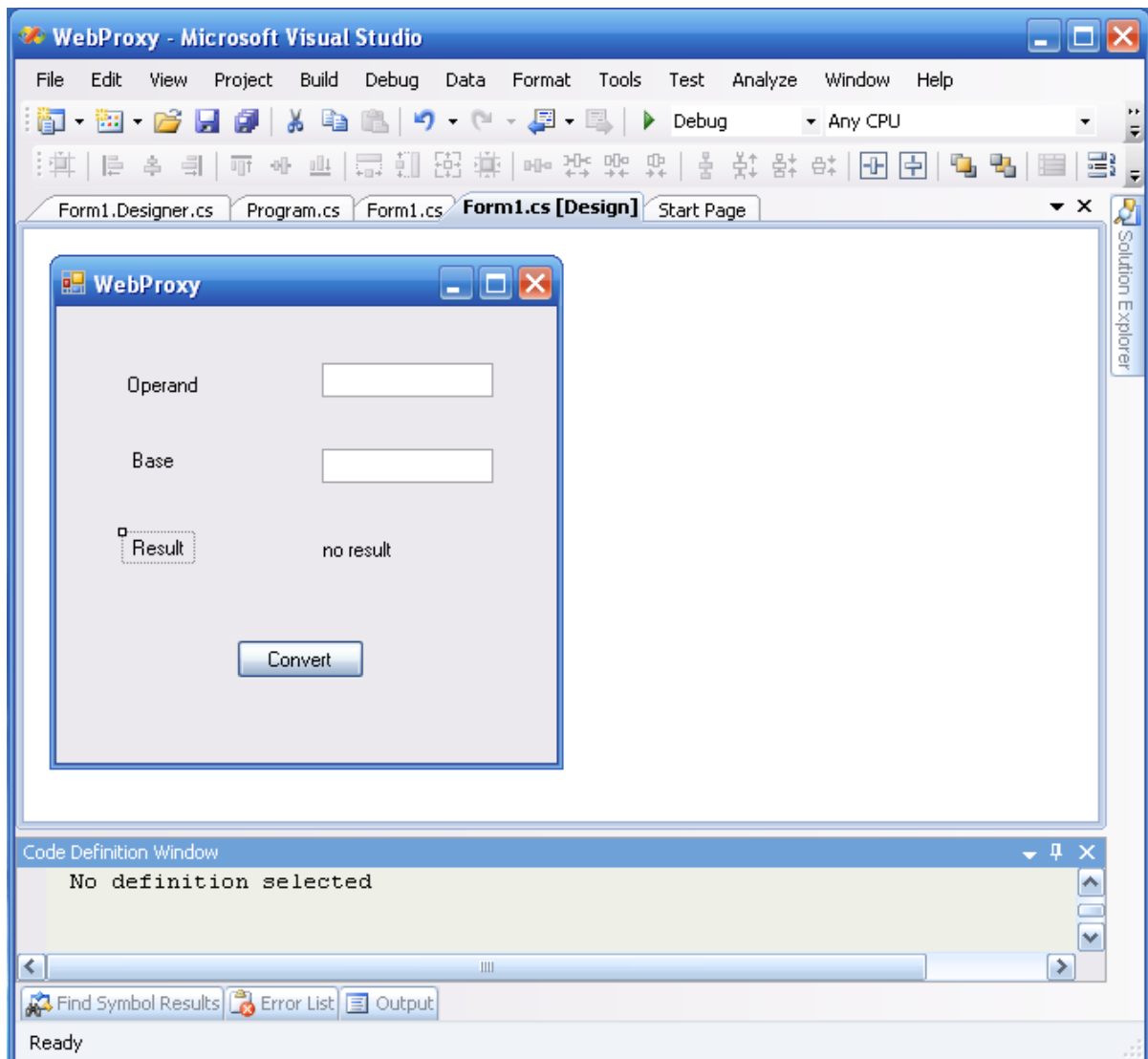
3) Відкомпілюйте проект і виконайте пробний запуск веб-служби. URL служби для завантаження в браузері:

<http://localhost/ASPNETCalcWebService/Service.aspx>

2. Створення проксі-збірки.

1) Створіть проект з ім'ям, наприклад *WebProxy* (тип проекту *Windows Forms Application*).

2) За допомогою дизайнера форм MS Visual Studio підготуйте інтерфейс взаємодії з користувачем у вигляді форми з двома текстовими полями введення даних, полем відображення результату і кнопкою:



Код користувацької форми, автоматично згенерований середовищем MS VS, знаходиться у файлі *Form1.Designer.cs*:

```
namespace WebProxy
{
    partial class WebProxy
    {
        private System.ComponentModel.IContainer components = null;

        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {

```



```
        components.Dispose();
    }
    base.Dispose(disposing);
}

#region Windows Form Designer generated code

private void InitializeComponent()
{
    this.label1 = new System.Windows.Forms.Label();
    this.textBox1 = new System.Windows.Forms.TextBox();
    this.textBox2 = new System.Windows.Forms.TextBox();
    this.button1 = new System.Windows.Forms.Button();
    this.label2 = new System.Windows.Forms.Label();
    this.label3 = new System.Windows.Forms.Label();
    this.label4 = new System.Windows.Forms.Label();
    this.SuspendLayout();
    //
    // label1
    //
    this.label1.AutoSize = true;
    this.label1.Location = new System.Drawing.Point(152, 135);
    this.label1.Name = "label1";
    this.label1.Size = new System.Drawing.Size(47, 13);
    this.label1.TabIndex = 0;
    this.label1.Text = "no result";
    //
    // textBox1
    //
    this.textBox1.Location = new System.Drawing.Point(155, 33);
    this.textBox1.Name = "textBox1";
    this.textBox1.Size = new System.Drawing.Size(100, 20);
    this.textBox1.TabIndex = 1;
    //
    // textBox2
    //
    this.textBox2.Location = new System.Drawing.Point(155, 83);
    this.textBox2.Name = "textBox2";
    this.textBox2.Size = new System.Drawing.Size(100, 20);
    this.textBox2.TabIndex = 2;
```

```
//  
// button1  
//  
this.button1.Location = new System.Drawing.Point(105, 194);  
this.button1.Name = "button1";  
this.button1.Size = new System.Drawing.Size(75, 23);  
this.button1.TabIndex = 3;  
this.button1.Text = "Convert";  
this.button1.UseVisualStyleBackColor = true;  
this.button1.Click += new  
System.EventHandler(this.button1_Click);  
//  
// label2  
//  
this.label2.AutoSize = true;  
this.label2.Location = new System.Drawing.Point(38, 39);  
this.label2.Name = "label2";  
this.label2.Size = new System.Drawing.Size(48, 13);  
this.label2.TabIndex = 4;  
this.label2.Text = "Operand";  
//  
// label3  
//  
this.label3.AutoSize = true;  
this.label3.Location = new System.Drawing.Point(41, 83);  
this.label3.Name = "label3";  
this.label3.Size = new System.Drawing.Size(31, 13);  
this.label3.TabIndex = 5;  
this.label3.Text = "Base";  
//  
// label4  
//  
this.label4.AutoSize = true;  
this.label4.Location = new System.Drawing.Point(41, 134);  
this.label4.Name = "label4";  
this.label4.Size = new System.Drawing.Size(37, 13);  
this.label4.TabIndex = 6;  
this.label4.Text = "Result";  
//  
// WebProxy
```

```
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F,
13F);
this.AutoScaleMode =
System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(292, 266);
this.Controls.Add(this.label4);
this.Controls.Add(this.label3);
this.Controls.Add(this.label2);
this.Controls.Add(this.button1);
this.Controls.Add(this.textBox2);
this.Controls.Add(this.textBox1);
this.Controls.Add(this.label1);
this.Name = "WebProxy";
this.Text = "WebProxy";
this.Load += new System.EventHandler(this.WebProxy_Load);
this.ResumeLayout(false);
this.PerformLayout();
}

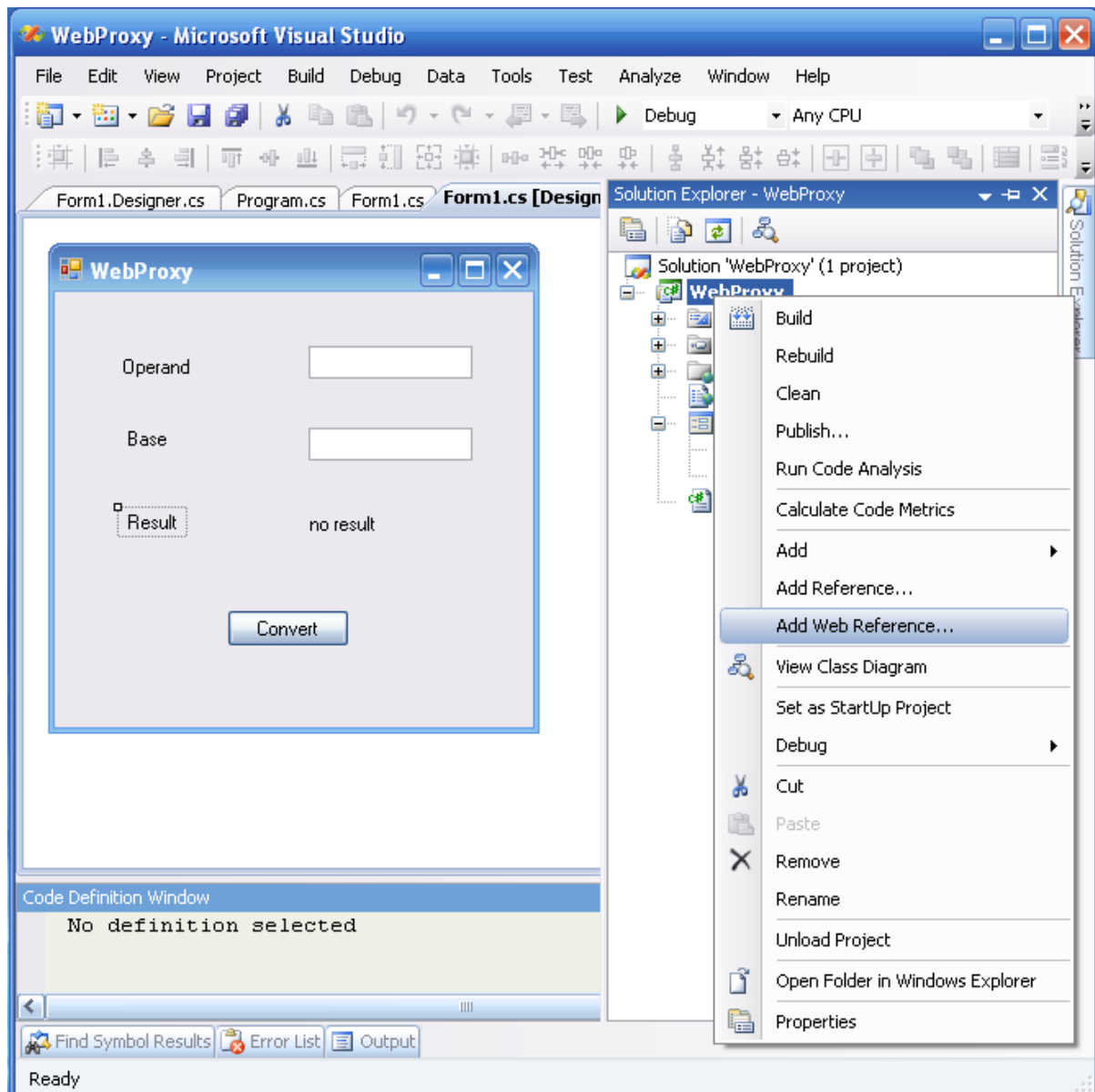
#endregion

private System.Windows.Forms.Label label1;
private System.Windows.Forms.TextBox textBox1;
private System.Windows.Forms.TextBox textBox2;
private System.Windows.Forms.Button button1;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.Label label3;
private System.Windows.Forms.Label label4;
}
}
```

3) Додайте веб-посилання:

<http://localhost/ASPNETCalcWebService/Service.aspx>

на веб-службу *ASPNETCalcWebService* в проект:



4) Додайте обробник події «натискання кнопки» у файлі *Form1.cs*:

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Text;  
using System.Windows.Forms;  
  
namespace WebProxy
```

```
{

// додано простір імен localhost

using localhost;
public partial class WebProxy : Form
{
    public WebProxy()
    {
        InitializeComponent();
    }

// обробка події натискання кнопки

    private void button1_Click(object sender, EventArgs e)
    {

// створення об'єкта, пов'язаного з веб-службою

        localhost.Service w = new localhost.Service();

        string ans;

// перетворення даних з текстових полів введення в формі в цілі
числа
// operand – операнд, NumBase - основа системи числення

        int NumBase = int.Parse(textBox2.Text);
        int operand = int.Parse(textBox1.Text);

// вибір відповідної функції веб-служби та взаємодія з нею

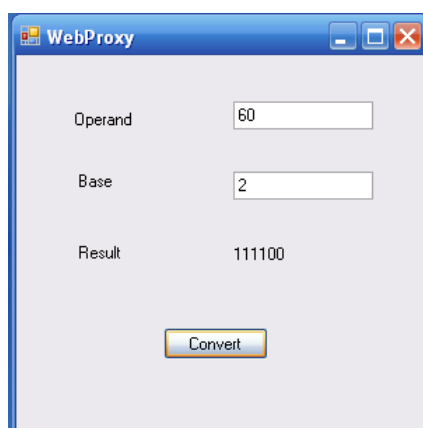
        switch (NumBase)
        {
            case 2: ans = w.Binary(operand); break;
            case 8: ans = w.Octal(operand); break;
            case 16: ans = w.Hexadecimal(operand); break;
            default: ans = "base is undefined"; break;
        }
    }
}
```

```
// відображення результату обчислення у формі
```

```
    label1.Text = ans;  
}  
  
}  
}
```

5) Відкомпілюйте проект.

6) Перевірка роботи програми. На скріншоті нижче показана робота програми *WebProxy* (спільно з веб-службою *ASPNETCalcWebService*):



Контрольне завдання

Створіть ASP.NET проксі-збірку на мові C# для веб-служби, що підтримує три методи, які повертають відповідно *назву* поточного дня *тижня*, номер поточного дня в *місяці*, номер поточного дня в *році*.

Для отримання необхідних даних на стороні сервера можна використовувати властивості і методи:

```
Label1.Text = DateTime.Now.DayOfWeek.ToString();  
Label1.Text = DateTime.Now.Day.ToString();  
Label1.Text = DateTime.Now.DayOfYear.ToString();
```

Вибір методу веб-служби повинен бути реалізований через форму з елементом типу *ListBox* або *RadioButton*.

Лабораторна робота №16.

Тема «Приклади розробки RSS-джерел і RSS-рідерів.»

Мета роботи:

1. Введення в технологію RSS.
2. Вивчення структури RSS документів, їх генерації і публікації.

Теоретичний матеріал

RSS – це спосіб поширення веб-контенту з веб-сайту на інших веб-сайтах. RSS дозволяє виконувати швидкий перегляд новин і змін.

На кого орієнтований формат RSS? В першу чергу RSS призначений для використання на дуже часто оновлюваних веб-сайтах, наприклад:

- Новинні сайти (списки новин).
- Сайти компаній (списки новин і продуктів).
- Календарі (списки майбутніх подій і знаменних днів).
- Зміни сайтів (список оновлених і нових сторінок).

Офіційного стандарту RSS як такого не існує. Реально використовуються наступні формати:

- Порядку 50% RSS-потоків використовують RSS 0.91.
- Порядку 25% використовують RSS 1.0.
- Решта 25% RSS-потоків розподілені між RSS 0.9 і RSS 2.0.

Як працює RSS?

1. Створюється RSS документ у вигляді файлу з розширенням .xml.
2. Цей файл розміщується на веб-сайті.
3. RSS-потік реєструється в RSS-агрегаторах.

1. Приклад RSS-документу

Розглянемо приклад RSS-документа.

```
<?xml version="1.0" encoding="windows-1251" ?>
<rss version="2.0">
<channel>
  <title>My Home Page</title>
  <link>http://www.MyHP.edu</link>
  <description>Free web building tutorials</description>
  <item>
    <title>RSS Demo</title>
    <link>http://www.MyHP.edu/rss</link>
    <description>New RSS demo on my home page</description>
  </item>
  <item>
    <title>XML Demo</title>
    <link>www.MyHP.edu/xml</link>
    <description>New XML demo on my home page</description>
  </item>
</channel>
</rss>
```

У першому рядку розміщено оголошення версії XML і кодування документа.

Другий рядок ідентифікує даний документ як RSS документ версії 2.0.

У третьому рядку міститься елемент `<channel>`, що описує RSS потік. Він у свою чергу містить дочірні елементи:

- `<title>` – описує заголовок RSS каналу.
- `<link>` – описує гіперпосилання на канал.
- `<description>` – містить коротку характеристику каналу.

Кожен елемент `<channel>` може містити один або більше `<item>` елементів. Кожен елемент `<item>` описує

окрему статтю RSS джерела. У свою чергу кожен елемент `<item>` має три обов'язкових дочірніх елементи:

- `<title>`;
- `<link>`;
- `<description>`.

Крім обов'язкових елементів `<channel>` може містити деякі додаткові дочірні елементи.

Елемент `<category>` описує категорію потоку і використовується RSS агрегатор для угруповання сайтів на основі категорій.

Елемент `<copyright>` інформує про авторське право на даний документ.

Елемент `<image>` призначений для відображення зображення при показі доки агрегатором. У свою чергу він містить три обов'язкові дочірніх елементи:

- `<url>` – URL зображення;
- `<title>` – альтернативний текст, що відображається при неможливості показу зображення;
- `<link>` – містить гіперпосилання на веб-сайт, що містить канал.

Приклад:

```
<image>
  <url>http://www.myhp.edu/images/me.gif</url>
  <title>My home page</title>
  <link>http://www.myhp.edu</link>
</image>
```

Елемент `<language>` інформує про мову документа. Може бути використаний агрегаторами для угруповання сайтів за мовою.

2. Список дочірніх елементів для елемента `<channel>`

Елемент	Опис	Статус
<category>	Вказується одна або кілька категорій потоку	Необов'язковий
<cloud>	Реєструє процеси, які отримують негайне повідомлення в разі поновлення потоку	Необов'язковий
<copyright>	Сповіщає про авторське право на документ	Необов'язковий
<description>	Короткий опис каналу	Обов'язковий
<docs>	Визначає URL документів за форматами, використовуваним в потоці	Необов'язковий
<generator>	Вказує на програму, що генерує потік	Необов'язковий
<image>	Дозволяє показувати зображення, що представляє потік в агрегатор	Необов'язковий
<language>	Вказується мова потоку	Необов'язковий
<lastBuildDate>	Визначається дата останнього оновлення змісту потоку	Необов'язковий
<link>	Вказується гіперпосилання на канал	Обов'язковий
<managingEditor>	Міститься поштову адресу редактора змісту потоку	Необов'язковий
<pubDate>	Визначає дату останньої публікації змісту потоку	Необов'язковий
<rating>	Міститься PICS рейтинг потоку	Необов'язковий
<skipDays>	Вказує дні, в які агрегатори не повинні оновлювати потік.	Необов'язковий
<skipHours>	Вказується кількість годин протягом яких агрегатори не повинні оновлювати потік	Необов'язковий
<textInput>	Описується текстове поле введення, яка відображається в потоці	Необов'язковий
<title>	Розміщується ім'я каналу	Обов'язковий
<ttl>	Зазначається кількість хвилин протягом яких потік може знаходитися в кеші до	Необов'язковий

	поновлення з джерела	
<webMaster>	Поміщається адресу електронної пошти веб-дизайнера потоку	Необов'язковий

3. Список дочірніх елементів для елемента <item>

Елемент	Опис	Статус
<author>	Адреса електронної пошти автора теми	Необов'язковий
<category>	Описує одну або більше категорій для даної теми	Необов'язковий
<comments>	Дозволяє зв'язати тему з коментарями з даної теми	Необов'язковий
<description>	Короткий опис теми	Обов'язковий
<enclosure>	Дозволяє включити мультимедіа файл в дану тему	Необов'язковий
<guid>	Містить унікальний ідентифікатор теми	Необов'язковий
<link>	Містить гіперпосилання на тему	Обов'язковий
<pubDate>	Містить останню дату публікації теми	Необов'язковий
<source>	Вказує зовнішнє джерело для даної теми	Необов'язковий
<title>	Назва теми	Обов'язковий

4. Публікація RSS файлу

Створення RSS потоку не обмежується розробкою RSS документа. Необхідно ще опублікувати цей файл. Для цього буде потрібно виконати наступну послідовність дій:

1. Вибрати відповідну назву для RSS файлу. Розширення повинно бути *.xml*.
2. Перевірити RSS файл на правильність за допомогою відповідної програми-валідатора, наприклад, взятої за адресою <http://www.feedvalidator.org>.

3. Розмістити RSS файл у відповідному веб-каталозі веб-сервера.
4. Скопіювати одну з «кнопок»: **RSS** або **XML** у веб-каталог.
5. Вставити обрану «кнопку» на вихідну сторінку RSS потоку у вигляді гіперпосилання на RSS файл, наприклад:

```
<a href = "www.myhp.edu/rss/myrss.xml">  
    
</a>
```

6. Розмістити створений RSS потік в популярних Каталогах RSS потоків, при цьому URL потік повинен посилатися саме на сам XML файл потоку, тобто *http://www.myhp.edu/rss/myrss.xml*.
7. Можна також зареєструвати потік в популярних пошукових системах, наприклад:
 - Yahoo -
http://publisher.yahoo.com/rss_guide/submit.php;
 - Google - <http://www.google.com/ig>;
 - MSN -
<http://w.moreover.com/site/products/ind/pingserver.html>

8. Періодичне оновлення потоку.

В принципі, можна самостійно займатися формуванням даних у відповідному форматі RSS для роботи потоку. Однак є досить зручні високорівневі засоби для автоматизації даної роботи. Наприклад:

- **MyRSSCreator** - <http://www.myrsscreator.com/>;
- **FeedFire** -
<http://www.feedfire.com/site/index.html>.

Для читання RSS потоків використовуються спеціальні програми читання - RSS рідери. Деякі з браузерів мають вбудовані RSS рідери. Зокрема, веб-

браузер *MS Internet Explorer 7* може інтерпретувати XML файли RSS потоків і коректно їх відображати.

5. Додавання RSS-каналу за допомогою Microsoft Internet Explorer версії 7

При використанні MS Internet Explorer версії 7 і Office Outlook 2007 можна додавати та переглядати RSS-канали за допомогою будь-якої з цих програм.

1. Під час перегляду веб-сторінки, що містить RSS, в *Internet Explorer* поряд з кнопкою  (Домашня сторінка) відображається кнопка .
2. Натисніть кнопку .
3. На веб-сторінці з'явиться список доступних RSS-каналів.
4. Виберіть RSS-канал, який необхідно додати.

Можна також натиснути кнопки ,  або , які розташовані на веб-сторінці.

Порядок виконання лабораторної роботи

1. Створення RSS документа за допомогою PHP сценарію і з використанням інтерфейсу DOM XML.
 - 1) Підготуйте файл, що містить сценарій на мові PHP:

```
<?php
// Створюється новий XML документ.
$dfeed = new DOMDocument('1.0', 'utf-8');

// Создание корневого элемента <rss>
$erss = $dfeed->createElement('rss');
$erss->setAttribute('version', '2.0');

// І додавання його в дерево документа
$dfeed->appendChild($erss);
```

```
// Створення елемента <channel> і додавання його до <rss>
$sechannel = $dfeed->createElement('channel');
$rsrss->appendChild($sechannel);

// Створення і додавання в channel
// вузлів <title>, <link>, <description>, <language>, <pubDate>
$sechannel->appendChild( $dfeed->createElement('title', 'RSS-channel
title') );
$sechannel->appendChild( $dfeed ->createElement('link',
'http://www.myhp.edu') );
$sechannel->appendChild( $dfeed ->createElement('description', 'my
RSS demo') );
$sechannel->appendChild( $dfeed ->createElement('language', 'en') );
$sechannel->appendChild( $dfeed ->createElement('pubDate', date('r'))
);

// Додавання до вузла <channel> 2 вузлів <item>
for ( $i = 1; $i <= 2; $i++ )
{
    $seitem = $dfeed->createElement('item');
    $sechannel->appendChild($seitem);

    // Створення дочірніх елементів для <item>
    $seitem->appendChild( $dfeed->createElement('title', 'Item' . $i) );
    $seitem->appendChild( $dfeed->createElement('link',
'http://www.myhp.edu/rss/' . $i . '.xml') );
    $seitem->appendChild( $dfeed->createElement('description',
'Description for ' . $i . ' item') );
}

// Збереження документа у файлі demo.rss
$dfeed->save('demo.rss');
?>
```

2) Створений файл розмістіть на веб-сервері, налаштуйте права доступу (право на запис) для веб-сервера до директорії, в якій розміщується сценарій, або до спеціальної директорії, в якій буде створений файл *demo.rss* (це більш безпечно).

- 3) Виконайте сценарій за запитом з веб-браузера.
- 4) У результаті виконання сценарію вийде наступний документ:

```
<?xml version="1.0" encoding="utf-8" ?>
<rss version="2.0">
  <channel>
    <title>RSS-channel title</title>
    <link>http://www.myhp.edu</link>
    <description>my RSS demo</description>
    <language>en</language>
    <pubDate>Thu, 06 Nov 2008 01:50:53 +0300</pubDate>
    <item>
      <title>Item1</title>
      <link>http://www.myhp.edu/rss/1.xml</link>
      <description>Description for 1 item</description>
    </item>
    <item>
      <title>Item2</title>
      <link>http://www.myhp.edu/rss/2.xml</link>
      <description>Description for 2 item</description>
    </item>
  </channel>
</rss>
```

2. Читання RSS документа за допомогою PHP сценарію і з використанням інтерфейсу DOM XML.

- 1) Підготуйте файл, що містить сценарій на мові PHP:

```
<?php
// Завантаження документа
$dfeed = DOMDocument::load('demo.rss');

// Читання елемента <channel>
$echannel = $dfeed->firstChild->firstChild;

// Читання першого дочірнього елемента вузла <channel> т.е.<
title>
```

```
$schild = $echannel->firstChild;

// Виведення вмісту дочірніх елементів вузла <channel> до
елемента <item>:

print "<strong><br>";
while ( $schild->tagName != 'item' )
{
    print $schild->nodeValue. "<br>";
    $schild = $schild->nextSibling;
}
print "</strong><br>";

// Отримання списку вузлів <item>
$litems = $echannel->getElementsByTagName('item');

print "<br>";
foreach ( $litems as $item )
{
    $title = $item->firstChild->nodeValue;
    $link = $item->firstChild->nextSibling->nodeValue;
    $descr = $item->lastChild->nodeValue;

    print "<a href=\".$link.\">\".$title.\"</a> :: \".$descr.\"<br>";
}
print "<br>";

?>
```

У методі *load()* в значенні параметра вкажіть шлях до RSS-файлу.

2) Створений файл розмістіть на веб-сервері, налаштуйте права доступу (право на запис) для веб-сервера до директорії, в якій розміщується сценарій, або до спеціальної директорії, в якій буде створений файл *demo.rss* (це більш безпечно).

3) Виконайте сценарій за запитом з веб-браузера.

4) Результат повинен виглядати наступним чином:

RSS-channel title
<http://www.myhp.edu>
my RSS demo
en
Thu, 06 Nov 2008 02:15:55 +0300

[Item1](#) :: Description for 1 item

[Item2](#) :: Description for 2 item

Контрольне завдання

Розробіть PHP-сценарій, що генерує RSS документ, що містить опис веб-сторінок на вашому локальному веб-сайті. Як джерело інформації для сценарію можна використовувати або заздалегідь підготовлений текстовий файл, або функції PHP з бібліотеки для роботи з файловою системою (див., Наприклад:

<http://www.softtime.ru/bookphp/help.php>)

Лабораторна робота №17.

Тема «Реалізація асинхронної взаємодії веб-браузера з веб-сервером за допомогою технології AJAX.»

Мета роботи:

1. Ознайомлення з принципами асинхронної взаємодії між веб-клієнтом і веб-сервером в рамках технології AJAX.

Теоретичний матеріал

AJAX = Asynchronous JavaScript and XML.

AJAX – не є новою мовою програмування, але технологія створення покращених, більш швидких і більшою мірою інтерактивних веб-додатків.

JavaScript сценарій за допомогою AJAX може безпосередньо взаємодіяти з сервером за допомогою об'єкту *XMLHttpRequest*. Використання даного об'єкту, обмін даними з веб-сервером можуть відбуватися без перезавантаження сторінки.

AJAX дозволяє веб-сторінкам запитувати невеликі обсяги інформації з сервера ніж цілком всю сторінку в результаті асинхронної передачі даних (в рамках HTTP протоколу) між браузером і сервером.

AJAX не залежить від програмного забезпечення веб-сервера і заснований на наступних веб-стандартах:

- JavaScript;
- XML;
- HTML;
- CSS.

Оскільки ці веб-стандарти чітко визначені і мають підтримку в найбільш поширених веб-браузерах, то AJAX додатки є браузеро- і платформо-незалежними.

Популярність AJAX пов'язана з появою сервісу *Google Suggest* в 2005 році. Даний сервіс на основі об'єкту *XMLHttpRequest* надає в розпорядження користувача досить динамічний веб-інтерфейс. У процесі введення символів користувачем в поле пошукового запиту JavaScript відправляє їх на сервер і отримує від нього список підказок.

Об'єкт *XMLHttpRequest* підтримується в *Internet Explorer* (починаючи з 5 версії і вище), *Safari 1.2*, *Mozilla 1.0 / Firefox*, *Opera 8+* і *Netscape 7*.

Порядок виконання лабораторної роботи

У даній лабораторній роботі розглядається приклад системи, що імітує роботу сервісу *Google Suggest* на основі AJAX.

1. Реалізація клієнтської частини.

Передбачається, що користувач може вводити в текстове поле форми назву автомобільної марки, отримуючи при цьому динамічно список варіантів назв, відповідних вже введенним символам, без перезавантаження сторінки.

1) Створіть наступну веб-сторінку:

```
<html>
<head>
  <script src="chint.js"></script>
</head>
  <body>
    <form>
      First Name:
      <input type="text" id="txt1"
        onkeyup="showHint(this.value)">
    </form>
    <p>Suggestions: <span id="txtHint"></span></p>
  </body>
</html>
```

Як видно з коду, при настанні події *onkeyup* (відтискання клавіші) викликається обробник *showHint()*.

2) У файлі *chint.js* збережіть наступний код обробника:

```
var xmlHttp;
function showHint(str)
{
    if (str.length==0)
    {
        document.getElementById("txtHint").innerHTML="";
        return;
    }
    xmlHttp=GetXmlHttpRequestObject();
    if (xmlHttp==null)
    {
        alert ("Your browser does not support AJAX!");
        return;
    }
    var url = "ghint.php";
    url = url + "?q=" + str;
    url = url + "&sid=" + Math.random();
    xmlHttp.onreadystatechange = stateChanged;
    xmlHttp.open("GET", url, true);
    xmlHttp.send(null);
}
function stateChanged()
{
    if (xmlHttp.readyState==4)
    {
        document.getElementById("txtHint").innerHTML =
        xmlHttp.responseText;
    }
}
function GetXmlHttpRequestObject()
{
    var xmlHttp=null;
    try
    {
```

```
        // Firefox, Opera 8.0+, Safari
        xmlHttp = new XMLHttpRequest();
    }
    catch (e)
    {
        // Internet Explorer
        try
        {
            xmlHttp=new
            ActiveXObject("Msxml2.XMLHTTP");
        }
        catch (e)
        {
            xmlHttp=new
            ActiveXObject("Microsoft.XMLHTTP");
        }
    }
    return xmlHttp;
}
```

З коду видно, що кожен раз, коли вводиться символ, викликається функція-обробник. Якщо при цьому зміст текстового поля форми непорожній ($str.length > 0$), функція виконує наступні дії:

- Формується *url* для відправки веб-серверу.
- Додається значення параметра *q*, рівне вмісту текстового поля, до *url*.
- Додається до *url* випадкове число для запобігання кешування.
- Створюється об'єкт *XMLHTTP*, при цьому вказується функція (*stateChanged*) підлягає виконанню при настанні події введення символу.
- Відкривається об'єкт *XMLHTTP* з вказаним значенням *url*.
- Відправляється HTTP запит веб-серверу.

Якщо поле введення пусте, відбувається очищення змісту розділу *txtHint* на веб-сторінці.

Ключовим моментом в даній системі є використання об'єкту *XMLHttpRequest*.

Даний об'єкт по-різному створюється в різних браузерах. Так, *Internet Explorer* для цього використовує *ActiveXObject*, в той час як інші браузери використовують вбудований в JavaScript об'єкт *XMLHttpRequest*.

Для підтримки роботи системи в різних браузерах використаний оператор «try-catch».

- Спочатку робиться спроба створити об'єкт *XMLHttpRequest* для браузерів *Firefox*, *Opera* або *Safari*: `xmlHttp = new XMLHttpRequest()`.
- У разі невдачі, робиться наступна спроба створення об'єкту для *Internet Explorer 6.0+*: `xmlHttp = new ActiveXObject ("Msxml2.XMLHTTP")`.
- Якщо це також не вдається, то робиться спроба створення об'єкту вже для *Internet Explorer 5.5+*: `xmlHttp = new ActiveXObject ("Microsoft.XMLHTTP")`.
- У випадку, якщо жодна з цих спроб не принесла успіху, видається повідомлення про відсутність підтримки AJAX браузером.

2. Реалізація серверної частини.

Розмістіть на веб-сервері у файлі з ім'ям *ghint.php* наступний PHP сценарій:

```
<?php
header("Cache-Control: no-cache, must-revalidate");
// Минулий дата
header("Expires: Mon, 1 Sep 2008 07:30:00 GMT");
// Ініціалізація масиву назв
$a[]="Audi";
$a[]="BMW";
$a[]="Buick";
$a[]="Chevrolet";
$a[]="Citroen";
$a[]="Dodge";
$a[]="Ferrari";
```

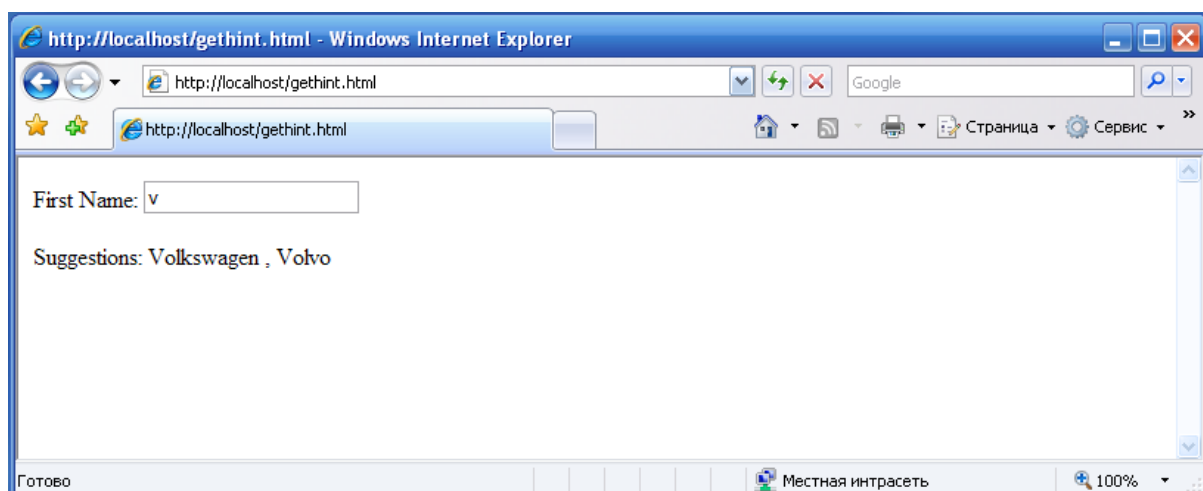
```
$a[]="Fiat";
$a[]="Ford";
$a[]="Honda";
$a[]="Hyundai";
$a[]="Cherokee";
$a[]="Cherry";
$a[]="Lada";
$a[]="Lamborghini";
$a[]="Lincoln";
$a[]="Mazda";
$a[]="Mercedes";
$a[]="Mitsubishi";
$a[]="Nissan";
$a[]="Opel";
$a[]="Peugeot";
$a[]="Plymoth";
$a[]="Pontiac";
$a[]="Renault";
$a[]="Rover";
$a[]="Saab";
$a[]="Subaru";
$a[]="Suzuki";
$a[]="Toyota";
$a[]="Volkswagen";
$a[]="Volvo";
// отримання параметра q з URL
$q = $_GET["q"];
// пошук відповідників з масиву якщо довжина q > 0
if (strlen($q) > 0)
{
    $hint = "";
    for($i = 0; $i<count($a); $i++)
    {
        if (strtolower($q) == strtolower(substr($a[$i],0,strlen($q))))
        {
            if ($hint == "")
            {
                $hint=$a[$i];
            }
        }
        else
    }
```

```
        {  
            $hint=$hint." , ".$a[$i];  
        }  
    }  
}  
// Повернення рядка "немає варіантів" якщо відповідностей,  
не знайдено  
// або знайдене відповідність  
if ($hint == "")  
{  
    $response = "no suggestion";  
}  
else  
{  
    $response = $hint;  
}  
// висновок результату  
echo $response;  
?>
```

3. Перевірка працездатності системи.

Перевірте за допомогою відповідного веб-браузера роботу системи.

На скріншоті показана робота в браузері з веб-сторінкою *ghint.html*, що використовує AJAX:



Контрольне завдання

З використанням РНР-сценарію №7 з лабораторної роботи №5 розробіть модифікований варіант калькулятора для чотирьох арифметичних операцій, в якому відбувається асинхронна передача даних між браузером і сервером (за допомогою AJAX).

Лабораторна робота №18.

Тема «Створення мобільних веб-додатків за допомогою ASP.NET.»

Мета роботи:

1. Демонстрація деяких можливостей розробки мобільних веб-додатків за допомогою керуючих елементів ASP.NET Mobile.

Теоретичний матеріал

У даній лабораторній роботі демонструється застосування керуючих елементів ASP.NET Mobile для створення веб-додатків для мобільних пристроїв.

.NET Mobile представляє собою розширення Microsoft ASP.NET і Microsoft .NET Framework. По суті, .NET Mobile представляє собою набір серверних керуючих елементів для форм, що орієнтовані на застосування в бездротових мобільних пристроях. Ці елементи управління генерують різний код для різних пристроїв на мовах WML, HTML або Compact HTML (cHTML).

Керуючі елементи ASP .NET Mobile розширюють функції SDP (Smart Device Programmability) і платформу .NET Compact Framework. Вони дозволяють використовувати можливості платформи .NET Compact Framework і середовища розробки Visual Studio .NET для створення мобільних веб-додатків завдяки можливості доставки даних на різноманітні мобільні пристрої за допомогою технології ASP.NET. Цей підхід дозволяє створити в середовищі Visual Studio .NET єдиний мобільний веб-додаток, що автоматично виконує формування даних для відображення на різноманітних пристроях: мобільних телефонах, пейджерів, смартфонів, Pocket PC і ін. При цьому інтегроване середовище розробки

дозволяє створювати мобільні веб-додатки, просто перетягуючи керуючі елементи на дизайнер форм.

Система .NET Mobile не встановлює жодних компонентів на клієнтський пристрій. Для адаптації форматування під конкретні браузері використовується серверна логіка. Вона генерує дані у форматах WML, HTML, і cHTML. Крім адаптивної генерації веб-сторінок, технологія .NET Mobile надає багатий набір засобів індивідуалізації та розширення, забезпечуючи простий спосіб здійснення підтримки нових пристроїв. Крім того, технологія .NET Mobile дозволяє розробникам керувати поданням даних для конкретного пристрою або класу пристроїв в одній програмній моделі і забезпечує легку реалізацію підтримки нових пристроїв без переробки існуючих веб-додатків.

Загальна послідовність роботи *.NET Mobile* виглядає наступним чином:

- *Мобільний пристрій*. Він посилає запит на завантаження веб-сторінки.
- *Інтернет*. Запит передається через мережу відповідному веб-серверу.
- *Веб-сервер IIS* отримує запит від мобільного пристрою і передає його обробнику.
- *.NET Framework* виконує обробку запиту.
- *ASP.NET* компілює запитуваний пристроєм документ.
- *.NET Mobile*. Реалізує елементи веб-сторінки з урахуванням вимог конкретного типу мобільного пристрою.
- *Веб-сторінка* повертається назад до клієнтського пристрою.

Найбільш поширеними елементами управління форм для мобільних пристроїв є наступні:

Назва	Функція
Command	Виконання дії
Form	Контейнер для елементів управління
Image	Зображення
Label	Текст
Link	Гіперпосилання
List	Список
MobilePage	Контейнер для елементів управління
ObjectList	Список об'єктів
Panel	Контейнер для інших елементів управління
SelectionList	Список з вибором
StyleSheet	Стилі для інших елементів управління
TextBox	Поле для вводу однострокового тексту
TextView	Поле для вводу багаторічкового тексту

Порядок виконання лабораторної роботи

Для роботи з прикладами, які приводилися в даній лабораторній роботі крім середовища розробки Visual Studio .NET потрібне встановлення компонентів .NET Mobile і WAP емулятора.

В якості WAP емулятора був використаний [Series 40 5th Edition SDK](#) (на основі Nokia 6267). Для встановлення емулятора необхідно завантажити з сайту [Series 40 5th Edition SDK](#) (необхідна попередня реєстрація) і встановити програму *Nokia Mobile Internet Toolkit* і сам емулятор *Nokia Series 40 5th Edition SDK*.

Загальна послідовність розробки веб-додатку з використанням .NET Mobile виглядає наступним чином:

- Створення проекту веб-додатку ASP.NET.
- Підключення інтерфейсу System.Mobile.UI.
- Вставка в веб-сторінку керуючих елементів .NET Mobile.

Розглянемо приклад створення найпростішої веб-сторінки, що відображає на екрані мобільного пристрою повідомлення «Hello world!».

1. Створіть новий проект типу **ASP.NET Web Application**.

2. Активуйте за допомогою меню вікно «*Solution Explorer*». Зі списку файлів проекту видаліть файли *Default.aspx.cs* і *Default.aspx.designer.cs*.

3. Сторінка *Default.aspx* повинна містити наступний код:

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage "
Language="C#" %>
<%@ Register TagPrefix="mobile" Namespace=
"System.Web.UI.MobileControls " Assembly="System.Web.Mobile "
%>

<mobile:Form runat="server">
  <mobile:Label runat="server">Hello world!</mobile:Label>
</mobile:Form>
```

Директива в першому рядку повідомляє оброблювачу ASP про необхідність обробки даної сторінки, яка призначена для мобільних пристроїв.

Директива у другому рядку реєструє префікс «mobile» для звернення до елементів управління для мобільних пристроїв.

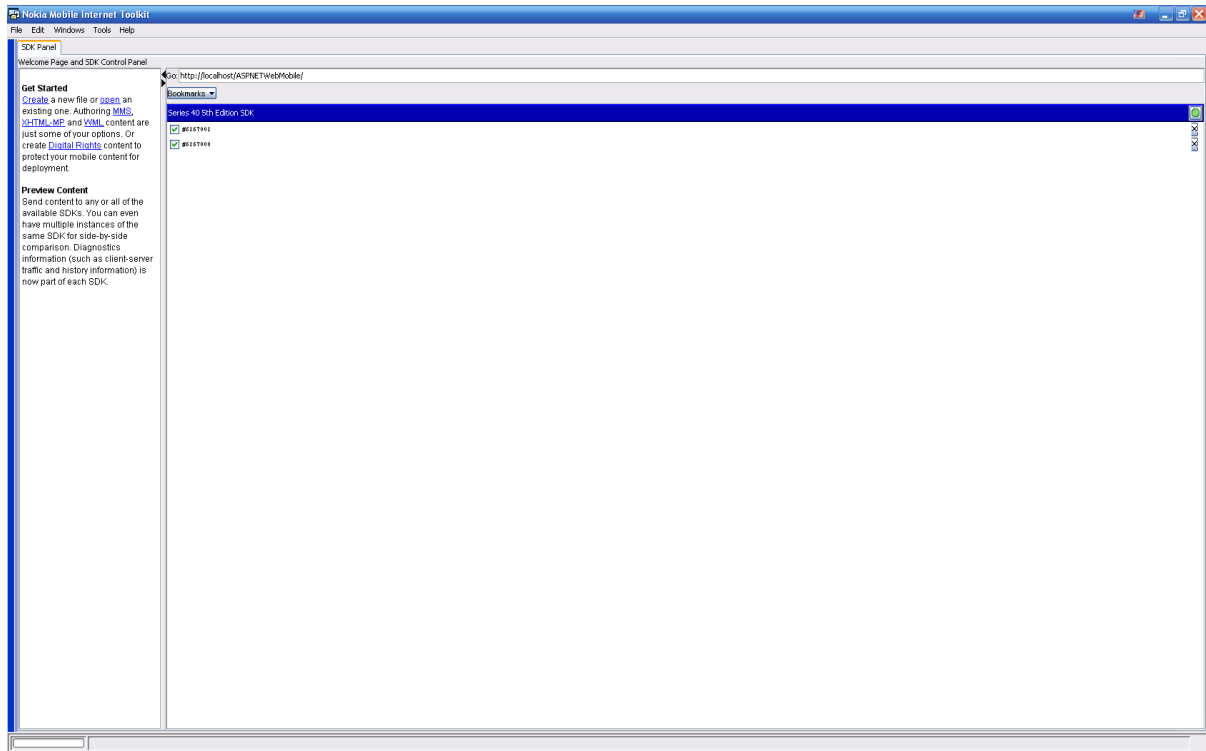
Тег <mobile: Form> вказує серверу на необхідність вставки форми.

Тег <mobile: Label> вказує серверу на необхідність вставки мітки з текстом «Hello world!».

4. Відкомпілюйте проект.

5. Створіть новий віртуальний каталог, який пов'язаний з директорією поточного проекту. Додайте до списку документів за замовчуванням сторінку *Default.aspx*.

6. Запустіть емулятор *Nokia Series 40 5th Edition SDK* і програму *Nokia Mobile Internet Toolkit*. У програмі *Nokia Mobile Internet Toolkit* вкажіть повний URL створеної за допомогою ASP.NET сторінки *Default.aspx*.



7. Після обробки даної сторінки для мобільного пристрою, що підтримує протокол WAP, буде згенерована сторінка, вид якої в емуляторі наведено нижче:



8. У емуляторі виберіть розділ меню «Tools>Diagnostics». У вікні діагностики, що з'явилося можна буде побачити наступний код сторінки, яка завантажена в емуляторі:

```
<?xml version='1.0'?>
<!DOCTYPE wml PUBLIC '-//WAPFORUM//DTD WML 1.1//EN'
'http://www.wapforum.org/DTD/wml_1.1.xml'><wml><head>
<meta http-equiv="Cache-Control" content="max-age=0" />
</head>
<card>
<p>Hello world!</p></card>
</wml>
```

Наступний приклад демонструє використання простого елемента форми для введення тексту (в режимі пароля):

```
<%@ Page Inherits= "System.Web.UI.MobileControls.MobilePage "
Language= "C# "%>
<%@ Register TagPrefix= "mobile " Namespace=
"System.Web.UI.MobileControls " Assembly= "System.Web.Mobile "
%>

<script runat="server" language="C#">

string uname;

protected void AgeClick(Object sender, EventArgs e)
{
    uname=text1.Text;
    ActiveForm=form2;
}

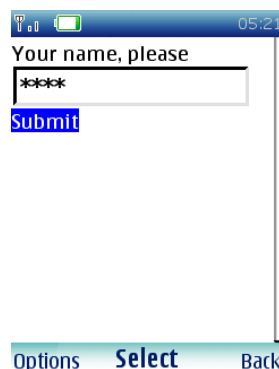
protected void Form2_Activate(Object sender, EventArgs e)
{
    message.Text="Wellcome " + uname + "!";
}
```

```
</script>

<Mobile:Form id="form1" runat="server">
<Mobile:Label runat="server">Your name, please</Mobile:Label>
<Mobile:TextBox runat="server" id="text1" password="true"/>
<Mobile:Command runat="server" OnClick="AgeClick" Text="Submit"
/>
</Mobile:Form>

<Mobile:Form id="form2" runat="server"
OnActivate="Form2_Activate">
<Mobile:Label runat="server" id="message" />
</Mobile:Form>
```

Якщо виконати як у попередньому прикладі всі кроки з 1 по 8, то після завантаження отриманої сторінки в емуляторі на екрані відобразиться перша форма:



Після введення рядка і натискання кнопки «*Submit*» активується друга форма:



Код сторінки для першої форми, що завантажений в мобільний пристрій, виглядає наступним чином:

```
<?xml version='1.0'?>
<!DOCTYPE wml PUBLIC '-//WAPFORUM//DTD WML 1.1//EN'
'http://www.wapforum.org/DTD/wml_1.1.xml'><wml><head>
<meta http-equiv="Cache-Control" content="max-age=0" />
</head>
<card>
<do type="prev" label="PḱP°P ·P°Pr"><prev /></do>
  <p>Your name, please<br/>
  <input name="mcsvprrws0" type="password" value="" />
    <anchor title="P'PiPμCḂPμPr">Submit
      <go href="Default.aspx?__ufps=058985"
        method="post">
        <postfield name="__EVENTTARGET"
          value="ctl01" />
        <postfield name="text1" value="$(mcsvprrws0)"
          />
      </go>
    </anchor>
  </p>
</card>
</wml>
```

Контрольне завдання

Розробіть за допомогою MS Visual Studio .NET серверний веб-додаток, що використовує керуючий ASP.NET Mobile елемент: *<Mobile: Calendar>*.

Додайте для цього елемента обробник події *OnSelectionChanged*, який повинен відображати вибраний користувачем мобільного пристрою день календаря за допомогою елемента *<Mobile: Label>*.

Код обробника події може виглядати наступним чином:

```
<script runat="server">

protected void CalChanged(Object sender, EventArgs e)
{
    label1.Text = "You selected: " + calendar1.SelectedDate;
}

</script>
```

Створіть необхідний віртуальний каталог на веб-сервері і перевірте роботу програми за допомогою емулятора мобільного пристрою.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО САМОСТІЙНОЇ РОБОТИ з дисципліни «Web-технології»

**Спеціальність: 6.010104 Професійна освіта
Напрямок підготовки: 0101 Педагогічна освіта**

ПЕРЕДМОВА

Самостійна робота є основним засобом засвоєння студентом навчального матеріалу в час, вільний від обов'язкових навчальних занять. Мета організації самостійної роботи – надавати максимальну можливість для розвитку особистих і професійних якостей, творчої індивідуальної майбутніх фахівців. Під час опрацювання запропонованого для самостійного опрацювання матеріалу студенти навчаються елементам організації власної роботи, отримують звичку до самонавчання, самовиховання.

Самостійна робота студента над засвоєнням навчального матеріалу з конкретної дисципліни може виконуватися у бібліотеці вищого навчального закладу, методичних кабінетах, навчальних лабораторіях, комп'ютерних класах, а також в домашніх умовах. Студентам також рекомендується для самостійного опрацювання відповідна наукова література та періодичні видання.

Навчальний матеріал навчальної дисципліни, передбачений робочим навчальним планом для засвоєння студентом в процесі самостійної роботи, виноситься на підсумковий контроль поряд з навчальним матеріалом, який опрацьовувався при проведенні навчальних занять.

Самостійна робота проводиться відповідно до заздалегідь складеного графіку, який доводиться до відома студентів на початку поточного семестру. Обов'язковим для студентів є конспектування матеріалу, винесеного на самостійне вивчення, в зошит для самостійної роботи.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО РОЗДІЛІВ І ТЕМ КУРСУ WEB-ТЕХНОЛОГІЇ

1. Опис основних подій і методів їхньої обробки.

Команда `onClick`.

Ми знаємо, що команда `onMouseOver` запускає подію, якщо навести курсор на посилання. Аналогічно, клацнувши по посиланню, можна з таким же успіхом запустити подію за допомогою команди `onClick`.

Щоб продемонструвати дія команди, скористаємося методом `alert`. Ось його схема:

```
alert('текст, що з'явиться у вікні')
```

Таким чином, застосовуючи той же підхід, що й для `onMouseOver`, одержуємо код:

```
<A HREF="http://www.mail.ru"  
onClick="alert('Подивитися пошту!');">
```

```
Клацніть тут</A>
```

При клацанні на посиланні з'явиться вікно з повідомленням "Подивитися пошту!"

Пам'ятайте, що усередині одинарних лапок не можна вживати слова з апострофами ', інакше браузер зрозуміє їх як закінчення тексту, а це буде помилка.

Команда `onFocus`.

Це чудовий оброблювач подій, що викликає дію, коли користувач "фокусується" на елементі сторінки. Він буде працювати для елементів форми: прапорців, текстових полів, текстових областей і ін.

Ось приклад:

```
<FORM>
```

```
<INPUT TYPE="text" SIZE="30"
```

```
onFocus="window.status='Текст у рядку стану';">
```

```
</FORM>
```

При використанні цього сценарію виводиться текстове поле, при клацанні в якому в рядку стану виводиться рядок "Текст у рядку стану".

Команда onBlur.

Якщо можна направити фокус на об'єкт, виходить, можна й "втратити фокус". Оброблювач подій onBlur дозволяє повідомити користувача про те, що він змінив свою відповідь. Цей оброблювач не так часто використовується, але наведемо все-таки приклад.

Створюється текстове поле з текстом. Треба змінити текст і потім клацнути мишею поза полем, імітуючи перехід фокуса до іншого елемента.

Легко догадатися, як це робиться, ось код:

```
<FORM>
```

```
<INPUT TYPE="text" SIZE="40"
```

```
VALUE="Впишіть своє ім'я й клацніть поза текстовим полем"
```

```
onBlur="alert('Ви змінили відповідь - ви впевнені,  
що вона правильна?');">
```

```
</FORM>
```

Команда onChange.

Дія цієї команди дуже схоже на дію попередньої, onBlur. Її головне завдання - перевірка.

Цей оброблювач події перевіряє, чи зробив користувач те, про що його просили. Приклад дуже схожий на попередній, але діє по-іншому.

```
<FORM>
```

```
<INPUT TYPE="text" SIZE="45"
```

```
VALUE="Змініть текст і клацніть поза полем - потім  
перевірте рядок стану"
```

```
onChange="window.status='Текст був змінений'">
```

```
</FORM>
```

Т.ч. при зміні тексту в полі й наступній зміні фокуса в рядку стану виводиться повідомлення 'Текст був змінений'.

Подія onSelect.

Ця команда працює так само, як і три попередні, указуючи, що в полі введення форми відбулися зміни.

Відмінність полягає в тому, що дана команда реагує, коли в полі введення щось було виділено.

Команда `onSubmit`.

Це дуже популярна команда. Вона дозволяє викликати яку-небудь дію, коли ви натискаєте кнопку `Submit` (відіслати, відправити). Коли користувач натискає на цю кнопку, команда виведе на екран сторінки текст: "Спасибі, що ви нам написали".

Формат команди наступний:

```
<FORM>
```

```
<INPUT TYPE="submit"
```

```
onSubmit="parent.location='спасибі.html';">
```

```
</FORM>
```

Тут використовуються нові команди. Схема начебто знайома, але якась дивна.

`parent.location` — це стандартна схема посилання на іншу сторінку. Можна подумати, що `parent` (джерело) — це об'єкт, а `location` (місцезнаходження) — метод. Невірно. У даному випадку `parent` є властивістю вікна браузера, а `location` — об'єктом, що повинен з'явитися в цьому вікні. Тобто для ясності просто майте на увазі, що `parent.location=""` означає посилання.

Команди `onLoad` і `onUnload`.

Обидві вони поміщаються усередину команди `<BODY>` документа HTML. Вони викликають подію, коли сторінка відкривається або закривається, тобто коли користувач іде зі сторінки. Ці команди будуть дуже корисні при роботі з функціями.

Література: [2, 3].

Завдання до самоперевірки.

Створіть форму, що буде взаємодіяти з користувачем (для цього необхідно знати про команди форми).

Форма повинна мати три елементи: поле введення із проханням увести ім'я; два поля для прапорців з питанням

про те, що воліє користувач - морозиво або шоколад; кнопку відправлення даних (submit).

З кожним елементом повинне відбутися наступне.

1. При введенні імені в рядку стану повинні з'явитися слова: Уведіть своє ім'я.

2. Два поля із прапорцями повинні відіслати в рядок стану слова: Ви вибрали... і вибір користувача.

3. При натисканні на кнопку відправлення повинне з'явитися вікно із вдячністю користувачеві за участь в опитуванні.

2 Структура об'єктів JavaScript, порядок їхнього підпорядкування.

В JavaScript є об'єкти, схожі на іменники або предмети. В об'єктів є властивості, які описують їх, як прикметники описують іменник. Ми посилаємося на властивості за допомогою схеми об'єкт.властивість.

Ще в об'єктів є методи, або дії, які можна виконати з об'єктом. Всі методи мають дужки й використовуються за схемою об'єкт.метод(). У різних об'єктів є різні властивості й методи.

Тепер ми познайомимосся з ієрархією об'єктів JavaScript.

Що мається на увазі:

Window

Parent

Self

Location

Href

Document

Image

Src

Form

Text

Submit

Checkbox

Top

Frames

Результат дії ієрархії.

Всі посилання починаються з найвищого об'єкта, window (вікно браузера), і йдуть по спадні. Вікна й рамки (frames) належать об'єкту window. На них не потрібно посилатися, якщо тільки їх не більше одного. Top, self, parent і frames — "убудовані" імена для вікон.

Ось кілька прикладів. Зверніть увагу на ієрархію.


```
document.mypic.src = "pic1.gif"
```

на самому початку window не потрібно. Передбачається, що це все й так перебуває усередині вікна. Команда `document.mypic.src` указує на зображення з ім'ям `mypic`, і змінює його вміст на `"pic1.gif"`. У цьому випадку `document` (документ) — це сторінка на якій перебуває елемент, `mypic` — ім'я елемента, а `SRC` — джерело елемента (`"pic1.gif"`).

```
document.write(location.href)
```

`write()` — це метод об'єкта `document`. `Location.href` містить повний URL вікна. Зверніть увагу, що `location` і `document` перебувають на одному рівні. Це значить, що ви одержуєте адресу документа того ж рівня.

Розбір ієрархії об'єктів

Window

Parent

Self

Location

Href

Document

Image

Src

Form

Text

Submit

Checkbox

Top

Frames

Сама велика плутанина в тім, що деякі об'єкти також є й властивостями.

- `window` — тільки об'єкт.
- `document` є властивістю вікна, але у свою чергу й об'єктом.

- `form` — це властивість документа, але також і об'єкт зі своїми властивостями.
- `value` (значення) і `SRC` (джерело) — тільки властивості.

Тут представлені не всі об'єкти й властивості. Однак цього досить, щоб зрозуміти концепцію в цілому... Всі посилання починаються зверху від `window` і йдуть по спадні. Тобто, не можна написати `document.mytext.myform` або `mypic.src.document`. Це неправильний порядок, варто писати зліва праворуч від більше загального до більше конкретного.

Важливе зауваження: щоб показати вміст поля форми, необхідно використовувати властивість `value` (значення), наприклад, `document.myform.mytext.value`. Якщо написати просто `document.myform.mytext`, то буде отримана інформація про поле форми, але не про його зміст.

Уважайте `value` ("значення") деяким показником того, що щось є або відсутній у певний час. Поле із прапорцем може мати значення `"on"` або `"off"`, залежно від того, заданий він чи ні. Текстове поле може мати значення `"hidden"` (сховане), якщо ви не хочете, щоб користувач його бачив. Текстове поле, як зазначено вище, може містити якийсь запис. Вона буде значенням цього поля.

Завдання до самоперевірки

Напишіть повні посилання, починаючи з `window`, навіть якщо це не потрібно. Зверніть увагу, що зображення має ім'я `mypic`, а форма — `myform`.

1. Посилання на всю форму;
2. Посилання на вміст поля `lname`;
3. Посилання на вміст поля `fname`;
4. Замініте зображення на `"marigold.gif"`.

Література: [2, 3].

3 Використання подій onUnLoad і onMouseOut.

onMouseOver викликає якусь подію, якщо навести мишу, приміром, на посилання. На противагу їй onMouseOut починає діяти, якщо курсор повести з посилання. Ми також знаємо, що команда onLoad запускає сценарій, коли сторінка завантажується. Команда onUnload діє, коли користувач іде зі сторінки.

Сценарій.

Наступний код використовує події при переміщенні покажчика миші:

```
<A                                HREF="index.htm"
onMouseOver="window.status='Агов! Убирайся з мене!';
return true"
onMouseOut="window.status=' Так-ото краще, спасибі!';
return true">
```

Наведіть курсор на це посилання й змістите убік</ A>

Використання команди onUnload при відході зі сторінки:

```
<BODY onUnload="alert('Уже йдете?')">
```

Результат роботи сценарію.

При розміщенні сценарію на сторінці виводиться текстове посилання "Наведіть курсор на це посилання й змістите убік". Якщо навести курсор на посилання й змістити убік кілька разів, то в рядку стану можна бачити повідомлення, що змінюються. Це перший результат. При натисканні на посилання можна побачити другий.

Розбір сценарію.

Ефекти з мишею, як легко бачити, створюються за допомогою команд onMouseOver і onMouseOut.

Зверніть увагу, що ці дві команди ніяк не зв'язані між собою. Вам не потрібно, щоб ці події відбувалися одночасно. Помнете, кілька уроків назад ми розділяли два оброблювачі подій комі, щоб вони відбувалися одночасно? Тут не той випадок. Потрібно, щоб одна подія відбувалася, коли курсор миші вказує на посилання, а інше — коли

курсор миші зміщується з посилання. Тому потрібно писати їх як дві зовсім незалежні команди, кожна з яких містить свою команду `return true`.

Повідомлення при відході зі сторінки створюється за допомогою команди `onUnload="alert('Уже йдете?')"`, що додана в рядок BODY документа HTML. Зверніть увагу на подвійні й одинарні лапки. Усередині подвійних - одинарні. Другі подвійні лапки означають для браузера кінець команди.

Завдання до самоперевірки.

У цьому завданні необхідно використовувати функції `onUnload`, `onMouseOver`, і `onMouseOut`. Зробіть наступне:

- Створіть сторінку з гіпертекстовим посиланням.
- Коли курсор указує на посилання, у рядку стану повинні з'являтися слова: "Привіт, користувач 'назва браузера!'".
- Коли курсор іде з посилання, у рядку стану повинен з'являтися текст: "Не нудьгуєте в нас на 'URL сторінки'?"
- Якщо клацнути на посиланні, повинне з'явитися вікно зі словами: "Уже йдете? Зараз усього тільки 'поточний час'";
- Час повинне визначатися за допомогою функції.
- Не користуйтеся командою `onClick`, щоб вивести вікно повідомлення, візьміть команду `onUnload`.

Література: [2, 3].

4. Відкриття вікна за допомогою функції.

Спробуємо створити функцію, що відкриє нове вікно, - причому й нове вікно, і весь його вміст буде втримуватися в тім же документі HTML. Тобто, у буквальному значенні слова ми вкладемо дві сторінки в одну.

Сценарій.

```
<SCRIPT type="text/javascript">
function openindex()
{
    var      OpenWindow=window.open("",      "newwin",
"height=300,width=300");
    OpenWindow.document.write("<HTML>")
    OpenWindow.document.write("<TITLE>Нове
вікно</TITLE>")
    OpenWindow.document.write("<BODY
BGCOLOR='00ffff'>")
    OpenWindow.document.write("<CENTER>")
    OpenWindow.document.write("<font      size=+1>Нове
вікно</font><P>")
    OpenWindow.document.write("<a
href='http://www.intuit.ru' target='main'>
    Це посилання<BR> відкриється в основному
вікні</a><p>")
    OpenWindow.document.write("<P><HR
WIDTH='60%'><P>")
    OpenWindow.document.write("<a      href="
onClick='self.close()'>
    Це посилання закриє вікно</a><p>")
    OpenWindow.document.write("</CENTER>")
    OpenWindow.document.write("</HTML>")
}
</SCRIPT>
і в команді BODY:
onLoad="openindex()"
Результат роботи сценарію.
```

Результат: відкрилося вікно з двома посиланнями. Але, все це було написано на одній сторінці.

Розбір сценарію.

Головна частина сценарію, що містить функцію, поміщається між тегами <HEAD> і </HEAD>, як більшість функцій.

За звичайною схемою для функції задається ім'я `openindex()`. Потім ідуть фігурні дужки. Тепер підходимо до основного моменту. Створюємо змінну `OpenWindow`, під якою ховається команда `window.open()`. Вона виглядає в такий спосіб:

```
var OpenWindow=window.open("", "newwin",  
"height=300,width=300");
```

Формат знайомий. Єдина різниця в тім, що не зазначений URL. Бачите порожні парні лапки? Вони говорять браузеру, що він повинен шукати в сценарії інформацію про нове вікно, - точно так само, як і у випадку відсутності URL у команді, що закриває вікно. Воно б не закрилося, якби почала завантажуватися нова сторінка. Те ж саме й отут. Браузер став би завантажувати нову сторінку, а не виконувати сценарій.

Тепер починаємо створювати сторінку HTML, що буде в новому вікні. От перший рядок тексту:

```
OpenWindow.document.write("<HTML>")
```

Команда говорить, що рядок тексту повинен бути записан в документ змінною `OpenWindow` (нове вікно) .

Подивіться на сценарій. Кожний новий рядок слідує тій же схемі. Можна написати сотню рядків, що створюють закінчену сторінку. Наш сценарій зовсім невеликий, тому що це навчальний приклад.

Пам'ятайте: коли ви пишете HTML усередині команди `document.write`, замість подвійних лапок з підкомандами ставте одинарні. Інакше буде помилка.

Нарешті оброблювач подій `onLoad` у команді `BODY` викликає функцію.

Завдання до самоперевірки.

- У цьому завданні необхідно написати функцію, що відкриває вікно. Документ, що з'явиться у вікні повинен мати зелене тло й заголовок TITLE: "Привіт, 'ім'я користувача", от твоє вікно!" Ім'я користувача можна довідатися за допомогою запиту (prompt). Додайте ще посилання, що закриє вікно.

Література: [2, 3].

Література

1. Котеров Д.В., Костарев А.Ф. "PHP 5 В Подлиннике".: БХВ-Петербург, 2006. – 1120 с.
2. Рева О.Н. "JavaScript в кармане".: Эксмо, 2008. - 256с.
3. А.Ю.Гаевский, В.А.Романовский. "100% самоучитель по созданию Web-страниц и Web-сайтов: HTML и javascript".: Триумф, 2008. - 454с.
4. Коггзолл Джон "PHP5 полное руководство".:Вильямс, 2006. - 752с.
5. Д.С. Зольников "PHP 5. Как самостоятельно создать сайт любой сложности".:ИТ Пресс, 2007. - 272с.

Пакет
комплексної контрольної роботи
з дисципліни «Web-технології»
Спеціальність: 6.010104 Професійна освіта
Напрямок підготовки: 0101 Педагогічна освіта

АНОТАЦІЯ ДО КОМПЛЕКСНОЇ КОНТРОЛЬНОЇ РОБОТИ

Призначення пакету ККР – об'єктивна оцінка якості підготовки студентів з дисципліни «Web-технології».

До пакету ККР з курсу «Web-технології» входить:

- комплект контрольних завдань з дисципліни (варіанти ККР);
- критерії оцінки виконання ККР;
- перелік довідкової літератури, використання якої дозволяється при виконанні комплексної контрольної роботи

Зміст складових пакету ККР.

Дисципліна «Web-технології» згідно чинної освітньо-професійної програми (ОПП) вищої освіти за спеціальністю 6.010104 «Професійна освіта». «Web-технології» – відноситься до циклу професійно-орієнтованої підготовки.

Зміст програми дисципліни відповідає анотації, що подана в ОПП. Програма структурована за темами з визначенням кількості годин відведених на кожну тему. Програма затверджена міністерством освіти і науки України.

Комплексна контрольна робота орієнтована на діагностику рівня сформованості знань студентами програми дисципліни.

Завдання, що видається студенту складається із трьох питань, що вимагають від студента глибокого розуміння тлумачення матеріалу, аналізу, обґрунтування та оцінки.

Обсяг і складність завдань відповідності часу, що визначений на виконання ККР – 2 академічні години.

КОМПЛЕКСНА КОНТРОЛЬНА РОБОТА
з дисципліни «Web-технології»
для студентів напряму підготовки 6.010104 Професійна
освіта

Варіант 1.

1. Охарактеризуйте сутність поняття Інтернет.
2. Визначте та проаналізуйте протокол SOAP.
3. Проаналізуйте стандарт DISCO.

Варіант 2.

1. Вкажіть стандарти Інтернету.
2. Проаналізуйте специфікацію WSDL.
3. Проаналізуйте специфікацію UDDI.

Варіант 3.

1. Охарактеризуйте стек протоколів TCP / IP.
2. Проаналізуйте Веб-сервіси.
3. Проаналізуйте система управління контентом (Content management system, CMS).

Варіант 4.

1. Охарактеризуйте систему доменних імен DNS.
2. Охарактеризуйте взаємодію в мережі Інтернет.
3. Проаналізуйте систему керування веб-контентом (Web content management system, WCMS або Web CMS).

Варіант 5.

1. Вкажіть нерекурсивні процедури DNS-імен.
2. Охарактеризуйте інтеграцію в мережі Інтернет.
3. Проаналізуйте систему WCMS Drupal.

Варіант 6.

1. Вкажіть рекурсивні процедури DNS-імен.
2. Охарактеризуйте модель XML DOM.
3. Проаналізуйте каркасну систему управління вмістом (Content Management Framework, CMF).

Варіант 7.

1. Вкажіть структуру і принципи WWW.
2. Проаналізуйте перетворення XML документів.
3. Охарактеризуйте синдекацію Веб-контенту.

Варіант 8.

1. Визначте та проаналізуйте проксі-сервери.
2. Проаналізуйте специфікацію XQuery.
3. Охарактеризуйте агрегування Веб-контенту.

Варіант 9.

1. Визначте та проаналізуйте протоколи Інтернет прикладного рівня.
2. Проаналізуйте специфікацію XSL-FO.
3. Охарактеризуйте бібліотек в просторі імен XML::RSS.

Варіант 10.

1. Проаналізуйте протокол Інтернету FTP.
2. Проаналізуйте специфікацію XPath.
3. Охарактеризуйте Веб-портали.

Варіант 11.

1. Проаналізуйте протокол Інтернету Telnet.
2. Проаналізуйте специфікацію XSLT.
3. Дайте класифікацію Веб-порталів.

Варіант 12.

1. Проаналізуйте протокол Інтернету SSH.
2. Охарактеризуйте XDR схему.
3. Охарактеризуйте загальнодоступні Веб-портали.

Варіант 13.

1. Проаналізуйте поштові протоколи.
2. Охарактеризуйте DTD схему.
3. Охарактеризуйте вертикальні Веб-портали.

Варіант 14.

1. Проаналізуйте протокол HTTP.
2. Охарактеризуйте введення в XML.
3. Охарактеризуйте корпоративні Веб-портали.

Варіант 15.

1. Поясніть яке є забезпечення безпеки передачі даних HTTP.
2. Проаналізуйте інтерфейси взаємодії веб-додатків з СКБД.
3. Проаналізуйте введення в технологію AJAX.

Варіант 16.

1. Проаналізуйте поняття механізм *cookie*.
2. Охарактеризуйте роботу з джерелами даних в ASP.NET.
3. Поясніть принципи розробки мобільних Веб-додатків.

Варіант 17.

1. Поясніть кроки створення матеріалу – метал.
2. Визначте та проаналізуйте серверні елементи керування ASP.NET.
3. Проаналізуйте формат передачі даних JavaScript Object Notation.

Варіант 18.

1. Перерахуйте програми, що виконуються на клієнт-машині.
2. Проаналізуйте серіалізацію об'єктів мови C#.
3. Охарактеризуйте протокол доступу Wireless Application Protocol.

Варіант 19.

1. Перерахуйте програми, що виконуються на сервері.
2. Охарактеризуйте інтерфейси C#.
3. Проаналізуйте сутність поняття «Дек».

Варіант 20.

1. Охарактеризуйте насичені інтернет-додатки.
2. Вкажіть вирази та оператори мови C#.
3. Проаналізуйте формат WAP Binary XML.

Варіант 21.

1. Проаналізуйте введення в JScript.
2. Проаналізуйте класи і методи в мові C#.
3. Проаналізуйте введення в Веб 2.0.

Варіант 22.

1. Вкажіть типи даних JScript.
2. Поясніть типи даних C#.
3. Охарактеризуйте сайти Веб 2.0.

Варіант 23.

1. Проаналізуйте функції та об'єкти JScript.
2. Проаналізуйте основи мови C#.
3. Поясніть сутність поняття «Мешап».

Варіант 24.

1. Охарактеризуйте *Visual Basic Scripting Edition*.
2. Охарактеризуйте платформу Microsoft Visual Studio.NET.
3. Проаналізуйте групи мешап.

Варіант 25.

1. Визначте та проаналізуйте Java-аплети.
2. Поясніть принципи програмування мовою PHP.
3. Проаналізуйте додатки для соціальних мереж.

Варіант 26.

1. Дайте загальну характеристику ActionScript.
2. Поясніть принципи програмування мовою Perl.
3. Охарактеризуйте поняття «Соціальний Веб» (Social Web).

Варіант 27.

1. Визначте мови інтерфейсів платформи Windows Vista.
2. Проаналізуйте ISAPI-додаток.
3. Вкажіть, які прикладні програми систем відносяться до соціального ПЗ.

Варіант 28.

1. Дайте поняття про DOM.
2. Проаналізуйте програмування на базі ASP.
3. Охарактеризуйте контент Фолксономія (folksonomy).

Варіант 29.

1. Визначте регулярні вирази.
2. Поясніть принцип програмування мовою Ruby.
3. Охарактеризуйте семантичну Веб-мережу (*Semantic Web*).

Варіант 30.

1. Охарактеризуйте стандарт CGI.
2. Поясніть принцип програмування мовою Python.
3. Дайте загальну характеристику семантичним Веб-сервісам.

КРИТЕРІЇ ОЦІНКИ ВИКОНАННЯ ЗАВДАНЬ КОМПЛЕКСНОЇ КОНТРОЛЬНОЇ РОБОТИ

Мета перевірки – виявити рівень знань наукового матеріалу з курсу «Web-технології».

При визначенні підсумкової оцінки за чотирьохбальною системою члени комісії керуються такими критеріями:

- оцінку «відмінно» заслуговує студент, який виявив всебічні, систематичні та глибокі знання, який здатний синтезувати знання окремих розділів дисципліни; послідовно, логічно, обґрунтовано та безпомилково викладає матеріал; вміло користується довідковою літературою;

- оцінку «добре» заслуговує студент, який виявив повне знання навчального матеріалу, передбаченого програмою, показав систематичний характер знань, здібний застосовувати свої знання у практичній діяльності;

- оцінку «задовільно» заслуговує студент, який виявив повне знання основ навчального матеріалу, передбаченого програмою, але припустив похибок у відповіді на запитання, спрощено викладав матеріал, проте має необхідні знання і спроможний використати їх для вирішення помилок;

- оцінку «незадовільно» виставляється студенту, у якого виявлено не повне знання основного програмного матеріалу, який припустився принципових помилок при виконанні завдань, передбачених програмою.

ПЕРЕЛІК ДОВІДКОВОЇ ЛІТЕРАТУРИ, ВИКОРИСТАННЯ ЯКОЇ ДОЗВОЛЯЄТЬСЯ ПРИ ВОКОНАННІ КОМПЛЕКСНОЇ КОНТРОЛЬНОЇ РОБОТИ

1. Троелсен Э. С# и платформа .NET. Библиотека программиста. – СПб. : «Питер», 2007. – 796 с.
2. Шеперд Д. Освой самостоятельно XML за 21 день. – М. : «Вильямс», 2002. – 432 с.
3. Форта Б. Освой самостоятельно регулярные выражения. 10 минут на урок. – М. : «Вильямс», 2005. – 184 с.

Пакет
ректорської контрольної роботи
з дисципліни «Web-технології»
Спеціальність: 6.010104 Професійна освіта
Напрямок підготовки: 0101 Педагогічна освіта

АНОТАЦІЯ ДО РЕКТОРСЬКОЇ КОНТРОЛЬНОЇ РОБОТИ

Призначення пакету РКР – об’єктивна оцінка якості підготовки студентів з дисципліни «Web-технології».

До пакету РКР з курсу «Web-технології» входить:

- комплект контрольних завдань з дисципліни (варіанти РКР);
- критерії оцінки виконання РКР;
- перелік довідкової літератури, використання якої дозволяється при виконанні ректорської контрольної роботи

Зміст складових пакету РКР.

Дисципліна «Web-технології» згідно чинної освітньо-професійної програми (ОПП) вищої освіти за спеціальністю 6.010104 «Професійна освіта». «Web-технології» - відноситься до циклу професійно-орієнтованої підготовки.

Зміст програми дисципліни відповідає анотації, що подана в ОПП. Програма структурована за темами з визначенням кількості годин відведених на кожну тему. Програма затверджена міністерством освіти і науки України.

Ректорська контрольна робота орієнтована на діагностику рівня сформованості знань студентами програми дисципліни.

Завдання, що видається студенту складається із трьох питань, що вимагають від студента глибокого розуміння тлумачення матеріалу, аналізу, обґрунтування та оцінки.

Обсяг і складність завдань відповідності часу, що визначений на виконання РКР – 2 академічні години.

РЕКТОРСЬКА КОНТРОЛЬНА РОБОТА
з дисципліни «Web-технології»
для студентів напряму підготовки 6.010104 Професійна
освіта

Завдання 1

1. Перерахуйте і охарактеризуйте три варіанти визначення типу документу HTML 4.01.
2. Перерахуйте і визначте основні складові частини URL.
3. Визначте елементи, які мають бути присутніми в таблиці HTML.
4. Охарактеризуйте метод, за допомогою якого можна знайти в документі усі елементи певного типу в динамічному HTML.

Завдання 2

1. Проаналізуйте ознаки строгого визначення типу документу HTML 4.01.
2. Охарактеризуйте параметри шрифту, які можна вказати за допомогою CSS-властивості font.
3. Проаналізуйте, яким чином сценарій Javascript вставляється в HTML-документ.
4. Перерахуйте і охарактеризуйте атрибути елементу текстового введення HTML-форми.

Завдання 3

1. Проаналізуйте ознаки перехідного визначення типу документу.
2. Проаналізуйте основні властивості CSS, що відносяться до тексту.
3. Поясніть, як оголошується змінна в Javascript, і які імена недопустимі.
4. Проаналізуйте спосіб діставання програмного доступу до об'єкту form HTML-форми.

Завдання 4

1. Визначте перший елемент структури HTML-сторінки, і його види в HTML 4.01.
2. Визначте властивості рамок, регульовані правилами CSS.
3. Проаналізуйте типи даних в Javascript.
4. Охарактеризуйте основні методи і події об'єкту form HTML-форми.

Завдання 5

1. Охарактеризуйте єдиний елемент найвищого кореневого рівня, що йде за ВТД, наведіть приклад його відкриваючого і закриваючого тега.
2. Проаналізуйте відмінність відступів від полів, визначте якими властивостями CSS вони задаються.

3. Поясніть створення масиву, і з чого розпочинається нумерація елементів масиву.

4. Охарактеризуйте основні методи і події елементів HTML-форми.

Завдання 6

1. Дайте визначення «тілу» і «шапці» документу HTML.

2. Визначте завдання положення і розмірів елементу при абсолютному позиціонуванні.

3. Перерахуйте і охарактеризуйте структури JavaScript, що управляють.

4. Поясніть, яких можливостей бракує в DHTML.

Завдання 7

1. Охарактеризуйте єдиний обов'язковий елемент розділу head, наведіть приклад мінімального розділу head.

2. Охарактеризуйте спадкоємство стилів.

3. Проаналізуйте варіанти синтаксису створення функцій в Javascript.

4. Дайте визначення Java – аплет, поясните, як вставити аплет у веб-сторінку.

Завдання 8

1. Проаналізуйте вміст документу, оформленого за стандартом HTML 4.01, наведіть приклад мінімального порожнього але коректного документу.
2. Визначте властивості, що викликають розриви сторінок при друці.
3. Охарактеризуйте спосіб створення декілька однотипних призначених для користувача об'єктів в Javascript.
4. Охарактеризуйте можливості і недоліки аплетів.

Завдання 9

1. Поясніть призначення каскадної таблиці стилів (CSS), і її розташування в документі.
2. Охарактеризуйте типи позиціонування CSS.
3. Визначте спосіб додавання методу до типу об'єкту в Javascript.
4. Проаналізуйте переваги технології Flash.

Завдання 10

1. Дайте розшифровку і поясніть аббревіатури HTML і CSS.
2. Проаналізуйте правила розміщення елементів в нормальному потоці.

3. Проаналізуйте створення об'єкту за відсутності необхідності визначення його типу.

4. Поясніть, що з себе представляє документ SVG.

Завдання 11

1. Дайте визначення і охарактеризуйте тег, елемент і атрибут.

2. Охарактеризуйте абсолютне позиціонування і сфера його застосування.

3. Проаналізуйте призначення динамічного HTML і технології, які він поєднує.

4. Проаналізуйте основну ідею технології AJAX та її використання.

Завдання 12

1. Охарактеризуйте вміст елемента.

2. Визначте позиціонування на основі плаваючих елементів.

3. Поясніть роль об'єктної моделі документу в динамічному HTML.

4. Охарактеризуйте особливості технології Microsoft Silverlight.

Завдання 13

1. Назвіть і охарактеризуйте останню версію стандарту HTML, дату і автора його створення.
2. Визначте сферу застосування відносного позиціонування.
3. Коротко охарактеризуйте об'єкти window, history, navigator, document, location, screen в динамічному HTML.
4. Проаналізуйте нові види елементів формив HTML
- 5.

Завдання 14

1. Визначте відмінність строгого і перехідного ВТД.
2. Визначте центрування елементу на сторінці по горизонталі.
3. Перерахуйте і охарактеризуйте найчастіше вживані методи об'єкту window в динамічному HTML.
4. Охарактеризуйте можливості HTML 5 по перевірці введених у форму даних.

Завдання 15

1. Проаналізуйте, які елементи обов'язково повинні входити в документ HTML.
2. Проаналізуйте можливість регулювання порядку накладення елементів.

3. Дайте визначення cookie, поясніть спосіб їх створення і читання.

4. Проаналізуйте відмінність технологій SVG і Canvas.

Завдання 16

1. Визначте правила, відрізняючі XHTML від HTML.

2. Визначте елемент, використовуваний для вставки зображення в документ, і його атрибути.

3. Охарактеризуйте метод, за допомогою якого можна знайти в документі елемент по його id.

4. Перерахуйте і охарактеризуйте додаткові селектори CSS3.

Завдання 17

1. Проаналізуйте значення заголовків в документі і визначте, за допомогою яких елементів вони створюються і як виглядають.

2. Проаналізуйте властивості, що дозволяють вказати або обмежити розміри зображення.

3. Проаналізуйте методи DOM, використовувані для створення і видалення елементів.

4. Проаналізуйте нові методи вибірки елементів документу введених в DOM в HTML5.

Завдання 18

1. Охарактеризуйте елемент, формуючий абзац.
2. Охарактеризуйте спосіб досягнення обтікання зображень текстом.
3. Поясніть призначення властивості `innerHTML`.
4. Охарактеризуйте нові властивості `CSS3`, вживані для роботи з рамками і кольорами.

Завдання 19

1. Проаналізуйте логічне і фізичне форматування і їх відмінність.
2. Проаналізуйте властивості, що визначають фонове зображення елемента.
3. Поясніть значення події `DOM`.
4. Проаналізуйте, які додаткові обережності слід приймати, користуючись новими засобами `HTML` та `CSS`.

Завдання 20

1. Назвіть декілька елементів логічного і фізичного форматування і поясніть їх значення.
2. Проаналізуйте використання ролловерів і їх створення за допомогою фонових зображень.
3. Перерахуйте і проаналізуйте основні події `DOM`.

4. Перерахуйте і охарактеризуйте три варіанти визначення типу документу HTML 4.01.

Завдання 21

1. Визначте засоби, використовувані для вказівки розривів рядків в тексті, окрім блокових елементів.

2. Визначте спосіб створення і дії клієнтської навігаційної карти.

3. Перерахуйте і проаналізуйте властивості об'єкту event DOM.

4. Проаналізуйте ознаки строгого визначення типу документу HTML 4.01.

Завдання 22

1. Проаналізуйте призначення елементу div.

2. Охарактеризуйте призначення колонтитулів таблиці.

3. Проаналізуйте спосіб діставання доступу до об'єкту event у функції-обробнику події – в ІЕ і інших браузерах.

4. Проаналізуйте ознаки перехідного визначення типу документу.

Завдання 23

1. Визначте способи маркіровки і нумерації списків закладені в HTML.

2. Проаналізуйте спосіб і призначення об'єднання стовпців в групі.

3. Визначте спосіб діставання доступу до елементу-джерела події, маючи об'єкт event.

4. Визначте перший елемент структури HTML-сторінки, і його види в HTML 4.01.

Завдання 24

1. Охарактеризуйте ефекти (що стосуються маркіровки), які можна досягти, застосовуючи правила CSS до елементів списку.

2. Охарактеризуйте атрибути, що використовуються для об'єднання осередків по горизонталі і вертикалі.

3. Проаналізуйте спосіб передання обробникові події посилання на елемент-джерело.

4. Дайте визначення «тілу» і «шапці» документу HTML.

Завдання 25

1. Проаналізуйте спосіб включення гіперпосилання в HTML-документ.

2. Проаналізуйте маніпуляції, які можна робити з вмістом таблиці за допомогою методів і властивостей об'єктної моделі таблиці.

3. Проаналізуйте можливість зв'язування елементу-джерела і події з функцією-обробником.

4. Охарактеризуйте єдиний обов'язковий елемент розділу head, наведіть приклад мінімального розділу head.

Завдання 26

1. Проаналізуйте призначення особливих позначень для спеціальних символів, наведіть приклади.

2. Проаналізуйте переваги і недоліки табличного верстання макету сторінки.

3. Поясніть значення ключового слова `this` у функціях-обробниках подій.

4. Проаналізуйте вміст документу, оформленого за стандартом HTML 4.01, наведіть приклад мінімального порожнього але коректного документ.

Завдання 27

1. Проаналізуйте і перерахуйте одиниці виміри, вживані в CSS, визначите, які з них використовуються найчастіше.

2. Охарактеризуйте переваги і недоліки верстання з фіксованою і непостійною шириною колонок.

3. Дайте визначення HTML – форми і проаналізуйте її призначення.

4. Поясніть призначення каскадної таблиці стилів (CSS), і її розташування в документі.

Завдання 28

1. Проаналізуйте способи завдання кольору в CSS.
2. Визначте прийом, використовуваний для заповнення фоном усієї висоти колонки з невеликою кількістю вмісту.
3. Охарактеризуйте специфічні атрибути HTML-форми.
4. Дайте визначення і охарактеризуйте тег, елемент і атрибут.

Завдання 29

1. Визначте способи завдання розміру шрифту.
2. Проаналізуйте HTML-структуру, яку має багато колоночний макет.
3. Охарактеризуйте, дані яких елементів управління вирушають на сервер в HTML-формах.
4. Назвіть і охарактеризуйте останню версію стандарту HTML, дату і автора його створення.

Завдання 30

1. Проаналізуйте основні властивості CSS, що відносяться до тексту.
2. Охарактеризуйте призначення мови JavaScript.

3. Перерахуйте і охарактеризуйте елементи управління HTML-форми.

4. Проаналізуйте, які елементи обов'язково повинні входити в документ HTML.

КРИТЕРІЇ ОЦІНКИ ВИКОНАННЯ ЗАВДАНЬ РЕКТОРСЬКОЇ КОНТРОЛЬНОЇ РОБОТИ

Мета перевірки – виявити рівень знань наукового матеріалу з курсу «Web-технології».

При визначенні підсумкової оцінки за чотирьохбальною системою члени комісії керуються такими критеріями:

- оцінку «відмінно» заслуговує студент, який виявив всебічні, систематичні та глибокі знання, який здатний синтезувати знання окремих розділів дисципліни; послідовно, логічно, обґрунтовано та безпомилково викладає матеріал; вміло користується довідковою літературою;

- оцінку «добре» заслуговує студент, який виявив повне знання навчального матеріалу, передбаченого програмою, показав систематичний характер знань, здібний застосовувати свої знання у практичній діяльності;

- оцінку «задовільно» заслуговує студент, який виявив повне знання основ навчального матеріалу, передбаченого програмою, але припустив похибок у відповіді на запитання, спрощено викладав матеріал, проте має необхідні знання і спроможний використати їх для вирішення помилок;

- оцінку «незадовільно» виставляється студенту, у якого виявлено не повне знання основного програмного матеріалу, який припустився принципів помилок при виконанні завдань, передбачених програмою.

ПЕРЕЛІК ДОВІДКОВОЇ ЛІТЕРАТУРИ, ВИКОРИСТАННЯ ЯКОЇ ДОЗВОЛЯЄТЬСЯ ПРИ ВОКОНАННІ РЕКТОРСЬКОЇ КОНТРОЛЬНОЇ РОБОТИ

1. Троелсен Э. С# и платформа .NET. Библиотека программиста. – СПб. : «Питер», 2007. – 796 с.
2. Шеперд Д. Освой самостоятельно XML за 21 день. – М. : «Вильямс», 2002. – 432 с.
3. Форта Б. Освой самостоятельно регулярные выражения. 10 минут на урок. – М. : «Вильямс», 2005. – 184 с.

**ПРОГРАМА ЕКЗАМЕНУ
З ДИСЦИПЛІНИ «WEB-ТЕХНОЛОГІЇ»**
Спеціальність: 6.010104 Професійна освіта
Напрямок підготовки: 0101 Педагогічна освіта

АНОТАЦІЯ

Екзамен з дисципліни «Web-технології» проводиться для студентів спеціальності 6.010104 «Професійна освіта».

До програми екзамену з дисципліни «Web-технології» входить 30 білетів. Кожен білет складається з 3-х питань теоретичного матеріалу. Завдання виконуються в письмовому вигляді.

Питання у білетах потребують уміння застосовувати інтегровані знання програмного матеріалу дисципліни «Web-технології» у відповідності до програми, затвердженої науково-методичною радою факультету фізико-математичної і технологічної освіти.

Зміст варіантів завдань в повній мірі відповідає програмним вимогам, які ставляться до підготовки фахівців за спеціальністю 6.010104 «Професійна освіта».

Варіанти білетів мають приблизно однакову складність. Відповідь на питання білету оцінюється за чотирибальною системою «відмінно», «добре», «задовільно», «незадовільно», що є обґрунтованим і відповідає установленим вимогам.

ПИТАННЯ ДО ЕКЗАМЕНУ З ДИСЦИПЛІНИ «WEB-ТЕХНОЛОГІЇ»

1. Що таке Інтернет?
2. Роль стандартизації в Інтернет?
3. Стек протоколи TCP / IP.
4. Система доменних імен DNS.
5. Структура і принципи WWW.

6. Проксі-сервери.
7. Протоколи Інтернет прикладного рівня.
8. Протокол HTTP.
9. Забезпечення безпеки передачі даних HTTP.
10. Cookie.
11. Програми, що виконуються на клієнт-машині.
12. Програми, що виконуються на сервері.
13. Насичені інтернет-додатки.
14. Коротка характеристика VBScript.
15. Java-аплети.
16. ActionScript - загальна характеристика.
17. XAML і Microsoft Silverlight.
18. Поняття про DOM.
19. DHTML.
20. Регулярні вирази.
21. Стандарт CGI.
22. Сценарії веб-додатків.
23. Мова Python.
24. Мова Ruby.
25. Веб-додаток ASP.
26. Веб-додаток ISAPI.
27. Мова Perl.
28. Мова PHP.
29. Платформа Visual Studio.NET.
30. Основи C#.
31. Типи даних C#.
32. Класи і методи в C#.
33. Вирази та оператори C#.
34. Інтерфейси C#.
35. Сериалізація об'єктів.
36. Платформа .NET Framework.
37. Серверні елементи керування ASP.NET.
38. Робота з джерелами даних в ASP.NET.
39. Взаємодія веб-додатків з СКБД.
40. Dodatok ADO.NET.

41. Мова SGML.
42. Мова XML.
43. DTD схема.
44. XDR схема.
45. Модель XML DOM.
46. Мови XSLT и XPath.
47. Мова XSL-FO.
48. Мова XQuery.
49. Інтеграція на основі XML.
50. Веб-сервіси (веб-служби).
51. Специфікація WSDL.
52. Протокол SOAP.
53. Стандарт DISCO.
54. Специфікація UDDI.
55. Система керування веб-контентом.
56. Система Drupal.
57. Синдикація веб-контенту.
58. Агрегування веб-контенту.
59. RSS – сімейство XML-форматів.
60. Веб-портали.
61. Класифікація веб-порталів.
62. Технологія AJAX.
63. Розробка мобільних веб-додатків.
64. Веб 2.0.
65. Мешапи.
66. «Соціальний Веб».
67. Фолксономія.
68. Семантична веб-мережа.
69. Онтологія.
70. Семантичні веб-сервіси.

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 1.

1. Що таке Інтернет?
2. Типи даних C#.
3. Класифікація веб-порталів.

Затверджено на засіданні кафедри технічних дисциплін

Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 2.

1. Роль стандартизації в Інтернет?
2. Класи і методи в C#.
3. Технологія AJAX.

Затверджено на засіданні кафедри технічних дисциплін

Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 3.

1. Стек протоколи TCP / IP.
2. Вирази та оператори C#.
3. Розробка мобільних веб-додатків.

Затверджено на засіданні кафедри технічних дисциплін

Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 4.

1. Система доменних імен DNS.
2. Інтерфейси C#.
3. Веб 2.0.

Затверджено на засіданні кафедри технічних дисциплін

Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 5.

1. Структура і принципи WWW.
2. Сериалізація об'єктів.
3. Мешапи.

Затверджено на засіданні кафедри технічних дисциплін

Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 6.

1. Проксі-сервери.
2. Платформа .NET Framework.
3. «Соціальний Веб».

Затверджено на засіданні кафедри технічних дисциплін

Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 7.

1. Протоколи Інтернет прикладного рівня.
2. Серверні елементи керування ASP.NET.
3. Фолксономія.

Затверджено на засіданні кафедри технічних дисциплін

Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 8.

1. Протокол HTTP.
2. Робота з джерелами даних в ASP.NET.
3. Семантична веб-мережа.

Затверджено на засіданні кафедри технічних дисциплін

Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 9.

1. Забезпечення безпеки передачі даних HTTP.
2. Взаємодія веб-додатків з СКБД.
3. Онтологія.

Затверджено на засіданні кафедри технічних дисциплін
Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 10.

1. Cookie.
2. Додаток ADO.NET.
3. Семантичні веб-сервіси.

Затверджено на засіданні кафедри технічних дисциплін
Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010103. Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 11.

1. Програми, що виконуються на клієнт-машині.
2. Мова SGML.
3. Роль стандартизації в Інтернет?

Затверджено на засіданні кафедри технічних дисциплін
Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 12.

1. Програми, що виконуються на сервері.
2. Мова XML.
3. Стек протоколи TCP / IP.

Затверджено на засіданні кафедри технічних дисциплін
Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 13.

1. Насичені інтернет-додатки.
2. DTD схема.
3. Система доменних імен DNS.

Затверджено на засіданні кафедри технічних дисциплін

Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 14.

1. Коротка характеристика VBScript.
2. XDR схема.
3. Структура і принципи WWW.

Затверджено на засіданні кафедри технічних дисциплін

Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 15.

1. Java-аплети.
2. Модель XML DOM.
3. Протоколи Інтернет прикладного рівня.

Затверджено на засіданні кафедри технічних дисциплін
Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 16.

1. ActionScript - загальна характеристика.
2. Мови XSLT и XPath.
3. Забезпечення безпеки передачі даних HTTP.

Затверджено на засіданні кафедри технічних дисциплін
Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 17.

1. XAML і Microsoft Silverlight.
2. Мова XSL-FO.
3. Програми, що виконуються на клієнт-машині.

Затверджено на засіданні кафедри технічних дисциплін
Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 18.

1. Поняття про DOM.
2. Мова XQuery.
3. Програми, що виконуються на сервері.

Затверджено на засіданні кафедри технічних дисциплін
Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 19.

1. DHTML.
2. Інтеграція на основі XML.
3. Насичені інтернет-додатки.

Затверджено на засіданні кафедри технічних дисциплін
Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 20.

1. Регулярні вирази.
2. Веб-сервіси (веб-служби).
3. Коротка характеристика VBScript.

Затверджено на засіданні кафедри технічних дисциплін
Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 21.

1. Стандарт CGI.
2. Специфікація WSDL.
3. ActionScript - загальна характеристика.

Затверджено на засіданні кафедри технічних дисциплін
Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 22.

1. Сценарії веб-додатків.
2. Протокол SOAP.
3. XAML і Microsoft Silverlight.

Затверджено на засіданні кафедри технічних дисциплін
Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 23.

1. Мова Python.
2. Стандарт DISCO.
3. Поняття про DOM.

Затверджено на засіданні кафедри технічних дисциплін

Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 24.

1. Мова Ruby.
2. Специфікація UDDI.
3. Стандарт CGI.

Затверджено на засіданні кафедри технічних дисциплін

Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ №25.

1. Веб-додаток ASP.
2. Система керування веб-контентом.
3. Сценарії веб-додатків.

Затверджено на засіданні кафедри технічних дисциплін

Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 26.

1. Веб-додаток ISAPI.
2. Система Drupal.
3. Мова Python.

Затверджено на засіданні кафедри технічних дисциплін

Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 27.

1. Мова Perl.
2. Синдикація веб-контенту.
3. Мова Ruby.

Затверджено на засіданні кафедри технічних дисциплін
Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 28.

1. Мова PHP.
2. Агрегування веб-контенту.
3. Веб-додаток ASP.

Затверджено на засіданні кафедри технічних дисциплін
Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 29.

1. Платформа Visual Studio.NET.
2. RSS – сімейство XML-форматів.
3. Веб-додаток ISAPI.

Затверджено на засіданні кафедри технічних дисциплін
Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

БЕРДЯНСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

Спеціальність: 6.010104. Професійна освіта.

Навчальна дисципліна: **Web-технології**

Семестр: VII

ЕКЗАМЕНАЦІЙНИЙ БІЛЕТ № 30.

1. Основи C#.
2. Веб-портали.
3. Мова Perl.

Затверджено на засіданні кафедри технічних дисциплін
Протокол № 1 від « 28 » серпня 2015 р.

Зав. кафедрою _____ В.І. Перегудова

Екзаменатор _____ С.В. Онищенко

ЛІТЕРАТУРА

1. Беллиньясо М. Разработка Web-приложений в среде ASP.NET 2.0: с примерами на C# / М. Беллиньясо. – М. : «Диалектика», 2007. – 640 с.
2. Дунаев В. В. JavaScript / В. В. Дунаев. – СПб. : «Питер», 2003. – 394 с.
3. Крейн Д. AJAX в действии: технология – Asynchronous JavaScript and XML = AJAX in Action / Д. Крейн, Э. Паскарелло, Д. Джеймс. – М. : «Вильямс», 2006. – 640 с.
4. Мельтцер К. Разработка CGI-приложений на Perl / К. Мельтцер. – М. : «Вильямс», 2001. – 395 с.
5. Ньюкомер Э. Веб-сервисы: XML, WSDL, SOAP и UDDI / Э. Ньюкомер. – СПб. : Питер, 2003. – 345 с.
6. Роджерс Д. Программирование на Microsoft JScript.NET / Д. Роджерс. – М. : «Вильямс», 2002. – 352 с.
7. Салмре И. Программирование мобильных устройств на платформе .NET Compact Framework / И. Салмре. – М. : «Вильямс», 2006. – 736 с.
8. Старыгин А. XML: разработка Web-приложений / А. Старыгин. – СПб. : «БХВ-Петербург», 2003. – 585 с.
9. Троелсен Э. C# и платформа .NET. Библиотека программиста / Э. Троелсен. – СПб. : «Питер», 2007. – 796 с.
10. Форта Б. Освой самостоятельно регулярные выражения. 10 минут на урок / Б. Форта. – М. : «Вильямс», 2005. – 184 с.
11. Храмцов П. Б. Основы WEB-технологий. / П. Б. Храмцов, С. А. Брик, А. М. Русак, А. И. Сурин – М. : ИТУИТ.РУ, 2003. – 512 с.
12. Шапошников И. PHP 5.1: учебный курс / И. Шапошников. – СПб. : «Питер», 2007. – 192 с.

13. Шеперд Д. Освой самостоятельно XML за 21 день / Д. Шеперд. – М. : «Вильямс», 2002. – 432 с.
14. Шорт С. Разработка XML Web-сервисов средствами Microsoft.NET / С. Шорт. – СПб. : «БХВ-Петербург», 2003. – 480 с.

ЗМІСТ

ВСТУП	3
РОБОЧА ПРОГРАМА ДИСЦИПЛІНИ	5
Лекція 1. Введення в веб-технології: структура та принципи Веб.	15
Лекція 2. Введення в клієнт-серверні технології Веб. Протокол http.	35
Лекція 3. Клієнтські сценарії та додатки.	56
Лекція 4. Серверні веб-додатки.	71
Лекція 5. Мови розробки сценаріїв Perl і PHP.	88
Лекція 6. Введення в C# і платформу Visual Studio.Net.	100
Лекція 7. Архітектура веб-додатків ASP.NET. Розробка веб-додатків на платформі .NET.	111
Лекція 8. Інтерфейси взаємодії веб-додатків з СКБД ...	118
Лекція 9. Введення в XML.	121
Лекція 10. Мови опису схем XML.	130
Лекція 11. DOM XML. Перетворення XML документів.	142
Лекція 12. Інтеграція та взаємодія в мережі Інтернет. .	151
Лекція 13. Організація процесу розробки веб-контенту CMS/CMF.	162
Лекція 14. Синдикація і агрегування веб-контенту.	166
Лекція 15. Веб-портали. Класифікація веб-порталів. ...	171
Лекція 16. Введення в технологію AJAX. Розробка мобільних веб-додатків.	177
Лекція 17. Введення в Веб 2.0.	185
Лекція 18. Додатки для соціальних мереж.	190
Лабораторна робота № 1	
Службові утиліти для роботи в Інтернет. Вивчення протоколу HTTP	198
Лабораторна робота № 2	
Принципи веб-дизайну. Знайомство з Microsoft Expression Web.	210
Лабораторна робота № 3	
Введення в JavaScript. Програмна взаємодія з HTML	

документами на основі DOM API.	226
Лабораторна робота № 4	
Клієнтські сценарії. Використання регулярних виразів.	243
Лабораторна робота № 5	
Розробка CGI-додатків на Perl і PHP.	256
Лабораторна робота № 6	
Знайомство з середовищем розробки Microsoft Visual Studio.NET. Структура програми на C#. Основи мови C#.	276
Лабораторна робота № 7	
Основи мови C#. Робота з масивами і рядками. Інтерфейси і колекції.	288
Лабораторна робота №8	
Основи розробки веб-додатків за допомогою ASP.NET.	306
Лабораторна робота №9	
Серверні елементи управління ASP.NET.	313
Лабораторна робота №10	
Робота з джерелами даних в ASP.NET.	323
Лабораторна робота №11	
Структура XML документа. XML-схеми.	340
Лабораторна робота №12	
Програмна обробка XML документів за допомогою XML DOM.	363
Лабораторна робота №13	
Форматування і перетворення XML документа за допомогою XSL. XSLT перетворення XML документа. .	377
Лабораторна робота №14	
Розробка веб-служби в ASP.NET.	387
Лабораторна робота №15	
Розробка веб-служби в ASP.NET. Створення проксі-збірки для веб-служби.	398
Лабораторна робота №16	
Приклади розробки RSS-джерел і RSS-рідерів.	408

Лабораторна робота №17

Реалізація асинхронної взаємодії веб-браузера з веб-сервером за допомогою технології AJAX. 419

Лабораторна робота №18

Створення мобільних веб-додатків за допомогою ASP.NET. 427

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО САМОСТІЙНОЇ РОБОТИ. 436

ПАКЕТ КОМПЛЕКСНОЇ КОНТРОЛЬНОЇ РОБОТИ. ... 449

ПАКЕТ РЕКТОРСЬКОЇ КОНТРОЛЬНОЇ РОБОТИ. 461

ПРОГРАМА ЕКЗАМЕНУ З ДИСЦИПЛІНИ «WEB-ТЕХНОЛОГІЇ» 478

ЛІТЕРАТУРА 496

Навчальне видання

Онищенко Сергій Вікторович

WEB-ТЕХНОЛОГІЇ:
навчально-методичний комплекс

Навчальний посібник

Комп'ютерна верстка та дизайн обкладинки

С. В. Онищенко

Технічний редактор В. М. Ліпич

Надруковано з оригінал-макету, наданого автором

Підписано до друку 05.12.2016 р.

Формат 60x84/16. Папір офсетний.

Гарнітура "Book Antiqua". Друк – лазерний.

Ум.-друк. арк. 24. Наклад 50 прим. Зам. № 08.

Бердянський державний педагогічний університет

Вул. Шмідта 4, м. Бердянськ, 71100

Свідоцтво суб'єкта видавничої справи ДК № 2961 від
05.09.2007 р.