

Міністерство освіти і науки України
ЗНТУ

Кафедра програмних засобів

"Програмування Інтернет-застосунків"

Конспект лекцій
для студентів спеціальності

121 Інженерія програмного забезпечення

усіх форм навчання

Запоріжжя 2016

ЗМІСТ

Тема 1. Вступ до Інтернет програмування	4
1.1. Інтернет-додатки - основні відомості	4
1.2. Принцип роботи Web-додатків	4
1.3. Можливості ASP. NET	5
1.4. Переваги ASP.NET	5
1.5. Структура Web-додатків	6
1.6. Компоненти Web-форм	7
1.7. Мови програмування	8
1.8. Створення проекту Web-додатка	8
1.8.1. Створення нового Web-додатка	8
1.8.2. Керування проектом за допомогою IIS	9
1.8.3. Створення віртуальних каталогів для Web-додатків	9
1.8.4. Створення каталогу subweb	10
1.8.5. Створення нового проекту у віртуальному каталозі	10
1.8.6. Розміщення елементів керування й написання коду	10
Тема 2. Аутентифікація і авторизація користувачів	12
2.1. Анонімний доступ	12
2.2. Доступ аутентифікованих користувачів	13
2.3. Аутентифікація запитів HTML-сторінок	14
2.4. Аутентифікація Windows	15
2.4.1. Включення аутентифікації Windows	15
2.4.2. Призначення прав доступу окремим користувачам	16
2.4.3. Застосування авторизації, заснованої на ролях	17
2.4.4. Витяг ідентифікаційних даних користувача	17
2.4.5. Параметри IIS і аутентифікація засобами Windows	17
2.5. Аутентифікація Forms	18
2.5.1. Як задіяти аутентифікацію за допомогою форм	19
2.5.2. Вибір методу аутентифікації	19
2.5.3. Створення вхідної Web-форми	20
2.5.4. Аутентифікація за допомогою бази даних	22
2.5.5. Додавання користувачів у базу даних	22
2.6. Аутентифікація Passport	24
2.6.1. Як задіювати аутентифікацію Passport	24
Тема 3. Обробка подій	26
3.1. Події життєвого циклу Web-додатка	26
3.2. Збереження даних Web-форми	27
3.3. Події об'єктів Application і Session	28
3.4. Події Web-форми	29
3.5. Події серверних елементів керування	30
3.6. Відключення змінної стану Session	30
3.7. Визначення моменту завершення додатка	30
Тема 4. Простори імен в Web-додатках	31
4.1. Огляд простору імен Web	31
4.2. Використання об'єкту Application	32
4.3. Використання об'єкту Page	33
4.4. Використання об'єкту Response	34
Тема 5. Збереження відомостей про стан	34
5.1. Способи збереження відомостей про стан	34
5.1.1. Використання рядків запитів	35
5.1.2. Використання файлів cookie	35

5.1.3. Використання стану відображення	36
5.1.4. Використання змінних стану Application і Session.....	36
5.2. Впорядкування доступу до змінних стану	37
Тема 6. Використання баз даних в додатках ASP.NET	38
6.1. Основні відомості про модель доступу до даних ADO.NET	38
6.1.1. Архітектура ADO.NET	38
6.2. Організація взаємодії з БД	41
6.2.1 Підключення до БД	41
6.2.2. Виконання команд над наборами даних	43
6.2.3. Використання команд, що параметризуються.....	45
6.2.4. Використання процедур, що зберігаються	46
6.3. Від'єднані набори даних.....	47
6.3.1. Клас DataSet 	47
6.3.2. Використання DataSet.....	48
6.3.3. Клас DataAdapter	48
6.4. Прив'язка і відображення даних	49
6.4.1. Прив'язка з одним значенням.....	49
6.4.2. Прив'язка з множинним значенням.....	50
6.5. Використання параметрів в запитах	53
6.6. Оновлення даних	54
Тема 7. Практична реалізація баз даних в інтернеті.....	54
7.1. Яким має бути ефективний Інтернет-магазин.....	54
7.1.2 . Послідовність роботи інтернет-магазину:	54
7.1.3. Переваги і можливості Інтернет-магазину	55
7.1.4. Недоліки Інтернет-магазину	55
7.2. Конструювання інтернет-магазину.....	55
7.3. Технологія ASP	56
7.4. Інструменти	57
7.5. Розміщення сайту в інтернеті	57
7.6. База даних інтернет-додатка	58
Таблиці	59
7.7. Створюємо інтерфейс інтернет-магазину.....	60
7.7.1. Короткий список основних сторінок сайту:	60
7.7.2. Реалізація каталогу товарів	61
7.7.3. Авторизація відвідувачів інтернет-магазину.....	61
7.7.4. Оформлення замовлення в інтернет-магазині	62
7.7.5. Розробка інтернет-додатків	63
<u>Література.....</u>	<u>65</u>

Тема 1. Вступ до Інтернет програмування

1.1. Інтернет-додатки - основні відомості

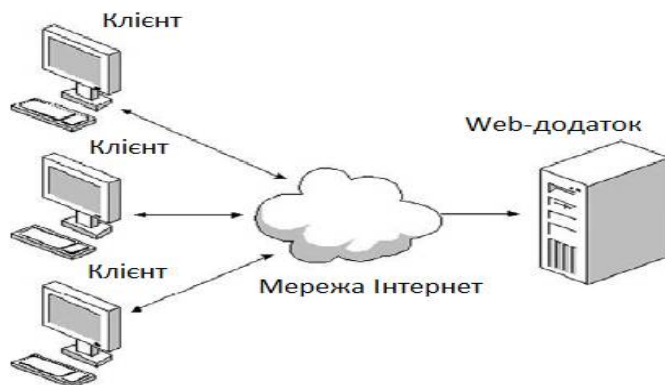
Інтернет-додатком можна вважати будь-який програмний продукт, якій так чи інакше, використовує Інтернет. Відповідно до такого визначення, додатки, користувачі яких реєструються або одержують доступ до довідкової системи через Інтернет, теж у якійсь ступені є Інтернет-додатками. Давайте спробуємо обмежити область нашого інтересу, виділивши чотири типи Інтернет-додатків;

- **Web-додатки**, які працюють на сервері, передаючи через Інтернет дані на клієнтські машини. Для їх застосування потрібні Web-браузери, такі, як **Microsoft Internet Explorer (IE)** і **Mozilla FireFox (FF)**;
- **Web-сервіси**, які дозволяють додаткам обробляти їхні дані на сервері. При цьому передача даних на сервер і повернення результатів здійснюється через Інтернет;
- додатки з підтримкою Інтернету - автономні програми з убудованими механізмами, що дозволяють їхнім користувачам реєструватися, одержувати відновлення, а також надають доступ до довідкової системи і інших допоміжних служб через Інтернет;
- **однорангові додатки** - автономні програми, що використовують Інтернет для взаємодії з іншими програмними продуктами цього ж типу.

1.2. Принцип роботи Web-додатків

Web-додатки використовують архітектуру «**клієнт - сервер**». Web-додаток перебуває на сервері і обробляє запити, які передають йому через Інтернет численні клієнти.

На стороні клієнта Web-додаток працює в браузері, наприклад в **IE** або **FF**. Користувальницький інтерфейс Web-додатка передається на клієнтську машину у вигляді сторінок мовою **HTML** (Hypertext Markup Language), де браузер інтерпретує й відображає їх.



На стороні сервера Web-додаток працює під керуванням **IIS (Internet Information Services)**. **IIS** управляє роботою додатка, передає йому клієнтські запити й повертає клієнтам результати виконання їхніх запитів. Запити і результати їх виконання передаються через Інтернет протоколом **HTTP (Hypertext Transport Protocol)**.

***Протокол** - це набір правил, що регламентують взаємодію двох і більше сутностей, що реалізується через*

середовище, таке, як Інтернет.

Web-додаток komponує відгук із серверних ресурсів, до яких відноситься:

- код, що працює на сервері (те, що традиційно вважається «додатком» в Windows-програмуванні);
- Web-форми, HTML-сторінки, графічні файли й інший уміст, що становить інформаційне наповнення додатків.

Web-додатки багато в чому нагадують традиційні Web-сайти, але на відміну від них відображають користувачеві динамічний уміст, який генерується кодом, додатка, а не статичні сторінки, що зберігаються на сервері в готовому вигляді.

Частина Web-додатка, що виконується, здатна робити багато чого з того, чого не можуть статичні Web-сайти, а саме:

- приймати дані від користувача й зберігати їх на сервері;
- виконувати для користувача різні дії: розміщати замовлення, робити складні обчислення і витягати інформацію з баз даних (БД);

- ідентифікувати користувача і відображати інтерфейс, побудований відповідно до його профілю;
- відображати постійно змінюваний уміст, наприклад інвентарні списки, оброблювані замовлення і відомості про товари, що відвантажують.

Цей перелік далеко не повний. У принципі, Web-додатки здатні вирішити будь-які уявлені завдання, доступні і клієнт-серверним додаткам. Особливість Web - додатків у тім, що взаємодія між клієнтом і сервером здійснюється через Інтернет.

1.3.Можливості ASP. NET

ASP.NET - це платформа для створення Web-додатків і Web-сервісів, що працюють під керуванням **IIS**. Звичайно, є і інші технології, що дозволяють створювати Web-додатки, наприклад CGI. ASP.NET відрізняється від них високим ступенем інтеграції із серверними продуктами, а також з інструментами Microsoft для розробки, доступу до даних і забезпечення безпеки.

Можливості ASP.NET зробили розробку Web-додатків набагато більше впорядкованою, подібно тому як Microsoft Office дозволив навести порядок у світі додатків для настільних систем. ASP.NET є частиною інфраструктури .NET Framework і складається з наступних компонентів:

- інструментів **Visual Studio .NET** для Web-розробок - графічних засобів розробки Web-сторінок, шаблонів Web-додатків і інструментів для керування проектами і розгортання Web-додатків;
- просторів імен **System.Web**, які є частиною .NET Framework і включають класи для роботи з елементами, специфічними для Web, такими, як HTTP-запити й відгуки, браузерери й електронна пошта;
- серверних елементів керування і HTML-елементів керувань-компонентів користувальницького інтерфейсу, застосовуваних для прийому даних від користувачів і відображення їм відгуку додатка,

Крім того, ASP.NET застосовує ряд більш універсальних компонентів і інструментальних засобів з ОС Windows. Хоча вони не вважаються частиною ASP.NET, вони мають ключове значення для програмування з використанням ASP.NET:

- **Internet Information Services (IIS)** - як сказано вище, IIS керує Web-додатками на серверах Windows;
- мови програмування **Visual Basic .NET, Visual C# і JScript**. **Visual Studio .NET** підтримує всі ці мови, тому Web-додатки можна писати на кожній з них;
- **.NET Framework** - повний набір класів, необхідний для Windows-програмування. Крім класів ASP.NET, у нього входять класи для рішення інших завдань програмування: файлового доступу, перетворення типів, маніпулювання масивами і рядками та інші;
- інструменти і класи **ADO.NET** для роботи із базами даних (БД), що надають доступ до БД **SQL Server** і ODBC. Механізми доступу до даних часто є ключовими компонентами Web-додатків;

Хоча ASP.NET є найбільш повною платформою для розробки Web-додатків, важливо пам'ятати, що вона функціонує винятково на серверах Windows, тому що вимагає наявності IIS. Для створення Web-додатків, що не вимагають IIS (а що використовують, скажемо, Web-сервер **Apache**) і працюючих на серверах під керуванням ОС, відмінної від Windows (наприклад, **Linux**), застосовуються інші технології (як правило, CGI).

1.4.Переваги ASP.NET

ASP.NET володіє цілим рядом переваг у порівнянні з іншими платформами для створення Web-додатків. Можливо, найважливіший з них - інтеграція із серверами Windows і інструментами програмування. Застосування ASP.NET спрощує розробку, налагодження і розгортання Web-додатків, оскільки всі ці завдання вдається вирішувати в одному і тім же середовищі розробки - **Visual Studio .NET**.

Основні переваги, які дає розроблювачам Web-додатків використання ASP.NET:

- компіляція частин Web-додатка, які виконуються, завдяки якій вони працюють швидше, ніж інтерпретуємі сценарії;
- відновлення розгорнутих Web-додатків «з льоту» без перезавпуску сервера;
- доступ до інфраструктури .NET Framework, що доповнює можливості API Windows;
- використання відомої мови програмування Visual Basic, що тепер повністю підтримує **об'єктно-орієнтоване програмування (ООП)**, і нової мови Visual C# - більше модифікованої версії C, що підтримує ООП і контроль типів;
- автоматичне керування станом елементів керування Web-сторінок [називаних *серверними елементами керування* (server controls)]. Завдяки цій особливості їхнє поводження мало відрізняється від звичайних елементів керування Windows;
- можливість створення нових, нестандартних серверних елементів керування на основі існуючих;
- забезпечення безпеки Web-додатків за допомогою убудованих захисних механізмів сервера Windows і інших методів аутентифікації й авторизації;
- інтеграція з **ADO.NET**, що дозволяє одержувати доступ до БД і застосовувати інструменти для конструювання БД із Visual Studio .NET;
- повна підтримка мови **Extensible Markup Language (XML)**, **каскадних таблиць стилів (CSS)** і інших нових стандартів Web;
- убудовані механізми кешування часто запитуваних Web-сторінок на сервері, локалізації вмісту (пристосування його для використання різних мов і регіональних стандартів) і визначення можливостей браузера.

1.5. Структура Web-додатків

Будь-який Web-додаток складається із трьох частин: інформаційного наповнення, коду, що реалізує логіку додатка, і відомостей про конфігурацію. Коротко ці частини із вказівкою їхнього місця в структурі Web-додатків ASP.NET описані в таблиці 1-1.

Таблиця 1-1. Компоненти Web-додатка ASP.NET

Компонент	Типи файлів	Опис
Інформаційне наповнення	Web-форми, HTML, графіка, аудіо, відео й інші дані	Ці файли визначають вигляд Web-додатка. Містять як статичні (текст або зображення), так і динамічні елементи, які компонуються кодом додатка «нальоту» (наприклад, запити до БД)
Код	Файли, які виконуються, і сценарії	Реалізують логіку, що визначає реакцію додатка на запити користувача. Код Web-додатків ASP.NET перебуває в бібліотеці, що підключає динамічно (DLL), яка працює на сервері, і (у деяких випадках) сценаріях, що виконуються на клієнтських машинах
Відомості про конфігурації	Файл Web.config , таблиці стилів параметри IIS	Конфігураційні файли і параметри визначають спосіб виконання додатка на сервері, коло осіб, що мають доступ до додатків, способи обробки помилок і ряд інших аспектів

Web-форма (Web form) — це ключовий елемент Web-додатка. Вона рівною мірою схожа на HTML-сторінку і на Windows-форму: Web-форма виглядає і працює як HTML-сторінка, але, подібно Windows-формі, містить елементи керування, що реагують на події і виконують код.

У готовому Web-додатку код Web-форми, який виконується, зберігається в бібліотеці (.dll-файл), що працює під керуванням IIS на сервері. Інформаційне наповнення Web-форми перебуває в каталозі вмісту Web-сервера.

Коли користувач відкриває в браузері сторінку Web Forms, відбувається наступне:

1. IIS запускає частину Web-додатка, яка виконується, якщо це ще не зроблено.
2. Частина додатка, яка виконується, компонує відгук для даного користувача із вмісту запитаної їм сторінки Web Forms і вмісту, динамічно сформованого кодом.
3. IIS повертає користувачеві результат виконання його запиту у вигляді HTML.

Одержавши запитану Web-форму, користувач може вводити в неї дані, установлювати потрібні перемикачі, клацати кнопки, тобто працювати з елементами керування форми. Маніпуляції над деякими елементами керування (наприклад, натискання на кнопки) ініціюють повернення форми на сервер для обробки подій, і все повторюється знову.

1.6. Компоненти Web-форм

На Web-формах можуть розміщуватися різні компоненти (таблиця 1-2).

Таблиця 1-2. Компоненти, розташовувані на Web-формах

Компонент	Приклади	Опис
Серверні елементи керування	<i>TextBox, Label, Button, ListBox, DropDownList, DataGrid</i>	Ці елементи керування реагують на події, що відбуваються у відповідь на дії користувача — виконання відповідних процедур на сервері. Мають убудовані механізми для збереження уведеної користувачем інформації при відправленні сторінки на сервер. Використовуються для конструювання користувацького інтерфейсу Web-форми
HTML-елементи керування	<i>Text area, Table, Image, Submit Button, Reset Button</i>	HTML-версії стандартних графічних елементів керування. Зручні у випадках, коли потрібна лише частина багатої функціональності серверних елементів керування
Елементи керування, пов'язані з даними	<i>SqlConnection, SqlCommand, OleDbConnection, OleDbCommand, DataSet</i>	Дозволяють підключатися до БД SQL і OLE, а також до файлів даних XML, виконувати над ними різні команди і витягати дані із цих джерел
Системні компоненти	<i>FileSystemWatcher, EventLog, MessageQueue</i>	Ці елементи керування надають доступ до системних подій сервера

Серверні елементи керування і HTML-елементи необхідні для створення користувацького інтерфейсу Web-форми. Елементи керування, пов'язані з даними, як і системні

компоненти, видно на поверхні форми тільки під час розробки, що дозволяє визначати їхні властивості і створювати оброблювачі для їхніх подій. У цих елементів керування немає графічного відображення часу виконання.

1.7. Мови програмування

Платформа ASP.NET (як і .NET Framework у цілому) **не залежить від мови програмування**. Це означає, що можна писати код на будь-якій мові, для якого існує CLS-сумісний компілятор. Microsoft не тільки створила власні мови програмування, але й розробила (у співробітництві з іншими виробниками) підтримку .NET для таких мов, як **Perl, Pascal, Eiffel, Cobol, Python, Smalltalk** і інші.

Ці мови функціонально еквівалентні, тобто надають рівні можливості для створення Web-додатків, відмінності між ними стосуються лише синтаксису й стилю написання коду. Більшість програмістів, швидше за все, віддадуть перевагу мові, з яким вони вже знайомі. Ті, хто зараз пише на Visual Basic, будуть створювати Web-додатки за допомогою Visual Basic .NET, а програмуючим на C і C++ підійде Visual C#.

Якщо ви недавно займаєтеся програмуванням або хочете збагатити свої знання, освоївши нову мову, то можете вивчати Visual Basic NET і Visual C# одночасно. У випадку розробки Web-додатків це ще простіше, тому що більшість завдань вирішується в них через класи .NET Framework. Це означає, що написаний на Visual Basic .NET і Visual C# код Web-додатків найчастіше виглядає майже ідентично.

1.8. Створення проекту Web-додатка

Web-форми — це об'єкти Web-додатка, що визначають його користувацький інтерфейс. Від розміщених на Web-формі тексту й елементів керування залежить, що бачить користувач, який запустив створений вами додаток, - у цьому Web-форми подібні з формами Windows-додатків.

Однак, на відміну від Windows-додатків, Web-додатки працюють на сервері, обмінюючись даними із клієнтами через Інтернет. Тому в Visual Studio .NET створення проекту Web-додатка і керування його файлами істотно відрізняються від аналогічних процедур для Windows-додатків. Ці відмінності розглядаються в наступних розділах.

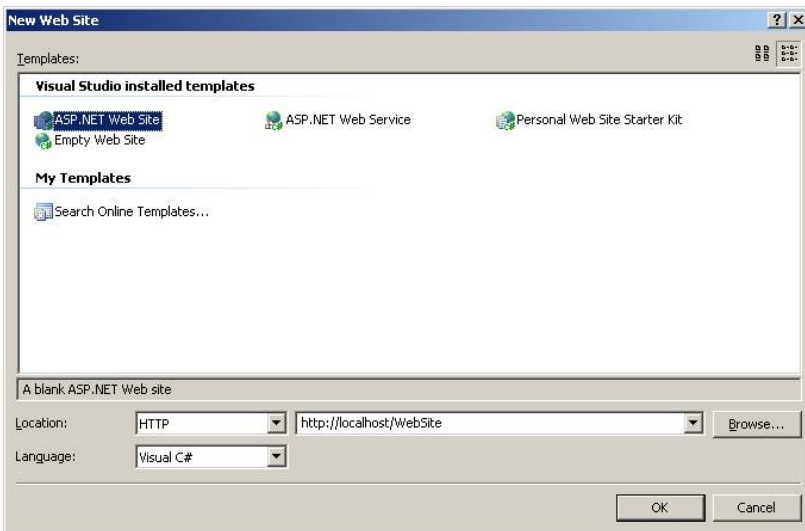
1.8.1. Створення нового Web-додатка

Щоб створити в Visual Studio .NET новий додаток Web Forms, насамперед потрібно відкрити новий проект. Для кожного типу додатків в Visual Studio .NET є відповідний шаблон. Шаблон Web-додатків називається «**ASP.NET Web Application**». При створенні нового проекту на основі цього шаблону Visual Studio .NET генерує файл проекту, порожню Web-форму й ряд допоміжних файлів, необхідні додатку.

Щоб створити проект Web-додатка в Visual Studio .NET, виконайте наступні дії:

1. На початковій сторінці Visual Studio .NET клацніть посилання New Project. У результаті відкриється діалогове вікно New Project. Visual Studio .NET розмішає Web-додаток у віртуальному каталозі **localhost**.
2. У діалоговому вікні New Project виберіть шаблон **ASP.NET Web Application**, уведіть у поле Location ім'я нового проекту й клацніть OK.

При створенні проекту Web-додатка Visual Studio .NET створює новий каталог і генерує в ньому файли проекту. Visual Studio .NET називає новий каталог по імені проекту і розмішає його в кореневому каталозі Web-сайту за замовчуванням, що обслуговується IIS на даному комп'ютері. У поле Location діалогового вікна New Project зазначена адреса каталогу проекту у вигляді ***http://localhost/ім'я_проекту***.



1.8.2. Керування проектом за допомогою IIS

Важливо розуміти, що Web-додаток може існувати тільки у *віртуальному каталозі* (virtual folder), опублікованому IIS. Віртуальний каталог - це поділюваний ресурс, який ідентифікується псевдонімом, що представляє фізичний каталог, розташований на сервері. Якщо в діалоговому вікні New Project спробувати вибрати для розміщення Web-дodatка звичайний каталог (наприклад, c:\MyFiles), кнопка OK буде недоступна, оскільки Web-додаток не можна створювати у звичайних каталогах.

Віртуальний каталог `//localhost` є кореневим Web-каталогом комп'ютера, його фізичне розміщення визначає IIS. За замовчуванням це каталог `\inetpub\wwwroot` у завантажувальному розділі жорсткого диска. Але якщо зберігати всі проекти в кореновому Web-каталозі, як задано за замовчуванням, він швидко заповниться хаотичною сумішшю проектів-прикладів, файлів робочих додатків і інших файлів. Щоб уникнути плутанини краще розміщувати приклади, додатки, які тестуються і робочий код в окремих каталогах. Вкладені каталоги, створені у кореновому Web-каталозі необхідно опублікувати в Web *до* створення в них нових проектів. Оскільки Visual Studio .NET автоматично створює нові проекти в нових каталогах, вам досить створити систему каталогів, що впорядковує проекти по типах.

1.8.3. Створення віртуальних каталогів для Web-додатків

IIS дозволяє створювати нові віртуальні каталоги і управляти розміщеними на комп'ютері Web-сайтами. Основні етапи створення віртуального каталогу, призначеного для використання в Visual Studio .NET такі:

1. Створення віртуального каталогу. Віртуальні каталоги визначають фізичне розміщення проектів Web-додатків і служать для їхнього впорядкування.
2. Додавання до віртуального каталогу **FrontPage Server Extensions** для створення каталогів subweb. **Subweb** - це просто віртуальний каталог, що містить Web-сайт. Додавши до віртуального каталогу FrontPage Server Extensions, ви дозволите Visual Studio .NET створювати і обслуговувати Web-додатки в цьому каталозі.

Щоб створити в IIS новий віртуальний каталог, виконаєте наступні дії

1. В IIS клацніть значок **Default Web Site** правою кнопкою миші й виберіть із контекстного меню команду **New\Virtual Directory**.
2. IIS запустить майстер Virtual Directory Creation Wizard, що буде направляти ваші дії під час створення віртуального каталогу. На заголовній сторінці майстра клацніть кнопку Next, у результаті відкриється сторінка **Virtual Directory Alias**.

3. Уведіть *псевдонім* (alias) для каталогу. Псевдонім знадобиться для звертання до ресурсів, розташованих у цьому каталозі, а в Visual Studio .NET - для вказівки розміщення проекту. Клацніть кнопку Next, щоб відкрити сторінку **Web Site Content Directory**.
4. Уведіть шлях до фізичного каталогу, з яким варто зв'язати новий віртуальний каталог. Цей фізичний каталог стане базовим, у ньому будуть розміщуватися папки ваших проектів. Клацніть Next, щоб відкрити сторінку **Access Permissions**.
5. Залишіть без змін дозволи, задані за замовчуванням. Клацніть кнопку Next, потім Finish, щоб створити віртуальний каталог і завершити роботу майстра.

1.8.4. Створення каталогу subweb

Щоб додати до віртуального каталогу FrontPage Server Extensions, виконайте наступні дії.

1. В IIS клацніть правою кнопкою значок Default Web Site і виберіть із контекстного меню пункт New Server Extensions Web.
2. IIS запустить майстер New Subweb Wizard, що допоможе вам додати до віртуального каталогу FrontPage Server Extensions. Клацніть кнопку Next на заголовній сторінці майстра, щоб перейти до сторінки Subweb Name.
3. У поле Directory name уведіть ім'я віртуального каталогу (воно відповідає псевдоніму, зазначеному вами при виконанні дії 3 попередні процедури). У поле Title можна ввести опис каталогу або залишити його незаповненим. Клацніть Next - майстер відкриє сторінку Access Control.
4. Прийміть параметри керування доступом, задані за замовчуванням, клацнувши кнопку Next. Потім клацніть кнопку Finish, щоб створити вкладений віртуальний каталог.

1.8.5. Створення нового проекту у віртуальному каталозі

Після додавання FrontPage Server Extensions віртуальний каталог готовий до створення в ньому нових проектів за допомогою Visual Studio .NET. Щоб створити проект Web-додатка в новому віртуальному каталозі, виберіть в Visual Studio .NET команду New Project з меню File і вкажіть у діалоговому вікні New Project ім'я віртуального каталогу.

У поле Location діалогового вікна New Project необхідно вказати шлях до проекту в наступному виді: `http://ім'я_сервера/ім'я_віртуального_каталогу`. Ім'я сервера localhost свідчить про те, що Web-сервер працює на тій же машині, де ведеться розробка Web-додатка. *Віртуальний_каталог* — це псевдонім, привласнений до цього віртуальному каталогу.

1.8.6. Розміщення елементів керування й написання коду

Після створення Web-додатка Visual Studio .NET відображає в центральному вікні нову Web-форму. Елементи керування можна перетаскувати з панелі Toolbox на поверхню такої форми, як при створенні звичайної Windows-форми.

Щоб додати до Web-форми код для обробки подій, двічі клацніть її - Visual Studio .NET відкриє вікно **Code**. Вставте потрібний код у процедуру оброблювача події цього елемента. Нижче наведено приклад, який покаже в Internet Explorer сторінку із кнопкою, клацнувши яку, ви побачите напис «Hello, Web!»

```
private void Button1_Click(object sender, System.EventArgs e)
{
    Response.Write("Hello, Web!<br>");
}
```

Запустіть додаток, натиснувши клавішу F5. Visual Studio .NET скомпонує додаток, запустить браузер і відкриє в ньому створену сторінку. Натиснувши кнопку **Button1**, ви змусите браузер відобразити вітання «Hello, Web!» (рис. 1-10). Закривши браузер, ви завершите роботу Web-додатка.

Цей приклад показує, наскільки схожі додатки Web Forms і Windows Forms. Проте за рядом параметрів додатки Web Forms істотно відрізняються:

- елементами керування. На Web-формах не можна використати стандартні елементи керування Windows, замість них застосовуються спеціальні елементи керування для Web-форм: *серверні елементи керування* (server controls), *HTML-елементи керування* (HTML controls), а також *нестандартні елементи керування* (custom controls);
- користувальницьким інтерфейсом. Вид Web-форми визначається її браузером, що відображає. *Web-додаток* можна відкрити в Internet Explorer, Mozilla FireFox або в будь-якому іншому HTML-сумісному браузері. Різні браузери (і навіть різні версії одного браузера) відрізняються наборами підтримуваних можливостей мови HTML, що впливає як на зовнішній вигляд, так і на поведінку Web-форм. Але не варто особливо турбуватися через це, тому що серверні елементи керування коректно справляються з подібними розходженнями;
- часом життя. Windows-форми існують стільки, скільки потрібно програмі, а потім знищуються. У Web-форм схожий життєвий цикл, але ця подібність чисто зовнішня. У дійсності вони створюються, передаються браузеру і відразу після цього знищуються. Отже, всі оголошені в Web-формі змінні і об'єкти після відображення цієї форми відразу знищуються! Щоб зробити щось корисне в такій ситуації, необхідно зберігати інформацію в спеціальних об'єктах стану, підтримуваних ASP.NET;
- способом виконання. Частина Web-додатка, що виконується, «живе» на Web-сервері. Із цього погляду Web-додатки є різновидом клієнт-серверних додатків, у яких єдиним клієнтським компонентом є браузер, а все інше (прикладна логіка і користувальницький інтерфейс) працює на сервері. Вся взаємодія між клієнтом і сервером відбувається з використанням HTML. Це означає, що навіть складні Web-додатки практично не загрожують безпеки клієнтів, тому брандмауери не стають перешкодою для роботи Web-додатків.

Тема 2. Аутентифікація і авторизація користувачів

Аутентифікацією (authentication) називається процес ідентифікації користувачів.

Авторизація (authorization) — це процес надання доступу користувачам на основі їхніх ідентифікаційних даних. Аутентифікація поряд з авторизацією являють собою засоби захисту.

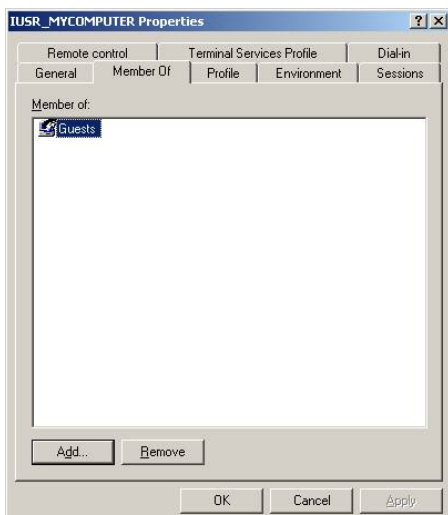
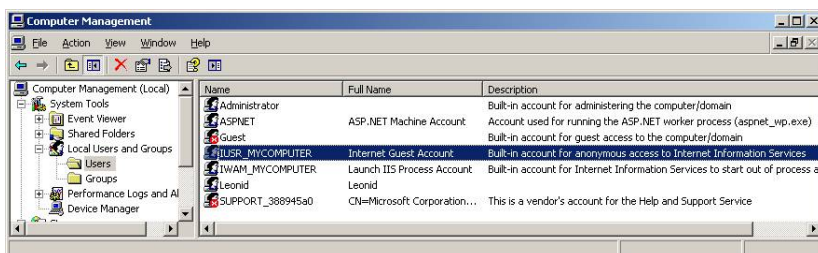
2.1. Анонімний доступ

Більшість відкритих Web-сайтів працюють у режимі анонімного доступу. Ці сайти містять загальнодоступну інформацію, яку можуть переглядати всі бажаючі, тому вони не проводять аутентифікацію користувачів. Web-додатки ASP.NET надають анонімний доступ до серверних ресурсів за допомогою уособлення. У даному контексті **уособлення (impersonation)** — це призначення облікового запису невідомому користувачеві.

За замовчуванням обліковий запис для анонімного доступу має ім'я у вигляді **IUSR_ім'я_комп'ютера** і використовується для керування доступом анонімних користувачів до серверних ресурсів. Щоб переглянути або змінити привілеї облікового запису для анонімного доступу за допомогою оснащення **Computer Management** (керування комп'ютером), виконайте наступні дії.

1. Увійдіть на сервер як адміністратор.
2. Виберіть із меню **Start (Пуск)** пункт **Administrative Tools\Computer Management** (Адміністрування\Керування комп'ютером), щоб запустити консоль Computer Management.
3. Виберіть у списку ліворуч елемент **Local Users And Groups** (Локальні користувачі й групи), потім папку **Users**, щоб відкрити список авторизованих користувачів для цього комп'ютера.
4. У списку праворуч двічі клацніть лівою кнопкою анонімний обліковий запис із ім'ям у формі **IUSR_ім'я_комп'ютера** - оснащення **Computer Management** відкриє вікно властивостей облікового запису.

Для перегляду груп, до яких належить цей обліковий запис, перейдіть на вкладку **Member Of** (Членство в групах). За замовчуванням анонімні користувачі є членами групи **Guests**, що володіють обмеженими привілеями.



Права доступу для анонімних користувачів можна обмежувати, установлюючи дозволи Windows на доступ до файлів. Для більш надійного захисту сервера варто використовувати файлову систему **NT (NTFS)**, тому що більш ранні файлові системи, такі, як FAT і FAT32, не забезпечують безпеку на файловому рівні.

2.2. Доступ аутентифікованих користувачів

Як говорилося раніше, анонімний доступ підходить для роботи з відкритою інформацією, але якщо додаток містить конфіденційні відомості або працює з ними, наприклад, розміщає замовлення, всі користувачі повинні проходити аутентифікацію і авторизацію.

Існує три основних способи аутентифікації і авторизації користувачів Web-додатків ASP.NET:

- **аутентифікація Windows** - застосовує для ідентифікації і авторизації список користувачів сервера. Доступ до серверних ресурсів надається або забороняється залежно від привілеїв облікового запису користувача. Працює аналогічно звичайним механізмам мережної безпеки Windows;
- **аутентифікація Forms** - направляє користувача на вхідну Web-форму, де він уводить своє ім'я і пароль, після чого форма ідентифікує його за допомогою списку користувачів або бази даних, підтримуваної додатком;
- **аутентифікація Passport** - направляє користувачів на сайт Microsoft, де вони за допомогою єдиного імені і пароля одержують доступ до декількох Web-сайтів. Зареєстрований користувач повинен увести ім'я і пароль служби Passport, які додаток потім звіряє зі списком користувачів служби Passport.

У кожного способу включаючи анонімний доступ, є свої переваги і недоліки. У таблиці 2.1 перераховані методи аутентифікації, оптимальні для різних типів Web-додатків.

Таблиця 2.1 Методи, що рекомендують, аутентифікації користувачів Web-додатків

Тип додатка	Метод, що рекомендує, аутентифікації	Описання
Відкриті Web-додатки, які працюють в Інтернеті	Анонімний доступ	Це звичайний метод доступу до Web-сайтів, не потребуючий реєстрації. Безпека закритих ресурсів забезпечується за допомогою файлових дозволів NTFS
Web-додатки, що працюють в інтрамережі	Аутентифікація засобами Windows	Аутентифікація користувачів мережі виконується контролером домена. Користувачі мережі одержують доступ до ресурсів Web-додатка відповідно до своїх привілеїв на сервері
Закриті корпоративні Web-додатки	Аутентифікація засобами Windows	Корпоративні користувачі одержують доступ до Web-додатків за допомогою облікових даних користувача корпоративної мережі. Адміністрування користувальницьких облікових записів виконується за допомогою інструментів мережної безпеки Windows
Комерційні Web-додатки	Аутентифікація за допомогою вхідної форми	У додатках, що збирають відомості для виставлення рахунків і відправлення товарів, варто застосовувати аутентифікацію за допомогою форм, які заодно дозволяють одержати і зберегти відомості про клієнта
Єдиний доступ до декількох комерційних Web-додатків	Аутентифікація за допомогою служби Passport	Для аутентифікації за допомогою служби Passport користувачеві досить один раз зареєструватися в центрі видачі паспортів. Після цього ідентифікаційні дані користувача стають доступними будь-якому додатку, що використовує Passport SDK. Відомості про клієнта

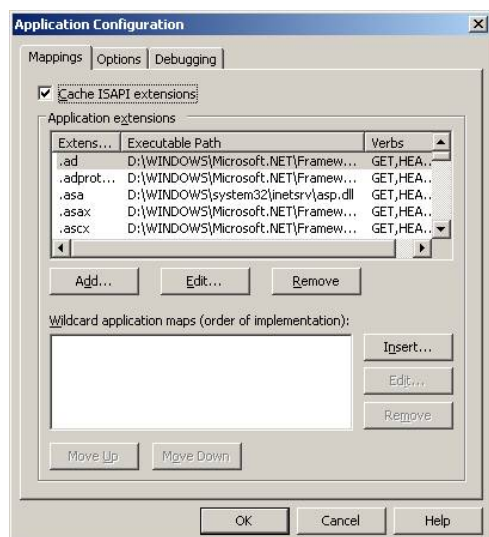
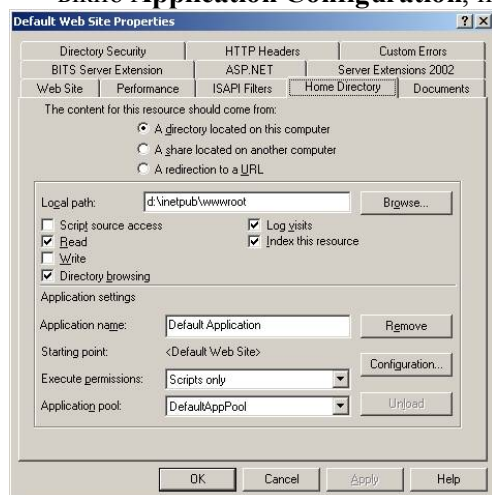
	зберігаються в профілі служби Passport, а не в локальній базі даних
--	---

2.3. Аутентифікація запитів HTML-сторінок

Описані методи аутентифікації ASP.NET застосовуються для файлів, що є частиною Web-додатка. До них відносяться Web-форми (.aspx-файли), модулі (.asax-файли) і інші ресурси, які обробляються виконуючими файлами Web-додатка. HTML-сторінки (.htm- або .html-файли) не включаються в число цих файлів автоматично. Такі сторінки обробляються IIS, а не ASP.NET. Якщо потрібно аутентифікувати користувачів, що звертаються до HTML-сторінок Web-додатка, засобами Windows або за допомогою форм або служби Passport, необхідно зареєструвати ці файли для обробки виконуючими компонентами ASP.NET.

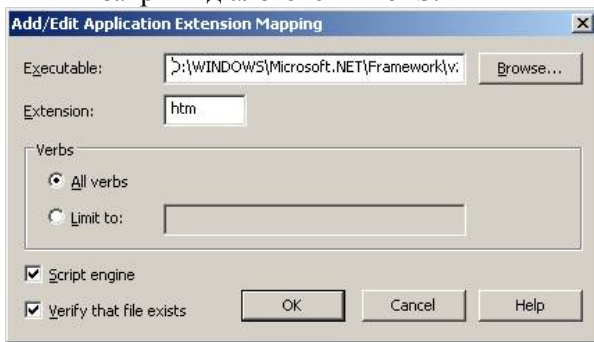
Щоб зареєструвати .htm- і .html-файли для обробки виконуючими файлами ASP.NET, відкрийте оснащення IIS і виконаєте наступні дії:

1. В оснащенні **IIS** виберіть папку з Web-додатком і виберіть із меню **Action** команду **Properties** - IIS відкриє діалогове вікно **Properties**.
2. Перейдіть на вкладку **Home Directory** і клацніть кнопку **Configuration**, IIS відкриє діалогове вікно **Application Configuration**, показане на рис. 2.2.



3. Клацніть **Add** - IIS відкриє діалогове вікно **Add/Edit Application Extension Mapping**, показане на рис. 2.3.
4. Клацніть кнопку **Browse** і виберіть файл **aspnet_isapi.dll**. Цей файл перебуває в каталозі Windows Microsoft .Net Framework, шлях до нього виглядає приблизно так: **C:\Windows\Microsoft.NET\Framework\versionnumber\aspnet_isapi.dll**.

5. Уведіть у текстове поле File Extension **.htm** і клацніть ОК.
6. Повторіть пункти 3-5, щоб зареєструвати розширення **.html**. Закінчивши, клацніть ОК, щоб закрити діалогове вікно IS.



2.4. Аутентифікація Windows

Аутентифікація Windows використовує убудовані механізми системи безпеки операційних систем Windows NT, Windows XP, Windows Vista. Перевага аутентифікації засобами Windows полягає в можливості використання Web-додатком схеми безпеки, прийнятої в корпоративній мережі, тобто застосування тих самих імен, паролів і дозволів для одержання доступу до ресурсів корпоративної мережі й ресурсам Web-додатків.

2.4.1. Включення аутентифікації Windows

У новому Web-додатку аутентифікація Windows використовується за замовчуванням. Щоб побачити, як працює аутентифікація Windows, виконайте наступні дії.

1. Створіть новий проект Web-додатка. Додайте до проекту файл **Web.config** і додайте в нього наступний елемент:

```
<authorization>
  <deny users="*" />
</authorization>
```

2. Додайте до коду початкової Web-форми проекту наступне HTML-визначення таблиці:

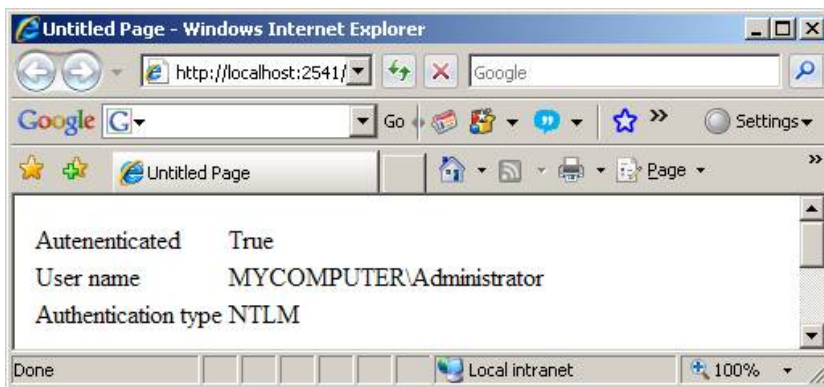
```
<TR><TD>Autnenticated</TD>
  <TD><span runat="server" id="spnAuthenticated"></span></TD></TR>
<TR><TD>User name</TD>
  <TD><span runat="server" id="spnUserName"></span></TD> </TR>
<TR><TD>Autlientication type</TD>
  <TD><span runat="server" id="spnAuthenticationType">
</span></TD></TR>
```

3. Перемкніть вікно форми в режим **Design** і додайте до об'єкта коду початкової Web-форми наступні рядки;

```
private void Page_Load(object sender, System.EventArgs e)
{
    spnAuthenticated.InnerText = User.Identity.IsAuthenticated.ToString();
    spnUserName.InnerText = User.Identity.Name;
    spnAuthenticationType.InnerText = User.Identity.AuthenticationType;
}
```

4. Запустіть проект.

Якщо проект запущений локально, ASP.NET аутентифікує за допомогою імені, зазначеного при вході в Windows (рис. 2.5):



Аутентифікація засобами **Windows**, яка виконана локально

При віддаленому виконанні проекту (наприклад, через Інтернет):

1. ASP.NET відкриває в браузері діалогове вікно для введення імені користувача й пароля.
2. Якщо ім'я і пароль, введені користувачем, збігаються із заданими в обліковому записі, який авторизований у даному домені, ASP.NET ідентифікує користувача і надає йому доступ до додатка.
3. Додаток виконується з дозволами, призначеними обліковому запису цього користувача.
4. Для авторизованого користувача ASP.NET створює сертифікат авторизації у вигляді файлу **cookie**, що зберігається протягом сеансу цього користувача.
5. Сеанс користувача завершується при закритті браузера або при настанні тайм-ауту сеансу. Найкраще аутентифікація засобами **Windows** працює в мережі, організованій у вигляді домену. У мережах, де замість доменів задіяні робочі групи, можливості застосування мережної безпеки обмежені.

Одне із **ключових переваг** аутентифікації засобами **Windows** полягає в тому, що вона звільняє користувачів, що зареєструвалися в мережі, від повторної реєстрації для доступу до Web-додатків. Додаткова перевага - надавана корпоративним користувачам **можливість застосовувати ті самі мережні імена і паролі для віддаленого доступу до корпоративного Web-сайту з домашнього комп'ютера і у відрядженнях**.

Аутентифікація засобами Windows також дозволяє організувати додаткову лінію захисту, надаючи або забороняючи доступ окремим користувачам або групам. Крім того, аутентифікація засобами Windows дублює ряд функцій IIS.

2.4.2. Призначення прав доступу окремим користувачам

Якщо додаток застосовує аутентифікацію засобами Windows, ASP.NET перевіряє список авторизації у файлі **Web.config** проекту, щоб визначити тих, кому дозволений доступ до додатка. Символи «*» і «?» мають особливе значення в списку авторизації: **зірочка** позначає всіх користувачів, а **знак питання** - всіх неавторизованих користувачів. Наприклад, якщо у файлі **Web.config** заданий наступний список авторизації, всім користувачам прийдеться пройти аутентифікацію, щоб одержати доступ до Web-додатка:

```
<authorization>
  <!-- Доступ неавторизованим користувачам заборонений -->
  <deny users="?" />
</authorization>
```

Щоб визначити обмежене коло користувачів, яким ви хочете дозволити доступ, вкажіть їхні імена (через кому) в елементі **<allow>**. Перевіряючи список авторизації, ASP.NET приймає першу знайдену відповідність. Обов'язково помістіть в список авторизації елемент **<deny>**, щоб заборонити доступ іншим користувачам:

```
<authorization>
  <!--Дозволити доступ двом користувачам. . .-->
  <allow users="Deanna Meyer, Michael Emanuel" />
  <!--...і заборонити його всім іншим. -->
```

```
<deny users="*" />
</authorization>
```

Цей список авторизації дозволяє доступ до Web-додатка двом користувачам - Deanna Meyer і Michael Emanuel (у них також повинні бути облікові записи в цьому домені) і забороняє його всім іншим.

2.4.3. Застосування авторизації, заснованої на ролях

При використанні авторизації на основі ролей можна призначати дозволи цілим групам користувачів, які виділяються на основі їхніх функцій в організації. Ролі відповідають групам Windows NT, Windows XP і Windows Vista. В Windows передбачено кілька убудованих груп, наприклад **Administrators**, **Users** і **Guests**. Для перегляду, модифікації або створення груп служить консоль **Computer Management**.

Щоб дозволити або заборонити доступ певним групам, додайте елемент **<roles>** до списку авторизації у файлі **Web.config** Web-додатка. На прикладі нижче список авторизації дозволяє доступ тільки користувачам із групи системних адміністраторів:

```
<authorization>
  <!-- Дозволити доступ тільки членам групи Administrators... -->
  <allow roles="Administrators" />
  <!-- ...і заборонити його всім іншим. -->
  <deny users="*" />
</authorization>
```

2.4.4. Витяг ідентифікаційних даних користувача

Додатки витягають ідентифікаційні дані користувачів, що пройшли аутентифікацію, за допомогою властивості **Identity** об'єкта **User**. Як показано в наступному фрагменті коду, властивість **Identity** повертає об'єкт, що містить ім'я користувача і роль, до якої він належить:

```
private void Page_Load(object sender, EventArgs e)
{
    spnAuthenticated.InnerText = User.Identity.IsAuthenticated.ToString();
    spnUserName.InnerText = User.Identity.Name;
    spnAuthenticationType.InnerText = User.Identity.AuthenticationType;
}
```

Крім того, об'єкт **User** підтримує метод **IsInRole**, що дозволяє визначити роль поточного користувача, наприклад, так:

```
if (User.IsInRole("Administrators"))
    // Виконати деякі дії.
```

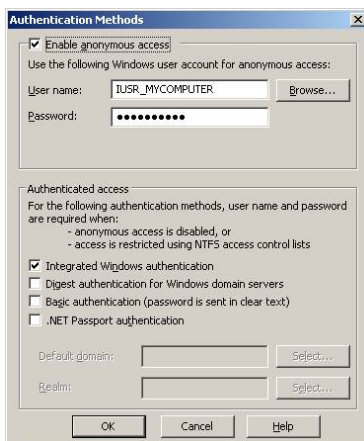
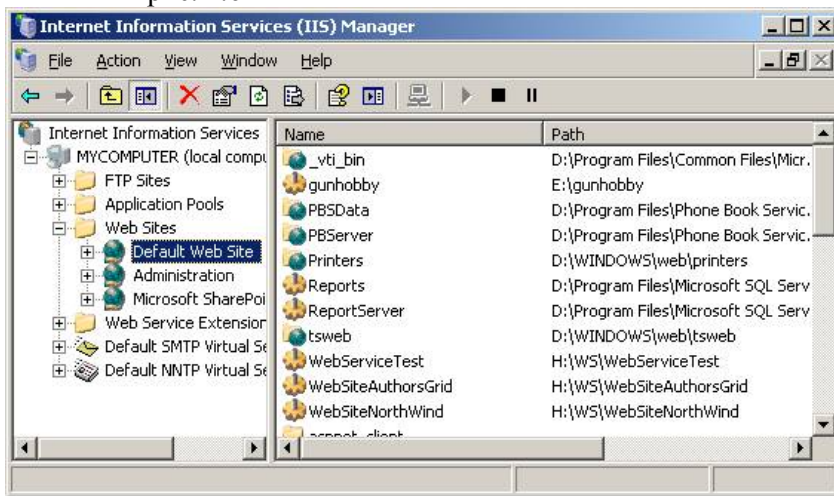
Ці методи і властивості разом з подією **AuthorizeRequest**, у модулі **Global**, дозволяють задати метод авторизації користувача. Наприклад, за допомогою події **AuthorizeRequest** звіряється ім'я користувача з ім'ям, не заданим у списку файлу **Web.config**.

2.4.5. Параметри IIS і аутентифікація засобами Windows

Параметри авторизації, що задають у файлі **Web.config**, дублюють ряд параметрів **IIS**. Якщо параметри авторизації задані як файлом **Web.config**, так і параметрами **IIS**, першими перевіряються саме вони, а лише потім - файл **Web.config**. У результаті, як правило, використовуються параметри, що задають мінімальні права доступу.

Для перегляду параметрів авторизації, заданих в **IIS**, виконайте наступні дії.

1. В оснащенні IIS клацніть правою кнопкою миші папку Web-додатку й виберіть із контекстного меню елемент **Properties** - IIS відкриє діалогове вікно властивостей папки (рис. 2...).
2. Перейдіть на вкладку **Directory Security** і в групі **Anonymous Access And Authentication Control** клацніть Edit. IIS відкриє діалогове вікно **Authentication Methods**, показане на рис. 2.6



Перша група параметрів у цьому діалоговому вікні управляє анонімним доступом користувачів, що не пройшли аутентифікацію. Зняття відповідного прапорця рівнозначно додаванню елемента `<deny users="?*>` у файл **Web.config**. Прапорці в другій секції діалогового вікна дозволяють додатку використати, крім механізмів аутентифікації Windows, аутентифікацію **Basic** або **Digest**. Ці методи менш безпечні, ніж аутентифікація за допомогою убудованих механізмів Windows, форм або служби Passport і реалізуються за участю IIS, а не ASP.NET.

Як видно з рисунку, IIS дозволяє задіяти кілька методів аутентифікації. Якщо включено кілька методів аутентифікації, метод *Authentication* об'єкта *Identity* дозволяє визначити в кодї, який із цих методів застосований для аутентифікації користувача. Це робиться в такий спосіб:

```
Response.Write(User.Identity.AuthenticationType);
```

2.5. Аутентифікація Forms

При застосуванні цього методу аутентифікації автоматично відображається особлива Web-форма, призначена для збору відомостей про користувача. Код, пов'язаний із цією формою, ідентифікує користувачів і надає їм доступ, звіряючи уведені ними дані зі списком користувачів, що зберігається у файлі **Web.config** або в спеціальній базі даних (БД).

Достоїнство аутентифікації за допомогою форм полягає в тім, що для одержання доступу до додатка користувач не обов'язково повинен бути членом домена. Крім того, цей метод аутентифікації

спрощує створення Web-додатків для електронної комерції. Ці додатки обробляють замовлення, розташовувані клієнтами, тому їм просто *необхідний* доступ до відомостей про користувача.

2.5.1. Як задіяти аутентифікацію за допомогою форм

Цей метод аутентифікації дозволяє створити власну базу даних з відомостями про користувачів і застосовувати її для перевірки ідентифікаційних даних користувачів, що відвідують ваш Web-сайт.

Щоб застосовувати для ідентифікації і авторизації користувачів аутентифікацію за допомогою форм, виконайте наступні дії.

1. У файлі **Web.config** встановіть режим аутентифікації в «Forms».
2. Створіть Web-форму для збору облікових даних.
3. Створіть файл або БД для зберігання імен і паролів користувачів.
4. Напишіть код, що додає відомості про нових користувачів у файл або БД.
5. Напишіть код, що виконує аутентифікацію користувачів із застосуванням файлу або БД із відомостями про користувачів.

При звертанні до Web-додатка, що використовує аутентифікацію Forms, ASP.NET відображає вхідну Web-форму, задану у файлі Web.config. Для користувача, що пройшов авторизацію, ASP.NET генерує сертифікат авторизації у вигляді файлу cookie і зберігає його протягом часу, заданого параметрами файлу **Web.config**.

Розходження між аутентифікацією засобами Windows і за допомогою форм полягає в тому, що в другому випадку всі завдання по аутентифікації і авторизації користувачів додаток виконує самостійно. Ви повинні створити Web-форми і написати код, які будуть приймати ім'я й пароль користувача і звіряти їх зі списком авторизованих користувачів.

Для реалізації аутентифікацій за допомогою форм застосовуються класи із простору імен **System.Web.Security**. Щоб зробити ці класи доступними коду, необхідно помістити в початок кожного модуля, що виконує аутентифікацію, оператори:

```
using System.Web.Security
```

2.5.2. Вибір методу аутентифікації

Використовуваний додатком метод аутентифікації встановлюють в елементі **<authorization>** файлу **Web.config**. Щоб настроїти додаток для використання аутентифікації за допомогою форм, внесіть у файл "Web.config" наступні зміни:

```
<authentication mode="Forms" >
  <!-- Режим аутентифікації -->
  <forms loginUrl="Default.aspx" >
    <!-- Вхідна форма -->
    <credentials passwordFormat="Clear">
      <!-- Список користувачів -->
      <user name="June" password="111111" />
      <user name="Walter" password="222222" />
    </credentials>
  </forms>
</authentication>

<authorization>
  <!-- Заборонити доступ кожному, хто не пройшов аутентифікацію -->
  <deny users="?" />
</authorization>
```

Показаний фрагмент файлу **Web.config** демонструє спрощений приклад аутентифікації за допомогою форм, у якому для більшості параметрів залишені значення за замовчуванням, а список користувачів включений у файл **Web.config**. У таблиці перераховані всі можливі атрибути для елементів, що визначають аутентифікацію за допомогою форм.

Елемент	Атрибут	Опис
<authorization>	mode	Для використання ідентифікації за допомогою форм необхідно встановити цей атрибут і Form
<forms>	name	Визначає ім'я файлу cookie, в якому зберігаються облікові записи дінні користувача
	loginURL	Використовується для посилання на Web-форму, яка відображається поки користувач не пройшов аутентифікацію. За замовчуванням задана форма Default.aspx.
	Protection	Визначає метод, який ASP .Net використовує для захисту файлу cookie з ідентифікаційними даними користувача на даному комп'ютері. За замовчуванням задане значення All , яке визначає захист шифруванням та перевірку істинності даних.
	Timeout	Встановлює час зберігання файлу cookie на користувальницькому комп'ютері у хвилинах. За замовчуванням - 30 хвилин.
	Path	Визначає шлях до файлу cookie на користувальницькому комп'ютері.
<credentials>	passwordFormat	Вказує алгоритм шифрування пароля користувача. За замовчуванням - SHA1.
Users	Name	Задає ім'я користувача
	Password	Задає пароль користувача.

Елемент **<credentials>** дозволяє зберігати у файлі **Web.config** список користувачів. Це зручно при простій аутентифікації, коли адміністратор сам додає нових користувачів і призначає їм паролі, але краще дозволити клієнтам самостійно створювати свої облікові записи і вибирати собі паролі.

Для цього варто створити файл або БД для зберігання імен користувачів до зашифрованих паролів. БД дає додаткову перевагу, оскільки в ній можна зберігати додаткові відомості про користувачів, наприклад адреси доставки і список замовлень.

2.5.3 Створення вхідної Web-форми

Для аутентифікації користувачів за допомогою форм необхідна насамперед вхідна Web-форма, що дозволить зареєстрованим користувачам увійти на сайт (а новим користувачам — зареєструватися). Ім'я цієї форми визначається елементом `<forms>` файла **Web-config**.

В якості вхідної форми годиться проста Web-форма, що містить лише пари текстових полів і кнопку. Інший варіант - включити вхідну форму в сторінку з незахищеним умістом. Наприклад, можна помістити поля для введення облікових даних на початкову сторінку додатка (рис.).



```
protected void butSignOn_Click(object sender, EventArgs e)
{
    // Звірити введені ім'я і пароль користувача
    // с такими з елемента <credentials>
    if (FormsAuthentication.Authenticate
        (txtUserName.Text, txtPassword.Text))
    {
        // Якщо вони знайдені в списку,
        // відобразити початкову сторінку додатка
        FormsAuthentication.RedirectFromLoginPage (txtUserName.Text,true);
    } else {
        // у протилежному випадку очистити поле для введення
        // пароля,
        txtPassword.Text = "";
        // Після третьої невдалої спроби
        // відобразити сторінку "Access Denied",
        if (System.Convert.ToInt32(ViewState["Tries"]) > 3)
            Response.Redirect("Denied.htm");
        else
            // у протилежному випадку збільшити число
            // використаних спроб,
            ViewState["Tries"] = System.Convert.ToInt32(ViewState["Tries"]) + 1;
    }
}
```

Варто звернути увагу на ряд моментів:

- клас **FormsAuthentication** визначений у просторі імен **System.Web.Security**, тому необхідно імпортувати цей простір імен за допомогою оператора **using** або посилатися на цей клас за допомогою повного імені;

- метод *Authenticate* класу *FormsAuthentication* звіряє ім'я і пароль користувача зі списком, заданим елементом `<credentials>` файлу **Web.config**;
- метод *RedirectFromLoginPage* класу *FormsAuthentication* відображає початкову сторінку додатка. Якщо поля для введення облікових даних розміщені на початковій сторінці додатка, після входу користувача необхідно відключити їх або вказати, що вхід виконаний успішно;
- якщо користувач увів невірне ім'я або пароль, цей код дає йому ще дві спроби, після чого відображає сторінку Access Denied. Це HTML-сторінка, а не Web-форма, оскільки без аутентифікації доступ до будь-яких Web-форм додатка заборонений. Якщо ви направляєте користувачів на цругую сторінку, переконайтеся, що вона перебуває поза областю видимості додатка.

Клас *FormsAuthentication* дозволяє користувачеві вийти з додатка, коли він завершив роботу, а також видалити файл cookie з паспортом аутентифікації з його комп'ютера. На наведеному прикладі код завершує користувацький доступ до додатка і при спробі знову одержати його просить увести ім'я і пароль.

```
using System.Web.Security;
private void butSignOut_Click(object sender,
    System.EventArgs e)
{
    // Видалити файл cookie з паспортом аутентифікації
    FormsAuthentication.SignOut();
    // Повернути користувача на цю (вхідну) сторінку,
    Response.Redirect ("UserInfo.aspx");
}
```

2.5.4 Аутентифікація за допомогою бази даних

Попередні розділи демонструють аутентифікацію за допомогою списку користувачів з файлу **Web.config** — метод *Authenticate* класу *FormsAuthentication* читає цей файл за замовчуванням. Це годиться, коли імена і паролі користувачів створюються і обслуговуються системним адміністратором, але якщо користувачам дозволено самим вибирати собі імена і паролі, краще зберігати цю інформацію в іншому місці. Справа в тому, що після модифікації **Web.config** необхідно перезапустити Web-додаток, що приведе до скидання змінних стани *Application* і *Session*, які використовуються додатком. Імена і паролі користувачів можна зберігати в будь-яких файлах, але БД дає ряд істотних переваг:

- ім'я користувача можна застосовувати як первинний ключ для зберігання інших відомостей про користувача;
- БД забезпечує високу швидкість при витягу користувацьких імен і паролів;
- SQL дозволяє стандартизувати додавання, модифікацію і витяг записів.

Імена і паролі користувачів можна зберігати у файлі або в БД у зашифрованому виді, для шифрування використовується метод *HashPasswordForStoringInConfigFile* класу *FormsAuthentication*, що застосовує алгоритми SHA1 або MD5:

```
Password = FormsAuthentication.HashPasswordForStoringInConfigFile
(Password, "SHA1");
```

2.5.5. Додавання користувачів у базу даних

Щоб додати користувачів у БД, треба одержати його ім'я і пароль із відповідних текстових полів, а також створити оброблювач події, що додає користувача в БД і повідомляє про успішне або невдале додавання користувача. Наступний оброблювач події викликає допоміжну функцію *AddUser* для додавання облікових даних користувача в БД:

```
private void butNewUser_Click(object sender, System.EventArgs e)
{
```

```

if (AddUser(txtUserName.Text, txtPassword.Text))
    spnNote.InnerText = "User added.";
else
    spnNote.InnerText = "User exists. Choose a
    different user name";
}

```

Допоміжна функція *AddUser*, яка показана в наступному коді, шифрує пароль перед записом облікових даних в БД, здійснюваною командою **SQL INSERT**. Якщо ім'я користувача, що додається, вже є в БД, блок обробки виключень перехоплює помилку і повертає значення **False**, що свідчить про невдачу додавання користувача.

```

private bool AddUser(string UserName, string Password)
{
    // Оголосити змінну для визначення стану операції
    bool bSuccess = false;
    // Зашифрувати пароль.
    Password = FormsAuthentication.
        HashPasswordForStoringInConfigFile
        (Password, "SHA1");
    // Створити команду для вставки імені і пароля
    // користувача
    OleDbCommand oleConnmand = new OleDbCommand
    ("INSERT INTO Users" + " VALUES
    (" + UserName + ",
    " + Password + ")", oledbUsers);
    // Якщо такий запис вже існує, перехопити помилку
    try
    {
        // Відкрити з'єднання з БД.
        oledbUsers.Open();
        // Якщо запис додано встановити змінну стану
        // у true...
        if (oleCommand.ExecuteNonQuery() != 0)
        {
            bSuccess = true
            // ... і закрити з'єднання
            oledbUsers.Close();
        }
    }
    catch
    {
        // якщо ні, вважати операцію невдалою
        bSuccess = false;
        // Закрити з'єднання
        oledbUsers.Close();
    }
    // Повернути повідомлення про успішне або ні
    // завершення
    return bSuccess;
}

```

Допоміжна функція *CheckPassword* (див. код нижче) шифрує введений пароль, пише в БД запис з таким же ім'ям користувача і порівнює зашифрований пароль з паролем із знайденим в БД записом. Звернення до БД проводиться в блоці обробки виключень, щоб користувач не побачив повідомлення про помилки, що виникають при конфліктах із-за блокування БД.

```

private bool CheckPassword(string UserName, string Password)
{

```

```

// Оголосити змінну для визначення стану операції
bool bSuccess = false;
// Зашифрувати пароль
Password = FormsAuthentication.HashPasswordForStoringInConfigFile
(Password,"SHA1");
// Створити команду для витягу рядка із заданим
// значенням UserName
OleDbCommand oleCommand = new OleDbCommand
("SELECT * FROM Users" + " WHERE UserName =
" + txtUserName.Text + "", oledbUsers);
// Слідкувати за помилками в БД.
try
{
// Відкрити з'єднання з БД
oledbUsers.Open();
//Знайти запис
OleDbDataReader rdrUsers = oleCommand.ExecuteReader();
while (rdrUsers.Read())
{
    if (Password == rdrUsers["Password"].ToString())
        bSuccess = true;
}
// Закрити з'єднання
oledbUsers.Close();
}
catch
{
// Якщо запис знайдено, вважати операцію невдалою
bSuccess = false;
// Закрити з'єднання
oledbUsers.Close();
}
return bSuccess;
}

```

2.6. Аутентифікація Passport

При аутентифікації **Passport** ідентифікація користувачів виконується за допомогою служби єдиного входу Microsoft Passport. Мета служби Passport — дозволити користувачам Інтернету входити на різні Web-сайти, що вимагають аутентифікації, за допомогою єдиних посвідчень. При цьому Web-додатки можуть витягувати відомості про користувача з профілю, Microsoft, що зберігається.

Перевага аутентифікації Passport в тому, що вона звільняє клієнта від необхідності запам'ятовувати окремі облікові дані для кожного Web-сайта і дозволяє йому централізовано зберігати свій профіль. Аутентифікація Passport дозволяє отримати доступ і до інших сервісів Microsoft, таких, як **Passport Express Purchase**.

2.6.1. Як задіювати аутентифікацію Passport

Аутентифікація Passport ідентифікує користувачів за допомогою централізованої служби аутентифікації Microsoft Passport дозволяє користувачам застосовувати єдині посвідчення для отримання доступу до декількох Web-додатків. Щоб включити аутентифікацію Passport в Web-додатку, необхідно встановити пакет **Passport SDK**, який знаходиться в розділі Downloads на сайті MSDN (див. сторінку за адресою <http://msdn.microsoft.com/downloads>).

Коли користувач звертається до додатка, що застосовує аутентифікацію Passport, ASP .NET перевіряє наявність на клієнтському комп'ютері файлу cookie служби Passport. Якщо такий cookie не знайдений, ASP.NET направляє користувача на вхідну сторінку служби Passport. Після входу користувача служба Passport аутентифікує його, записуючи на призначений для користувача комп'ютер файл cookie з ідентифікаційними даними, і направляє користувача на запитану Web-сторінку.

Процес аутентифікації Passport виконується у наступний спосіб:

1. Встановіть пакет Passport SDK. Хоча цей пакет не входить в Visual Studio, в .NET Framework є класи, що дозволяють працювати з Passport SDK після установки цього пакету.
2. Встановіть режим аутентифікації у файлі **Web.config** у **Passport** і забороніть доступ всім користувачам, що не пройшли аутентифікацію.
3. За допомогою події **passportAuthentication_OnAuthenticate** отримайте доступ до призначеного для користувача профілю служби Passport і авторизації користувача.
4. Напишіть процедуру для завершення роботи користувача, що видаляє файли cookie служби Passport з призначеного для користувача комп'ютера.

Наприклад, наступні параметри файлу Web.config включають аутентифікацію Passport і примушують всіх користувачів проходити її:

```
<authentication mode="Passport" />
<authorization>
<deny users="*" />
<!-- Заборонити доступ користувачам, що не пройшли аутентифікацію -->
</authorization>
```

Якщо запустити додаток з такими параметрами Web.config на локальному комп'ютері, він автоматично направить користувача на вхідну сторінку служби Passport.

Після входу користувача Passport записує на його комп'ютер файл cookie, підтверджуючий авторизацію, і направляє користувача на початково запитану ним сторінку. Відомості про користувача зберігаються в профілі служби Passport. Доступ до цього профілю дістають за допомогою події **PassportAuthentication_OnAuthenticate**, визначеної в модулі Global.asax, наприклад, так:

```
// Цей рядок має бути на початку модуля
using System.Web.Security;
protected void PassportAuthentication_OnAuthenticate(Object sender,
PassportAuthenticationEventArgs e)
{
// Якщо аутентифікація успішна
// отримати ідентифікаційні дані Passport для сеансу
if (e.Identity.Authenticated)
{
Response.Write("Name: " + e.Identity["FirstName"] +
" " + e.Identity["LastName"] + "<br>");
Response.Write("Address: " + e.Identity["City"] +
" " + e.Identity["PostalCode"] + "<br>");
Response.Write("Email: " + e.Identity["PreferredEmail"] +
<br>");
Response.Write("Passport ID: " + e.Identity.Name + "<br>");
}
}
```

Цей код відображує ім'я і адресу користувача, а також ідентифікаційні дані з його профілю служби Passport.

Підтвердження авторизації Passport і дані профілю зберігаються в п'яти окремих файлах cookie на комп'ютері користувача. Passport SDK вимагає видаляти ці cookie після закінчення роботи користувача з Web-додатком. Як це робиться, показано на прикладі наступного обробника події:

```
private void butSignOut_Click(object sender, System.EventArgs e)
{
// Завершити роботу, видаливши файли cookie служби
// Passport.
Response.Cookies["MSPPProf"].Expires = DateTime.Now;
Response.Cookies["MSPAuth"].Expires = DateTime.Now;
Response.Cookies["MSPSecAuth"].Expires = DateTime.Now;
Response.Cookies["MSPPProfC"].Expires = DateTime.Now;
Response.Cookies["MSPConsent"].Expires = DateTime.Now;
// Направити користувача на вхідну сторінку.
Response.Redirect("UserInfo.aspx");
}
```

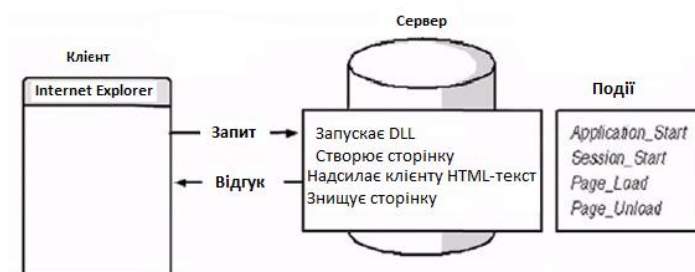
Аутентифікація за допомогою служби Passport надає ряд додаткових комерційних можливостей, у тому числі і захист дітей від небажаної інформації. Ці можливості описані в документації на Passport SDK.

Тема 3. Обробка подій

3.1. Події життєвого циклу Web-додатка

Web-додатки живуть, поки в них є хоч один активний сеанс, а Web-форми живуть буквально миттєвості. Життя Web-додатка складається з наступних етапів:

1. Браузер запитує його початкову сторінку.
2. У гру вступає Web-сервер, що запускає частину коду додатка, яка виконується (DLL), яка і формує відгук.
3. Частина коду, яка виконується, створює екземпляр запитаної Web-форми, генерує HTML-текст відгуку й посилає його браузеру, після чого DLL додатку знищує створений їй екземпляр Web-форми у себе на сервері.



4. Після одержання браузером HTML, згенерованого додатком, користувач може заповнювати текстові поля форми, установлювати потрібні перемикачі й виконувати інші дії, поки він не **спровокує подію**, що викликає відправлення форми на сервер (наприклад, клацнувши кнопку).
5. Ця подія змусить браузер відіслати дані сторінки (стан її відображення) назад на сервер для обробки події.
6. Одержавши стан відображення, сервер створює новий екземпляр Web-форми, заповнює його даними стану відображення і обробляє всі події.
7. Завершивши обробку, сервер повертає браузеру отриманий у результаті HTML і знищує створений ним екземпляр Web-форми.

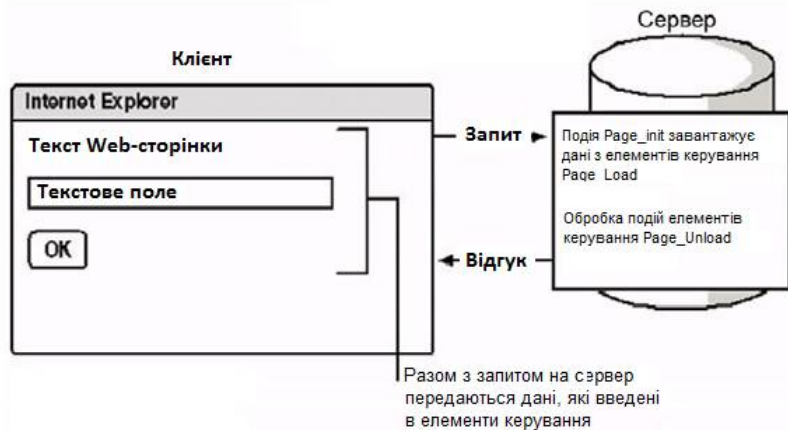
Закінчивши роботу, користувач закриває браузер або переходить на інший Web-сайт, у результаті **сеанс цього користувача** завершується.

Якщо з додатком більше ніхто не працює (тобто сеансів більше немає), він завершується. Однак це не завжди відбувається відразу. ASP.NET керує пам'яттю за допомогою збору сміття. При використанні цього механізму сервер періодично перевіряє посилання об'єктів. Виявивши об'єкт, що

більше не використовується, сервер знищує його, звільняючи зайняту їм пам'ять. Як наслідок, не можна сказати напевно, коли саме спрацює подія *Application_End*.

3.2. Збереження даних Web-форми

Оскільки Web-форми живуть дуже недовго, ASP.NET вживає спеціальні заходи для збереження даних, уведених в елементи керування Web-форми. З кожним запитом на сервер відправляються всі дані, які було введено в елементи керування, де вони записуються в елементи керування під час обробки події *Page_Init*. Тепер ці дані будуть доступні оброблювачеві події *Page_Load*.



Дані, які ASP.NET зберігає між передачею запитів, називаються *станом відображення (view state)* Web-форми. Стан відображення Web-форми доступно тільки усередині цієї Web-форми. Щоб зробити введені в Web-форму дані доступними іншим Web-формам додатка, необхідно зберегти їх у змінних станів (в об'єктах *Application* або *Session*). Ці об'єкти відрізняються областю дії:

- змінні станів на основі *об'єкта Application* доступні всім користувачам додатка, тому їх можна розглядати як глобальні змінні, які доступні декільком користувачам. Ці змінні дозволяється читати і записувати з будь-яких сеансів;
- змінні станів на основі *об'єкта Session* доступні в межах одного сеансу (тільки одному користувачеві). Змінні *Session* нагадують глобальні змінні стандартних Windows-додатків. Ці змінні станів доступні тільки в поточному сеансі.

Змінні *Application* і *Session* оголошуються інакше, ніж звичайні. Змінні станів створюються кодом «на льоту». Наступний код підраховує, скільки разів клацнули кнопку, і записує результат у змінну стану *Session*.

```
// Код з файлу Global.asax.es
protected void Session_Start(Object sender, EventArgs e)
{
    // Ініціалізувати змінну стану Clicks Session
    Session["Clicks"] = 0;
}

// Цей код узятий з файлу StateNEvents.aspx.cs
private void Button1_Click(object sender, System.EventArgs e)
{
    // Збільшити число натискань
    Session["Clicks"] = (int)Session["Clicks"] + 1;
    // Показати число натискань
    Response.Write("Number of clicks: " + Session["Clicks"] + "<br>");
}
```

У змінних станів можна зберігати дані будь-якого типу, від звичайних цілочислених значень до об'єктів. Оскільки змінні станів відносяться до глобальних, необхідно виробити стратегію їх використання в додатку.

Увага! Як правило, у програмах на Visual C# змінні станів *Application* треба ініціалізувати перед використанням. Наприклад, перед перетворенням типів (int)Session[«Clicks»] змінну стану *Clicks* необхідно ініціалізувати, у противному випадку під час виконання ви одержите повідомлення про помилку: «Value null was found where an instance of an object was required».

3.3. Події об'єктів Application і Session

Код, що обробляє події об'єктів *Application* і *Session*, можна помістити у файл **Global.asax**. Події *Application* будуть використовуватися для ініціалізації об'єктів і даних, які повинні бути доступними всім поточним сеансам Web-додатка. Події *Session* служать для ініціалізації даних, які потрібно зберігати в межах одного сеансу, не роблячи їх доступними іншим сеансам. У таблиці перераховані оброблювачі різних подій *Application* із вказівкою обставин, при яких вони викликаються.

Таблиця Оброблювачі подій *Application*

Ім'я	Коли викликається
<i>Application_Start</i>	Перший користувач відкрив початкову сторінку Web-додатка
<i>Application_End</i>	З додатком більше ніхто не працює
<i>Application_BeginRequest</i>	На початку кожного запиту до сервера. Запит відбувається щоразу, коли браузер відкриває яку-небудь сторінку додатка
<i>Application_EndRequest</i>	Наприкінці кожного запиту до сервера
<i>Session_Start</i>	Новий користувач відкрив початкову сторінку додатка
<i>Session_End</i>	Користувач покинув додаток, заклавши браузер, або наступив тайм-аут по бездіяльності

У додатках Web Forms *сеанс (session)* представляє унікальний екземпляр браузера. Якщо користувач запустить у себе на комп'ютері кілька екземплярів браузера і відкриє в кожному з них сторінки Web-додатка, для кожного екземпляра браузера буде створений окремий сеанс.

Щоб побачити події *Application* і *Session* у дії, додайте у файл **Global.asax** проекту Web Forms наступний код:

```
protected void Application_Start(Object sender, EventArgs e)
{
    // Оголосити змінні станів Application
    Application["AppCount"] = 0;
    Application["SessCount"] = 0;
    // Зареєструвати запуск додатка.
    Application["AppCount"] = (Int)Application["AppCount"] + 1;
}

protected void Session_Start(Object sender, EventArgs e)
{
    // Підрахувати сеанси
    Application["SessCount"] = (Int)Application["SessCount"] + 1;
    // Показати число екземплярів додатка.
    Response.Write("Number of applications: " + Application["AppCount"] + "<br>");
    // Показати число сеансів.
    Response.Write("Number of sessions: " + Application["SessCount"] + "<br>");
}

protected void Session_End(Object sender, EventArgs e)
{
    // Зменшити число сеансів.
    Application["SessCount"] = (Int)Application["SessCount"] - 1;
}
```

Щоб побачити, як працюють ці події, виконайте попередній код, потім запустіть новий екземпляр браузера і відкрийте в ньому свій Web-додаток, набравши його адресу. З кожним новим екземпляром браузера число сеансів збільшується, але число додатків залишається рівним 1.

Зверніть увагу, що внутрішні об'єкти, такі, як ***Session i Response***, не доступні на момент події ***Application Start***. Щоб задіяти ці об'єкти, прийдеться дочекатися подій, які відбуваються при їхньому створенні.

3.4. Події Web-форми

Події Web-форми використовуються для обробки і збереження даних, уведених на Web-сторінці, зв'язування даних, а також для обробки виключень Web-сторінки. У таблиці ці події перераховані в порядку їхньої генерації. Найчастіше в коді доводиться мати справу з оброблювачами подій ***Page_Load*** і ***Page_Error***.

Таблиця Події Web-форми

<i>Ім'я оброблювача</i>	<i>Коли відбувається</i>
<i>Page_Init</i>	При завантаженні і ініціалізації серверних елементів керування з використанням стану відображення Web-форми. Це перший етап життєвого циклу Web-форми
<i>Page_Load</i>	При завантаженні серверних елементів керування в об'єкт <i>Page</i> . У цей момент доступні дані стану відображення, тому код для зміни параметрів елементів керування або відображення тексту на сторінці варто помістити в оброблювач саме цієї події
<i>Page_PreRender</i>	Перед візуалізацією об'єкта <i>Page</i>
<i>Page_Unload</i>	При вивантаженні сторінки з пам'яті
<i>Page_Error</i>	При виникненні неопрацьованого виключення
<i>Page_AbortTransaction</i>	При скасуванні транзакції
<i>Page_CommitTransaction</i>	При фіксації транзакції
<i>Page_DataBinding</i>	При прив'язці до джерела даних серверного елемента керування, розташованого на сторінці
<i>Page_Disposed</i>	При звільненні об'єкта <i>Page</i> . Це остання подія життєвого циклу <i>Page</i>

Щоб виконати ініціалізацію при першому відвідуванні клієнтом Web-форми, можна використати подію ***Page_Load*** разом із властивістю ***IsPostBack***. У результаті вийде «подія», що нагадує ***Session_Start***, але відбувається на рівні сторінки, а не на рівні додатка. Наступний код ініціалізує об'єкт і при першому перегляді сторінки зберігає його в змінній стану ***Session***:

```
// Оголосити новий об'єкт
FlashcardClass FlashCard = new FlashCardClass();
private void Page_Load(object sender, System.EventArgs e)
{
    If (!IsPostBack)
    {
        // Перемішати картки.
        FlashCard.Shuffle();
        // Зберегти об'єкт у змінній Session
        Session["FlashCard"] = FlashCard;
    }
    // Одержати значення змінної FlashCard з об'єкта Session.
    FlashCard = (FlashCardClass) Session["FlashCard"];
    RefreshDisplay();
}
```

Інші сторінкові події дозволяють побудувати вид сторінки і обробляти події, пов'язані з даними.

3.5. Події серверних елементів керування

Події, що спрацьовують у відповідь на певні дії користувача, є у всіх серверних елементів керування (таких, як **Button**, **TextBox** і **DropDownList**), але не всі ці події однакові. Існує три типи подій серверних елементів керування:

1. Події, що ініціюють повернення форми (**post-back events**) - викликають повернення Web-сторінки на сервер для негайної обробки подій. Вони помітно знижують швидкодію через додатковий обмін даними із сервером.
2. Події які кешуються (**cached events**) - зберігаються в стані відображення сторінки. Обробляються, коли відбувається подія, що ініціювала повернення форми.
3. Події перевірки (**validation events**) - обробляються прямо на сторінці, не вимагають повернення форми або кешування.

Виконується наступна черговість подій серверних елементів керування Web-форми:

1. Події елементів перевірки обробляються до відправлення сторінки на сервер.
2. Події **Page_Init** і **Page_Load** обробляються при поверненні сторінки, потім виконується обробка внутрішніх подій
3. В останню чергу обробляється подія, що ініціювала повернення форми.

Елементи керування **Button**, **LinkButton** і **ImageButton** генерують події, що викликають повернення форми. На відміну від них елементи керування **TextBox**, **DropDownList**, **ListBox**, **RadioButton** і **CheckBox** генерують кешуємі події, але їх поведінку можна змінити, привласнивши властивості **AutoPostBack** значення **True**.

3.6. Відключення змінної стану Session

За замовчуванням ASP.NET підтримує змінну стану **Session** для кожної сторінки додатка. Якщо сторінці не потрібна сеансова змінна стану, то для підвищення продуктивності її можна відключити.

Щоб відключити змінну стану **Session** для однієї Web-форми: викличте вікно Web Form Properties для форми і встановіть **EnableSessionState** як **False**.

Щоб відключити змінну стану **Session** у всьому додатку: у файлі **Web.config** встановіть для тегу **<sessionstate mode>** значення off: **<sessionstate mode="off" ...>**

3.7. Визначення моменту завершення додатка

Коли користувач у перший раз запитує сторінку Web-додатка, IIS запускає цей додаток (якщо він ще не запущений) і створює для цього користувача сеанс. Всі наступні запити цього користувача є частиною його сеансу. Сеанси важливі по двох причинах:

- вони дозволяють ASP.NET зберігати специфічні для користувача дані в змінній стану **Session**;
- вони визначають момент завершення додатка. Як тільки закінчується останній сеанс, IIS завершує додаток.

Тривалість сеансу визначається значенням тайм-ауту, заданим у файлі **Web.config**. За замовчуванням воно становить 20 хвилин, тобто по закінченні 20 хвилин після останнього запиту користувача його сеанс завершується, а якщо немає інших сеансів, завершується і додаток.

IIS вважає Web-додатком будь-який файл, що виконується ним, з каталогів Web-сайту, тому, якщо протягом 20 хвилин користувач звернеться до кожного .aspx- або .htm-файлу в одному з каталогів додатка, його сеанс продовжиться.

Примітка: Можна лише побічно визначити момент завершення додатка, задаючи час тайм-ауту. Це важливий момент, оскільки від нього залежить життєвий цикл додатка на сервері.

Менше значення тайм-ауту сеансу дозволяє швидше звільнити ресурси сервера, оскільки в цьому випадку швидше минає строк дій сеансових даних. Однак занадто мале значення тайм-ауту може змусити сервер завершити сеанси, які ще використовуються, що чревато збоями в роботі додатка. Щоб змінити значення тайм-ауту сеансу, виконаєте дії, описані далі.

1. Відкрийте проект в Visual Studio .NET.
2. Відкрийте файл **Web.config**, двічі клацнувши його значок у вікні Solution Explorer.

3. Змініть значення тайм-ауту, відредагувавши наступний рядок:
`<sessionState mode="InProc" stateConnectionString="tcpip=127.0.0.1:42424" sqlConnectionString="data source=127.0.0.1;user id=sa;password=" cookieless="false" timeout="20" />`
4. Збережіть файл **Web.config**.

Тема 4. Простори імен в Web-додатках

4.1. Огляд простору імен Web

Визначення класів, що представляють об'єкти, які використовуються в Web-додатках, знаходяться в просторі імен **System.Web**. У таблиці ... визначена ієрархія просторів імен, розташованих в **System.Web** і описані визначення класів, які знаходяться в кожному з них. Ця таблиця стане в нагоді вам при пошуку об'єктів, необхідних для виконання різних завдань.

Таблиця Ієрархія простору імен **System.Web**

Простір імен	Які класи містить
<i>System.Web</i>	<i>Application, Browser, Cache, Cookies, Exception, Request, Response, Server і Trace</i> . Ці класи необхідні для вирішення більшості завдань Web-програмування. Об'єкт <i>Application</i> , визначуваний у файлі Global.asax , є екземпляром класу <i>Application</i>
<i>System.Web.Session.State</i>	<i>Session</i> . Використовується для збереження і витягання елементів із змінних стану <i>Session</i>
<i>System.Web.Services</i>	<i>WebService</i> . Потрібний для створення Web-сервисов і роботи з ними
<i>System.Web.UI</i>	<i>Page і Control</i> . Застосовується в Web-формах для створення призначеного для користувача інтерфейсу і управління ним, Web-форми представлені класом <i>Page</i>
<i>System.Web.UI.Web.Controls</i>	Класи всіх серверних елементів управління, використовуваних в Web-формах
<i>System.Web.UI.HtmlControls</i>	Класи всіх HTML-елементів управління, вживаних в Web-формах
<i>System.Web.Caching</i>	<i>Cache</i> . Необхідний для управління кешуванням на стороні сервера з метою підвищення продуктивності додатка
<i>System.WebMail</i>	<i>MailMessage, MailAttachment і SmtMail</i> . Потрібні для відправки з додатка повідомлень по електронній пошті

<i>System.Web.Security</i>	Містить класи для об'єктів і модулів аутентифікації, використовуваних для аутентифікації користувачів і забезпечення конфіденційності при роботі з додатками
----------------------------	--

При програмуванні Web-додатків доводиться безпосередньо мати справу з двома типами об'єктів, створеними на основі класів з простору імен Web:

- об'єкт **Application**. Походить від класу **HttpApplication**. У додатку відповідне визначення знаходиться у файлі **Global.asax**;
- об'єкти Web-форм. Походять від класу **Page**. У додатку відповідне визначення зберігається в модулях форм.

Наступний код демонструє оголошення класів **Global** і **WebForm1**, що генеруються Visual Studio. ASP.NET автоматично створює екземпляри цих класів під час виконання.

```
public class Global: System.Web.HttpApplication
{
}
public class WebForm1: System.Web.UI.Page
{
}
```

Об'єкти **Global** і **WebForm1** — точки входу, що відкривають доступ до решти Web-об'єктів додатка. З ними, а також з об'єктами **Request** і **Response** в коді доводиться працювати особливо часто.

4.2. Використання об'єкту Application

Об'єкт **Application** займає верхній рівень ієрархії об'єктів Web-приложения і служить для конфігурації додатка і збереження відомостей про стан. Властивості і методи об'єкту **Application** відкривають доступ до інших об'єктів Web-дodatка.

Запускаючи додаток, ASP.NET автоматично створює екземпляр об'єкту **Global**, визначеного у файлі **Global.asax**. За допомогою подій об'єкту **Global** вдається конфігурувати додаток і ініціалізувати змінні стану, оголошені на рівні застосування.

Від базового класу (**HttpApplication**) об'єкт **Global** успадковує властивості і методи для доступу до своїх підлеглих об'єктів. Ці властивості і методи служать для отримання інших об'єктів з об'єкту **Application**.

Таблиця Властивості і методи об'єкту **Application**

Ім'я	Призначення
Application	Збереження елементів даних в змінних стану Application
Context	Отримання об'єктів Trace , Cache , Error і інших для поточного контексту
Modules	Доступ до модулів HTTP
Request	Читання запитів і отримання об'єктів Browser , CtientCertificates , Cookies і Files з поточного запиту
Response	Запис у відгук тексту і даних, а також отримання об'єктів Cache , Cookies і Output з поточного відгуку
Server	Обробка запитів і відгуків. Допоміжні методи для кодування і декодування URL підтримуються об'єктом Server
Session	Збереження даних в змінних стану Session

<i>User</i>	Отримання результатів аутентифікації користувача, що зробив поточний запит. За умовчанням Web-приложения вирішують анонімний доступ
-------------	---

Наступний код перевіряє за допомогою об'єктів *Request* і *Browser*, доступних з об'єкту *Application*, чи присланий запит браузером з номером версії 4.0 або вище.

```
protected void Application_BeginRequest(Object sender, EventArgs e)
{
    // Цей код виконується перед обробкою кожного запиту
    if (Request.Browser.MajorVersion < 4)
    {
        // Відключити додаткові можливості
    }
}
```

4.3. Використання об'єкту Page

Об'єкт *Page* управляє призначенням для користувача інтерфейсом додатка. Властивості і методи об'єкту *Page* забезпечують доступ до інших об'єктів призначеного для користувача інтерфейсу. Властивості і методи об'єкту *Request* відкривають доступ до його підлеглих об'єктів (таблиця ...).

Таблиця Властивості і методи об'єкту Request

Ім'я	Призначення
Browser	Визначення можливостей запрошуючого браузера. Властивості цього об'єкту указують номер версії браузера, чи є він браузером AOL, чи підтримує він файли cookie і деяку іншу інформацію
ClientCertificates	Аутентифікація клієнта
Cookies	Отримання інформації від клієнта у вигляді файлів cookie
Files	Отримання файлів, що закачуються клієнтом
InputStream	Читання і запис присланого запиту у вигляді неструктурованих даних

Наступний код використовує об'єкт *Request*, щоб перед збереженням вмісту cookie в змінній стану *Session* перевірити, чи підтримує браузер файли cookie і чи існує цей файл cookie.

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Цей код виконується при першому відображенні сторінки.
    if(!IsPostBack)
    // Перевірити, чи підтримує браузер файли cookie.
    if(Request.Browser.Cookies)
    // Перевірити, чи існує cookie на ім'я UName
    if (Request.Cookies["UName"]!= null)
    // Набути значення cookie.
    Session["User"]= Request.Cookies["UName"].Value;
}
```

4.4. Використання об'єкту *Response*

Об'єкт *Response* застосовується для формування відгуку сервера, що пересилається клієнтському браузеру. Властивості і методи об'єкту *Response* забезпечують доступ до решти об'єктів, складаючих запит.

У таблиці ... перераховані властивості і методи об'єкту *Response*, що забезпечують доступ до його підлеглих об'єктів.

Таблиця Властивості і методи об'єкту *Response*

Ім'я	Призначення
<i>Cashe</i>	Визначає, як сервер кешує відгуки перед відправкою їх клієнтові
<i>Cookies</i>	Задає вміст файлів cookie , призначених для пересилання клієнтові
<i>Output</i>	Читання і запис неструктурованих даних, що повертаються клієнтові у вигляді відгуку

Наступний код створює cookie і посилає його клієнтові у складі відгуку.

```
private void Page_Load(object sender, System.EventArgs e)
{
    //Виконується при першому відображенні сторінки
    if(!IsPostBack)
    // Якщо браузер підтримує cookies...
    if (Request.Browser.Cookies)
    {
        //...створити cookie...
        HttpCookie cookUname = new HttpCookie("UName");
        cookUname.Value = "Wombat";
        //... і додати його до відгуку.
        Response.Cookies.Add(cookUname);
    }
}
```

Тема 5. Збереження відомостей про стан

5.1. Способи збереження відомостей про стан

Web-форми створюються і знищуються при кожному запиті клієнтського браузера. Через ці особливості значення змінних, оголошених у Web-формі, пропадають після її відображення. У ASP.NET існують різні засоби збереження змінних між запитами, що дозволяють вирішити цю проблему:

- рядки запитів — необхідні для передачі інформації між запитами і відгуками у складі Web-адреси. Рядки запитів видно користувачеві, тому в них не можна передавати секретну інформацію, наприклад паролі;
- файли **cookie**—використовуються для зберігання невеликих порцій інформації на клієнтській машині. Клієнт може відмовитися від запису cookie, це потрібно передбачити при написанні коду;
- стан відображення — ASP.NET зберігає елементи, записані в сторінкову властивість *ViewState*, в прихованих полях сторінки;
- змінні стани **Session**—потрібні для зберігання даних, локальних для поточного сеансу (доступних єдиному користувачеві);
- змінні стани **Application** — використовуються для зберігання даних, які мають бути доступні всім користувачам додатка.

5.1.1 Використання рядків запитів

Рядки запитів дозволяють пересилати разом з адресою додаткову інформацію. У HTML-тегах гіперпосилань рядок запити слідує після знаку питання:

```
<A HREF="WebForm1.aspx?UName=Wombat">Query string sample.</A>
```

Щоб переслати рядок запити в кодї, її слід додати до адреси, передаваного методу *Redirect*. Нижче показаний обробник події *Click*, функціонально еквівалентний попередньому HTML-тегу:

```
private void Button1_Click(object sender, System.EventArgs e)
{
    // Відображувати сторінку ще раз з урахуванням рядка запити.
    Response.Redirect("Webform1.aspx?UName=Womba");
}
```

Для витягання рядків запитів в кодї застосовується метод *QueryString* об'єкту *Request*. Наступний код відображує елемент *UName*, що витягує з рядка запити, створеного в попередніх прикладах:

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Відображувати рядок запити
    Response.Write(Request.QueryString["UName"]);
}
```

5.1.2 Використання файлів cookie

Файли *cookie* застосовуються для зберігання невеликих порцій даних на комп'ютері клієнта. Web-сайти часто звертаються до файлів *cookie* для зберігання особистих параметрів і іншої інформації клієнтів. Оскільки клієнт може відмовитися від *cookie*, перед спробою запису *cookie* важливо переконатися, що це дозволено браузером. Наступний код перевіряє, чи вирішує браузер запис *cookie*, і якщо так, зберігає особисті параметри користувача.

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Цей код виконується при першому відображенні сторінки
    if(!IsPostBack)
    // Якщо браузер підтримує файли cookie...
    if(Request.Browser.Cookies)
    {
        //...створити cookie...
        HttpCookiecookUPrefs = new HttpCookie("UPrefs");
        cookUPrefs.Value = "English";
        //...і додати у відгук.
        Response.Cookies.Add(cookUPrefs);
    }
}
```

Наступний код перевіряє наявність файлу *cookie*, і якщо перевірка успішна, отримує його.

```
private void Page_Load(object sender, System.EventArgs e)
{
    //Цей код виконується при першому відображенні сторінки.
    if(!IsPostBack)
    // Якщо браузер підтримує файли cookie...
    if(Request.Browser.Cookies)
    // ... перевірити, чи існує cookie UPrefs.
```

```

if (Request.Cookies["UPrefs"] != null)
// Зберегти значення cookie.
Session["Lang"] = Request.Cookies["UPrefs"].Value;
}

```

5.1.3. Використання стану відображення

Властивість *ViewState* служить для зберігання даних в прихованих полях сторінки. Оскільки *ViewState* зберігає дані в сторінці, цей спосіб годиться лише для тих елементів, які можна серіалізувати. Для збереження в *ViewState* складніших елементів їх доведеться спочатку перетворити в рядок, а після витягання — повернути в початковий вигляд.

Наступний код витягує текст з текстового поля і поміщає його в елементи таблиці, розташовані на сторінці. Оскільки не можна зберігати об'єкти безпосередньо у властивості *ViewState*, процедура *Button1_Click* спочатку зберігає отримані рядки в *ViewState*, а потім процедура *Page_Load* створює на основі збережених текстових рядків елементи управління, складові таблиці.

```

private void Button1_Click(object sender, System.EventArgs e)
{
    // Запустити текст в ViewState.
    ViewState.Add(ViewState.Count.ToString(), TextBox1.Text);
}
private void Page_Load(object sender, System.EventArgs e)
{
    if (IsPostBack)
    // Для кожного елементу ViewState
    foreach (StateItem staItem in ViewState.Values)
    {
        TableRow rowNew = new TableRow();
        TableCell celNew = new TableCell();
        // Запустити текст в осередок.
        celNew.Text = staItem.Value.ToString();
        // Додати осередок в рядок.
        rowNew.Cells.Add(celNew);
        // Додати рядок в таблицю.
        Table1.Rows.Add(rowNew);
    }
}

```

ASP.NET кодує приховані сторінкові дані, тому користувач не зможе прочитати їх з початкового тексту сторінки. Якщо, скориставшись попереднім кодом, додати декілька елементів в таблицю, а після вибрати в меню браузера елемент View\View Source, приховані поля виглядатимуть приблизно так:

```

</HEAD>
<body MS_POSITIONING="GridLayout">
<form name="Form1" method="post" action="WebForm1.aspx" id="Form1">
<input type="hidden" name="VIEWSTATE" value="test" />

```

5.1.4. Використання змінних стану Application і Session

Змінні стани *Application* і *Session* необхідні для зберігання даних, які потрібні впродовж часу життя додатка або окремого сеансу відповідно. У змінних стану *Application* і *Session* можна зберігати дані будь-якого типу, включаючи об'єкти. Але врахуйте наступне, перш ніж користуватися цим механізмом:

- змінні стани *Application* і *Session* створюються «на льоту», при цьому не виконується звичайна перевірка імені змінної і контроль типів. Слід обмежувати число точок доступу до цих змінних;
- обслуговування змінних стану *Session* знижує продуктивність. Ці змінні дозволяється відключати на рівні додатка або сторінки;
- змінні стану *Application* доступні тільки в поточному процесі, доступ до них з іншого процесу неможливий. В разі застосування на декількох серверах (або на багатопроцесорному сервері), у кожного процесу цього застосування буде своя змінна стану *Application*;
- зона видимості змінних стану *Application* визначається межами додатка, заданими IIS.

5.2. Впорядкування доступу до змінних стану

Змінні стану *Application* і *Session* — потужний і тому небезпечний засіб. Якщо використовувати їх безсистемно, простіше простого наробити помилок в коді, як часто бувало в старому Basic. Наприклад, наступний код створює дві змінних стану *Application*.

```
Application["Uname"] = "Wombat";
Response.Write(Application["Uname"]);
```

Код в першому рядку створює змінну і зберігає в ній деякий текст. Із-за банальної друкарської помилки в імені змінної код наступного рядка створить і відобразить на сторінці нову, порожню змінну, в результаті на сторінці теж буде порожньо. В цьому випадку знайти помилку просто, чого не скажеш, якщо така помилка виникне десь в глибині *структури рішень* (decision structure).

Щоб уникнути подібних проблем слід упорядковувати доступ до змінних стану *Application* і *Session*. Найпростіше зробити це таким чином. Для кожного необхідного елементу даних оголошуються змінні на рівні сторінки, потім при виконання обробника події *Page_Load* в них записуються значення змінних стану *Application* і *Session*, а при виконанні обробника події *Page_Unload* значення змінних, оголошених на рівні сторінки, повертаються в змінні стану. Впорядковане читання і запис змінних стану — корисний прийом програмування, який рекомендується узяти на озброєння. Наступний код ілюструє впорядкований доступ до змінних стану:

```
string mstrUname = "";
private void Page_Load(object sender, System.EventArgs e)
{
    // Перевірити, чи існує змінна стану.
    if(Application["Uname"] != null)
        // Набути значення змінної стану.
        mstrUname = Application["Uname"].ToString();
    // Привласнити значення.
    mstrUname = "Wombat";
    // Використовувати змінну
    Response.Write(mstrUname);
}

private void Page_UnLoad(object sender, System.EventArgs e)
{
    // Повернути значення в змінну стану.
    Application["Uname"] = mstrUname;
}
```

Увага! В разі Visual C#, перш ніж викликати будь-які методи змінної стану, наприклад *ToString()*, обов'язково переконайтеся, що її значення відмінне від *null*, інакше ви отримаєте помилку часу виконання із-за порожньої змінної стану.

Тема 6. Використання баз даних в додатках ASP.NET

Діяльність більшості Web-додатків зосереджена на витяганні, відображенні і модифікації даних. Ці завдання здаються достатньо прямолінійними, але за останнє десятиліття способи використання даних постійно мінялися. Розробники перейшли від простих клієнтських застосувань з локальними базами даних до розподільних систем, заснованих на централізованих базах даних і виділених серверах. В той же час розвивалися технології доступу до даних.

Сьогодні понад 95% всіх інформаційних систем так чи інакше використовують бази даних. Ось чому питання організації взаємодії додатка з базою даних є актуальними, а увага провідних розробників програмного забезпечення прикована до цієї проблеми. Дана лекція присвячена вивченню питань, пов'язаних з організацією взаємодії Web-приложений з базами даних.

6.1. Основні відомості про модель доступу до даних ADO.NET

ADO.NET є набором бібліотек, що входять в Microsoft .NET Framework і призначених для взаємодії з різними сховищами даних з .NET-додатків. Бібліотеки ADO.NET включають всі необхідні класи для підключення до джерел даних практично довільного формату, виконання запитів до цих джерел і отримання результату.

Крім того, безперечною перевагою ADO.NET є можливість роботи з від'єднаними джерелами даних, що є структурами, які організовують дані в оперативній пам'яті комп'ютера і працювати з якими можливо з використанням тих засобів доступу до даних, що стали вже звичними. Таким чином, ADO.NET можна використовувати як надійний, ієрархічно організований від'єднаний від джерела кеш даних для автономної роботи, що незамінне при побудові застосувань, що масштабуються, особливо орієнтованих на Web.

На сьогоднішній день ADO.NET є найбільш розвинутою технологією доступу до даних серед технологій, розроблених корпорацією Microsoft. Вона розвиває ті принципи, які були закладені в таких технологіях, як DAO і ADO, роблячи їх простішими в застосуванні, потужнішими і універсальними. В той же час ADO.NET є вже іншою технологією доступу до даних.

6.1.1. Архітектура ADO.NET

Оскільки ADO.NET є набором класів для організації взаємодії клієнтського застосування з базою даних, розглянемо об'єктну модель ADO.NET

На рис.....показані класи, складові об'єктної моделі ADO.NET

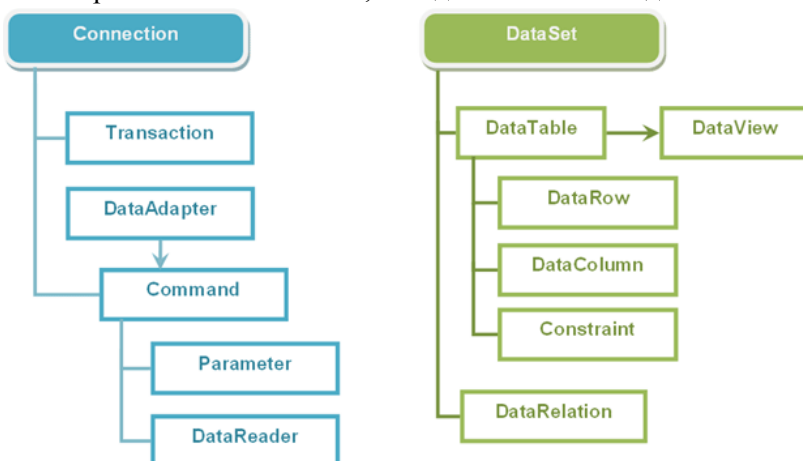


Рис. Ієрархія об'єктів ADO.NET

Об'єкти, розташовані в лівій частці, називаються **приєднаними** і необхідні для управління з'єднанням, транзакціями, вибіркою даних і передачею змін даних в БД. Вони безпосередньо взаємодіють з базою даних.

Об'єкти, розташовані в правій частці, називаються **від'єднаними**. Вони дозволяють працювати з даними автономно. Від'єдані об'єкти не взаємодіють безпосередньо з приєднаними для отримання даних з БД. Точніше, можна сказати, що вони не взаємодіють зі всіма об'єктами лівої частки ієрархії об'єктів. І хоча питання заповнення даними від'єднаних об'єктів обговорюватимуться

пізніше, можна сказати, що однією з **форм взаємодії** цих двох груп є **використання об'єктом DataAdapter об'єкту DataSet** для заповнення останнього набором даних, що витягують безпосередньо з БД.

Одній з основних ідей, лежачих в основі ADO.NET, є наявність постачальників даних.

Постачальник даних - це набір класів, призначених для взаємодії з сховищем даних певного типу. За рахунок цього модель ADO.NET є надзвичайно гнучкою і розширюваною. Розглянемо рівні моделі постачальників ADO.NET.

Додаток .NET взаємодіє з базою даних за допомогою **постачальників даних**. **Кожен постачальник даних може забезпечувати доступ тільки до бази даних певного формату**. Так, для доступу до БД Microsoft SQL Server використовується постачальник SQL Server для .NET, для доступу до БД Oracle - постачальник Oracle для .NET і так далі. Названі бази даних є одними з найпоширеніших у всьому світі, тому постачальники даних для них виділені в окремі елементи моделі ADO.NET; проте існує безліч баз даних іншого формату, до яких необхідно здійснювати доступ з додатка .NET. Для цього можуть застосовуватися постачальники даних OleDb для .NET або ODBC для .NET, що забезпечують доступ до будь-яких даних, для яких існує драйвер OleDb або ODBC відповідно.

При побудові додатка, що використовує доступ до даних, необхідно спочатку спробувати знайти "рідного" постачальника даних .NET. Коли ж такого не існує, **можна скористатися OleDb, якщо існує відповідний драйвер для джерела даних, до якого встановлюється підключення**. Так як технологія OleDb існує достатньо давно, створено багато драйверів для різних джерел даних, що підтримують її. Якщо в системі не встановлений відповідний драйвер OleDb, його можна спробувати знайти на сайті розробника тієї бази даних, до якої здійснюється доступ, або на сайті Microsoft.

Те ж саме справедливо і для технології ODBC. У ряді випадків існує декілька альтернативних варіантів організації доступу до даних певного формату. Наприклад, доступ до джерела даних на основі SQL Server можна організувати використовуючи або постачальник SQL Server для .NET, або постачальник OleDb. Проте завжди переважно використовувати той постачальник даних, який спеціально призначений для забезпечення доступу до конкретного джерела даних, оскільки він враховує його особливості.

Кожен постачальник .NET реалізує однакові базові класи - Connection, Transaction, DataAdapter, Command, Parameter, DataReader імена яких залежать від постачальника. Наприклад, у постачальника SQL Server існує об'єкт **SqlDataAdapter**, у постачальника OleDb — **OleDbDataAdapter** і так далі

У кожного постачальника даних існує власний простір імен. Хоча всі постачальники відносяться до простору імен System.Data, кожен з них містить свій підрозділ цього простору, який містить об'єкти, специфічні для даного постачальника. Наприклад, об'єкт **SqlDataAdapter** знаходиться в просторі імен **System.Data.SqlClient**.

Всі постачальники даних .NET реалізують однакові базові функції, тому код, що створюється для доступу до даних, виглядає приблизно однаково незалежно від постачальника. Це означає, що, розглядаючи використання інтерфейсів окремого постачальника, ми фактично розглядаємо можливість використання цих же інтерфейсів застосовно і до інших постачальників даних.

Розглянемо їх детальніше об'єкти доступу до даних.

Об'єкт Connection - *представляє з'єднання з джерелом даних*. З його допомогою можна задати типа джерела даних, його місцезнаходження, параметри доступу і ряд інших атрибутів. Перш ніж почати взаємодію з джерелом даних, необхідно встановити підключення до нього за допомогою об'єкту Connection.

Об'єкт Command - *представляє запит до джерела даних, виклик процедури, що зберігається, або прямий запит на повернення вмісту конкретної таблиці*. Як відомо, існує декілька типів запитів. Частку з них повертають дані, витягнуті з джерела даних, інші змінюють записи, треті, - управляють структурою БД. За допомогою об'єкту Command можливо виконати будь-який з перерахованих типів запитів. Відмінності в поведінці об'єкту Command починають виявлятися тоді, коли необхідно виконати той або інший запит. Так, наприклад, при необхідності виконання запиту, що не повертає запису, необхідно використовувати метод **ExecuteNonQuery** об'єкту

Command, а при витяганні даних з БД - метод **ExecuteReader**, який, у свою чергу, повертає об'єкт **DataReader**, що дозволяє проглядати отримані в результаті запиту записи.

Об'єкт DataReader призначений для максимально швидкої вибірки і проглядання записів, що повертаються запитом. Проте він дозволяє проглядати весь результуючий набір записів шляхом переміщення від одного запису до іншої і, таким чином, проглядати тільки одну запис за раз. Крім того, **DataReader** не має можливості для оновлення значень записів і тому може працювати в режимі **"тільки для читання"**, за рахунок чого володіє високою продуктивністю.

Об'єкт Transaction дозволяє здійснювати угруповання записів в логічну одиницю роботи, званою транзакцією. Транзакція логічно об'єднує декілька різних дій, пов'язаних з маніпулюванням даними, в єдине ціле. В процесі виконання дій, здійснюваних в рамках транзакції, СУБД зазвичай кэширует зміни, що вносяться цими діями в дані до моменту завершення транзакції. Це дозволяє проводити відміну будь-яких змін, виконаних в рамках транзакції, у випадку, якщо хоч би одна з дій транзакції завершилася невдало.

Об'єкт Parameter дозволяє вводити в запит елемент, значення якого може бути задане безпосередньо перед виконання запиту. За рахунок цього відпадає необхідність кожного разу змінювати текст самого запиту.

Об'єктом DataAdapter є сполучна ланка між від'єднаними об'єктами ADO.NET і базою даних. З його допомогою здійснюється заповнення таких об'єктів, як **DataSet** або **DataTable**, значеннями, отриманими в результаті виконання запиту до бази даних, для подальшої автономної роботи з ними. Окрім цього **DataAdapter** реалізує ефективний механізм виконання оновлення даних, що зберігаються в базі даних, змінами, внесеними до даних об'єктів **DataSet** і **DataTable**.

Об'єкт DataTable дозволяє проглядати дані у вигляді наборів записів і стовпців. Фактично, він є аналогом таблиці БД, розміщеним в пам'яті. Гідністю такої організації є можливість автономної роботи з даними. Це означає, що після встановлення з'єднання з базою даних, читання даних і заповнення ними об'єкту **DataTable** можна відключитися від джерела даних і продовжувати працювати з ним в автономному режимі. Така можливість надзвичайно корисна при організації Web-приложений, які мають бути такими, що добре масштабуються і орієнтовані на многопользовательський режим роботи. При цьому, проте, виникають і побічні ефекти, один з яких пов'язаний з тим, що людина, що працює з автономним набором даних, не може побачити змін, що вносяться до даних у цей момент іншими користувачами.

Об'єктом DataColumn є стовпець об'єкту DataTable. Набор же всіх стовпів об'єкту **DataTable** є колекцією **Columns**. За допомогою цього об'єкту можна дістати доступ до значення осередку відповідного стовпця.

Об'єкт DataRow є рядком об'єкту DataTable. Набор всіх рядків цього об'єкту є колекцією **Rows**. **DataRow** дуже часто використовується для доступу до значення конкретного поля певного запису. При цьому застосовується властивість **Item**.

Об'єктом **DataSet** є від'єднаний набір даних, який може розглядатися як контейнер для об'єктів **DataTable**. **DataSet** дозволяє організовувати усередині себе структуру, повністю відповідну реальній структурі таблиць і зв'язків між ними в БД. Це зручно у тому випадку, коли при роботі з базою даних необхідні дані з різних таблиць. В цьому випадку замість того, щоб багато разів звертатися до сервера і вибирати дані з однієї таблиці за раз, можна помістити всі дані в об'єкт **DataSet**, а потім передати його клієнтському застосуванню. **DataSet** є дуже потужним інструментом для роботи з від'єднаними наборами даних. Всі зміни, які вносяться до даних, що зберігаються в **DataSet** кэшируются в об'єктах **DataRow**. Коли виникає необхідність передачі змін з **DataSet** в БД, можливо здійснити передачу тільки змінених даних, замість того щоб передавати всі дані об'єкту, і це значно знижує об'єм даних, що передаються між клієнтським комп'ютером і сервером.

Об'єктом DataRelation є опис зв'язків між таблицями реляційної бази даних. Він надається об'єктом **DataSet** і дозволяє організовувати взаємозв'язку між таблицями від'єданого набору даних об'єкту **DataSet**. Об'єкт **DataRelation** виконує функцію, аналогічну тій, яку виконують зв'язки, визначувані в СУБД між таблицями при створенні структур даних. Це стосується і дотримання принципів посилальної цілісності інформації. Наприклад, **DataRelation** можна набудувати так, щоб зміни значення первинного ключа батьківської таблиці автоматично передавалися (каскадували) дочернім записам, а при видаленні запису в батьківській таблиці автоматично віддалялися записи в дочірніх таблицях, пов'язаних з нею.

Об'єкт DataView призначений для організації можливості переглядання вмісту DataTable різними способами. Це відноситься до таких операцій, як сортування і фільтрація записів. За допомогою DataView допускається переглядати вміст одного об'єкту DataTable з різними установками фільтрації і сортування. Для цього необхідно використовувати два різні об'єкти DataView, пов'язаних з одним об'єктом DataTable. Така можливість виключає необхідність зберігання одного набору даних в двох різних структурах.

6.2. Організація взаємодії з БД

Для того, щоб додаток .NET могло здійснювати взаємодію з джерелом даних, необхідно встановити з'єднання з ним. Найбільш типовим сценарієм роботи Web-додатку є наступний:

1. Встановлюється з'єднання, відкривається підключення до бази даних.
2. Виконується один або декілька запитів, що здійснюють внесення змін до наборів даних джерела даних, а також вибірки даних з БД.
3. Здійснюється відключення від джерела даних. При цьому користувач працює з від'єднаним набором даних, переглядаючи його, виконуючи фільтрацію, вносячи зміни і так далі
4. При необхідності перенесення змін з від'єданого набору даних в БД, а також при необхідності переглядання змін, внесених в БД іншими користувачами, здійснюється підключення до джерела даних, виконуються необхідні дії, після чого проводиться відключення від БД.

Розглянемо ці етапи детальніше.

6.2.1 Підключення до БД

Як вже говорилося вище, для підключення до джерела даних з використанням ADO.NET необхідно скористатися об'єктом **Connection**.

Для розгляду прикладів підключень до джерел даних перш за все необхідно **вибрати постачальника даних**, з яким працюватиме додаток. Потім необхідно підключити відповідні простори імен, що містять визначення об'єктів вибраного постачальника. Як приклади підключення до джерел даних розглядуватимемо бази даних Access і SQL Server 2005 Express Edition. Такий вибір продиктований перш за все тим, що Access дуже добре підходить для побудови невеликих інформаційних систем, невимогливий до ресурсів комп'ютера, не вимагає установки спеціального програмного забезпечення. SQL Server Express є вільно поширюваній СУБД, що володіє всіма достоїнствами комерційного SQL Server 2005 і що має ряд обмежень, зокрема на максимальний об'єм бази даних (не більше 4 Гб). Всі прийоми роботи з базами даних, описувані нижче, можуть бути використані також для роботи з комерційними версіями продуктів Microsoft і іншими СУБД.

Отже, для того, щоб підключитися до бази даних під час виконання програми, необхідно створити **об'єкт Connection**, а також задати його властивості, що визначають поточні параметри підключення.

Основним параметром, що встановлює необхідні опції для підключення до БД, є рядок з'єднання, який є набором пар "ім'я-значення", розділених крапкою з комою. Порядок проходження значень цих параметрів, а також їх регістр не важливі. Рядок з'єднання залежить від СУБД, до якої здійснюється підключення, а також від використовуваного постачальника даних. Проте існує декілька фрагментів інформації, що вказується в рядку підключення, які необхідні практично завжди. Перерахуємо їх і прокоментуємо їх призначення.

1. **Сервер**, на якому знаходиться база даних. Якщо СУБД, до якої здійснюється підключення, розташована на клієнтському комп'ютері (так буде у всіх прикладах, що розглядаються в рамках даного курсу), то замість імені сервера необхідно вказувати ім'я localhost або IP-адрес 127.0.0.1.
2. **Ім'я бази даних**, до якої проводиться підключення.
3. **Спосіб аутентифікації користувача.** Існуючі клієнт-серверні СУБД (до яких відноситься SQL Server, Oracle і ряд інших) дозволяють вказувати в рядку підключення ім'я користувача і пароль, які будуть потрібні для перевірки можливості доступу до бази даних, або використовувати параметри поточного користувача.

Побудова Web-додатків для роботи з базами даних різних видів практично ідентичні. Основна відмінність полягає в способі організації доступу до самої бази, тобто в способі підключення до неї.

Для підключення до БД використовується об'єкт **Connection** з простору імен **System.Data.SqlClient** у випадку з **SQL Server** і **System.Data.OleDb** - у випадку з іншими джерелами даних, наприклад, **Access**. Рядки з'єднання з базами даних при цьому виглядатимуть таким чином:

```
string strSqlConnection = "Data Source=localhost\\sqlexpress;Initial Catalog = TEST_DB; Integrated Security=SSPI  
string strOleDbConnection = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source =  
C:\\Projects\\Ex_Db\\App_Data\\TEST_DB.mdb";
```

Для відкриття з'єднання з базою даних необхідно викликати метод **Open** об'єкту **Connection**. При цьому рядок з'єднання з БД можна передати як параметр конструктора об'єкту, так і потім за допомогою установки відповідної властивості.

```
SqlConnection sqlCon=new SqlConnection(strSqlConnection);  
sqlCon.Open();
```

```
OleDbConnection oleDbCon=new OleDbConnection();  
oleDbCon.ConnectionString=strOleDbConnection;  
oleDbCon.Open();
```

Рядок з'єднання з базою даних можна жорстко прописати в початковому коді додатка, проте це не самий кращий варіант, оскільки при зміні шляху до бази даних або інших параметрів з'єднання доведеться вносити зміни в початковий код додатка і перекомпілювати його. У зв'язку з цим краще всього використовувати рядок з'єднання, збережений у файлі **web.config**.

```
<connectionStrings>  
<add name="TEST_DBConnectionString" connectionString="Provider = Microsoft .Jet.OLEDB.4.0; Data  
Source = C:\\Projects\\Ex_Db\\App_Data\\TEST_DB.mdb"/>  
</connectionStrings>
```

Згодом цей рядок можна витягувати з файлу **web.config**.

```
string strOleDbConnection = WebConfigurationManager.ConnectionStrings  
["TEST_DBConnectionString"].ConnectionString;
```

Управління з'єднанням здійснюється дуже легко. **Методи Open і Close об'єкту Connection виконують всю роботу.** Проте слід враховувати, що при підключенні до бази даних може статися збій, в результаті якого встановити з'єднання з нею опиниться неможливо. Це може бути особливе актуальним при розміщенні бази даних на іншому сервері, який у момент підключення може виявитися недоступним. Для того, щоб невдала спроба з'єднання з базою даних не приводила до фатальних наслідків при роботі додатка, необхідно використовувати конструкції **try catch**, що дозволяють адекватно реагувати на виниклу помилку. Наступний приклад демонструє можливість використання такого підходу.

```
try  
{  
    sqlCon.Open();  
    lbl_DB.Text = "<b>Сервер:</b>" + sqlCon.ServerVersion;  
    lbl_DB.Text += "</br><b>Соединение:</b>" +  
    sqlCon.ToString();  
}  
catch(Exception ex)  
{  
    lbl_DB.Text = "При з'єднанні з БД сталася помилка ";  
    lbl_DB.Text += ex.Message;  
}  
finally  
{  
    sqlCon.Close();  
    lbl_DB.Text += "</br><b>Соединение:</b>";  
    lbl_DB.Text += sqlCon.State.ToString();  
}
```

}

Для обслуговування з'єднання з базою даних витрачаються ресурси комп'ютера, на якому виконується Web-додаток. У зв'язку з цим рекомендується відкривати з'єднання якомога пізніше, а закривати якомога раніше, відразу після того, як всі необхідні дії з об'єктами бази даних були виконані. Крім того, бажано будувати програмний код так, щоб з'єднання з базою даних закривалося при будь-якому результаті з'єднання з нею. У попередньому прикладі код, розташований в блоці `finally`, виконається при будь-якому результаті спроби підключення, що гарантує звільнення ресурсів сервера, не чекаючи моменту, коли пам'ять буде звільнена складальником сміття.

6.2.2. Виконання команд над наборами даних

Встановлення з'єднання є необхідною умовою підключення до бази даних. Проте підключення необхідне для того, щоб виконувати певні команди над елементами бази. Розглянемо можливості виконання команд над джерелами даних.

Одним з основних елементів з набору класів ADO.NET, здатним виконувати будь-якого SQL-оператор, є клас **Command**. Для того, щоб використовувати клас **Command**, необхідно встановити тип команди, встановити текст запиту SQL і прив'язати її до з'єднання з БД.

Існує 3 типи команд класу **Command**:

CommandType. Text	Виконання прямого оператора SQL, текст якого встановлюється у властивості CommandText . Це значення за умовчанням.
CommandType. StoredProcedure	Виконання процедури, що зберігається, ім'я якої встановлене у властивості CommandText .
CommandType. TableDirect	Виконання опиту всієї таблиці бази даних, ім'я якої задається у властивості CommandText . Цей тип команди використовується для зворотної сумісності з деякими драйверами OLE DB і не підтримується постачальником даних SQL Server.

Приклад створення об'єкту **Command** для виконання SQL-запиту і процедури, що зберігається, представлений нижчим.

```
SqlCommand cmd_SQL = new SqlCommand("Select * From Товари", sqlCon);
cmd_SQL.CommandType = CommandType.Text;
```

```
SqlCommand cmd_Proc=new SqlCommand("GetGoods",sqlCon);
cmd_Proc.CommandType = CommandType.StoredProcedure;
```

Для виконання створеної команди необхідно використовувати один з наступних методів.

ExecuteReader()	Виконання запиту SQL і повернення об'єкту DataReader , що представляє однонаправлений курсор, з доступом тільки для читання.
ExecuteNonQuery()	Виконання SQL-команд, призначених для вставки, зміни, видалення записів БД. Результатом роботи команди є кількість рядків, оброблених командою.
ExecuteScalar()	Виконання SQL-команди і повернення першого рядка результату запиту. Зазвичай використовується для виконання команди, що містить агрегуючі функції типу COUNT() , MAX() і так далі

При використанні методу **ExecuteReader()** створюється об'єкт **DataReader**, за допомогою якого можна організувати перебір всіх рядків поверненого набору даних. Для цього необхідно реалізувати цикл по рядках результуючого набору даних. Об'єкт **DataReader** є одним з найпростіших і швидших способів діставання доступу до даних БД. У наступному прикладі демонструється спосіб відображення вмісту таблиці "Товари" у вигляді списку.

```
SqlDataReader rdr_SQL = cmd_SQL.ExecuteReader();
StringBuilder strResult=new StringBuilder("");
while (rdr_SQL.Read())
{
```

```

strResult.Append("<li>");
strResult.Append("Код товару <b>");
strResult.Append(rdr_SQL["Кодтовара"]);
strResult.Append("</b>, Найменування товару <b>");
strResult.Append(rdr_SQL.GetString(1));
strResult.Append("</b>");
strResult.Append(", Ціна <b>");
strResult.Append(rdr_SQL.GetDouble(2));
strResult.Append("</b></li>");
}
rdr_SQL.Close();
lbl_Result.Text = strResult.ToString();

```

Звернення до полів таблиці може відбуватися як по імені, так і за допомогою індексу. У випадку з індексом необхідно застосовувати метод, відповідний типові даних того поля, до якого відбувається звернення. Після завершення звернення до даних за допомогою об'єкту `DataReader` необхідно закрити модуль читання.

У даному прикладі використовується клас `StringBuilder`, який, як вже демонструвалося раніше, ефективніше працює з рядками. Ті ж самі операції з рядками можливі і із застосуванням стандартної операції конкатенації.

Використаний підхід достатньо трудомісткий для того, щоб його застосовувати на практиці для виведення інформації, що міститься в базі даних. У ASP.NET існує декілька потужніших класів, здатних проводити дану операцію. Одним з таких класів є клас **GridView**. Застосування даного класу дозволяє здійснювати виведення інформації на екран дуже простим способом. У наступному прикладі показаний спосіб виведення інформації, що отримується з БД за допомогою об'єкту `DataReader`.

```

GridView1.DataSource = rdr_SQL;
GridView1.DataBind();

```

Метод `ExecuteNonQuery()` використовується для виконання команд, які не повертають результуючих наборови даних. До таких команд відносяться команди **вставки, видалення і оновлення даних**. Результатом роботи методу `ExecuteNonQuery()` є кількість оброблених записів. У наступному прикладі демонструється можливість видалення товару з таблиці "Товари".

```

string strSqlConnection = "Data Source=localhost\\sqlexpress;
Initial Catalog=TEST_DB;Integrated Security=SSPI";
sqlCon = new SqlConnection(strSqlConnection);
SqlCommand cmdDelete = new SqlCommand("DELETE FROM Товари
WHERE Кодтовара=5",sqlCon);
try
{
    sqlCon.Open();
    int n = cmdDelete.ExecuteNonQuery();
    lbl_Delete.Text += String.Format("Видалено {0}
записів<br>",n);
}
catch (SqlException ex)
{
    lbl_Delete.Text += String.Format("Помилка: {0}<br>"
ex.Message);
}
finally
{
    sqlCon.Close();
}

```

У реальних ситуаціях практично завжди користувачеві потрібно ввести деяку умову, відповідно до якої виконуватиметься запит SQL. Такою умовою, наприклад, може бути введення

найменування товару, підмета видаленню. Для реалізації такої можливості в попередньому прикладі додамо елемент управління TextBox, призначений для введення найменування товару, який необхідно видалити, а також змінимо текст рядка, що формує команду, таким чином:

```
SqlCommand cmdDelete = new SqlCommand("DELETE FROM Товари WHERE НазваТовара =  
" + tb_delete.Text + "'", sqlCon);
```

Як видно, в даному випадку, рядок SQL-запроса видалення запису з таблиці "Товари" формується динамічно залежно від введеного в елемент tb_Delete значення. Така практика динамічного формування запитів в реальних застосуваннях допустима, проте при цьому можуть виникати проблеми, пов'язані з безпекою Web-приложения, наприклад атаки впровадженням SQL. Докладніше про такого роду атаках можна прочитати в [1]. Суть цього виду порушення безпеки додатка полягає в тому, що користувач може ввести в поле введення параметра текст, відмінний від того, чого від нього чекає додаток. Це може приводити до того, що текст SQL-запта фактично змінюється і виконує не ту дію, на яку розраховував програміст при його реалізації. Наприклад, якщо в попередньому прикладі в елемент tb_Delete ввести наступний текст "Якийсь товар' OR '1'='1'", то в результаті будуть видалені всі записи з таблиці "Товари", оскільки текст SQL-запта в цьому випадку, після підстановки значення введеного в елемент tb_Delete, буде

```
"DELETE FROM Товари WHERE НазваТовара='якийсь товар' OR '1'='1'"
```

Як видно з цього прикладу, найменування введеного товару стає неважливим, оскільки умова завжди буде істинною за рахунок додавання оператора OR зі свідомо вірною умовою. Можна навести приклади і складніших атак впровадженням SQL, що приводять до ще серйозніших і масштабніших наслідків і створюють реальну загрозу функціонування всього Web-додатку.

Кращою практикою захисту від атаки впровадженням SQL є використання команд, що параметризуються. Крім того, команди SQL, що параметризуються, простіші при формуванні і застосуванні.

6.2.3. Використання команд, що параметризуються

Команда, що параметризується, - це звичайна SQL-команда, в тексті якої використовуються спеціальні символи, які указують місце для динамічної підстановки значень, що передаються об'єкту Command через колекцію Parameters. Наприклад, команда видалення запису з таблиці "Товари", використана вище, виглядатиме таким чином при використанні параметра:

```
DELETE FROM Товари WHERE НазваТовара = @productname
```

Тут @ProductName є параметром, значення якого має бути встановлене до того, як буде запущено виконання запиту. Синтаксис команд, що параметризуються, відрізняється в різних постачальниках даних. Наведений вище приклад справедливий для взаємодії з SQL Server. Для використання тієї ж команди при взаємодії з Access той же запит повинен виглядати таким чином:

```
DELETE FROM Товари WHERE НазваТовара=?
```

Текст програми, що реалізовує підключення до наборів даних SQL Serve і Access, установку значень параметрів запитів і їх виконання, приведений нижче.

```
OleDbConnection AccessCon=new OleDbConnection(strOleDbConnection);  
string strSQLServer = "DELETE FROM Товари WHERE НазваТовара = @productname";  
string strAccess = "DELETE FROM Товари WHERE НазваТовара =?";  
SqlCommand cmdDeleteSQLServer = new SqlCommand(strSQLServer,sqlCon);  
OleDbCommand cmdDeleteAccess=new OleDbCommand(strAccess,AccessCon);  
try  
{  
    cmdDeleteSQLServer.Parameters.Add("@ProductName"  
    tb_Delete.Text);  
    sqlCon.Open();  
    int n = cmdDeleteSQLServer.ExecuteNonQuery();  
    lbl_Delete.Text += String.Format("З бази даних SQL Server видалене {0}  
    записів</br>",n);  
  
    cmdDeleteAccess.Parameters.Add("ProductName", tb_Delete.Text);  
    AccessCon.Open();
```

```

int del = cmdDeleteAccess.ExecuteNonQuery();
lbl_Delete.Text += String.Format("3 бази даних Access видалене {0}
записів</br>", del);

}
catch (Exception ex)
{
    lbl_Delete.Text += String.Format("Помилка: {0}</br>", ex.Message);
}
finally
{
    sqlCon.Close();
    AccessCon.Close();
}

```

Крім того, що робота із запитом, що параметризується, простіше з погляду динамічного формування тексту самого запиту, вона ще забезпечує захист від атаки впровадженням SQL. Так, попередній приклад використання такої атаки в даному випадку не працює.

6.2.4. Використання процедур, що зберігаються

При використанні складної СУБД, що реалізовує повний набір функцій клієнт-серверної СУБД, розробник додатка, що взаємодіє з базою даних, отримує в своє розпорядження декілька додаткових дуже потужних інструментів, що дозволяють полегшити, прискорити і забезпечити процес взаємодії з базою даних в порівнянні з локальними базами даних. Одними з основних таких можливостей є способи організації і використання процедур, що зберігаються, і тригерів. Питання створення і програмування процедур, що зберігаються, і тригерів, а також всілякі питання, пов'язані з особливостями їх реалізації в конкретних СУБД, виходять за рамки даного курсу. Тут же в основному розглядають питання використання процедур, що зберігаються, при роботі з SQL Server без пояснення того, як саме створювалася та або інша процедура, що зберігалася.

Процедура, що зберігається, є набором операторів SQL, що збережений в базі даних (на стороні сервера), має ім'я, а також набір параметрів, за допомогою яких можуть бути передані значення в процедуру, що зберігається. Вони подібні до процедур, використовуваних в мовах програмування, з тією лише різницею, що в даному випадку призначені для обробки даних, що містяться в базі даних.

Процедури, що зберігаються, володіють рядом переваг, головними з яких є такі, як підвищення швидкодії додатка при використанні процедур, що зберігаються, для обробки даних, підвищення безпеки додатка при роботі з даними.

У реальних застосуваннях, якщо використовується СУБД, що підтримує можливості використання процедур, що зберігаються, рекомендується скористатися цим і створювати додаток з їх застосуванням. Крім того, рекомендується все найбільш типові і часто виникаючі дії, які пов'язані з обробкою даних, що містяться в базі даних, а також операції вставки, видалення і зміни проводити за допомогою виклику заздалегідь створених процедур, що зберігаються.

Розглянемо приклад створення і використання простої процедури, що зберігається, дозволяє додати новий товар в таблицю "Товари" бази даних. Текст процедури, що зберігається, реалізованої в СУБД SQL Server 2005, виглядає таким чином:

```

CREATE PROCEDURE AddProduct
@PRODUCTID int
@ProductName varchar(100)
@ProductPrice float
AS
INSERT INTO Товари
VALUES (@ProductID,@ProductName,@ProductPrice)

```

Дана процедура, що зберігається, використовує три значення змінних для формування запиту на додавання даних в таблицю "Товари".

Для її виклику можна скористатися об'єктом **SqlCommand**. Текст програми, що реалізовує виклик даної процедури, що зберігається, приведений нижче.

```
SqlCommand cmd_SQL=new SqlCommand ("AddProduct",sqlCon);
cmd_SQL.CommandType = CommandType.StoredProcedure;
string[] strProduct = tb_AddProduct.Text.Split(new char[]{' '});
cmd_SQL.Parameters.Add(new SqlParameter("@ProductID", SqlDbType.Int, 4));
cmd_SQL.Parameters["@ProductID"].Value = Convert.ToInt32(strProduct[0]);
cmd_SQL.Parameters.Add(new SqlParameter("@ProductName"
SqlDbType.NVarChar, 100));
cmd_SQL.Parameters["@ProductName"].Value = strProduct[1];
cmd_SQL.Parameters.Add(new SqlParameter("@ProductPrice", SqlDbType.Float, 8));
cmd_SQL.Parameters["@ProductPrice"].Value = Convert.ToDouble(strProduct[2]);

try
{
    sqlCon.Open();
    int do = cmd_SQL.ExecuteNonQuery();
}
finally
{
    sqlCon.Close();
}
```

В процесі роботи даної програми текст, введений користувачем в елемент TextBox, перетвориться в масив рядків. У колекцію Parameters об'єкту SqlConnection додається три параметри, відповідних параметрам процедури, що зберігається. Для них встановлюються значення, раніше поміщені в масив рядків. Після чого відкривається з'єднання і посилається запит на виконання процедури, що зберігається, за допомогою команди ExecuteNonQuery().

З даного прикладу видно, що при додаванні нового параметра, промовця в ролі параметра процедури, що зберігається, необхідно обов'язково вказувати його типа, а також довжину (для рядкових типів це кількість символів, для числових - кількість байт).

6.3. Від'єднані набори даних

Логіка взаємодії клієнтського додатку, що розглядала раніше, з базою даних, заснована на з'єднанні, виправдана для реалізації однобічного зв'язку з базою даних. Це може бути або отримання даних, або виконання запитів, пов'язаних з внесенням змін до неї. Реалізація ж складних операцій взаємодії з БД при цьому є дуже трудомістким процесом і вимагає написання великої кількості програмного коду, його відладки і так далі ADO.NET надає досконаліші способи організації двосторонньої взаємодії додатка з базою даних, заснованою на від'єднаних наборах даних.

Основною ідеєю використання від'єднаних наборів даних є зміна алгоритмів взаємодії додатка з базою даних за рахунок підключення до набору даних, виконання запиту і створення копії даних на стороні клієнта, відключення від БД, здійснення маніпуляцій з даними на стороні клієнта, при необхідності внесення змін до бази даних, підключень до неї, передачі змін і відключення. Таким чином, всі основні маніпуляції з даними відбуваються у від'єданому наборі даних, який є копією даних, що зберігаються в БД, а внесення змін відбувається в одній пакетній операції. Все це зменшує час, протягом якого має бути відкрите з'єднання з БД, прискорює роботу і спрощує логіку взаємодії додатка з даними. Розглянемо об'єкт ADO.NET, реалізуючий даний механізм.

6.3.1. Клас DataSet

Об'єктом DataSet є контейнер, що містить об'єкти DataTable і Relation. DataTable є таблицею, що складається з рядків і стовпців. Рядки таблиці представлені об'єктом DataRow, який, у свою чергу, є колекцією стовпців таблиці (об'єкт DataColumn). Дані в DataSet від'єднані від БД. Всі зміни даних кэшуються в об'єктах DataRow. При виникненні необхідності передачі змін в даних об'єкту DataSet існує можливість передачі частки даних, що тільки змінилася, що дозволяє значно економити ресурси каналу зв'язку, оскільки в цьому випадку передається набагато менший об'єм даних.

Узагальнюючи все вищесказане, можна зробити висновок про те, що використання об'єкту DataSet у ряді випадків виявляється ефективнішим, ніж DataReader. Найбільш типовими ситуаціями, в яких рекомендується застосовувати об'єкт DataSet, є наступні:

1. Необхідність реалізації серіалізації даних на диск. DataSet дозволяє легко зберігати дані у файлі XML. При цьому можливі варіанти збереження або тільки даних, або тільки структури даних, або і того і іншого.
2. Необхідність організації навігації по набору даних в двох напрямках. Як уже згадувалося, DataReader забезпечує переміщення по набору тільки вперед. З допомогою ж DataSet можлива організація посторінкового перегляду даних.
3. Необхідна прив'язка декількох елементів управління до одного набору даних. DataSet, на відміну від DataReader, містить засоби організації сортування і фільтрації даних.

6.3.2. Використання DataSet

Вище вже мовилося про те, що DataSet складається з таблиць, які, у свою чергу, складаються з рядків, а вони, у свою чергу, - із стовпців. Кожен з перерахованих елементів реалізований у вигляді класу. Для управління автономними змінами DataSet відстежує інформацію про версію кожного об'єкту DataRow. Це означає, що коли відбувається редагування рядка, її початкове значення зберігається в пам'яті, а рядок позначається як змінена. Аналогічні дії відбуваються і при додаванні і зміні рядків DataTable. Надалі в базу даних можна перенести значення тільки тих рядків, які торкнулися змінами. Таким чином, DataSet ніколи не підтримує постійного з'єднання з базою даних.

Для витягання даних з бази даних і наповнення ними об'єкту DataSet необхідно використовувати ще один об'єкт DataAdapter, який окрім іншого дозволяє оновлювати дані БД на основі внесених до DataSet змін.

6.3.3. Клас DataAdapter

DataAdapter є сполучною ланкою між базою даних і DataSet. Точніше, він зв'язує БД і об'єкт DataTable, розташований усередині DataSet.

DataAdapter містить три основні методи, які дозволяють йому виконувати всі необхідні операції, пов'язані з витяганням і оновленням даних.

Fill	Виконання запиту типа Select, визначеного у властивості SelectCommand, і додавання таблиці, що отримується в результаті даного запиту, в DataSet.
FillSchema()	Виконання запиту типа Select, текст якого розташований у властивості SelectCommand, і додавання таблиці, яка містить тільки структуру даних, отриманих в результаті виконання запиту, в DataSet.
Update()	Застосовує всі зміни, внесені до DataTable, до джерела даних. При цьому виконуються команди вставки, оновлення і видалення, розташовані у властивостях InsertCommand, UpdateCommand, DeleteCommand.

Розглянемо приклад використання об'єктів DataSet і DataAdapter для витягання даних з БД, наповнення DataSet і відображення даних на сторінці Web-додатка.

Перш за все, необхідно встановити підключення до джерела даних. Для цього потрібно використовувати рядок підключення, а також об'єкт Connection. У даному прикладі підключення відбуватиметься до бази даних Test_Db, розташованої на локальному сервері SQL Server Express 2005:

```
string strCon = "Server= .\SQLEXPRESS;Integrated Security=S SPI; Initial  
Catalog=Test_Db";  
string sqlString = "SELECT * FROM Товари";  
SqlConnection sqlCon = new SqlConnection(strCon);
```

Створимо об'єкт DataAdapter і передамо йому як параметри рядок запиту, а також рядок підключення до БД:

```
SqlDataAdapter da = new SqlDataAdapter(sqlString,sqlCon);
```

Тепер необхідно створити об'єкт DataSet і заповнити його даними за допомогою DataAdapter:

```
DataSet ds = new DataSet();
```

da.Fill(ds, "Goods");

З наведеного вище прикладу видно, що метод Fill об'єкту da як параметри використовує ім'я об'єкту DataSet, куди необхідно помістити дані, виконання запиту, який визначений в рядку sqlString, що повертаються в результаті. У другому параметрі можна вказати ім'я, яке буде зіставлено з таблицею, створеною в DataSet. З прикладу видно, що в даному випадку не використовується явний виклик методу Open об'єкту Connection - даний метод викликається неявно при виконання методу Fill. Таким чином, DataAdapter спершу відкриває з'єднання з БД, потім виконує необхідні маніпуляції з даними, після чого закриває відкрите раніше з'єднання. Якщо по якихось причинах такий алгоритм взаємодії з базою даних потрібно змінити, необхідно до виклику методу Fill відкрити з'єднання з БД. В цьому випадку DataAdapter використовуватиме вже існуюче з'єднання.

Після того, як необхідний набір даних витягував з бази даних, його необхідно відображувати на Web-сторінці. Для цього можна скористатися способом, описаним раніше, при розгляді об'єкту DataReader. Проте в даному випадку набагато ефективнішого використання можна добитися із застосуванням можливостей прив'язки даних до візуальних об'єктів, здатних виводити дані на екран. Основна ідея, лежача в основі прив'язки даних, полягає в створенні зв'язку між об'єктом даних і елементом управління. Всю подальшу роботу, пов'язану з витяганням даних і виводом їх на екран, виконує ASP.NET Розглянемо основні способи прив'язки даних і особливості їх виводу на екран, реалізовані в різних елементах управління.

6.4. Прив'язка і відображення даних

Прив'язка даних є дуже потужним засобом організації автоматичного відображення даних в елементі управління без використання програмування.

Більшість елементів управління ASP.NET підтримують прив'язку даних. При цьому прив'язка може працювати як для даних з одним значенням, так і для даних з множинними значеннями. Прив'язка з одним значенням означає, що елемент управління може відображувати єдине значення, витягване з джерела даних. Такі принципи використовуються елементами управління TextBox, LinkButton, Image, Hyperlink. Прив'язка з множинними значеннями означає, що елемент управління може відображувати декілька значень, витяганих з джерела даних. Елементи управління, що підтримують прив'язку з множинними значеннями, будуються на основі списків і електронних таблиць. Типовими представниками таких елементів управління є ListBox і GridView.

Для прив'язки елементів управління до джерела даних необхідно встановити значення властивості DataSource, в якій вказати найменування об'єкту, який містить дані, необхідні для відображення. При установці властивості DataSource створюється логічний зв'язок між елементом управління, здатним відображувати дані, і об'єктом даних, що підлягають відображенню. Після того, як джерело даних визначене, необхідно заповнити його даними. Для цього використовується метод DataBind() елементу управління, що заповнюється даними, який витягує дані з джерела, проходить в циклі по джерелу, визначеному в DataSource, і оновлює сторінку. Прив'язка з множинними значеннями є найбільш потужним типом прив'язки, оскільки в цьому випадку не потрібно програмувати циклічний перебір значень джерела даних і вивід їх на екран - ця логіка вже реалізована в елементі управління, що підтримує множинну прив'язку.

6.4.1. Прив'язка з одним значенням

Елементи управління, що підтримують прив'язку даних з одним значенням, дозволяють здійснити прив'язку деяких з їх властивостей до даних за допомогою вираження прив'язки даних. Вираз прив'язки даних вводиться в тексті сторінки .aspx і полягає усередині обмежувачів <%# %>.

Як вираз прив'язки даних може бути використана загальнодоступна або захищена змінна, а також будь-яке інше вираз, яке може бути обчислене у момент виконання сторінки. Так, як вираз прив'язки даних допустимо використовувати функції, властивості, об'єкти, визначені в класі сторінки.

Для того, щоб обчислити вираження прив'язки, необхідно в коді додатка викликати метод Page.DataBind(). У момент виклику даного методу ASP.NET перевіряє всі вирази на поточній сторінці, при необхідності проводить обчислення і замінює їх відповідними значеннями.

Як приклад розглядемо прив'язку елементу Label до даних таблиці "Товари". Для цього створимо метод `GetProductName()`, що повертає як значення вміст першого рядка стовпця "Найменованітовара" таблиці "Товари".

```
public string GetProductName()
{
    return ds.Tables["Goods"].Rows[0]["НазваТовара"].ToString();
}
```

Тепер необхідно додати елемент Label на сторінку .aspx і встановити вираження прив'язки для властивості Text даного елементу.

```
<asp:Label ID = "Label1" runat="server" Text="< %#GetProductName() %>" > </asp:Label>
```

У методі `Page.Load()` проведемо виклик методу `Page.DataBind()`.

Повний текст початкового коду даного прикладу виглядає таким чином:

```
public partial class _Default : System.Web.UI.Page
{
    private DataSet ds;
    protected void Page_Load(object sender, EventArgs e)
    {
        string strCon = WebConfigurationManager.ConnectionStrings
        ["Test_Db"].ConnectionString;
        string sqlString = "SELECT Кодтовара,НазваТовара,ціна FROM Товари";
        SqlConnection sqlCon = new SqlConnection(strCon);

        SqlDataAdapter da = new SqlDataAdapter(sqlString,sqlCon);
        ds = new DataSet();
        da.Fill(ds, "Goods");
        this.DataBind();
    }

    public string GetProductName()
    {
        return ds.Tables["Goods"].Rows[0]["Найменованітовара"].
        ToString();
    }
}
```

Як видно з даного прикладу, в такий спосіб можна відображувати тільки одне значення якого-небудь поля таблиці. Про те, яким чином можливо здійснювати множинну прив'язку і виводити всі значення, витягнуті з бази даних, буде розказано нижче.

6.4.2. Прив'язка з множинним значенням

Для відображення набору значень необхідно використовувати прив'язку з множинним значенням. Прив'язуваний набір значень повинен міститися в об'єкті, здатному зберігати колекції значень (наприклад, об'єкти `ArrayList`, `Hashtable` і ін.).

Кожен елемент, що підтримує можливість множинної прив'язки, містить наступний набір властивостей:

DataSource	Об'єкт, що містить прив'язувані дані.
DataSourceID	Властивість, яка містить ідентифікатор об'єкту, що використовується для підключення до джерела даних. На відміну від <code>DataSource</code> , значення <code>DataSourceID</code> можна встановити в режимі дизайну на етапі розробки додатка.
DataTextField	Містить найменування стовпця (для рядка) таблиці або властивість (для об'єкту) елементу даних, що включає значення, що відображується на екрані.
DataTextFormatString	Містить необов'язковий рядок форматування, вживаний для відображення

	даних.
DataValueField	Містить унікальний ідентифікатор або поле первинного ключа, яке використовується для витягання даних, пов'язаних з вибраним користувачем елементом.

Принципи використання всіх спискових елементів, здатних відображувати зв'язані дані, однаковий. Для прив'язки такого елемента до джерела даних необхідно встановити значення властивостей `DataSource` і `DataTextField`.

У наступному прикладі на сторінці розміщено два елементи - `DropDownList1` і `ListBox1`, які пов'язані з одним і тим же джерелом даних - як таке виступає таблиця "Товари":

```
protected void Page_Load (object sender, EventArgs e)
{
    string strCon = WebConfigurationManager.ConnectionStrings
    ["Test_Db"].ConnectionString;
    string sqlString = "SELECT Кодтовара,НазваТовара,ціна FROM Товари";
    SqlConnection sqlCon = new SqlConnection(strCon);

    SqlDataAdapter da = new SqlDataAdapter(sqlString,sqlCon);
    DataSet ds = new DataSet();
    da.Fill(ds, "Goods");
    DropDownList1.DataSource = ds.Tables["Goods"];
    DropDownList1.DataTextField = "Наіменованієтовара";
    DropDownList1.DataValueField = "Кодтовара";

    ListBox1.DataSource = ds.Tables["Goods"];
    ListBox1.DataTextField = "НазваТовара";
    ListBox1.DataValueField = "Кодтовара";

    this.DataBind();
}
```

Оскільки об'єкт `DataSet` може містити багато таблиць, у властивості `DataSource` необхідно вказати ім'я таблиці, до якої необхідно здійснити прив'язку. Властивість `DataTextField` містить найменування стовпця таблиці, значення якого відображуватимуться елементом управління. Властивість `DataValueField` є необов'язковою і в даному випадку позначає те, що при виборі значення в елементі управління значення властивості `SelectedItem` цього елемента буде встановлено рівним значенню поля "Кодтовара", відповідного вибраному запису. Рахувати вибрані значення елементів можна таким чином:

```
protected void Button1_Click(object sender, EventArgs e)
{
    StringBuilder str = new StringBuilder();
    str.Append("У Випадному списку вибраний елемент");
    str.Append(DropDownList1.SelectedItem.Text);
    str.Append(" йому відповідає значення ");
    str.Append(DropDownList1.SelectedValue);
    str.Append("<br>");
    str.Append("У Списку вибрані елементи:<br>");
    foreach (ListItem li in ListBox1.Items)
    {
        if (li.Selected)
        {
            str.Append(li.Text);
            str.Append(" значення - ");
            str.Append(li.Value);
            str.Append("<br>");
        }
    }
    Label2.Text = str.ToString();
}
```

```
}
```

У даному прикладі елемент `ListBox1` підтримує можливість вибору відразу декількох елементів (для цього необхідно встановити значення властивості `SelectionMode` даного елемента рівним `Multiple`). Читання значення виділеного елемента в списку здійснюється за допомогою властивості `SelectedItem`. Проте у випадку з `ListBox1` дана властивість використовувати не можна, оскільки можливе множинне виділення. Для визначення кожного елемента, виділеного в `ListBox1`, необхідно застосувати цикл, де відбувається звернення до колекції елементів `Items`, для кожного з яких є властивість `Selected`. Якщо значення даної властивості встановлене рівним `true`, даний елемент виділений і його необхідно відображувати.

Аналогічний принцип можна застосувати з будь-яким іншим елементом управління, що підтримує множинне виділення елементів.

Окрім використаних спискових елементів, ASP.NET містить також і складніші елементи, здатні прив'язуватися до даних і відображувати їх. До таких елементів відносяться `GridView`, `DetailsView`, `FormView`.

`GridView` є таблицею, здатною прив'язуватися до даних і відображувати вміст таблиць. Підтримує вибір, редагування, сортування і переміщення по сторінках даних. `DetailsView` призначений для показу одному запису з таблиці за раз. Підтримує редагування і переміщення по сторінках. `FormView` практично аналогічний `DetailsView`, з тією лише різницею, що `FormView` заснований на шаблонах, і це робить його гнучкішим при відображенні інформації.

Найпотужнішим елементом по кількості можливостей є `GridView`. Дуже часто його доводиться використовувати для відображення інформації у вигляді таблиці. Для прив'язки `GridView` до джерела даних необхідно встановити значення властивості `DataSource`. У наступному прикладі на сторінці розміщений об'єкт `GridView`, який прив'язується до таблиці "Товари" і відображує її вміст:

```
protected void Page_Load(object sender, EventArgs e)
{
    string strCon = WebConfigurationManager.ConnectionStrings
    ["Test_Db"].ConnectionString;
    string strQuery = "SELECT * FROM Товари";
    SqlConnection sqlCon = new SqlConnection(strCon);
    DataSet ds = new DataSet();
    SqlDataAdapter da = new SqlDataAdapter();
    da.SelectCommand = new SqlCommand(strQuery, sqlCon);
    da.Fill(ds, "Goods");
    GridView1.DataSource = ds.Tables["Goods"];
    this.DataBind();
}
```

Показані приклади демонструють способи прив'язки до джерел даних з використанням програмного коду, проте це не єдина можливість здійснити прив'язку елемента управління до джерела даних. Visual Studio 2005 дозволяє здійснити прив'язку з використанням візуальних засобів. Для цих цілей .NET Framework містить декілька елементів управління джерелами даних. Вони розташовані на вкладці `Data` панелі `Toolbox`. Серед них такі як:

SqlDataSource - дозволяє підключатися до будь-якого джерела даних, яке має постачальника даних ADO.NET. До таких джерел даних відносяться SQL Server, OLE DB, Oracle та інші.

ObjectDataSource - дозволяє підключатися до призначеного для користувача класу доступу до даних. Цей підхід доцільно застосовувати при побудові великомасштабних і складних застосувань. Докладніше про використання призначених для користувача класів доступу до даних і елемент `ObjectDataSource` буде розказано в розділі, присвяченому реалізації тривірневої архітектури доступу до даних в ASP.NET

AccessDataSource - дозволяє підключатися до баз даних Microsoft Access.

XmlDataSource - дозволяє підключатися до XML-файлу.

SiteMapDataSource - дозволяє підключатися до файлу `Web.Sitemap`, що містить опис навігаційної структури Web-приложения.

Для використання елемента управління джерелами даних досить перетягнути його з панелі Toolbox в область Web-сторінки. При цьому він відображується у вигляді сірого прямокутника, що містить усередині найменування елемента управління джерелом даних.

6.5. Використання параметрів в запитах

У більшості реальних додатків роботи з даними неможливо застосовувати жорстко закодовані запити, оскільки залежно від використовуюваного режиму вони повинні містити різні дані. Для вирішення цього завдання задіюються параметри, застосування яких вже розглядалося в розділі "Використання команд, що параметризувалися". Для демонстрації цієї техніки спільно з елементами управління джерелами даних створимо форму, що відображує головну і підлеглу таблиці. Роль головної таблиці виконуватиме таблиця "Товари", роль підпорядкованою - "Закупівлі". При виборі певного товару на форму повинні виводитися тільки операції закупівлі даного товару.

Помістимо на форму елементи DropDownList і SqlDataSource. Набудуємо дані елементи на підключення до джерела даних, що містить список товарів. Визначення цих елементів виглядають таким чином:

```
<asp:DropDownList ID="DropDownList1" runat="server"
DataSourceID="SqlDataSource1"
DataTextField="НазваТовара" DataValueField="Кодтовара" Width = "196px"
AutoPostBack="True">
</asp:DropDownList>
```

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
ConnectionString="<%%$
ConnectionStrings:TEST_DBConnectionString %>"
SelectCommand="SELECT DISTINCT [Кодтовара], [Назватовара] FROM [Товари]">
</asp:SqlDataSource>
```

Слід звернути увагу на те, що властивість **AutoPostBack** елемента **DropDownList** встановлена рівним true. Це необхідно для забезпечення автоматичної ініціації зворотного відсилання при зміні поточного елемента списку, щоб сторінка змінювалася при виборі іншого елемента списку. Потрібно зробити так, щоб при виборі міста із списку вміст елемента **GridView**, що відображує закупівлі, оновлювався. Для реалізації даного механізму помістимо на форму елементи **GridView** і ще один - **SqlDataSource**. Набудуємо **SqlDataSource** відповідно до наступного визначення:

```
<asp:SqlDataSource ID="SqlDataSource2" runat="server" ConnectionString = "<%%$
ConnectionStrings: TEST_DBConnectionString%>"
SelectCommand = "SELECT Закупівлі.КодОперації, Закупівлі.ДатаОперації,
Контрагенти.НазваКонтрагента AS Контрагент, Закупівлі.Кількість, Закупівлі.Ціна,
Закупівлі.Кількість FROM Закупівлі INNER JOIN Контрагенти ON Закупівлі.КодКонтрагента =
Контрагенти. Кодконтрагента WHERE (Закупівлі.КодТовара = @Product)">
<SelectParameters>
<asp:ControlParameter ControlID="DropDownList1" Name="Product" PropertyName =
"SelectedValue" />
</SelectParameters>
</asp:SqlDataSource>
```

Ключовими елементами налаштування **SqlDataSource** є рядок запиту, визначений у властивості **SelectCommand**, і параметри, визначені в розділі **<SelectParameters>**. У рядку запиту визначений параметр **@Product**, який використовується для фільтрації списку закупівель за значенням коду товару. У розділі параметрів визначений один параметр, для якого встановлено ім'я **Product**. Значення для цього параметра витягується з властивості **SelectedValue** об'єкту **DropDownList1**.

У даному прикладі значення параметра **Product** підставляється автоматично з властивості **SelectedValue** елемента **DropDownList**.

6.6. Оновлення даних

SqlDataSource надає можливість зміни даних, що містяться в джерелах даних. Для того, щоб скористатися цією можливістю, необхідно визначити відповідні запити, що виконують операції вставки, оновлення і видалення записів. Всі вони створюються по аналогії із запитом Select на основі запиту, що параметризується. Вище вже згадувалося про наявних в SqlDataSource можливостях побудови запитів даного типу. Як приклад додамо можливість зміни даних в сторінку переглядання товарів і замовлень, створену раніше. Для цього виконаємо нижченаведені операції.

Визначимо в SqlDataSource запит на оновлення даних (UpdateQuery). Текст запиту виглядатиме таким чином:

UPDATE Закупівлі SET Датаоперациі = @ДатаОперации, Кількість = @Кількість, Ціна = @Ціна, Кодконтрагента = @Контрагент WHERE (Кодоперації = @КодОперации)

При складанні запиту важливо давати імена параметрам так, щоб вони збігалися з іменами стовпців таблиці. Це пояснюється тим, що ASP.NET автоматично відправляє значення параметрам, узятим із стовпців, у яких імена збігаються з іменами параметрів.

У даному прикладі присутня ще одна особливість, яку слід враховувати. Вона полягає в тому, що раніше для зручності відображення даних був використаний запит, в якому відбувається внутрішнє з'єднання даних з таблиць "Контрагенти" і "Закупівлі", - він необхідний для підміни номерів контрагентів їх найменуваннями. Тому при відображенні інформації в GridView ми бачимо найменування контрагентів замість їх номерів. Проте, це створить деякі труднощі при реалізації оновлення даних. Річ у тому, що тепер ми не можемо реалізувати оновлення даних контрагентів, оскільки вони знаходяться в таблиці "Контрагенти", а SqlDataSource пов'язаний з таблицею "Закупівлі". Все, що ми можемо зробити, - це змінити номер поточного контрагента в редагованій операції закупівлі. При цьому необхідно буде вводити не найменування контрагента, а його номер. Саме цим пояснюється наявність в запиті параметра Кодконтрагента = @Контрагент, оновлюючого і інформацію про контрагента поточної операції закупівлі.

Тема 7. Практична реалізація баз даних в інтернеті

7.1. Яким має бути ефективний Інтернет-магазин.

За помірну ціну в Інтернеті можна придбати найрізноманітніші товари, продажі в Інтернеті вигідні і для продавців, і для покупців. Але, на жаль, не всі інтернет-магазини «приречені» на успіх. І, як часто це буває, причина банальна – незнання розробниками і власниками елементарних правил.

Далі ми спробуємо розібратися, чи потрібний інтернет-магазини, яким чином можна збільшити кількість продажів, що впливає на остаточне рішення про покупку товару, чим керується покупець, вибираючи Інтернет-магазин, і чи зробить він покупку саме у вас...

Спершу дамо визначення Інтернет-магазину. *Інтернет-магазин – це багатофункціональний програмний модуль, вбудований в web-ресурс (web-сайт, інтернет-представництво), що забезпечує продажі за допомогою мережі Інтернет, що має електронний каталог для вибору товарів і послугу подальшій їх доставки покупцеві.*

7.1.2 . Послідовність роботи інтернет-магазину:

1. Надання клієнтові інформації про товар (послуги).
2. Отримання замовлення від клієнта (замовлення здійснюється через заповнення форми, процедуру реєстрації).
3. Отримання оплати (при використанні он-лайнних платіжних систем).
4. Відправка сплаченого товару.

7.1.3. Переваги і можливості Інтернет-магазину

Якщо ви – власник віртуального магазину:

1. Відсутність витрат на водопостачання, торгівельні площі, заробітну плату обслуговуючому персоналу, рекламні заходи.
2. Доступність магазину з будь-якої точки світу.
3. Заощадження часу: немає необхідності кожному покупцеві розповідати про товар, оскільки надається більший об'єм інформації про продукцію/послугу, необхідну для ухвалення рішення про покупку, ніж в традиційному магазині.
4. Можливість запропонувати значно більше товарів.
5. Ведення історії відвідин користувачами магазину, на підставі наявних даних можлива персоналізація підходу, автоматичні планування.

Якщо ви – покупець:

1. Ціни у віртуальних магазинах, частенько, набагато нижче, ніж в звичайних.
2. Мережеві магазини дуже зручні, відкриті цілу добу, можна розглядувати товар за декілька тижнів до покупки.
3. Можливість отримати докладну інформацію про товар/послугу не виходячи з будинку.
4. Великий асортимент товарів.
5. Заощадження часу – не потрібно кудись їхати за покупками.
6. Відсутність нав'язливого тиску з боку продавця.
7. Можливість поспілкуватися, запитати думки інших покупців, продавця.

7.1.4. Недоліки Інтернет-магазину

Якщо ви – власник віртуального магазину:

1. Оформлення «лжезамовників».
2. Загроза атаки «хакера» (якщо ви маєте базу користувачів з вказівкою номерів кредитних карток).

Якщо ви – покупець:

1. Немає можливості поторкати, помацати товар, протестувати його.
2. Очікування замовлення.

7.2. Конструювання інтернет-магазину

Так, щоб зробити ваш Інтернет-магазин ефективним, необхідно враховувати наступні чинники:

1. Зручність (чи продуманий інтерфейс вашого магазину, чи є можливість у користувача провести налаштування під себе).
2. Сервіс (наскільки зручна система оплати, чи є можливість повернути куплений товар і як швидко здійснюється доставка).
3. Інтерактивність (спілкування покупця з продавцем, спілкування покупців між собою, отримання консультацію он-лайн).
4. Інформативність (якомога докладніша і зрозуміліша інформація про товар, про сервіс, про послуги, бонуси, знижках і так далі).
5. Безпека (виключить вірогідність втрати даних; для цього упроваджується функція «резервне копіювання даних» і вибирається надійний хостинг).

Уявіть себе відвідувачем власного Інтернет-магазину і відповідайте на наступні питання:

1. Чи достатньо ясні і докладні описи приведені для представлених товарів, щоб ви не купили kota в мішку.
2. Наскільки нижче/вище ціни в порівнянні з іншими віртуальними магазинами.

Головна сторінка інтернет-магазину – найважливіша його складова. Вона має бути спроектована так, щоб відвідувач відразу розумів, що на сайті працює Інтернет-магазин. Відвідувач завжди повинен знати, де знайти відповіді на наступні питання:

1. Що у вас можна купити.
2. Які характеристики того, що ви хочете продати.
3. Як він може купити те, що ви хочете продати.
4. Як він отримає те, що купує.
5. Чому він повинен купити це саме у вас.
6. Чи надійна репутація компанії.
7. Чи зможе він повернути неякісний товар.

Звідси, щоб ефективність була стовідсотковою – на час потрібна стаття **Покупцем** власного Інтернет-магазину, і, проаналізувавши і критично оцінивши всі недоліки, намагатися внести виправлення. До покупця має бути особливий підхід, і в Інтернеті реалізувати його простіше, ніж в реальному магазині - зручна і зрозуміла навігація по сайту; відповіді, на питання, що виникають у покупця, розташовані, що називається, «під рукою» - рівноцінні хорошему продавцеві. А підсумком цієї роботи будуть збільшена кількість продажів і захоплені відгуки ваших покупців.

Для того, щоб теорія носила конкретніший характер, ми вибрали як мету - **створення простого інтернет-магазину**. Простого - в сенсі спрощеного. Це буде не повноцінне комерційне застосування, а всього лише ілюстрація інтернет-технологій з нашими поясненнями і коментарями.

Надалі ви зможете використовувати ці матеріали як основу для створення справжнього інтернет-магазину або для розробки інтернет-додатків інших типів. Для цього вам доведеться вирішити масу проблем, характерних для реальних інтернет-застосувань.

7.3. Технологія ASP

Програмний продукт **Microsoft Active Server Pages (ASP)**, назва якого в переказі означає Активні Серверні Сторінки, призначений для виконання серверної частки інтернет-додатків, що працюють під управлінням **MS Internet Information Server (IIS)**. ASP є виключно зручною технологією створення WEB-сторінок і дозволяє застосовувати будь-яку мову написання сценаріїв, що задовольняє стандарту ACTIVEX. У ASP використовується комбінація **HTML-тексту** і скриптових вставок на мові **VBScript** або **JavaScript**. IIS включає сервер автоматизації OLE, який виконує скрипти і посилає результати виконання в потрібному форматі клієнтові. Як правило, клієнтом виступає **браузер**, але можна використовувати і будь-який інший клієнт, який в стані адекватного обробляти відповіді сервера. Оскільки сценарії ASP виконуються на сервері, то з ASP-додатком можна працювати з будь-якого **WEB-браузера (MSIE, NC, FireFox, Opera і інші)**.

Для роботи з ASP вам необхідно встановити у себе **IIS**, якщо ви працюєте в середовищі Windows-NT/ 2000/XP. Для Windows-98/ME необхідно встановити Personal Web Server.

При установці IIS в Windows-NT/2000/XP, встановлюється стандартна конфігурація для веб-вузла. Це дуже добре, оскільки не доводиться відразу налаштовувати і конфігурувати служби IIS. Можна використовувати те, що є і присвятити свій дорогий час важливішим справам. За умовчанням файли і папки веб-вузла розташовуються за адресою **c:\inetpub\wwwroot**. Ви можете переписати туди створені вами сторінки, набрати в оглядачі http://ім'я_сервера/ім'я_сторінки і проглянути отриманий результат.

Більшість інтернет-додатків використовують для зберігання інформації бази даних. За допомогою таких компонентів як **ADO (об'єкт доступу до бази даних)** ASP-додатки легко можуть звертатися до сховища даних для створення, редагування або видалення даних.

7.4. Інструменти

Парадоксально, але в той час, як **Internet** відкрив двері до глобальної комунікаційної середовища мільйонам користувачів, основою мова розмітки гіпертексту (HTML) - колись простий набір легко читаних кодів - прийняв зрозумілу лише присвяченим форму. Колись практично весь авторський Web-інструментарій був представлений лише простим текстовим редактором (деяким екстремальним розробникам і сьогодні достатньо FAR або вбудованого в Windows редактора Notepad). Проте, факт - навіть найбільш досвідчені дизайнери сторінок і Web-майстра користуються спеціалізованими інструментами для редагування HTML.

Нове покоління Web-редакторів дозволить вам упевнено впоратися з будь-якими складнощами синтаксису мови HTML. Деякі редактори - це ультрасучасні засоби програмування, інші повністю звільняють автора від написання початкового тексту.

Наступне, що вам буде потрібно – це інструментарій для створення бази даних, який залежить від вибраного типу сховища даних. Ви можете використовувати **MS SQL Server** або **MDB-файли** (файли MS Access). Тоді як MS SQL Server має ряд істотних переваг перед MS Access, останній не такий дорогий і менш вимогливий до ресурсів системи. Тим паче, що з ряду причин інколи неможливо використовувати MS SQL Server. **Наприклад, якщо провайдер хостингу не підтримує бази даних MS SQL Server.**

7.5. Розміщення сайту в інтернеті

Інтернет-додаток потрібно розмістити в інтернеті. Комплекс послуг з розміщення сервера, забезпечення його роботи, організація з'єднання з Інтернетом називається «**хостингом**».

Залежно від ваших потреб і фінансових можливостей ви можете вибрати відповідний варіант хостингу.

Безкоштовний хостинг

Переваги:

- безкоштовний (хоча це швидше недолік);
- он-лайнова реєстрація (зазвичай на це йде декілька хвилин);
- дозволяє економити на початковій стадії проекту, особливо, коли потрібний тільки статичний сайт;
- не вимагає реєстрації свого доменного імені.

Недоліки:

- ім'я сайту, рідко буває легким для читання і таким, що добре запам'ятовується. Наприклад, таке: [www.irk.ru/~\[ваше_ім'я\]_ім'я.boom.ru](http://www.irk.ru/~[ваше_ім'я]_ім'я.boom.ru) (домен 3 рівня);
- обмежена функціональність;
- неконтрольоване включення на ваших сторінках чужої реклами (замовникові може не сподобатися реклама порноресурсів на своєму сайті);
- можлива повільна або нестійка робота системи.

Платний хостинг

Достоїнства:

- реєстрація домена другого рівня (можна підібрати зрозуміле ім'я, що запам'ятовується, для сайту);
- доступ до великого числа системних і прикладних бібліотек, що реалізують ряд корисних сервісних функцій;
- контрольовані вами сервіси сайту;
- можливість застосовувати майже всі передові технології.

Недоліки

- регулярні платежі, об'єм яких залежить від рівня сервісу, що надається.

При виборі варіанту хостингу і хостинг-провайдера переконайтеся в тому, що на сервері підтримується технологія ASP і забезпечується робота з SQL-сервером або MDB-файлами.

7.6. База даних інтернет-додатка

Інтернет-додаток використовує бази даних для зберігання інформації. Інформація може оброблятися, зберігатися і витягуватися з бази даних. У базі даних можна зберігати добре структуровану інформацію. Таку, як: список користувачів, список замовлень, прайси. Проте, цим вид інформації, що зберігається, не обмежується. У сучасних інформаційних системах в базах даних зберігаються і тексти, і зображення, і навіть виконувані програми або скрипти. Якщо буде потрібно, то в своєму застосуванні ви можете зберігати в базі даних інструкції і опису товарів.

Ви можете використовувати MS SQL Server або сховище у вигляді MDB-файлу Access. Надалі ми орієнтуватимемося на MS SQL Server. Для MDB-файлів можливі відмінності не принципові і полягають в незначних відмінностях в синтаксисі SQL-запитів і параметрах підключення до бази даних.

У цій статті ми визначимо мінімально необхідну структуру бази даних для нашого інтернет-магазину. Структура бази даних залежить від тих функціональних завдань, які повинен виконувати Інтернет-додаток.

Давайте спробуємо визначити функціональні завдання, що вирішуються нашим застосуванням.

Спочатку визначимо категорії користувачів, що працюють з магазином. Це можуть бути:

- анонімні відвідувачі (користувачі, які не реєструвалися при вході в магазин);
- покупці (користувачі, зареєстровані в магазині);
- менеджери (користувачі з розширеними правами).

Анонімні відвідувачі можуть "ходити" по магазину і "дивитися" товари.

Покупці можуть здійснювати покупки в магазині, але при цьому вони повинні надати деякий мінімум інформації про себе для того, щоб менеджери магазину могли уточнити і погоджувати з покупцем деталі по доставці покупки. Покупець може бути зареєстрованим (постійним) або разовим.

Разовий покупець може оформити разове замовлення.

Зареєстрований покупець може не лише оформляти замовлення, але і проглядати історію і стан своїх замовлень. Більш того, для постійного покупця заголовні реквізити замовлення заповнюються додатком автоматично (ім'я покупця, контактні реквізити).

Менеджери магазину повинні мати можливість додавання товарів в каталог (виставити товар на вітрину), редагувати назви і ціни товарів, проглядати списки замовлень і список зареєстрованих покупців.

Кожен магазин має **каталог товарів**, виставлених на продаж. У каталозі перераховуються назви товарів і їх ціни. Для нашого магазину цього вистачає. Проте в реальних електронних магазинах в каталозі зазвичай є описи товарів і їх зображення.

Всі покупки в магазині оформляються за допомогою **замовлень**. Замовлення - це електронний документ, в якому є дві частки: заголовок (дата замовлення, реквізити покупця) і вміст замовлення (перелік замовлених товарів і їх ціни).

Для кожного виду інформації створюється одна або декілька таблиць. Кожна таблиця складається з полів. Перелік, назви і типи полів визначаються призначенням таблиці.

Ми рекомендуємо в кожній таблиці використовувати спеціальні службові поля:

- **Id** - ідентифікатор запису (використовується в первинних ключах для адресації записів);
- **Actual** - ознака актуальності запису (використовується для розділення актуальних і неактуальних даних);

- **UpdateDate** - дата зміни запису (може використовуватися для сортування записів в хронологічному порядку);
- **UpdateUser** - ідентифікатор користувача, що створив або змінив запис (використовується для авторизації факту зміни запису).

Службовим полям доцільно дати імена однакові у всіх таблицях. В цьому випадку вам буде простіше надалі розбиратися з кодом додатка, супроводжувати і змінювати його.

Призначення поля, в загальному випадку, має бути зрозуміле з його назви. Наприклад, **Users** - таблиця користувачів, **Wares** – каталог товарів.

Хорошою практикою є використання спеціальних префіксів в назвах або використання назв, що складаються з декількох слів. При цьому перше слово повинне означати суть інформаційного об'єкту, а друге слово (або решта слів) - уточнення.

Наприклад, якщо **Ware** – товар, а **Cost** – ціна, то полю «**ціна товару**» можна привласнити ідентифікатор **WareCost**.

Звичайно, використання цих нехитрих правил, не є обов'язковим, але дозволяє значно полегшити вам життя при подальшій розробці і супроводі програмного продукту.

Таблиці

Wares - таблиця каталога товарів містить:

- службові поля (**Id**, **Actual**, **UpdateDate**, **UpdateUser**);
- **WareName** - назва товару;
- **WareCost** - вартість товару;
- **WareHouse** - ознака наявності або відсутності товару на складі.

Записи в каталозі товарів зберігаються вічно або, принаймні, до тих пір, поки є хоч одне замовлення, в якому присутній відповідний товар.

Якщо який-небудь товар потрібно прибрати з магазину, то для цього можна встановити в полі **Actual** ознаку неактуальності запису. Тоді, при відображенні каталогу покупцеві, можна виключати із списку такі товари.

Orders і **OrderRecords** - містять інформацію про замовлення. Дві таблиці потрібно для того, щоб розділити заголовки замовлень і списки замовлених товарів. В цьому випадку забезпечується мінімізація об'ємів бази даних і підвищується швидкість обробки інформації.

Orders – таблиця заголовків замовлень містить загальну інформацію про замовлення:

- службові поля (**Id**, **Actual**, **UpdateDate**, **UpdateUser**);
- **OrderDate** – дата замовлення;
- **ClientId** – ідентифікатор запису зареєстрованого користувача в таблиці **Users**;
- **ClientName** – контактна особа, з якою повинен зв'язатися менеджер при виконанні замовлення;
- **ClientAddress** – адреса, куди потрібно доставити замовлення;
- **ClientPhone** – телефон;
- **ClientEMail** – адреса електронної пошти;
- **OrderStatus** – поточний стан замовлення.

Призначення полів очевидне і не вимагає пояснень. Окрім поля **OrderStatus**. У цьому у полі можна зберігати код поточного стану замовлення. Наприклад: 0 – створюється, 3 – відправлений менеджером, 5 – обробляється, 7 – виконаний. В цьому випадку можна організувати подібність **document workflow** (процес документообігу). Менеджер знатиме чи можна починати обробку замовлення (раптом клієнт сьогодні не закінчив створення замовлення і вирішив продовжити цей процес завтра). З іншого боку, клієнт знатиме чи прийняв менеджер магазину замовлення для виконання.

Для анонімних покупців поле **ClientId** містить значення **NULL**.

Для зареєстрованих покупців поля **ClientName**, **ClientAddress**, **ClientPhone**, **ClientEMail** заповнюються системою автоматично на підставі вмісту реєстраційної картки покупця.

OrderRecords – таблиця вмісту замовлень:

- службові поля (**Id**, **Actual**, **UpdateDate**, **UpdateUser**);
- **OrderId** – ідентифікатор запису заголовка замовлення в таблиці **Orders**;
- **WareId** – ідентифікатор запису товарної позиції в таблиці **Wares**;
- **WareCost** – вартість товару;
- **WareCount** – кількість.

За допомогою поля **OrderId** забезпечується зв'язок між таблицями **Orders** і **OrderRecords**.

Поле **WareId** зв'язує запис замовлення з таблицею **Wares**.

Поле **WareCost** містить вартість одиниці даного товару у момент оформлення замовлення. Якщо менеджер в майбутньому поміняє вартість товару в каталозі, то на раніше оформлених замовленнях це не відіб'ється. Загальні суми по кожній позиції і за замовленням в цілому зберігатися не повинні, оскільки вони елементарно розраховуються в процесі відображення замовлення.

Users – таблиця користувачів зберігає інформацію про зареєстрованих покупців і менеджерів електронного магазину.

- службові поля (**Id**, **Actual**, **UpdateDate**, **UpdateUser**);
- **LogonName** – реєстраційне ім'я;
- **LogonPassword** – пароль;
- **ClientName** – ФІО користувача і назва компанії;
- **ClientAddress** – адреса;
- **ClientPhone** – телефон;
- **ClientEMail** – електронна адреса;
- **UserLevel** – тип користувача.

Реєстраційне ім'я і пароль потрібні для авторизації відвідувача при роботі з магазином.

Поле **UserLevel** використовується для визначення категорії користувача (наприклад: 1 – покупець, 3 – менеджер, і так далі). При авторизації користувача Інтернет-додаток може визначити вигляд і об'єм інформації, що надається користувачеві і вирішити або заборонити виконання деяких функцій. Наприклад, менеджерів магазину можна надати функції по редагуванню каталогу товарів і перегляданню загального списку замовлень.

У сучасному електронному магазині кількість таблиць і полів може бути істотно більше. Наприклад, можна товари згрупувати по категоріях, ввести поняття дилерської ціни, зберігати в каталозі опис товару і його зображення. Не зайвими в магазині будуть: інформація про новини магазину або вашої компанії, умови і порядок оформлення покупки, способи оплати і доставки товарів, сторінки зворотного зв'язку з покупцями (гостьова книга, форум, анкети). Можливо, регулярна публікація технічних описів і інструкцій з експлуатації товарів, що продаються вами, підвищить привабливість магазину для ваших клієнтів. Але ми не ускладнюватимемо структуру нашого застосування, щоб зосередитися на основних моментах побудови інтернет-додатка.

7.7. Створюємо інтерфейс інтернет-магазину

Далі ми приступимо до формування сторінок сайту. Для зручності користування необхідно "уніфікувати" сторінки сайту. Припустимо, що ліва частка сторінок міститиме локальні навігаційні елементи (наприклад, пошукова форма), верхня і нижня частка – загальне навігаційне меню сайту (розділи сайту), а центральна зона сторінки – головна інформаційна зона, в якій відображується специфічна для даної сторінки інформація. Основне меню сайту повинне містити опції швидкого переходу по сторінках сайту.

7.7.1. Короткий список основних сторінок сайту:

- початкова сторінка (default.asp);

- реєстраційна форма (regForm.asp);
- сторінка результатів пошуку товарів (search.asp);
- сторінка з поточним замовленням (order.asp);
- список замовлень (orderList.asp).

Додаткові сторінки (доступні тільки менеджерам):

- список користувачів (userList.asp);
- картка товару (wareCard.asp).

Функціональні модулі виконуються сервером і реалізуються у вигляді текстових файлів з розширенням ASP (ASP-файли або ASP-сторінки).

ASP-файл може містити команди підключення інших файлів, текст програми, HTML-коди.

Текст програми може бути написаний з використанням таких скриптових мов, як: JavaScript, VBScript, Perl. Принципової різниці при виборі мови немає. Це справа смаку. У наших прикладах ми використовуватимемо JavaScript (не плутати з Java!), оскільки ця мова дозволяє використовувати подібність технології об'єктно-орієнтованого програмування, включаючи спадкоємство, інкапсуляцію, поліморфізм.

При визначенні об'єктів прагніть їх описи розміщувати в окремих (include) файлах з відповідною назвою, це дозволить краще орієнтуватися в проекті і включати такі файли в ASP-сторінку в міру необхідності.

7.7.2. Реалізація каталогу товарів

Раніше ми розглядали ряд допоміжних функцій і описів об'єктів, використовуваних в інтернет-магазині.

Тепер розглядемо просту ASP-сторінку (search.asp), що забезпечує логіку роботи з каталогом товарів.

Для роботи з каталогом, буде потрібно мінімальний набір функцій:

- пошук і переглядання записів каталогу;
- редагування;
- додавання записів.

У реальному електронному магазині, набір функцій може бути значно ширше і включати:

- протоколювання операцій по зміні каталогу;
- обчислення індивідуальних цін на підставі знижок, встановлених для клієнта;
- публікація описів і зображень товарів.

При великій кількості товарів, не обійтися також і без функції синхронізації каталогу товарів магазину з номенклатурним довідником в офф-лайнній обліковій системі (1С, Галактика і так далі). Синхронізація, може виконуватися шляхом експорту номенклатурного довідника, з облікової системи і імпорту даних в каталог інтернет-магазину. У простому випадку можна обійтися і без автоматичної синхронізації.

Оскільки ми вже визначили ряд низько рівневих об'єктів і функцій, то початковий текст модуля для управління каталогом товарів буде досить простим.

7.7.3. Авторизація відвідувачів інтернет-магазину

Авторизація відвідувачів потрібна по двох основних причинах:

- надання відвідувачеві відповідних прав для зміни інформації на сайті;
- забезпечення можливості створення кошика замовлення для конкретного покупця.

При роботі інтернет-додатка для кожного відвідувача створюється спеціальний програмний об'єкт, який називається «**сесією**» (**Session**).

При першому зверненні до сайту для відвідувача створюється нова сесія. Все подальші звернення виконуються в рамках створеної сесії. Час життя сесії (**session timeout**) визначається налаштуваннями сервера і за умовчанням дорівнює 20 хвилинам. Протягом цього часу сервер зберігає всі дані сесійної змінної, після чого сесія знищується і виділена під сесію область пам'яті звільняється. Кожне звернення відвідувача до сервера, продовжує час на чергових 20 хвилин.

У кожен момент часу, число активних сесій відповідає числу активних відвідувачів за останніх 20 хвилин.

Ідентифікатор сесії це унікальний ключ, він створюється при першому зверненні до сторінки сервера і передається назад браузеру. При черговому запиті, браузер передає цей ключ серверу, а той у свою чергу перевіряє наявність сесії з таким ключем. Якщо така сесія існує, то вона, і всі збережені в ній значення, стають доступними серверному застосуванню.

Об'єкт **Session** може зберігати не лише унікальний ключ сесії, але і різні змінні. Наприклад, ідентифікатор користувача і корзину товарів, що дозволяє виключити необхідність передачі ідентифікатора і пароля користувача при кожному зверненні до сайту, забезпечуючи при цьому необхідний рівень інформаційної безпеки.

Механізм сесій в MS Internet Information Server (MS IIS) працює тільки за умови підтримки браузером відвідувача так званих **session cookies**. У налаштуваннях браузера можна заборонити підтримку session cookies, і тоді при кожному зверненні до сервера створюватиметься нова порожня сесія.

У переважній більшості випадків підтримка, session cookies у відвідувачів включена. Але інколи користувачі Інтернету, що начулися про проломи в системі безпеки браузерів, відключають цю підтримку. Такі проломи дозволяють недобросовісним сайтам прочитувати cookies, що залишилися в кеші браузера після відвідин інших сайтів (відмітимо, що не дуже технічно грамотних сайтів).

Предбачати чи ні можливість роботи відвідувача Вашого сайту без cookies – вирішувати Вам. Для нашого інтернет-застосування безпека cookies неістотна, оскільки ми не збираємося зберігати в cookies ні номери кредитних карт, ні ідентифікаторів, ні паролів. І вам не радимо.

Для критичніших до інформаційної безпеки застосувань можемо порекомендувати використовувати механізми «емуляції сесій». Наприклад, шляхом додавання до URL-адресу власного сесійного ключа. Ще безпечніше використовувати систему аутентифікації на базі цифрових сертифікатів. Проте, ці питання виходять за межі нашої статті.

Щоб зробити покупку в магазині, покупець може бути не реєстрованим користувачем (має на це право). І в цьому випадку описаний вище механізм чудово виконує свої завдання. Він дозволяє анонімному покупцеві проглядати каталог, формувати корзину замовлення, відправляти сформоване замовлення на обробку. При цьому всі необхідні для процесу параметри зберігаються в об'єкті Session.

Якщо покупець є зареєстрованим користувачем, то після авторизації система може автоматично заповнювати адресні поля замовлення і зберігати реєстр замовлень по кожному покупцеві. Погодитися – це зручно.

У нашому застосуванні для зберігання даних про зареєстрованих користувачів використовується таблиця Users. В процесі авторизації користувач повинен вказати свій ідентифікатор і пароль. Якщо в таблиці є запис з таким ідентифікатором і паролем, то система ідентифікує відвідувача і наділяє його необхідним рівнем прав на роботу з інформацією.

7.7.4. Оформлення замовлення в інтернет-магазині

Раніше нами розглянуті принципи побудови Інтернет додатка.

Які питання вже були розглянуті раніше?

1. Як швидко створювати нові ASP-сторінки з єдиним візуальним уявленням (об'єкт HTMLPage)?

2. Як підключатися до бази даних і працювати із списками записів (об'єкти OBJECTSQL, OBJECTDB)?
3. Як редагувати каталог товарів (модуль search.asp)?
4. Як управляти реєстраційними записами відвідувачів магазину і виконувати їх авторизацію (модулі logon.asp, regForm.asp, userList.asp)?

Далі ми зробимо останній крок і реалізуємо логіку роботи із замовленнями.

Нагадаємо про завдання:

Відвідувачі сайту, як анонімні так і зареєстровані, мають можливість проглядати каталог товарів і оформляти замовлення, які фіксуються в спеціальному журналі. Зареєстровані відвідувачі можуть проглядати журнал своїх замовлень. Крім того, для них автоматично заповнюються контактні реквізити замовлення на підставі даних в реєстраційній картці відвідувача.

Менеджери сайту володіють додатковими правами. Такими як: редагування каталогу товарів, переглядання загального журналу замовлень і списку зареєстрованих в магазині покупців.

7.7.5. Розробка інтернет-додатків

В процесі створення нашого невеликого навчального курсу був реалізований простою інтернет-магазин, який володіє багатьма необхідними для електронної комерції функціями.

Менеджер магазину може створювати каталог товарів, проглядати списки зареєстрованих покупців, відстежувати стан журналу замовлень.

Відвідувачі магазину, як анонімні, так і зареєстровані, можуть проглядати каталог товарів і оформляти замовлення.

Цей демонстраційний магазин цілком працездатний і функціональний. Його можна розміщувати для роботи в інтернеті на будь-якому сервері, на якому підтримуються MS ASP, JavaScript, ADO.

Питання WEB-дизайну ми не піднімали. Але ви можете самостійно виконати оформлення магазину або доручити цю роботу дизайнерові.

А зараз ми хотіли б звернути вашу увагу на питання, що залишилися неосвітленими в попередніх статтях:

- інтеграція електронного магазину з обліковою системою;
- оперативність обробки замовлень, що поступили;
- інформаційна безпека.

Питання **інтеграції з обліковою системою**, в якій ведеться товарний облік на вашому підприємстві, є дуже важливим в реальних електронних торговельних системах. Якщо кількість товарних позицій, виставлених в електронному каталозі, велике, то ручний супровід каталогу стає серйозною проблемою. При щоденній зміні цін, переліку товарів, наявності на складі доводиться витрачати значний час, щоб відновити каталог з 100 позицій. А якщо позицій 5000, то ручний супровід в принципі неможливий.

Інтеграцію з обліковою системою можна виконувати за допомогою операцій експорту/імпорту довідника товарів. Експорт довідника може відбуватися у файли різного формату. Добре, коли облікова система може підтримувати MDB або DBF формат для експорту. В цьому випадку процес імпорту значно спрощується. Можна виконувати експорт і в текстовий файл, але в цьому випадку процедура імпорту складніша і можуть виникнути проблеми, пов'язані з передачею бінарних значень (наприклад, зображень товарів).

При реалізації модуля імпорту товарів необхідно, щоб для кожного запису в каталозі зберігався код товару в обліковій системі. В цьому випадку при кожному імпорті створюватимуться нові записи тільки для нових товарів, що з'явилися на складі. А для раніше імпортованих товарів в записях оновлюватимуться дані про ціну, назву і наявність товару на складі. При цьому описи і зображення товарів, введені в ручному режимі, залишатимуться без змін.

Окрім інтеграції на рівні довідника товарів можливі складніші механізми. Наприклад, можна організувати автоматичну передачу оформлених замовлень в облікову систему для подальшої їх обробки в режимі **off-line**.

Для підвищення оперативності обробки замовлень, що поступили, в електронному магазині мають бути реалізовані **механізми сповіщення** менеджера магазину про нові замовлення.

Звичайне таке сповіщення реалізується за допомогою електронної пошти. Якщо замовлень не дуже багато, але час їх обробки грає важливу роль для вашого бізнесу, то можна реалізувати сповіщення за допомогою SMS-повідомлень на мобільний телефон.

Отримавши сповіщення про нове замовлення менеджер магазину може оперативно зв'язатися з покупцем (по телефону або електронній пошті) і уточнити з ним умови виконання замовлення.

Розробник інтернет-додатків повинен звернути особливу увагу на **інформаційну безпеку**. При цьому потрібно розуміти, що в поняття «Інформаційна безпека» входять не лише заходи щодо захисту даних від несанкціонованого доступу. Має бути реалізований цілий комплекс заходів, направлених на забезпечення стійкої, безпомилкової, безперебійної роботи додатка.

До заходів по забезпеченню інформаційної безпеки для інтернет-додатка можна віднести наступні дії:

1. Реалізація механізмів **авторизації доступу** до інформації. Такі механізми забезпечують комфортну роботу менеджерів при зміні даних. В той же час, вони виключають несанкціонований доступ ординарних відвідувачів до функцій і даних додатка.
2. Детальне **протоколювання подій**, що відбуваються на сайті. Таке протоколювання допомагає не лише в разі розбору нештатних ситуацій. Детальний протокол дозволяє виявити переваги відвідувачів. Корисною властивістю інтернет-додатка є автоматична реакція на підозрілі дії відвідувачів. Наприклад, в разі декількох послідовних спроб ввести неправильний пароль система може автоматично заблокувати всі запити, що поступають з відповідної IP-адреса.
3. Регулярне **резервне копіювання** даних для швидкого відновлення системи в разі краху.
4. При розробці програмного коду необхідно коректно ідентифікувати і обробляти всі **виняткові ситуації**. Особливу увагу необхідно приділити тим модулям, які приймають HTTP-запити від користувачів. Фільтрація, парсинг і аналіз параметрів запиту є обов'язковою умовою.
5. Необхідно вибрати «правильного» **сервіс-провайдера**, який забезпечить якісний рівень хостингу. Звичайно, можна встановити сервер для інтернет-магазину в своїй локальній мережі і підключитися до Інтернету за допомогою модему. Але в цьому випадку відвідувачів в магазині не буде. Вони просто не зможуть «достукатися» до вашого сервера.

Література

1. Спейнауэр С., Куэрсиа В. Справочник Web-мастера. - К: «ВНУ», 1997. – 368 с.
2. Яргер Р., Риз Дж., Кинг Т. MySQL и mSQL. Базы данных для небольших предприятий и Интернета. - СПб: Символ-Плюс, 2000 - 560 с.
3. Хилайер С., Мизик Д. Программирование Active Server Pages. - М: «Русская редакция», 1999. – 296 с.
4. Холзнер С. Perl: специальный справочник. – СПб: «Питер». 2000. – 496 с.
5. Шварц Р., Кристиансен Т. Изучаем Perl. - К: «ВНУ», 2000. – 320 с.
6. Ратшиллер Т., Геркен Т. PHP4: разработка Web-приложений. - СПб: Питер, 2001. - 384 с.
7. Томсон Л., Веллинг Л. Разработка Web-приложений на PHP и MySQL. - К.: "ДиаСофт", 2001. - 672 с.
8. Flanagan D. JavaScript. The Definitive Guide. – O'Reilly, 1997. - 420 стор.
9. Баррет Д. JavaScript. Web-профессионалам. — БХВ-Киев, 2001. — 240 с.
10. Бранденбау Дж. JavaScript: Сборник рецептов. — Питер, 2001. — 416 с.
11. Хейз, Д. Освой самостоятельно HTML и XHTML / Пер. с англ. - М.: Вильямс, 2002.-224 с.
12. Хоумер, А. Dynamic HTML. Справочник./ А.Хоумер, К. Улмен — СПб.: Питер, 1999. – 512 с.

Питання, що виносяться на перший модульний контроль

- 1 Назвіть інструменти для роботи з html-кодом.
- 2 Наведіть структуру html-документа.
- 3 Які теги визначають структуру html-документа?
- 4 Формат запису тегів.
- 5 Які правила формування назви сторінки?
6. Які основні елементи форматування тексту?
- 7 В чому відмінність тегів фізичного й логічного форматування?
8. Яких вимог необхідно дотримуватися при створенні html-документа у відповідності зі стандартом XHTML 1.0?
- 8 В чому особливість адресації в Інтернеті?
- 9 В чому різниця між відносними та абсолютними посиланнями?
- 10 Як додати посилання на елементи поточної сторінки?
- 11 Атрибути елемента `<img />`?
- 12 Які основні правила створення посилань?
- 13 Які класи мереж Ви знаєте?
- 14 З чого складається IP-адреса?
- 15 Для чого використовується доменна система імен (*DNS*)?

Питання, що виносяться на другий модульний контроль

1. Поняття CSS.
2. Способи підключення CSS.
3. Базовий синтаксис CSS.
4. Блокові елементи і їхні властивості.
5. Для чого використовуються форми?
6. Які атрибути тегу `<form>`?
7. Елементи створювані з допомогу тегу `<input/>` й їхні атрибути.
8. Елемент *textarea* та його атрибути.
9. Елемент *select* і його атрибути.
10. Яким чином можна регулювати зовнішній вигляд форми?
11. Який тег дозволяє включити *JavaScript* в HTML документ?
12. Перерахуйте основні конструкції мови *JavaScript*.
13. Розкрийте ієрархію об'єктів документа.
14. Перерахуйте основні властивості й методи об'єкта *window*.
15. Перерахуйте основні властивості й методи об'єкта *document*.
16. Які події використовують при обробці даних форми?
17. Що таке подія?
18. Як виконується доступ до властивостей форми?
19. Що зберігається в масиві *elements*?
20. Як одержати доступ до об'єктів форми?