НАВЧАЛЬНО-МЕТОДИЧНА ЛІТЕРАТУРА

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ
УНІВЕРСИТЕТ ІМЕНІ ІВАНА ПУЛЮЯ

Кафедра приладів і контрольно-вимірювальних системи

М Е Т О Д И Ч Н І В К А З І В К И
для виконання практичних робіт
з дисципліни

Основи інформаційних систем

для студентів спеціальності
176 «Мікро- та наносистемна техніка»

ТЕРНОПІЛЬ, 2023

Стрембіцький М.О. Методичні вказівки для виконання практичниї робіт з дисципліни «Основи інформаційних систем» для студентів спеціальності 176 – Мікро-та наносистемна техніка. Тернопіль: ТНТУ, 2023. 60 с.

Укладач: к.т.н., Стрембіцький М.О.

Відповідальний за випуск: завідувач кафедри приладів і контрольно-вимірювальних систем Паламар М.І.

Розглянуто та затверджено на засіданні приладів і контрольно-вимірювальних систем Тернопільського національного технічного університету імені Івана Пулюя, протокол № 7 від «1» травня 2023 р.

Схвалено та рекомендовано до друку науково-методичною комісією факультету прикладних інформаційних технологій та електроінженерії ТНТУ, протокол № 10 від «5» травня 2023 р.

ЗМІСТ

ПРАКТИЧНА РОБОТА № 1 СТВОРЕННЯ НОВОГО ASP.NET MVC ПРОЄКТУ. ДИНАМІЧНИЙ ВИВІД ДАНИХ

Мета: отримання навиків щодо створення нового проєкту ASP.NET MVC

Теоретичні відомості

Почнемо з створення нового MVC проекту Visual Studio. Виберіть *New Project* із меню *File*, щоб відкрити діалогове вікно *New Project*. Якщо ви оберете в розділі Visual C# шаблони Web, ви побачите, що одним із доступних типів проекту є *ASP.NET MVC 4 Web Application*. Виберіть цей тип проекту, як показано на рис. 1.1.

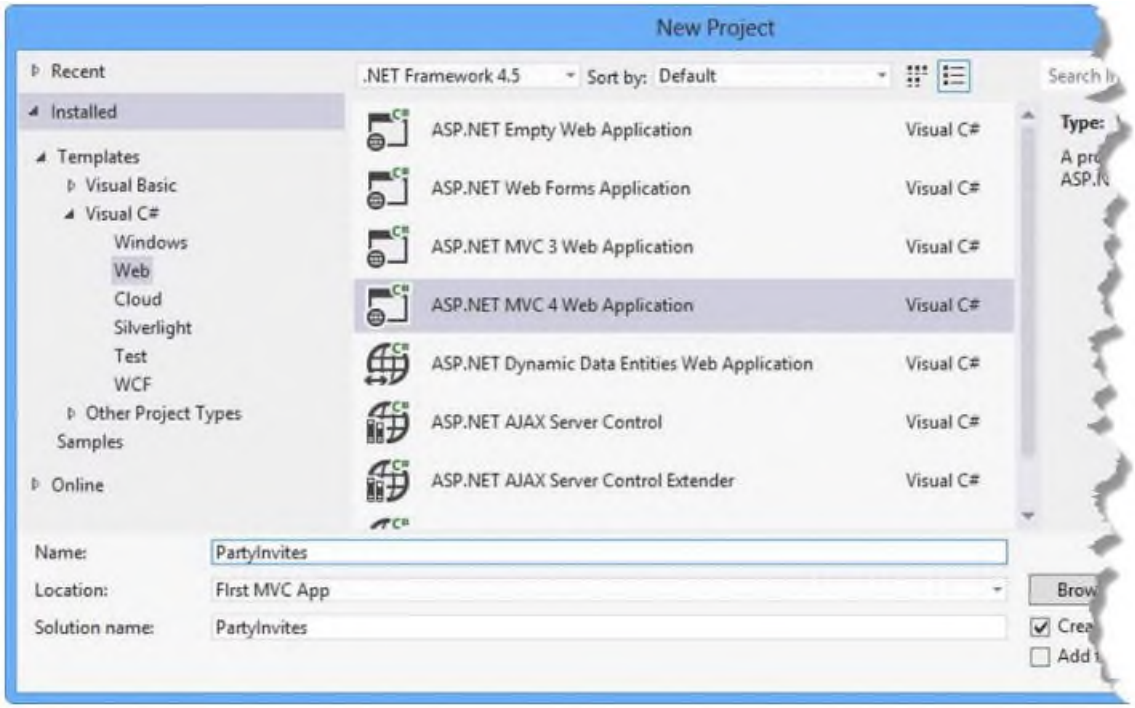


Рисунок 1.1 –Шаблон *Visual Studio MVC 4* проєкту

Увага

Visual Studio 2012 включає підтримку MVC 3, так само як і MVC 4, і ви бачите, що старі шаблони доступні разом із новими. При створенні нового проекту зверніть на цю увагу та виберіть правильний.

Назвіть новий проект *PartyInvites* та натисніть кнопку **ОК**, щоб продовжити. Ви побачите інше діалогове вікно, показане на рис.1.2, де вас попросять вибрати між трьома різними типами шаблонів MVC проєкту.

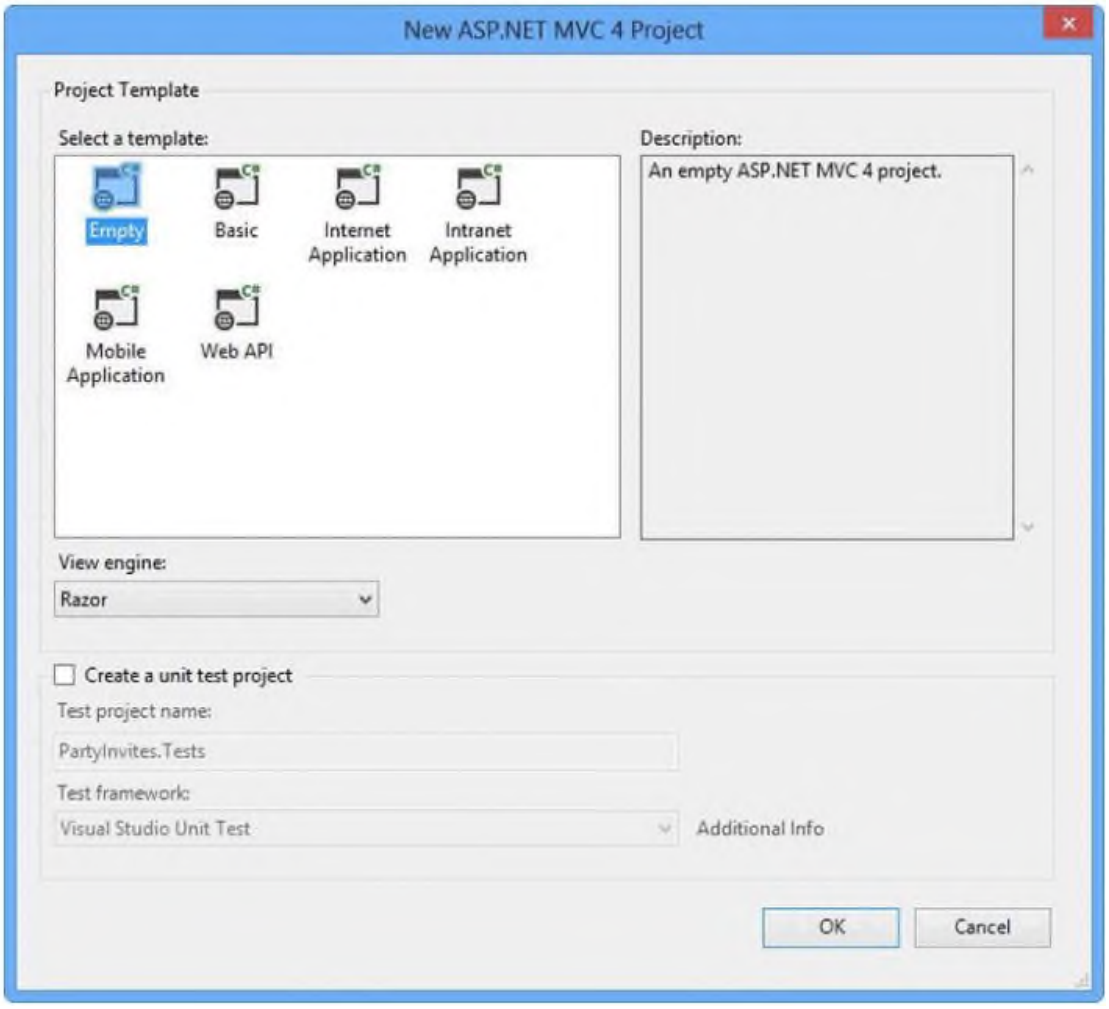


Рисунок 1.2 – Вибір типу MVC4 проєкта

Різні шаблони MVC проектів створюють проекти з різною базовою підтримкою таких функцій, як автентифікація, навігація та стилі. Ми не будемо все ускладнювати у цьому розділі. Виберіть варіант *Empty*, який створює проект із базовою структурою папок, але без файлів, необхідних для створення програм MVC. Ми будемо додавати файли, які нам знадобляться, в міру прочитання глави, і щоразу пояснюватимемо, що ми робимо.

Натисніть кнопку **OK**, щоб створити новий проект.

Примітка

На рис.2 ви можете побачити меню, що випадає, яке дозволяє вам вказати вид двигуна уявлення. У MVC 3 Microsoft представила новий та покращений вид движка, який називається ***Razor****, і ми будемо використовувати* ***Razor*** *у цій книзі. Ми рекомендуємо вам зробити те саме. Але якщо ви хочете використовувати стандартний вид ASP.NET движка (відомий як ASPX).*

Коли Visual Studio створить проект, ви побачите файли та папки, які відображаються у вікні *Solution Explorer*. Це стандартна структура MVC 4 проекти. Ви можете спробувати запустити програму, вибравши *Start Debugging* з меню *Debug* (якщо він попросить вас увімкнути налагодження, просто натисніть кнопку **OK**). Результат показаний рис. 1.3. Оскільки ми почали з порожнього шаблон проекту, програма нічого не містить, так що ми отримуємо помилку **404 Not Found**.

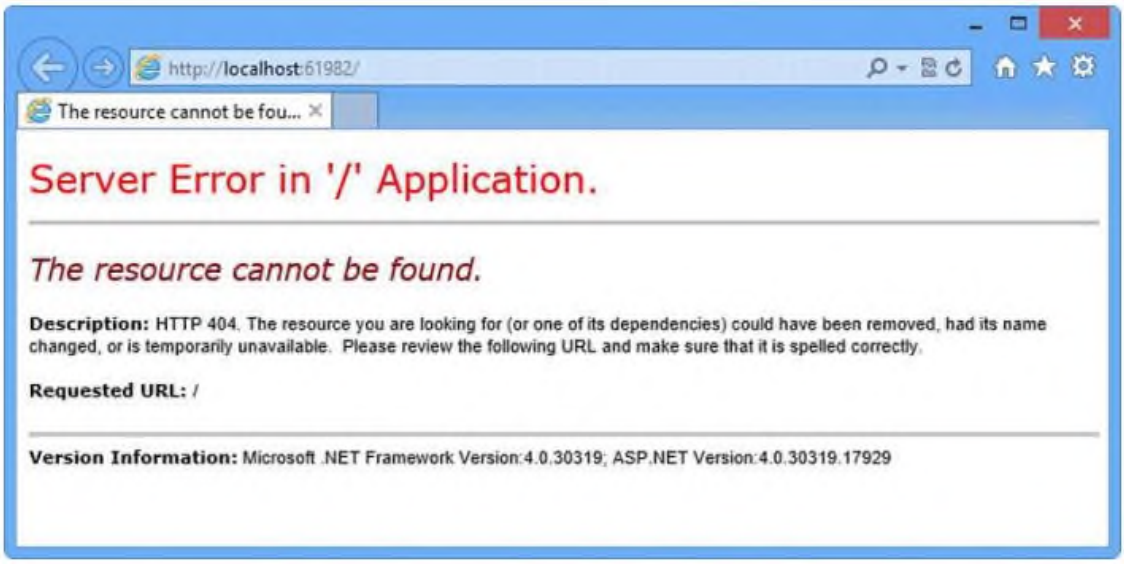


Рисунок 1.3 – Спроба запустити пустий проєкт

Коли ви закінчите, не забудьте зупинити відлагодку, закривши вікно браузера, який показує помилку, або поверніться до Visual Studio та виберіть *Stop Debugging* у меню *Debug*.

Visual Studio відкриває браузер для відображення проекту, і ви можете змінити браузер, який використовується в меню, показаному на рис. 1.4. Ви бачите, що тут представлені Microsoft Internet Explorer та Google Chrome

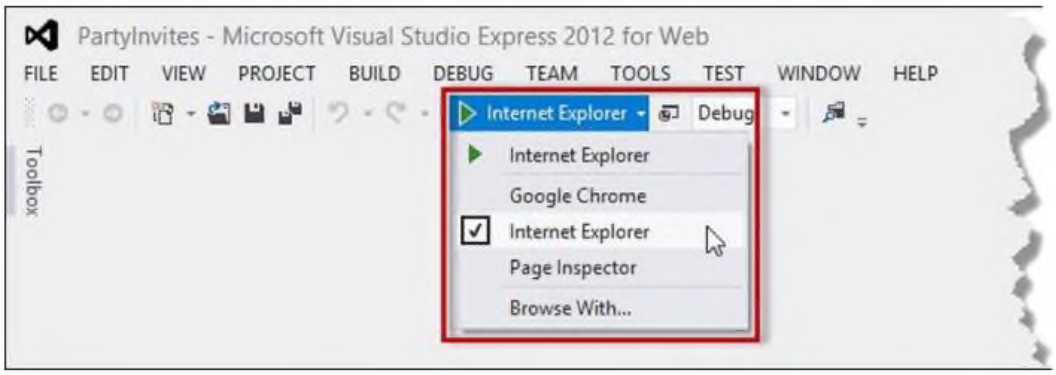


Рисунок 4 - Зміна браузера, який Visual Studio використовує для запуску проєкту

Додавання нового контролера

В архітектурі MVC вхідні запити обробляються *контролерами*. В ASP.NET MVC контролери є простими C# класами (як правило, успадковуються від *System.Web.Mvc.Controller*, вбудованих у фреймворк базових класів контролерів). Кожен відкритий метод у контролері відомий як *метод дії*, тобто ви можете викликати його з Інтернет через деякі URL, щоб виконати дію. У MVC контролери знаходяться в папці під назвою *Controllers*, яку Visual Studio створила для нас під час створення проекту. Вам не потрібно стежити за цим та більшістю інших погоджень MVC, але рекомендовано вияснити як це відбувається.

Щоб додати контролер до нашого проекту, просто клацніть правою кнопкою миші по папці Controllers у вікні Solution Explorer Visual Studio і потім виберіть Add у спливаючому меню, як показано на рис.1.5.

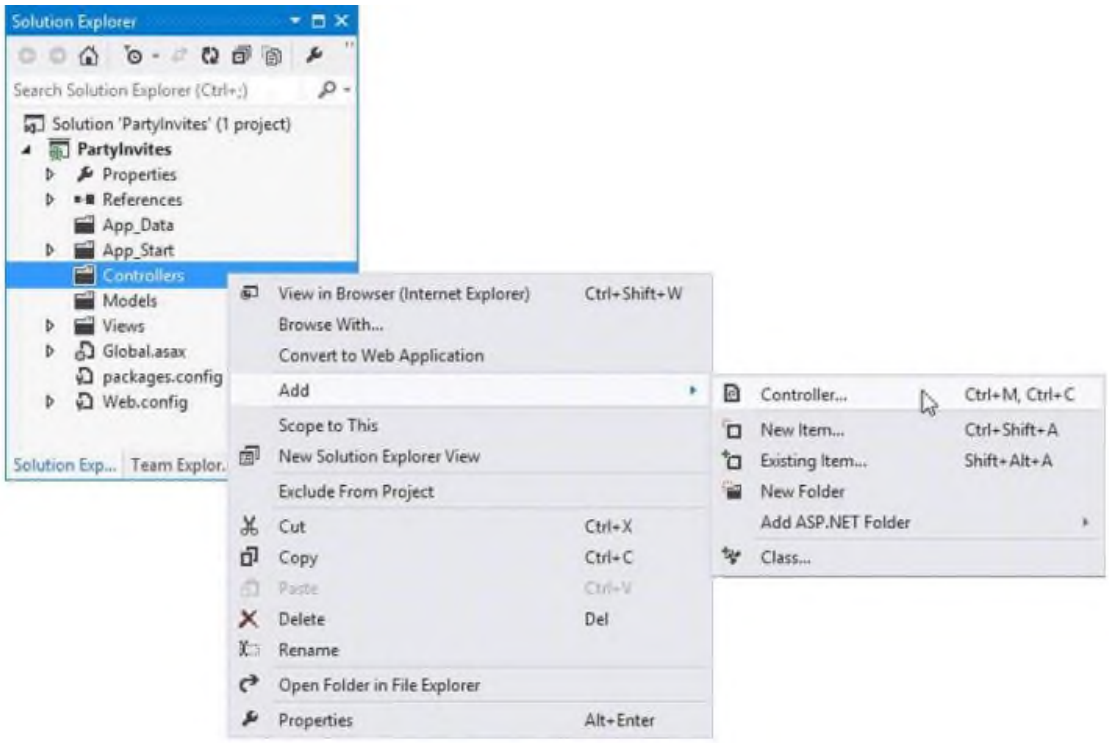


Рисунок 1.5 – Додавання контролера в MVC проєкт

Коли з'явиться діалогове вікно *Add Controller*, назвіть контролер *HomeController*, як показано на рис.1.6. Це ще одна угода: імена, які ми даємо контролерам, мають бути описовими та закінчуватися *Controller*.

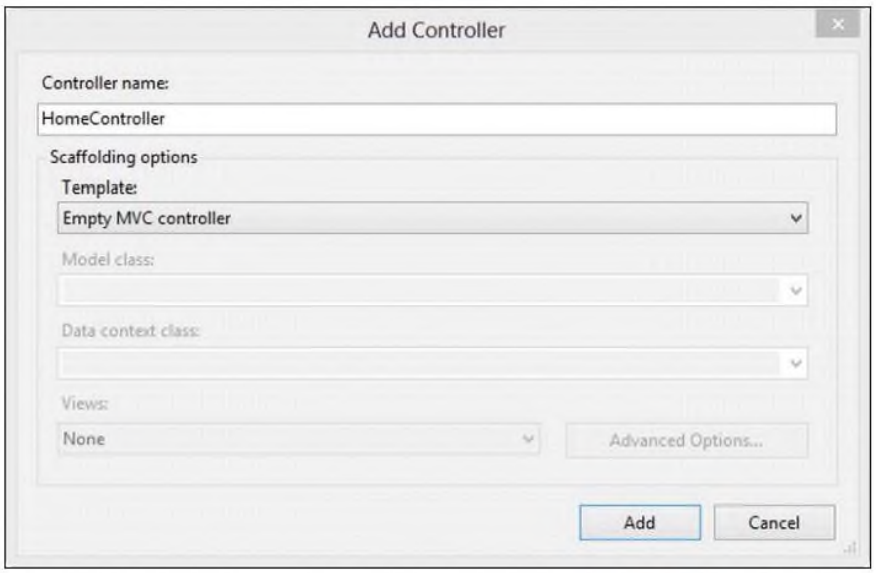


Рисунок 1.6 – Називаємо контролер

Розділ діалогового вікна *Scaffolding options* дозволяє нам створити контролер за допомогою шаблону із загальними функціями. Ми не збираємося використати цю можливість, тому переконайтеся, що в меню *Template* вибрано *Empty MVC controller*, як показано на рис.1.6.

Натисніть кнопку *Add*, щоб створити контролер. Visual Studio створить новий файл з C# кодом назвою *HomeController.cs* у папці *Controllers* та відкриє його для

редагування. Ми показали стандартний контент, який Visual Studio поміщає в класовий файл, в **лістингу 1**. Ви бачите, що клас називається *HomeController*, і він є похідним від *System.Web.Mvc.Controller*.

Лістиг 1: Текс класу *HomeController* за замовчування

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
namespace PartyInvites.Controllers
{
public class HomeController : Controller
                {
                public ActionResult Index()
                {
                                return View();
                        }
                }
}
```

Виконаємо декілька змін у класі контролера. Змінимо код файлу *HomeController.cs* так, щоб він відповідав коду лістингу 2. Виділено зміни, щоб їх було легше помітити.

Листинг 2: Изменение класса *HomeController*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using PartyInvites.Controllers
{
        public class HomeController : Controller
        {
                public string Index()
                {
                        return "Hello World";
                }
        }
}
```

Проведено зміни в методі дії (action method) *Index* в такий спосіб, що він повертає рядок *"Hello, world"*. Запустіть проект ще раз, вибравши *Start Debugging* у Visual Studio меню *Debug*. Браузер відобразить результат методу дії Index, як показано рис. 1.7.

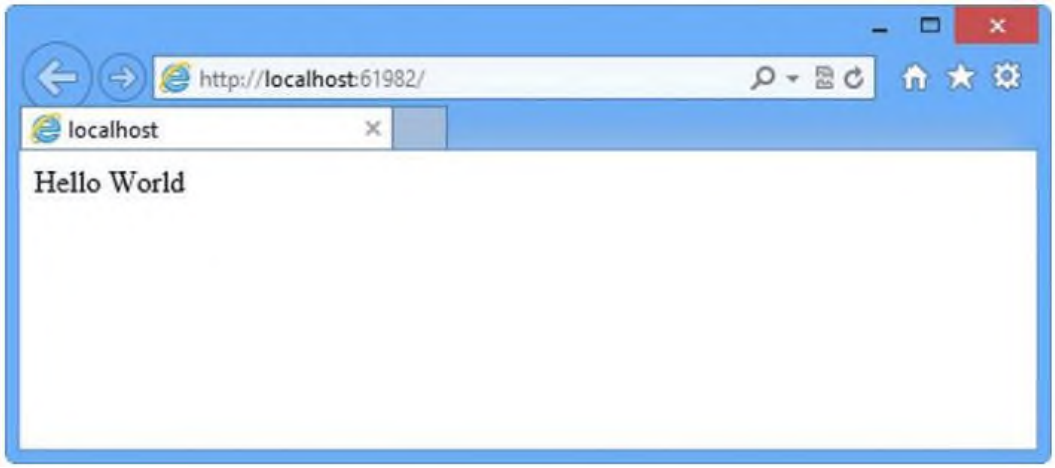


Рисунок 1.7 – Результат повернений методом контролера

Роути

Також як і моделі, уявлення та контролери, MVC програми використовують систему маршрутизації (роутингову систему) ASP.NET, яка вирішує, як URL-адреси картують конкретні контролери та дії. Коли Visual Studio створює MVC проект, вона спочатку додає деякі роути за замовчуванням. Ви можете запросити будь-яке з наступних посилань, та вони будуть спрямовані на *HomeController* метод *Index*:

- /
- /Home
- /Home/Index

Тому, коли браузер запитує *http://yoursite/* або *http://yoursite/Home*, він отримує вихідні дані *HomeController* методу *Index*. Ви можете спробувати зробити це самостійно, змінивши URL у браузері. На даний момент це буде *http://localhost:61982/*, за винятком того, що порт може бути іншим. Якщо додати до URL */Home* або */Home/Index* і оновити сторінку, ви побачите той же Hello World MVC програми.

Це добрий приклад користі від MVC угод. У даному випадку угода укладається те, що у нас є контролер *HomeController* і що він буде відправною точкою для нашого MVC програми. За замовчуванням роути, які Visual Studio створює для нового проекту, припускають, що ми слідуватимемо цій угоді. І оскільки ми дотримувалися угоди, ми отримали підтримку для URL-адрес з попереднього списку.

Якби ми не дотримувалися угоди, ми мали б змінити роути до точки до того контролеру, який ми створили натомість. Для цього простого прикладу конфігурація по замовчуванням це саме те, що нам потрібно.

Подання (рендерінг) веб-сторінок

Результатом попереднього прикладу не був HTML, це був просто рядок «*Hello World*». Щоб створити на запит браузера HTML відповідь, ми повинні створити представлення.

Створення та обробка представлення

Перше, що ми повинні зробити, це змінити наш метод Index, як показано у лістингу 2-3.

Лістинг 3: Зміна контролера для обробки представлення

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
namespace PartyInvites.Controllers
{
        public class HomeController : Controller
        {
                public ViewResult Index()
                {
                        return View();
                }
        }
}
```

Зміни у лістингу 3 виділені жирним шрифтом. Коли ми повертаємось до об'єкту *ViewResult* методу дії, ми доручаємо MVC зробити уявлення. Ми створюємо *ViewResult*, викликаючи метод *View* без параметрів. Це вказує MVC обробляти для методу дії подання за промовчанням.

Якщо ви зараз запустите програму, ви побачите, як MVC Framework намагається знайти потрібне подання за промовчанням, і це показано в повідомленні про помилку, яке представлене на рис. 1.8.

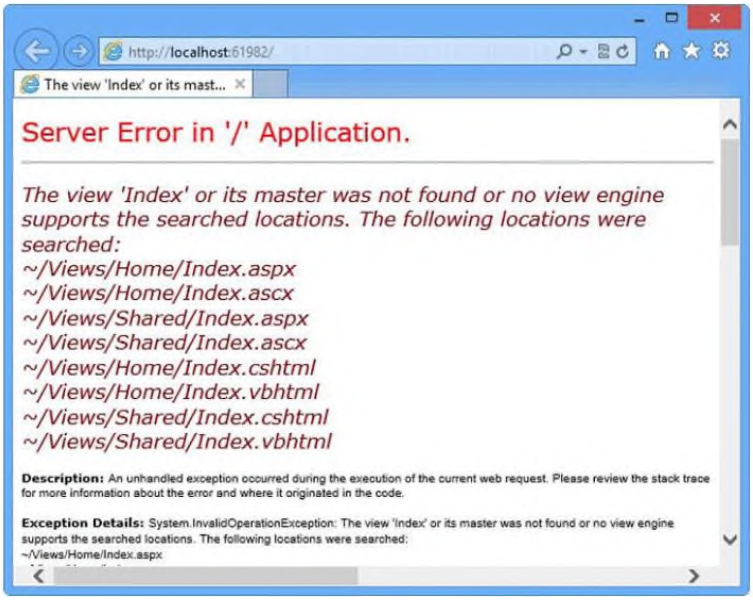


Рисунок 1.8 - MVC Framework намагається найти представлення по замовчуванні

Це повідомлення про помилку дуже корисне. Воно пояснює не лише те, що MVC не зміг знайти уявлення для нашого методу, але воно також показує, де шукав. Це ще один добрий Приклад MVC угоди: уявлення пов'язані з методами за допомогою імен.

Наш метод дії називається *Index*, а наш контролер називається *Home*, і ви можете побачити на рис. 8, що MVC намагається знайти різні файли в папці *Views* з таким ім'ям.

Щоб створити предстaлення, зупиніть відладчик і клацніть правою кнопкою миші за методом дії в кодовому файлі *HomeController.cs* (або за назвою методу або всередині тіла методу), а потім виберіть зі спливаючого меню *Add View*. Відкриється діалогове вікно *Add View*, яке показано на рис. 1.9.

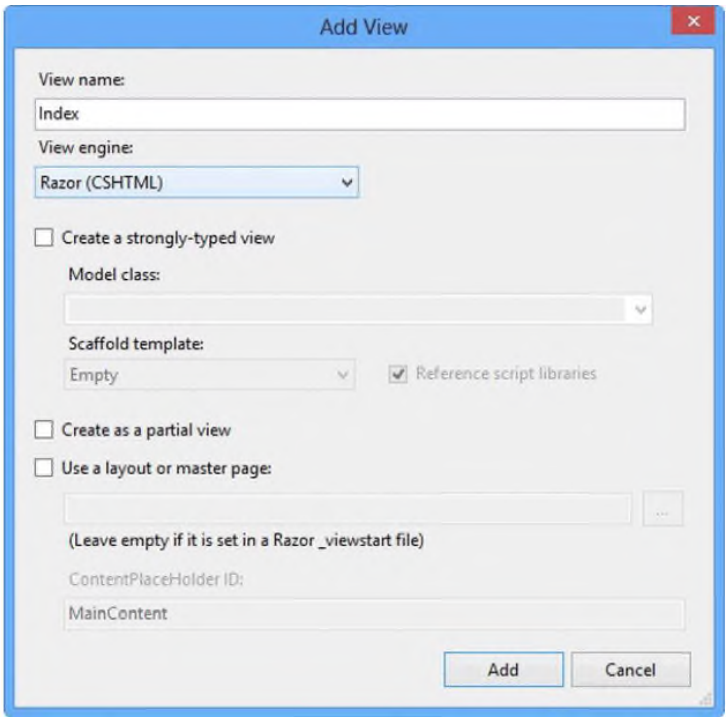


Рисунок 1.9 – Діалогове вікно *Add View*

Зніміть галочку з *Use a layout or master page*. У цьому прикладі ми не використовуємо макети, однак це буде розглядатися в подальшому. Натисніть кнопку *Add* і Visual Studio створить новий файл з ім'ям *Index.cshtml*, у папці *Views/Home*. Якщо ви подивитеся на повідомлення про помилку на рис. 1.8, ви побачите, що новий файл є одним із тих, що намагався знайти MVC.

Порада

Розширення файлу .cshtml означає C# уявлення, яке буде оброблятися Razor. Попередні версії MVC спиралися на движок уявлень ASPX, для якого файли уявлення мали розширення .aspx.

Visual Studio відкриває файл *Index.cshtml* для редагування. Ви бачите, що цей файл містить переважно HTML. Виняток становить лише та частина, яка має такий вигляд чином:

```
@{
 Layout = null;
}
```

Даний вираз буде інтерпретовано двигуном уявлення *Razor*. Це дуже простий приклад. Він просто каже *Razor*, що ми вирішили не використати майстер-сторінку. На цей момент ми збираємося проігнорувати *Razor* та повернутися до нього пізніше. Доповніть файл *Index.cshtml* тим виразом, який виділено жирним шрифтом у лістингу 4.

Листинг 2-4: Додавання представлення

```
@{
 Layout = null;
}
<!DOCTYPE html>
<html>
<head>
        <meta name="viewport" content="width=device-width" />
        <title>Index</title>
</head>
<body>
        <div>
                Hello World (from the view)
        </div>
</body>
</html>
```

З доповненням бачимо інше просте повідомлення. Виберіть *Start Debugging* у меню *Debug*, щоб запустити програму і перевірити наше уявлення. Ви повинні побачити щось схоже, що зображено рис. 1.10.

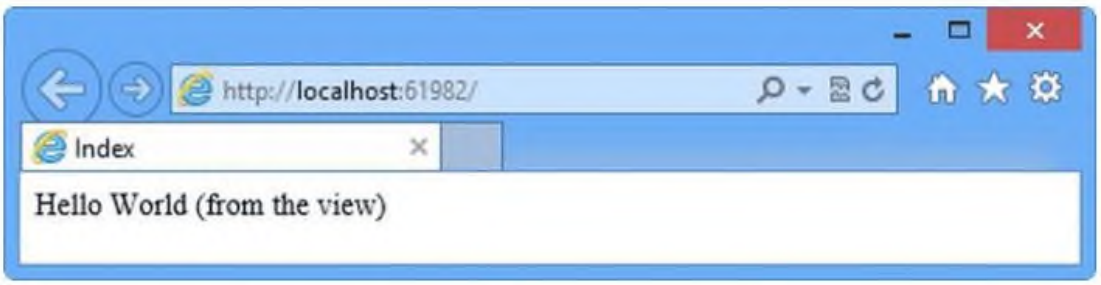


Рисунок 1.10 - Тестування представлення

Коли ми вперше редагували метод дії *Index*, він повернув рядкове значення. Це означало, що MVC не зробив нічого, крім як передав рядкове значення браузеру. Тепер, коли метод *Index* повертає *ViewResult*, ми доручаємо MVC обробити представлення та повернути HTML. Ми не говорили MVC, яке уявлення має бути використане, тому він використовував угоду за назвами, щоб знайти потрібне автоматично. Угода полягає в тому, що уявлення має назву методу дії і міститься в папці, названої після контролера: *~/Views/Home/Index.cshtml*.

Метод дії може повертати інші результати, крім рядків та об'єктів *ViewResult*. Наприклад, якщо ми повертаємо *RedirectResult*, ми змушуємо браузер перенаправитися на іншу адресу. Якщо ми повертаємо *HttpUnauthorizedResult*, ми змушуємо користувача увійти до систему (залогінитись). Ці об'єкти відомі як **результати дії**, і всі вони походять з класу *ActionResult*. Система результатів дій дозволяє нам інкапсулювати та повторно використовувати спільні відповіді певні дії.

Додавання динамічних вихідних даних

Весь сенс використання платформи для веб додатків полягає у побудові та відображення динамічних вихідних даних. У MVC це робота контролера – створити

деякі дані і передати їх уявленню, яке відповідає за те, щоб представити їх у вигляді HTML.

Одним із способів передачі даних від контролера до подання є використання об'єкт *ViewBag*, який є членом базового класу *Controller*. *ViewBag* – це динамічний об'єкт, якому можна привласнити довільні властивості, що робить ці значення доступні для будь-якого уявлення, яке буде з ними далі працювати. В лістингу 5 показана передача таким способом деяких простих динамічних даних у файл *HomeController.cs*.

Лістинг 5: Установка деяких даних представлення

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
namespace PartyInvites.Controllers
{
        public class HomeController : Controller
        {
                public ViewResult Index()
                {
                        int hour = DateTime.Now.Hour;
                        ViewBag.Greeting = hour < 12 ? "Good Morning" : "Good Afternoon";
                        return View();
                }
        }
}
```

Ми передали дані для представлення, коли ми надали значення властивості *ViewBag.Greeting.ViewBag* є прикладом динамічного об'єкта, а властивість *Greeting* не існувало доти, доки ми не надали йому значення. Це дозволяє передати дані з контролера на подання вільним і плавним чином, без необхідності достроково визначати класи.

Ми знову посилаємось на властивість *ViewBag.Greeting* у поданні, щоб отримати значення даних, як показано у лістингу 6, який демонструє зміни, що ми зробили у файлі *Index.cshtml*.

Лістинг 6: Отримання значення даних ViewBag

```
@{
        Layout = null;
}

<!DOCTYPE html>
<html>
<head>
```

```
        <meta name="viewport" content="width=device-width" />
        <title>Index</title>
</head>
<body>
        <div>
                @ViewBag.Greeting World (from the view)
        </div>
</body>
</html>
```

Доповненням у лістингу 6 є вираз **Razor**. Коли ми викликаємо метод *View* у методі контролера Index, фреймворк MVC знаходить файл представлення *Index.cshtml* і вимагає двигун **Razor** розібрати (відпарсувати) вміст файлу. **Razor** шукає вираз, як те, що ми додали у лістинг і обробляє його. У цьому прикладі обробка виразу означає вставку значення, яке ми надали властивості *ViewBag.Greeting* методу дії, в уявлення.

Там немає нічого особливого в імені властивості *Greeting*, ви можете замінити його будь-яким ім'ям властивості, і вона працюватиме так само. Крім того, ви можете передати кілька значень даних з контролера на подання шляхом присвоєння значень більш ніж одній властивості. Якщо ми запустимо проект, ми побачимо наші перші динамічні вихідні дані MVC як показано рис. 1.11.

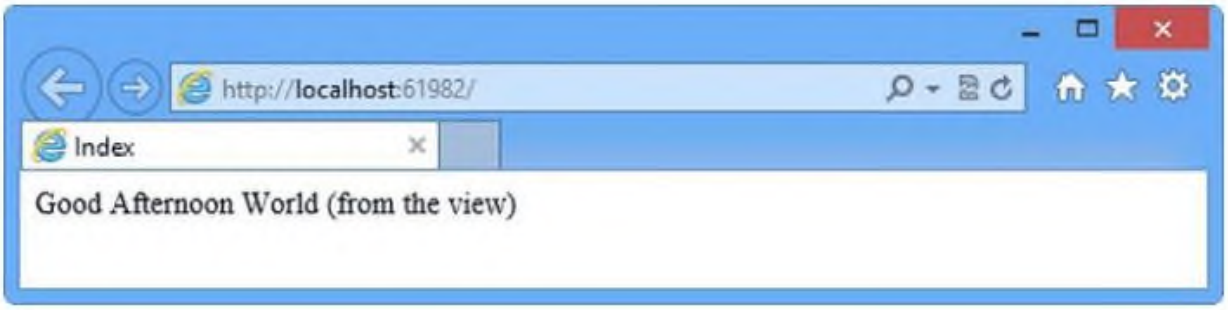


Рисунок 1.11- Динамічна відповідь MVC

ПРАКТИЧНА РОБОТА № 2 ДОДАТОК ДЛЯ ВВОДУ ДАНИХ

Мета: отримання навиків щодо створення елементів введення даних

Теоретичні відомості

Давайте уявімо, що Ви вирішила провести зустріч напередодні Нового року, і Вам створити веб сайт, який дозволить друзям та знайомим прийняти запрошення з RSVP (підпис на запрошення, що закликає одержувача дати відповідь на участь у заході).

На сайті має бути наступне:

- Головна сторінка, де відображається інформація про зустріч;
- Форма, яка може бути використана для RSVP.

У наступних заняттях ми будемо нарощувати MVC проект, який створили на початку практичних робіт, та додаємо ці можливості. Ми можемо зробити перший пункт зі списку, застосувавши ті знання, які ми отримали раніше, тобто ми можемо додати HTML для наших існуючих представлення, де буде надана докладна інформація про вечірку. У лістингу 1 показані доповнення, які ми зробили у файлі *Views/Home/Index.cshtml*.

Лістинг 1: Відображення інформації про зустріч

```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
    @ViewBag.Greeting World (from the view)
    <p>
        We're going to have an exciting party.<br />
        (To do: sell it better. Add pictures or something.)
    </p>
    </div>
</body>
</html>
```

Якщо ви запустите програму, ви побачите інформацію про вечірку, ну, вірніше, мітку-заповнювач (placeholder) для цієї інформації, але ви можете вловити суть. Приклад показаний на рис. 2.1.

Рисунок 2.1 – Додавання представлення

Проектування моделі даних

У MVC *M* позначає *модель*, і це найважливіша частина програми. Модель є поданням реальних об'єктів, процесів та правил, які визначають об'єкт, відомий як *домен*, нашого додатка. Модель, яка часто згадується як *доменна модель*, містить **C#** об'єкти (відомі як *доменні об'єкти*), які становлять суть нашого програми, та методи, які дозволяють нам маніпулювати ними. Подання та контролери розкривають домен клієнтам в узгодженому порядку, та добре продумане MVC, тому додаток починається з добре продуманої моделі, яка потім є координаційним центром, коли ми додаємо контролери та представлення.

Нам не потрібна складна модель для програми *PartyInvites*, але створимо один доменний клас, який назвемо *GuestResponse*. Цей об'єкт буде відповідати за зберігання, перевірку та підтвердження RSVP.

Додавання класу моделі

За MVC угодою класи, які складають модель, розміщуються в папці *Models*. Клацніть правою кнопкою миші по *Models* у вікні *Solution Explorer* і виберіть *Add*, за яким випливає *Class*, зі спливаючого меню. Назвемо файл *GuestResponse.cs* та натиснемо кнопку *Add*, щоб створити клас. Змінимо вміст класу відповідно до листингу 2.

Порада

Нам не потрібна складна модель для програми ***PartyInvites****, але ми створимо один доменний клас, який назвемо* ***GuestResponse****. Цей об'єкт буде відповідати за зберігання, перевірку та підтвердження RSVP.*

Листинг 2: Домений клас *GuestResponse*

```
namespace PartyInvites.Models
{
        public class GuestResponse
        {
                public string Name { get; set; }
                public string Email { get; set; }
                public string Phone { get; set; }
                public bool? WillAttend { get; set; }
        }
}
```

Порада

Ви, можливо, помітили, що властивість ***WillAttend*** *має тип* ***bool? (Nullable<bool>)****, тобто воно може бути* ***true****,* ***false*** *або* ***null****. Ми пояснимо це в розділі «Додавання валідації» далі в роботі.*

Силка на метод дії

Одна з цілей нашого додатку полягає у включенні RSVP форми, тому потрібно додати посилання на неї з нашої думки *Index.cshtml*, як показано в лістингу 3.

Листинг 3: Додавання силки для RSVP форми

```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
    @ViewBag.Greeting World (from the view)
    <p>
        We're going to have an exciting party.<br />
        (To do: sell it better. Add pictures or something.)
    </p>
    @Html.ActionLink("RSVP Now", "RsvpForm")
    </div>
</body>
</html>
```

Html.ActionLink є **допоміжним методом** HTML. MVC Framework розроблений з набором вбудованих допоміжних методів, які зручні для обробки HTML посилань, текстових вступних даних, прапорців, вибірок і навіть власних елементів управління. Метод *ActionLink* приймає два параметри: перший – це текст для відображення у засланні, а другий – це дія, коли користувач натискає на посилання. На рис. 2.2 показано посилання, яке додали.

Рисунок 2.2 – Додавання в представлення силки

Якщо наведете курсор на посилання в браузері, ви побачите, що посилання вказує на *http://yourserver/Home/RsvpForm*. Метод *Html.ActionLink* перевірив конфігурацію URL нашої програми і визначив, що */Home/RsvpForm* є URL для дії *RsvpForm* контролера *HomeController*. Зверніть увагу, що на відміну від традиційних програм ASP.NET, URL-

адреси MVC не відповідають фізичним файлам. Кожен метод дії має свій URL, і MVC використовує систему маршрутизації ASP.NET перевести ці URL на дії.

Створення методу дії

Якщо натиснете на посилання, побачите помилку *404 Not Found*. Це тому, що поки що ми не створили метод дії, який відповідає URL */Home/RsvpForm*. Ми зробимо це шляхом додавання методу *RsvpForm* до нашого класу *HomeController*, як показано в лістингу 4.

Лістинг 4: Додавання до контролера нового методу дії

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
namespace PartyInvites.Controllers
{
        public class HomeController : Controller
        {
                public ViewResult Index()
                {
                        int hour = DateTime.Now.Hour;
                        ViewBag.Greeting = hour < 12 ? "Good Morning" : "Good Afternoon";
                        return View();
                }
                public ViewResult RsvpForm()
                {
                        return View();
                }
        }
}
```

Додавання суворо типізованого уявлення

Ми хочемо додати уявлення для нашого методу дії *RsvpForm*, але ми збираємось зробити щось більше: тобто створити **строго типізоване** уявлення. Суворо типізовані уявлення призначені для обробки певного типу доменів. Якщо вкажемо тип, з яким ми хочемо працювати (у цьому прикладі *GuestResponse*), MVC надасть додаткові можливості, щоб спростити завдання.

Увага!

Переконайтеся, що ваш MVC проект скомпільований перед роботою. Якщо ви створили клас ***GuestResponse****, але не скомпілювали* ***його****, MVC не зможе створити суворо типизоване уявлення для цього типу. Щоб скомпілювати програму, виберіть* ***Build Solution*** *у Visual Studio меню Build.*

Клацніть правою кнопкою миші всередині методу дії *RsvpForm* і виберіть для створення уявлення *Add View* зі спливаючого меню. У діалоговому вікні *Add View* поставте галочку на *Create a strongly-typed view* і виберіть опцію *GuestResponse* з меню, що випадає. Зніміть прапорець з *Use a layout or master page* і переконайтеся, що Razor обраний як движка уявлення, і що опція *Scaffold template* встановлена на *Empty*, як показано на рис. 2.3.

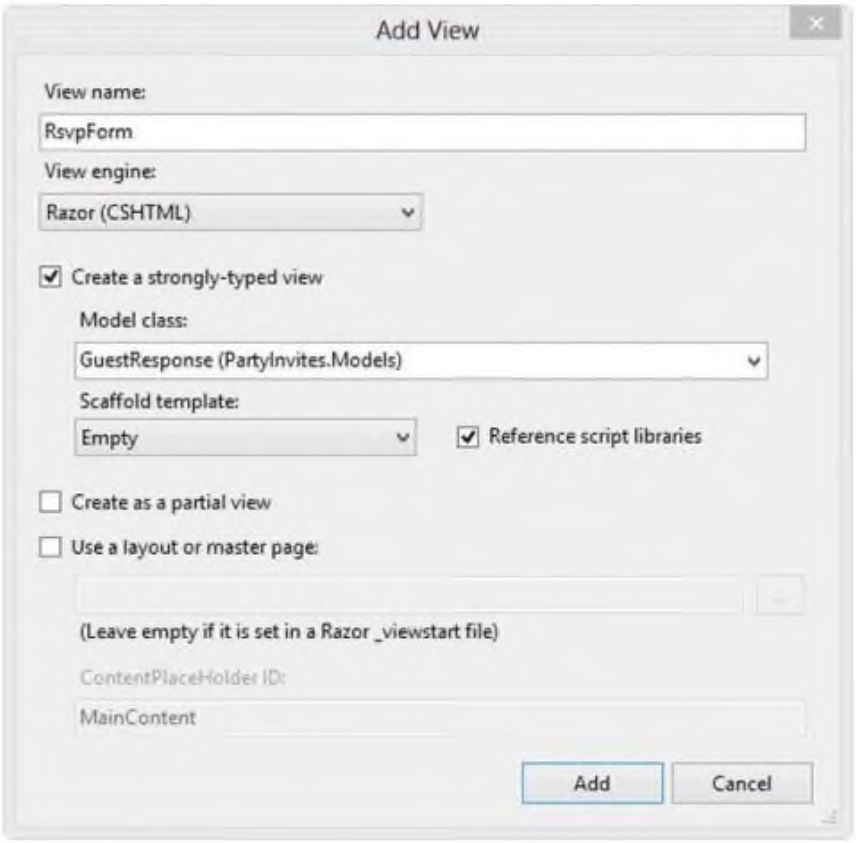


Рисунок 2.3 – Додавання строго типізованого представлення

Натисніть кнопку *Add* і Visual Studio створить новий файл з ім'ям RvspForm.cshtml і відкриє його для редагування. Ви можете побачити початковий зміст у лістингу 5. Як ви помітили, це інший HTML файл, але він містить Razor вираз **@model**. Ви зараз побачите, що це є ключем до строго типізованого уявлення та можливостей, які воно пропонує.

Лістинг 5: Початковий вміст файлу **RsvpForm.cshtm**

```
@model PartyInvites.Models.GuestResponse
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>RsvpForm</title>
</head>
<body>
    <div>
    </div>
</body>
</html>
```

Побудова форми

Тепер, коли ми створили строго типізоване уявлення, ми можемо побудувати зміст *RsvpForm.cshtml*, щоб перетворити його на HTML форму для редагування *GuestResponse* об'єктів. Змініть уявлення так, щоб воно відповідало лістингу 6.

Лістинг 6: Створення представлення форми

```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>RsvpForm</title>
</head>
<body>
    @using (Html.BeginForm())
    {
    <p>Your name: @Html.TextBoxFor(x => x.Name) </p>
    <p>Your email: @Html.TextBoxFor(x => x.Email)</p>
    <p>Your phone: @Html.TextBoxFor(x => x.Phone)</p>
    <p>
    Will you attend?
    @Html.DropDownListFor(x => x.WillAttend, new[] {
    new SelectListItem() {Text = "Yes, I'll be there", Value = bool.TrueString},
    new SelectListItem() {Text = "No, I can't come", Value = bool.FalseString}
    }, "Choose an option")
    </p>
    <input type="submit" value="Submit RSVP" />
    }
</body>
</html>
```

Для кожної властивості класу моделі *GuestResponse* ми використовуємо допоміжний метод

HTML, щоб обробити відповідний HTML елемент управління *input*. Ці методи дозволяють вибрати властивість, до якого належить елемент *input*, за допомогою лямбда-вираження, наприклад ось так:

```
@Html.TextBoxFor(x => x.Phone)
```

Допоміжний метод HTML *TextBoxFor* генерує HTML для елемента *input*, встановлює параметр *type* на *text* та встановлює атрибути *id* та *name* на *Phone*, ім'я вибраної властивості доменного класу, ось так:

```
<input id="Phone" name="Phone" type="text" value="" />
```

Як альтернатива лямбда-виразам ви можете звернутися до імені властивості типу моделі як до рядка, наприклад, ось так:

```
@Html.TextBox("Email")
```

Ми вважаємо, що методика лямбда-виражень допомагає нам не робити помилок у імені властивості типу моделі, тому що спливає Visual Studio IntelliSense і дозволяє нам вибрати властивість автоматично, як показано на рис. 2.4.

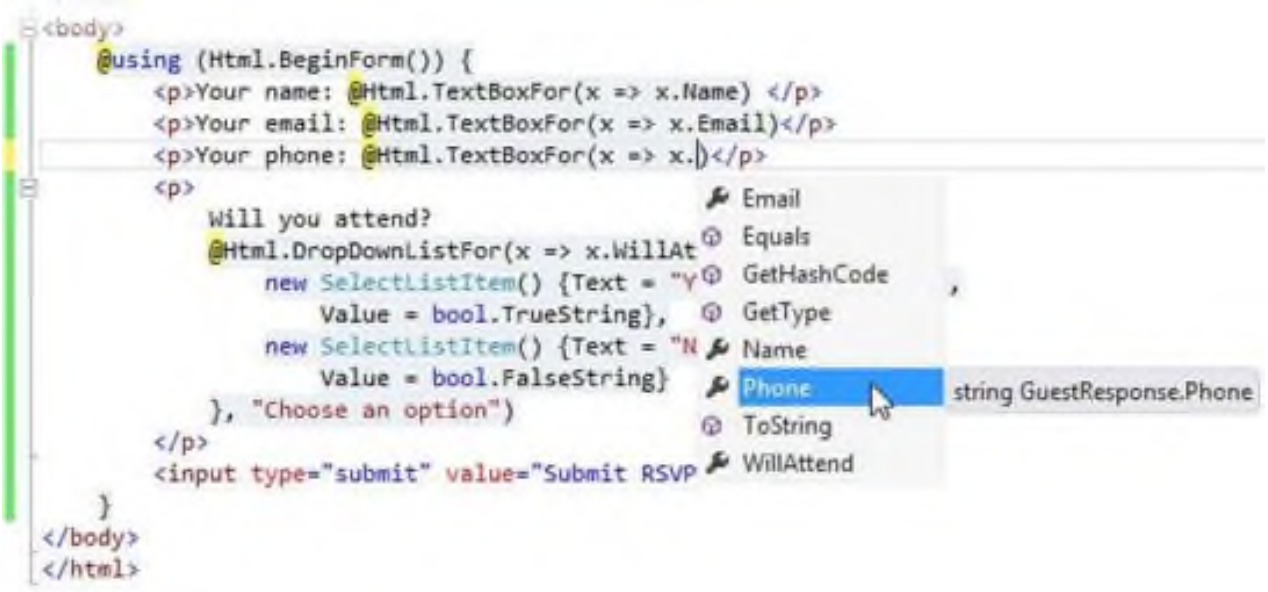


Рисунок 2.4 - Visual Studio IntelliSense для лямбда-виражень у допоміжних методах HTML

Іншим зручним допоміжним методом є *Html.BeginForm*, який генерує елемент HTML форми, налаштований зворотну передачу даних методу дії. Оскільки ми не передали допоміжному методу жодних параметрів, він припускає, що ми хочемо передати назад той самий URL. Спритним трюком є те, щоб обернути це в C# вираз *using*, ось таким чином:

@using (Html.BeginForm()) {
...form contents go here...
}

Зазвичай, коли воно застосовується таким чином, вираз *using* гарантує, що об'єкт видаляється, коли виходить із області видимості. Воно широко використовується для підключення до базі даних, наприклад, щоб переконатися, що вона закривається, як тільки запит було виконано. (Це застосування ключового слова *using* відрізняється від того, що стосується області видимості класу).

Замість видалення об'єкта, помічник *HtmlBeginForm* закриває HTML елемент *form*, коли він виходить із області видимості. Це означає, що допоміжний метод *Html.BeginForm* створює обидві частини елемента *form*, наприклад:

<form action="/Home/RsvpForm" method="post">
...form contents go here...
</form>

Не хвилюйтеся, якщо ви не знайомі з видаленням об'єктів C#. Наша мета зараз полягає в тому, щоб показати, як створити форму за допомогою допоміжного методу HTML. Ви можете бачити форму в поданні RsvpForm, коли ви запустите програму і натиснете посилання RSVP Now. На рис. 2.5 показаний результат.

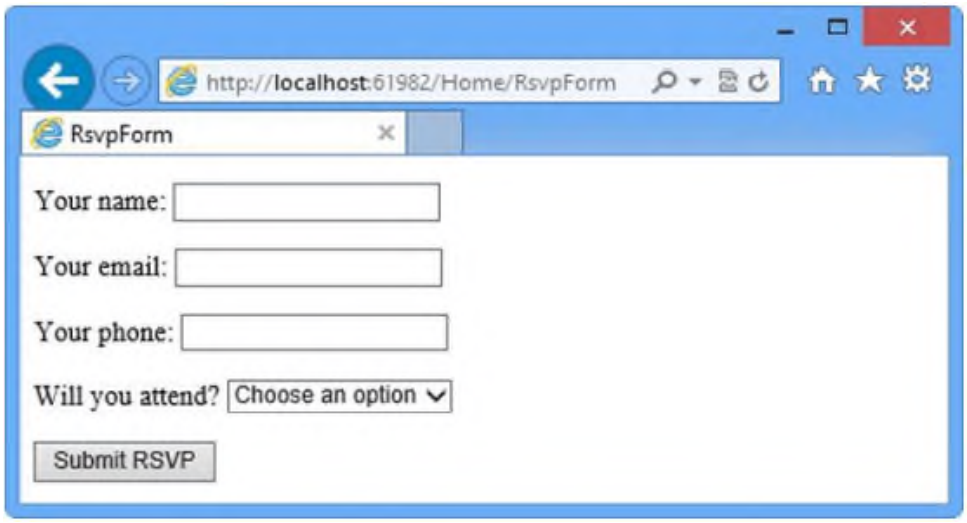


Рисунок 2.5 - Представления *RspvForm*

Примітка

*Даний курс не про **CSS** або веб дизайн. Здебільшого ми будемо створювати приклади, зовнішній вигляд яких може бути описаний як застарілий (хоча ми воліємо термін класичний, в якому відчуває менше зневаги). MVC уявлення генерують дуже чистий і простий **HTML**, та ви можете повністю керувати верстkою елементів та класів, до яких вони належать, тому у вас не буде проблем з використанням дизайнерських або готових шаблонів, щоб зробити ваш MVC проект красивим.*

Опрацювання форм

Ми не сказали MVC, що хочемо зробити, коли форма відправляється на сервер. На цей момент натискання на кнопку *Submit RSVP* просто видаляє будь-які значення, які ви ввели форму. Це тому, що форма відправляється назад методом дії *RsvpForm* у контролері *Home*, який просто каже MVC опрацювати представлення ще раз.

Примітка

*Ви можете бути здивовані тим фактом, що вхідні дані губляться, коли подання знову обробляється. Якщо це так, то ви, мабуть, розробляли програми за допомогою **ASP.NET Web Forms**, який автоматично зберігає дані у цій ситуації. В подальшому розглянемо випадок, як добитися такого ж результату з MVC.*

Щоб отримати та обробити надіслані дані форм виконаємо наступну дію: додамо другий метод дії *RsvpForm* для того, щоб створити:

- *Метод, який відповідає на **HTTP GET** запити*: **GET** запит є тим, з чим браузер має справу після кожного кліку за посиланням. Цей варіант дій буде відповідати за відображає початкову порожню форму, коли хтось перший раз відвідає */Home/RsvpForm*.
- *Метод, який відповідає на запити **HTTP POST***: За замовчуванням, форми, оброблювані за допомогою *Html.BeginForm()*, відправляються браузером як **POST** запити. Цей варіант дій буде відповідати за отримання відправлених даних та вирішувати, що з ними робити.

Опрацювання **GET** та **POST** запитів в окремих методах **C#** допомагає зберегти наш код охайним, оскільки обидва методи мають різні обов'язки. Обидва методи дії викликаються одним і тим ж URL, але MVC гарантує, що буде викликаний відповідний метод залежно від того, чи маємо ми справу з **GET** або з запитом **POST**. У лістингу 7 показані зміни, які ми повинні зробити у класі *HomeController*.

Лістинг 7: Додавання методу дії для підтримки запитів **POST**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using PartyInvites.Models;
namespace PartyInvites.Controllers
{
        public class HomeController : Controller
        {
                public ViewResult Index()
                {
                        int hour = DateTime.Now.Hour;
                        ViewBag.Greeting = hour < 12 ? "Good Morning" : "Good Afternoon";
                        return View();
                }

                [HttpGet]
                public ViewResult RsvpForm()
                {
                        return View();
                }

                [HttpPost]
                public ViewResult RsvpForm(GuestResponse guestResponse)
                {
                 // TODO: Email response to the party organizer
                        return View("Thanks", guestResponse);
                }
        }
}
```

Ми додали атрибут *HttpGet* для нашого існуючого *RsvpForm* методу дії. Це каже MVC, що цей метод повинен використовуватися лише для *GET* запитів. Потім ми додали перевантажену версію *RsvpForm*, який приймає параметр *GuestResponse* та застосовує атрибут *HttpPost*. Атрибут говорить MVC, що новий метод буде мати справу з *POST* запитами. Зверніть увагу, що ми також імпортували простір імен *PartyInvites.Models* – таким чином ми можемо звернутися до типу моделі *GuestResponse* без необхідності вказувати ім'я класу. Ми розповімо, як працюють наші доповнення у лістингу, у наступних темах.

Використання зв'язування даних моделі

Перший варіант перевантаженого методу дії *RsvpForm* обробляє те ж уявлення, що й раніше. Вона генерує форму, показану рис. 2.5. Другий варіант перевантаженого методу є більш цікавим через параметр, але, враховуючи, що метод дії буде викликатися у відповідь на HTTP *POST* запит, і що *GuestResponse* є типом класу **C#**, то як вони об'єднані?

Відповіддю є **зв'язування даних моделі** – надзвичайно корисна функціональна особливість MVC, коли вхідні дані розбиваються (парсяться) і пари ключ/значення в HTTP Запит використовується для заповнення властивостей типу доменної моделі. Цей процес є протилежністю використання допоміжних методів HTML; це коли при створенні даних форми для відправки клієнту, ми генерували HTML елементи *input*, де значення атрибутів *id* та *name* були отримані з назв властивостей класів моделей

На відміну від цього, із зв'язуванням даних моделі, імена елементів *input* використовуються для вказівки значень властивостей в екземплярі класу моделі, які потім передаються нашому методу дії з підтримкою *POST*.

Модель представлення даних є потужною та налаштовуваною функцією, яка позбавляє від рутини роботи безпосередньо з HTTP-запитами і дозволяє нам працювати з **C#** об'єктами, а не мати справу зі значеннями *Request.Form*[] та *Request.QueryString*[]. Об'єкт *GuestResponse*, який передається як параметр нашого методу дії, автоматично заповнюється даними з полів форми. Ми розглянемо докладно модель представлення даних, а також у тому числі як її можна налаштувати у наступних роботах.

Опрацювання інших представлень

Другий варіант перевантаженого методу дій *RsvpForm* також показує, як ми можемо вказати MVC обробляти конкретне подання, а не подання за промовчанням, у відповідь на запит. Ось відповідний вираз:

```
return View("Thanks", guestResponse);
```

Цей виклик методу *View* говорить MVC знайти та обробити уявлення, яке називається *Thanks*, і передати представленню наш об'єкт *GuestResponse*. Щоб створити уявлення, яке ми вказали, клацніть правою кнопкою миші всередині одного з методів *HomeController* та Виберіть *Add View* зі спливаючого меню. Встановіть ім'я подання на *Thanks*, як показано на рис. 2.6.

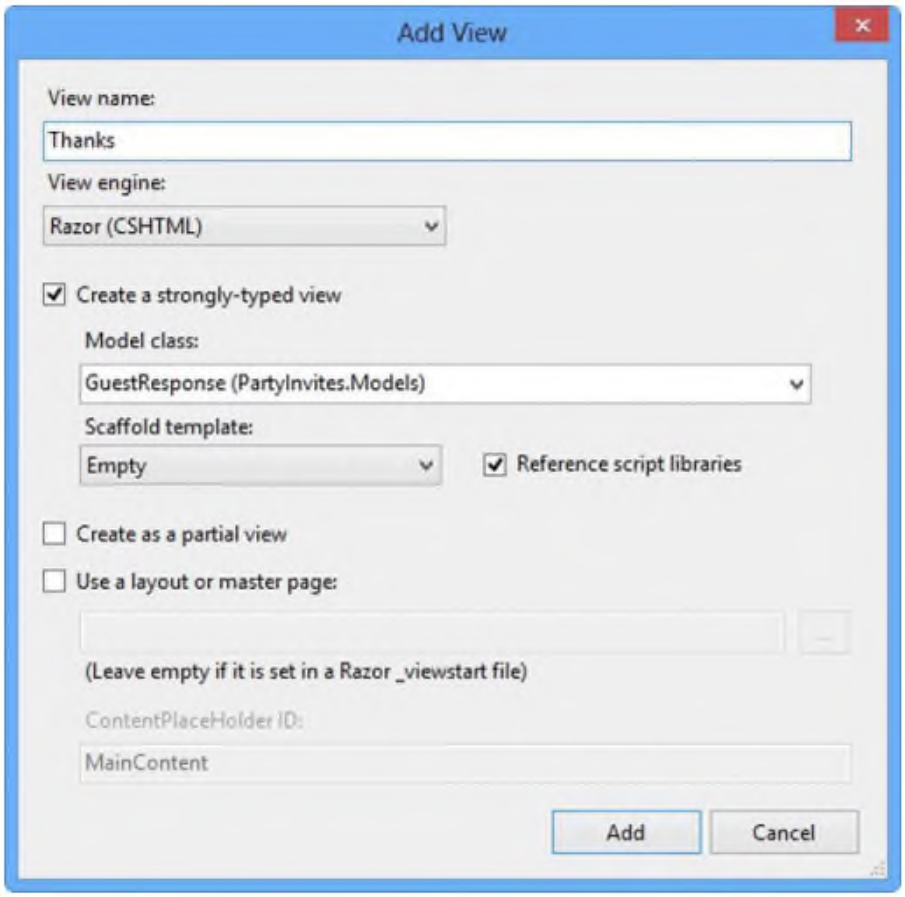


Рисунок 2.6 -Додавання представлення *Thank*

Створимо ще одне строго типізоване уявлення, тому поставте галочку на цьому пункті у діалоговому вікні *Add View*. Клас даних, який ми виберемо для цього уявлення, повинен відповідати класу, який ми передали уявленню при за допомогою методу *View*. Тому переконайтеся, що випадає меню *GuestResponse*. Переконайтеся, що немає галочки на *Use a layout or master page, View engine* встановлений на Razor, а також *Scaffold template* встановлено на *Empty*.

Натисніть кнопку *Add*, щоб створити нову виставу. Оскільки уявлення пов'язане з контролером *Home*, MVC створить уявлення як *~/Views/Home/Thanks.cshtml*. Змініть нове уявлення так, щоб воно відповідало лістингу 8 (курсивом виділено те, що потрібно додати).

Лістигн 8: Представлення *Thanks*

```
@model PartyInvites.Models.GuestResponse
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Thanks</title>
</head>
<body>
    <div>
        <h1>Thank you, @Model.Name!</h1>
        @if (Model.WillAttend == true)
        {
            @:It's great that you're coming. The drinks are already in the fridge!
        }
        else
        {
            @:Sorry to hear that you can't make it, but thanks for letting us know.
        }
    </div>
</body>
</html>
```

Представлення *Thanks* використовує Razor для відображення контенту на основі значення властивості *GuestResponse* ми передали методу *View* в методі дії *RsvpForm*. Razor оператор *@model* визначає тип доменної моделі, з якою пов'язане уявлення. Щоб отримати доступ до значення властивості доменного об'єкта ми використовуємо *Model.PropertyName*. Наприклад, щоб отримати значення властивості *Name*, ми викликаємо *Model.Name*. Синтаксис Razor детальніше буде пояснено в наступних роботах.

Тепер, коли ми створили уявлення *Thanks*, у нас є базовий робочий приклад обробки форми за допомогою MVC.

Запустіть програму в Visual Studio, натисніть на посилання *RSVP Now*, додайте у форму дані та натисніть кнопку *Submit RSVP*. Ви побачите результат, показаний на рис. 2.7 (хоча він може відрізнятися, якщо ваше ім'я не Джо, і ви сказали, що не зможете бути присутнім).

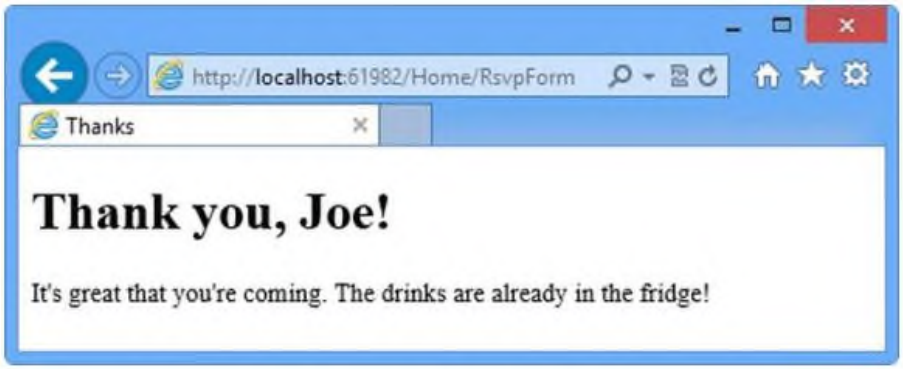


Рисунок 2.7 - Опрацювання представлення *Thanks*

ПРАКТИЧНА РОБОТА № 3 ВАЛІДАЦІЯ ФОРМИ RSVP»

Мета: отримання навиків щодо підтримки відповідного формату введених даних.

Теоретичні відомості

На сайті має бути наступне:

- Валідація форми RSVP, яка відображає сторінку з подякою;
- Заповнена та відправлена відповідь про згоду взяти участь у зустріч.

Додавання валідації

Зараз ми підійшли до того, щоб додати валідацію до нашої програми. Якби ми цього не зробимо, наші користувачі зможуть ввести безглузді дані або навіть надіслати порожню форму.

У додатку MVC перевірка, як правило, застосовується до доменної моделі, а не до інтерфейсу користувача. Це означає, що ми визначаємо наші критерії валідації у одному місці, і вони набирають чинності в будь-якому місці використовуваного класу моделі. ASP.NET MVC підтримує правила перевірки, що визначаються атрибутами простору імен *System.ComponentModel.DataAnnotations*. У лістингу 1 показано, як ці атрибути можуть бути застосовані до класу моделі *GuestResponse*.

Лістинг 1: Застосування валідації до класу моделі *GuestResponse*

```
using System.ComponentModel.DataAnnotations;
namespace PartyInvites.Models
{
    public class GuestResponse
    {
        [Required(ErrorMessage = "Please enter your name")]
        public string Name { get; set; }
        [Required(ErrorMessage = "Please enter your email address")]
        [RegularExpression(@".+\@.+\..+", ErrorMessage = "Please enter a
valid email address")]
        public string Email { get; set; }
        [Required(ErrorMessage = "Please enter your phone number")]
        public string Phone { get; set; }
        [Required(ErrorMessage = "Please specify whether you'll attend")]
        public bool? WillAttend { get; set; }
    }
}
```

Правила валідації виділені жирним шрифтом. MVC автоматично виявляє атрибути та використовує їх для перевірки даних під час представлення моделі даних. Зверніть увагу, що ми імпортували простір імен, що містить правила валідації, тому ми можемо посилатися ними без необхідності вказувати їх імена.

Порада

*Як зазначалося раніше, ми використали тип **bool**? для властивості **WillAttend**, тому ми змогли застосувати атрибут валідації **Required**. Якби ми використовували звичайний **bool**, значення, яке ми могли б отримати завдяки моделі представлення даних, може бути тільки **true** або **false**, і ми не були б сказати, чи вибрав користувач значення. Тип **bool**? має три можливі значення: **true**, **false** і **null**. Значення **null** буде використано, якщо користувач не вибрав значення, і це змушує атрибут **Required** повідомити про помилку валідації.*

Ми можемо перевірити, чи помилка валідації була перевіркою за допомогою властивості *ModelState.IsValid* у нашому класі контролера. У лістингу 2 показано, як це можна зробити у нашому методі дії *RsvpForm* за допомогою **POST**.

Лістинг 2: Перевірка на наявність помилок під час валідації форми

```
[HttpPost]
public ViewResult RsvpForm(GuestResponse guestResponse)
{
    if (ModelState.IsValid)
    {
        // TODO: Email response to the party organizer
        return View("Thanks", guestResponse);
    }
    else
    {
        // there is a validation error
        return View();
    }
}
```

Якщо помилки валідації немає, ми говоримо MVC обробляти уявлення *Thanks*, як ми робили раніше. Якщо помилка валідації є, ми знову опрацьовуємо уявлення *RsvpForm*, викликавши метод *View* без параметрів.

Просто відображати форму, коли є помилка, не дуже корисно, ми маємо дати користувачеві деяку інформацію про те, в чому полягає проблема, і чому ми не можемо прийняти її форму. Ми робимо це за допомогою допоміжного методу *Html.ValidationSummary* в поданні *RsvpForm*, як показано в лістингу 3.

Лістинг 3: Використання допоміжного методу *Html.ValidationSummary*

```
@model PartyInvites.Models.GuestResponse
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>RsvpForm</title>
</head>
<body>
    @using (Html.BeginForm())
    {
        @Html.ValidationSummary()
        <p>Your name: @Html.TextBoxFor(x => x.Name) </p>
        <p>Your email: @Html.TextBoxFor(x => x.Email)</p>
        <p>Your phone: @Html.TextBoxFor(x => x.Phone)</p>
        <p>
        Will you attend?
        @Html.DropDownListFor(x => x.WillAttend, new[] {
```

```
                new SelectListItem() {Text = "Yes, I'll be there", Value =
            bool.TrueString},
                new SelectListItem() {Text = "No, I can't come", Value =
            bool.FalseString}
                }, "Choose an option")
            </p>
            <input type="submit" value="Submit RSVP" />
        }
</body>
</html>
```

Якщо помилок немає, метод *Html.ValidationSummary* створює прихований елемент списку як заповнювача у формі. MVC робить мітку-заповнювач видимою та додає повідомлення про помилку, обумовлені атрибутами валідації. Ви можете побачити, як це виглядає на рис. 3.1.

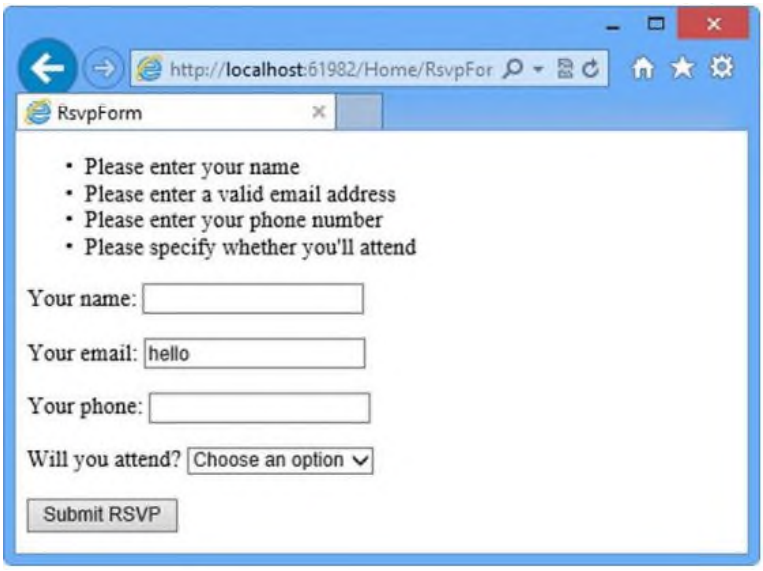


Рисунок 3.1 - Зведені результати валідації

Користувачеві не буде показано подання *Thanks*, доки не будуть задоволені всі правила валідації, які ми використали до класу *GuestResponse*. Зверніть увагу, що дані, введені у форму, були збережені і відображаються знову, коли подання показано з зведенням валідації. Це ще одна перевага зв'язування даних.

Примітка

У Web Forms існує поняття «серверні елементи управління», що зберігають стан, серіалізуючи значення у прихованому полі форми ***__VIEWSTATE****. Зв'язування даних ASP.NET MVC не прив'язано до Web Forms концепції серверних елементів управління, зворотного передачі даних або View State. ASP.NET MVC не вводить приховане поле* ***__VIEWSTATE*** *в оброблювані HTML сторінки.*

Виділення невалідних полів

Допоміжні методи HTML, які створюють текстові поля, списки та інші елементи мають дуже зручну функцію, яка може бути використана в поєднанні з зв'язуванням даних. Той самий механізм, який зберігає дані, введені користувачем у форму, також може бути використаний для виділення окремих полів, які не пройшли валідацію.

Якщо якість класу моделі не пройшла валідацію, допоміжні методи HTML будуть генерувати трохи інший HTML. Як приклад, ось HTML, який генерує виклик *Html.TextBoxFor (x => x.Name)*, коли немає помилки валідації:

<input data-val="true" data-val-required="Please enter your name" id="Name" name="Name" type="text" value="" />

В HTML, який генерує той самий виклик, коли користувач не надав значення (що є помилкою валідації, тому що ми застосували атрибут *Required* властивості *Name* у класі моделі *GuestResponse*):

<input **class="input-validation-error"** data-val="true" data-val-required="Please enter your name" id="Name" name="Name" type="text" value="" />

Ми виділили різницю. Це допоміжний метод додав клас з ім'ям *input-validation error*. Ми можемо скористатися цією функцією, створивши таблицю стилів CSS, що містить стилі для цього класу та інших, що застосовують різноманітні допоміжні методи HTML.

Угода в MVC проектах полягає в тому, що статичний контент, такий як таблиці стилів CSS, розміщується в папці під назвою *Content*. Ми створили папку *Content*, натиснувши правою кнопкою миші за проектом *PartyInvites* у Solution Explorer і вибравши зі спливаючого меню *Add New Folder*. Ми створили таблицю стилів, клацнувши правою кнопкою миші по папці *Content*, вибравши *Add New Item* і потім обравши *Style Sheet* у діалоговому вікні *Add New Item*. Ми назвали нашу таблицю стилів *Site.css*, і це ім'я, яке Visual Studio використовує під час створення проект із використанням іншого шаблону MVC, а не *Empty*. Ви можете переглянути вміст файлу *Content/Site.css* у лістингу 4.

Листинг 4: Зміст файла *Content/Site.css*

```
.field-validation-error {color: #f00;}
.field-validation-valid { display: none;}
.input-validation-error { border: 1px solid #f00; background-color: #fee; }
.validation-summary-errors { font-weight: bold; color: #f00;}
.validation-summary-valid { display: none;}
```

Щоб використати цю таблицю стилів, ми додали нове посилання в розділ *head* уявлення *RsvpForm*, як показано у лістингу 5. Ви додаєте уявленням елементи *link* так само, як у звичайному статичному HTML файлі.

Лістинг 5: Додавання елемента *link* до представлення *RsvpForm*

```
@model PartyInvites.Models.GuestResponse
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <link rel="stylesheet" type="text/css" href="~/Content/Site.css" />
    <title>RsvpForm</title>
</head>
<body>
    @using (Html.BeginForm())
    {
```

```
            @Html.ValidationSummary()
            <p>Your name: @Html.TextBoxFor(x => x.Name) </p>
            <p>Your email: @Html.TextBoxFor(x => x.Email)</p>
            <p>Your phone: @Html.TextBoxFor(x => x.Phone)</p>
            <p>
                Will you attend?
                @Html.DropDownListFor(x => x.WillAttend, new[] {
                new SelectListItem() {Text = "Yes, I'll be there", Value =
bool.TrueString},
                new SelectListItem() {Text = "No, I can't come", Value =
bool.FalseString}
                }, "Choose an option")
            </p>
            <input type="submit" value="Submit RSVP" />
        }
</body>
</html>
```

Порада

Якщо ви використовували MVC 3, то могли очікувати, щоб ми додамо CSS файл до подання, вказавши атрибут ***href*** *як* ***@Href("~/Content/Site.css")*** *або* ***@Url.Content("~/Content/Site.css")****. З MVC 4 Razor автоматично виявляє атрибути, які починаються з ~/, та автоматично вставляє вам* ***@Href*** *або* ***@Url****.*

Тепер буде відображатися візуально очевидніша помилка валідації, якщо були представлені дані, що спричинили цю помилку, як показано на рис. 3.2.

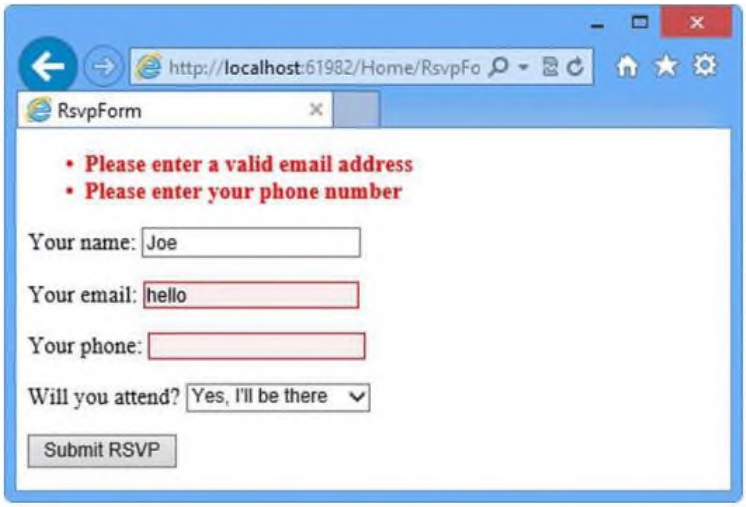


Рисунок 3.2 - Автоматично виділені помилки валідації

Остання вимога до нашого прикладу програми полягає в тому, щоб відправити електронний лист із завершеними RSVP нашому другу, організатору зустрічі. Ми могли б зробити це, додавши метод дії, щоб створити та надіслати електронною поштою повідомлення, використовуючи *e-mail* класи .NET Framework. Натомість ми збираємося використовувати допоміжний метод *WebMail*. Це не входить у рамки MVC, але це дозволить нам завершити даний приклад, не ув'язавши у деталях створення інших засобів надсилання електронної пошти.

Примітка

*Ми використовували допоміжний метод **WebMail**, тому що він дозволяє нам з мінімальними зусиллями продемонструвати надсилання повідомлень електронної пошти. Однак, у звичайній ситуації доцільно б перекласти цю функцію на спосіб дії, такий спосіб розглянемо в наступних роботах.*

Ми хочемо, щоб *e-mail* повідомлення було надіслано, коли ми обробляємо уявлення *Thanks*. У лістингу 6 показані зміни, які ми маємо зробити.

Лістинг 6: Використання допоміжного методу *WebMail*

```
@model PartyInvites.Models.GuestResponse
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Thanks</title>
</head>
<body>
    @{
        try
        {
            WebMail.SmtpServer = "smtp.example.com";
            WebMail.SmtpPort = 587;
            WebMail.EnableSsl = true;
            WebMail.UserName = "mySmtpUsername";
            WebMail.Password = "mySmtpPassword";
            WebMail.From = "rsvps@example.com";
            WebMail.Send("party-host@example.com", "RSVP
        Notification",
            Model.Name + " is " + ((Model.WillAttend ?? false) ? "" : "not")
        + "attending");
        }
        catch (Exception)
        {
            @:<b>Sorry - we couldn't send the email to confirm your
        RSVP.</b>
        }
    }
    <div>
        <h1>Thank you, @Model.Name!</h1>
        @if (Model.WillAttend == true)
        {
            @:It's great that you're coming. The drinks are already in the fridge!
        }
        else
        {
            @:Sorry to hear that you can't make it, but thanks for letting us know.
```

```
            }
        </div>
    </body>
    </html>
```

Додали **Razor** вираз, який використовує допоміжний метод *WebMail* для налаштування інформації про наш сервер електронної пошти, включаючи ім'я сервера, чи вимагає сервер SSL з'єднання та даних облікового запису. Після того, як ми всі налаштували, ми використовуємо метод *WebMail.Send* для надсилання електронної пошти.

Включено весь e-mail код у блок try...catch, щоб була змога попередити користувача, якщо електронна пошта не надійшла. Ми робимо це шляхом додавання текстового блоку до вихідним даним подання Thanks. Краще було б відобразити окреме представлення помилки, якщо повідомлення електронної пошти не може бути надіслано, але не будемо ускладнювати нашу першу MVC програму-додаток.

ПРАКТИЧНА РОБОТА № 4 ОСНОВНІ ЗАСОБИ HTML

Мета: Познайомитися із синтаксисом, основними тегами й атрибутами мови

Знати: Структуру html-документу, основні теги мови й методи форматування тексту.

Вміти: Компонувати текстову інформацію при створенні гіпертекстових документів з використанням простих текстових редакторів. Формувати гіпертекстові посилання й списки й створювати на цій основі зв'язані гіпертекстові сторінки.

Теоретичні відомості

HTML (англ. HyperText Markup Language) – стандартна мова розмітки гіпертекстових документів (веб-сторінок) в Інтернеті.

Мова HTML інтерпретується браузерами. Отриманий в результаті інтерпретації відформатований текст відображається на екрані монітора комп'ютера або мобільного пристрою.

Мета розробки HTML5 – поліпшення рівня підтримки мультимедіа-технологій з одночасним збереженням зворотної сумісності, зручності читання коду для людини і простоти аналізу для парсерів.

Розробкою та впровадженням стандартів для мережі Інтернет, в тому числі і стандартів для мови HTML займається організація W3C (англ. World Wide Web Consortium – консорціум Всесвітньої павутини). Переглянути інформацію про html5 на їхньому веб-сайті можна за наступним посиланням:

https://www.w3.org/TR/html5/

Розглянемо структуру html-документів та основні html-теги.

Синтаксис html

<назва_тегу**>** контент **</**назва_тегу**>**

Приклад 1. Приклад HTML-тегу:

```
<p>Мій перший HTML-параграф</p>
```

Приклад 2. Приклад структури HTML5-документу:

```
<!DOCTYPE html>
<html>
<head>
<title>Назва сторінки</title>
</head>
<body>
/body>
<h1>Заголовок статті</h1>
<p>Абзац тексту</p>
</html>
```

Для того, щоб побачити результат роботи даного коду, зробіть наступні кроки:

Крок 1: Відкрийте Notepad (або ін. текстовий редактор).

Крок 2: Скопіюйте код з Прикладу 2.

Крок 3: Збережіть файл з розширенням html.

Крок 4: Відкрийте збережений файл у будь-якому Інтернет-браузері.

Крім Notepad можна використовувати спеціалізовані текстові редактори, що полегшують роботу з кодом за допомогою додаткових функцій, таких, наприклад, як підсвічування коду. Наведемо деякі безкоштовні редактори для роботи з html-кодом.

Безкоштовні редактори для Windows:

– Notepad++ – Підтримує FTP та SFTP за допомогою плагіну; підсвічування синтаксису.

– HTML-Kit – Підсвічування синтаксису, підтримка FTP.

– Crimson Editor – Легковагий редактор. Підтримка FTP.

– PSPad – Підтримка FTP; підсвічування синтаксису.

– Вбудований редактор Total Commander.

Безкоштовні редактори для Linux:

– Gedit – вільний текстовий редактор робочого середовища GNOME, Mac OS X і Microsoft Windows з підтримкою Юнікода.

– Kate (KDE Advanced Text Editor) – текстовий редактор, який входить у склад середовища робочого столу KDE. Поширюється згідно GNU General Public License.

При виборі необхідно враховувати зручність, функціональність, наскільки свіжа остання версія і чи триває підтримка.

Кросплатформені редактори:

– Eclipse – Проекти PHPEclipse і PHP Development Tools. З додатковими плагінами підтримує SVN, CVS, моделювання баз даних, доступ по SSH/FTP, навігація по базі даних, інтеграція з Trac, та інше.

– Bluefish – Багатоцільовий редактор з підтримкою синтаксиса PHP, встроенной PHP документацией и т.д. З GVFS підтримує SFTP, FTP, WebDAV і SMB.

– jEdit –Versatile вільний/open source редактор. Підтримує SFTP і FTP.

– Sublime Text – кросплатформений пропріетарний текстовой редактор. Підтримує плагіни на мові програмування Python.

Приклад 3. Приклад структури сайту (стрілки ілюструють один з можливих варіантів посилань сторінок одна на одну):

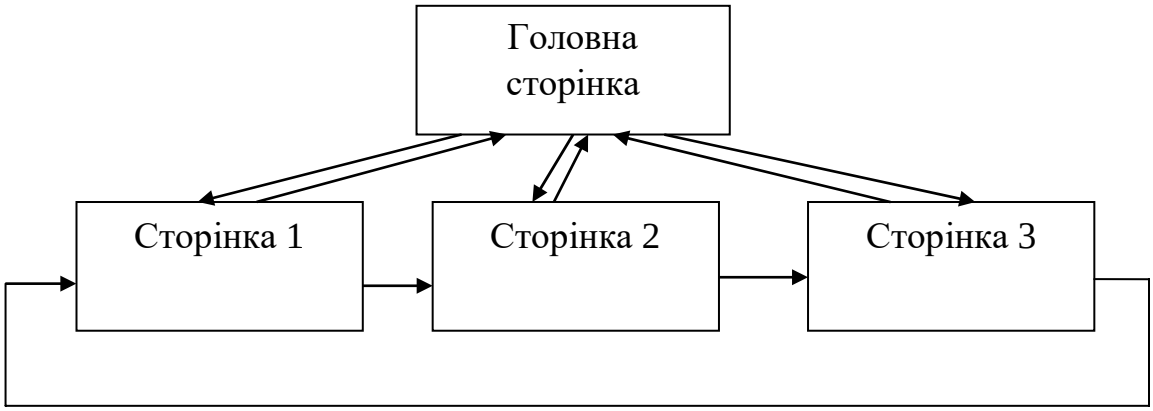


Рисунок 4.1

Розглянемо детальніше html-код з прикладу 2. DOCTYPE декларує визначення типу документа HTML.

Після оголошення версії й типу документа необхідно позначити його початок і кінець. Це робиться за допомогою тега-контейнера <HTML>. Потім, між тегами <HTML> і </HTML> варто розмістити заголовок і тіло документа.

Текст між <head> та </head> містить інформацію про поточний документ, таку як заголовок, ключові слова, які можуть використовуватися пошуковими машинами, та інші дані, які не вважаються вмістом документа.

Текст між <title> та</title> служить для ідентифікації вмісту документа (зовнішнього заголовка документа).

Текст між <body> і </body> описує видимий вміст сторінки. Текст між <h1> і </h1> описує заголовок.

Текст між і визначає параграф.

HTML заголовки

HTML заголовки визначаються тегами від <h1>до <h6>, наприклад:

<h1>This is a heading</h1>

<h2>This is a heading</h2>

<h3>This is a heading</h3>

Цифра визначає розмір заголовку – найбільший заголовок – 1, найменший – 6.

HTML параграфи

HTML параграфи визначаються тегом , наприклад:

Це параграф

Це інший параграф

HTML Посилання

HTML посилання визначаються тегом <a>, наприклад:

Це посилання

HTML Зображення

HTML зображення визначається тегом **<img>.**

Файл зображення (**src**), альтернативний текст (**alt**), розмір зображення (**width** and **height**) надаються як атрибути, наприклад:

<img src="img1.jpg" alt="зображення 1" width="104"height="142">

HTML Атрибути

HTML елементи можуть мати атрибути.

Атрибути містять додаткову інформацію про елемент, завжди визначаються в початковому тегу. Атрибути є парами ім’я/значення як: **name="value"**. Приклади:

Атрибут мови <html lang="en-US">

Атрибут назви

Атрибут посилання Це посилання **Атрибути розміру** <img src="site.jpg" width="104" height="142"> **Атрибут alt** <img src=" img2.jpg" alt="рис.2" width="104" height="142">

Таблиця 1 – Html-атрибути

Атрибути	**Опис**
Alt	Вказує альтернативний текст для зображення
Disabled	Вказує, що вхідний елемент повинен бути відключений
href	Вказує URL-адресу (веб-адресу) для зв'язку
Id	Вказує унікальний ідентифікатор для елемента
Src	Вказує URL-адресу (веб-адресу) для зображення
Style	Визначає CSS стиль вбудованого елементу
Title	Визначає додаткову інформацію про елемент (відображається у вигляді підказки)
Value	Визначає значення (текстовий зміст) для вхідного елемента.

Приклад 4. Приклад HTML5-документу з використанням css-елементів для вказання стилю тегів:

```
<!DOCTYPE html>
<html>
<body>
<h2 style="color:red">Я червоний заголовок</h2>
<h2 style="color:blue">Я синій заголовок</h2>
</body>
</html>

<body style="background-color:lightgrey">
<h1 style="font-family:verdana">Це заголовок</h1>

<p style="font-family:courier">Це параграф</p>
<h1 style="font-size:300%">Це заголовок</h1>

<p style="font-size:160%">Це параграф>
<h1 style="text-align:center">Це заголовок</h1>
```

Таблиця 4.2 – Html-елементи форматування тексту

Тег	**Опис**
<b>	Жирний текст
	Жирний текст, визначає важливість виділеного тексту
<i>	Курсив
	Курсив, визначає важливість виділеного тексту
<small>	Зменшує розмір шрифту на одну умовну одиницю. В HTML розмір шрифту вимірюється в умовних одиницях від 1 до 7. Середній розмір тексту, що використовується за замовчуванням – 3.Допускається застосування вкладених тегів <small>, при цьому розмір шрифту буде менше з кожним вкладеним рівнем, але не може бути менше, ніж 1.
<sub>	Нижній індекс
<sup>	Верхній індекс

Вибір кольору шрифта html - сторінки

Найм. коліру Код	Приклад:
aqua ##00FFFF black ##000000 blue ##0000FF fuchsia##FF00FF gray ##808080 green ##008000 lime ##00FF00 maroon ##800000 navy ##000080 olive ##808000 purple ##800080 red ##FF0000 silver ##C0C0C0 teal ##008080 white ##FFFFFF yellow ##FFFF00	HTML-код: червоний колір Синій колір Відображення в браузері: Червоний колір Синій колір

Таблиця 4.3 – Html-теги списків

Тег	Опис
	Ненумерований список
	Нумерований список
	Елемент списку

Приклад 5. Приклад ненумерованого списку.

```
<!DOCTYPE html>
<html>
<body>
<h2>Unordered List with Default Bullets</h2>
<ul>
<li> Яблука </li>
<li> Банани </li>
<li> Лимони </li>
<li> Мандарини </li>
</ul>
</body>
</html>
```

Таблиця 4.4 – Html-теги таблиці

Тег	Опис
<table>	Таблиця

<th>	Комірка заголовку таблиці
<tr>	Рядок таблиці
<td>	Комірка таблиці
<caption>	Заголовок таблиці

Приклад 6. Приклад таблиці з використанням css-елементів для вказання стилю її елементів:

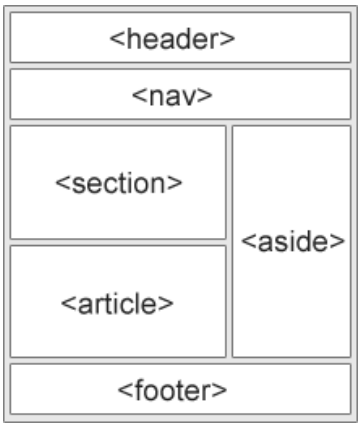
```
<!DOCTYPE html>
<html>
<head>
<style> table, th, td {
border: 1px solid black; border-collapse: collapse;
}
</style>
</head>
<body>
<table style="width:100%">
<tr>
<td>Jill</td>
<td>Smith</td>
<td>50</td>
</tr>
<tr>
<td>Eve</td>
<td>Jackson</td>
<td>94</td>
</tr>
<tr>
<td>John</td>
<td>Doe</td>
<td>80</td>
</tr>
</table>
</body>
</html>
```

Jill	**Smith**	**50**
Eve	**Jackson**	**94**
John	**Doe**	**80**

Елементи розмітки веб-сторінок в HTML5

HTML5 пропонує нові семантичні елементи, які визначають різні частини веб-сторінки, наприклад: <header>, <nav>, <section>, і <footer>.

Приклад 7. Макет сайту у кілька колонок:



Таблиця 4.5 – Елементи розмітки веб-сторінок в HTML5

Тег	Опис
Header	Заголовок документа або розділу
Nav	Контейнер для навігаційних посилань
Section	Розділ в документі
Article	Незалежна самодостатня стаття
Aside	Бічна панель
Footer	Нижній колонтитул документа або розділу

Таблиця 4.6 – HTML-елементи, що містять інформацію про поточний документ

Тег	Опис
<head>	Інформація про документ
<title>	Назва документа
<base>	Адреса за замовчуванням або мета за замовчуванням для всіх посилань на сторінці
<link>	Встановлює зв'язок із зовнішнім документом на зразок файлу зі стилями або з шрифтами.
<meta>	Визначає мета теги, які використовуються для зберігання інформації призначеної для браузерів і пошукових систем. Наприклад, механізми пошукових систем звертаються до метатегів для отримання опису сайту, ключових слів та інших даних.
<script>	Визначає сценарій на стороні клієнта
<style>	Визначає інформацію про стилі для документу

ПРАКТИЧНА РОБОТА № 5 ВИВЧЕННЯ Й ПРАКТИЧНЕ ЗАСТОСУВАННЯ СТИЛІВ CSS

Мета: Познайомитися із стилями CSS, синтаксисом, основними елементами.

Знати: Структуру HTML документа, основні підходи стилів CSS.

Вміти: Компонувати текстову інформацію при створенні гіпертекстових документів з використанням простих текстових редакторів.

Теоретичні відомості Стилі CSS

Керовані WEB-дизайнером стилі з'явилися в HTML відносно недавно й відразу наблизили HTML до сучасних засобів підготовки публікацій.

Стиль - це можливість, один раз описавши правила форматування ділянки документа, використовувати потім ці правила простим посиланням на назву стилю.

Стилі дозволяють легко вирішити проблему "фірмового" оформлення сайтів. Е можливість не форматувати тисячі об'єктів вручну, а просто описати стиль і посилатися на нього там, де потрібно. Тоді Ви змогли б, змінивши форматування стилю в одному єдиному місці, домогтися зміни форматування відповідних об'єктів у всьому сайті.

Насправді стилі як такі були присутні в HTML із самого початку. Наприклад, теги **<H1>**, **<H2>**, **<BLOCKQUOTE>**, і т.д. є ні що інше, як стилі. Однак WEB-дизайнер не міг управляти ними. Наприклад, не можливо було вказати, що всі заголовки рівня **<H1>** повинні бути в 20 пунктів висотою й синім кольором. З CSS така можливість з'явилася.

Крім того, стилі дають цілий ряд нових, раніше не доступних, засобів форматування документа. І, нарешті, стилі - річ динамічна! Це означає, що зовнішній вигляд документа можна змінювати із програм-сценаріїв прямо на льоту *(наприклад, у відповідь на те, що користувач завів покажчик миші на картинку або на посилання)*.

На жаль, існує деякий розбрід у реалізації всіх цих можливостей у найбільш популярних браузерах. Фірма **Netscape Communications**, наприклад, до останнього часу випливала своєму унікальному стандарту **JSSS** (**J**ava**S**cript **S**tyle **S**heets), що не підтримувався ніякими іншими клієнтами. Компанія **Microsoft** просувала стандарт **CSS** (**C**ascading **S**tyle **S**heets), який підтримувався WWW консорціумом. Нарешті точка зору консорціуму взяла гору й компанію *Netscape Communications* оголосила про підтримку **CSS**, починаючи з **Netscape Navigator 4.04.** Проте реалізація **CSS** у різних браузерах трохи різна. Якоюсь мірою в цьому може допомогти таблиця сумісності, складена людьми, які просто взяли, і все зрівняли. Однак, кращий спосіб не помилитися, це **перевіряти**, як виглядають сторінки в **різних** браузерах перед тим, як "випустити їх у світло".

Отже, перелічимо по пунктах, що ми можемо робити, використовуючи стилі:

- використовувати поля, кеглі й гарнітури шрифтів, індивідуальні кольори тексту й фону для окремих абзаців, слів і навіть букв. Оформляти нові рядки, буквиці та ін.;
- змінювати форматування сторінок і всього сайту в цілому, не доторкаючись до HTML файлів;
- зменшити кількість тегів в HTML тексті й зробити його кращим

для сприйняття;

- визначати варіації оформлення за рахунок спадкування класів. Наприклад, визначивши дизайн для тега **<P>**, можна пронаслідувати від нього теги **<P.exclamation>**, **<P.question>** і т.д., які будуть виглядати як **<P>**, за винятком тих атрибутів, які явно перевизначені в похідних стилях;
- гнучко управляти розміщенням елементів документа, включаючи накладення їх один на одного, і інші ефекти;
- управляти форматуванням сторінок під час друку HTML документа;
- створювати різні ефекти для тексту й зображень (тіні, димку, і т.д.).

Стилі: **Вставка в документ**

Існує три способи вставити стиль в HTML документ.

- посиланням з HTML документ на спеціальний файл, що містить визначення стилів. При такому підході Ви можете посилатися на той самий файл стилів з будь-якої кількості документів. Тоді, при зміні стилів у цьому файлі всі документи поміняють свій вид;
- визначити всі стилі, що використовуються на початку документа, а потім використовувати. При такому підході Ви можете змінювати стилі на початку документа (один раз) і Ваші зміни відіб'ються у всіх місцях документа, де ці стилі використовуються;
- вставляти вказівки на правила форматування безпосередньо в тег. Звичайно це використовується для унікального форматування, що більше (крім як у цьому місці документа) ніде не потрібно.

Звичайно при розробці сайту доводиться використовувати всі три підходи. При цьому, у випадку конфлікту, найвищий пріоритет у властивостей, вставлених безпосередньо в теги, потім - у блоків стилів, вбудованих у сторінку, і найнижчий - у блоків, на які є посилання.

Як резюме: якщо потрібно однаково відформатувати дві й більше ділянки тексту, то Вам належить визначити стиль. Двічі писати однакове форматування - ***дуже дурний тон.***

Стилі: **Синтаксис**

Синтаксис визначення стилю дуже простий. Пишеться ім'я стилю, потім у фігурних дужках перераховуються властивості, розділені символом ";". Перед закриваючою фігурною дужкою ";" не ставиться. Властивостей може бути дуже багато, пізніше ми розглянемо деякі з них.

Кілька описів стилів утворять блок стилів. Наприклад, визначимо блок, що задає червоний колір і кегль в 36 пунктів для заголовків **<H1>** і голубий колір і кегль в 18 пунктів для заголовків **<H2>**:

```
H1 {
font-size: 36pt; color: red
}
H2 {
font-size: 18pt; color: blue
}
```

Стилі: **Вставка в документ**

Стилі можна використовувати в документі трьома способами: визначаючи посилання на них, вставляючи в початок документа або вставляючи властивості стилю прямо в теги.

Вставка за допомогою посилання

Для того, щоб зробити в документі посилання на окремий файл стилів, варто записати:

<LINK REL=STYLESHEET TYPE="text/css" HREF="URL">

Де URL - адреса файлу з визначенням стилів (стандартне розширення - CSS). Наприклад,

<LINK REL=STYLESHEET TYPE="text/css" HREF="style2.css">

у файлі **style2.css** утримується наведене вище визначення **<H1>** і **<H2>**.

Вставка безпосередньо в документ

Для того, щоб вставити блок стилів прямо в документ, потрібно укласти його в тег **<STYLE> ... </STYLE>**. Наприклад, це може виглядати так:

```
<STYLE TYPE="text/css">
<! ---i
{
font-size: 36pt; color: red
}
H2 {
font-size: 18pt; color: blue
}
---i>
</STYLE>
<H1>Заголовок рівня 1</H1> Звичайний текст<BR> Звичайний текст<BR>
<H2>Заголовок рівня 2</H2> Звичайний текст<BR> Звичайний текст<BR>
<H2>Ще заголовок рівня 2</H2> Звичайний текст<BR> Звичайний текст<BR>
<H1>Ще заголовок рівня 1</H1> Звичайний текст<BR> Звичайний текст<BR>
<H2>Знов заголовок рівня 2</H2> Звичайний текст<BR>
Звичайний текст<BR>
```

Результат повинен бути таким же, як і минулого разу.

Важливе зауваження:

Хоча, тут цього не зроблено, існують ***дуже поважні причини*** *завжди вставляти*

<STYLE> ... </STYLE> *у* ***заголовну частину документа****. Тобто між тегами*

<HEAD> *і* ***</HEAD>****. Намагайтеся завжди робити саме так. Вставка властивостей у теги*

Властивості стилів можна вставляти прямо у відповідні теги. Розглянемо такий приклад:

<P>Цей абзац відформатований за замовчуванням. Цей абзац відформатований за замовчуванням. Цей абзац відформатований

<P STYLE=" margin-left: 2cm; margin-right: 2cm">

Цей абзац має відступи праворуч і ліворуч по 2 сантиметри. Цей абзац має відступи праворуч і ліворуч по 2 сантиметри.

<P STYLE=" font-size: 1in; color: red; font-style: italic">

Цей абзац написаний червоними похилими буквами висотою в один дюйм.

Елементи оформлення можна вставляти практично в будь-який тег. Наприклад, у тег **<DIV STYLE="...">** вони будуть ставитися до описуваної секції, у тег **<UL STYLE="...">** - до списку, у тег **<LI STYLE="...">** - до елемента списку й т.д.

Існує новий спеціальний тег **<SPAN>**, він у всім аналогічний **<DIV>** з тої лише різницею, що ніяк не впливає на розбивку документа на рядки. У такий спосіб його можна використовувати для завдання стильових особливостей окремих слів і навіть букв.

Класи

Приємною властивістю стилів є те, що вони можуть успадковувати властивості один одного. У програмуванні об'єкти, що успадковують властивості "батьків", називаються *класами*.

Давайте визначимо, що всі теги **<P>** у документі будуть задавати абзаци, написані шрифтом розміром в 20 пунктів блакитного кольору. Потім визначимо, що в нас ще будуть спеціальні абзаци, такі ж, але зрушені ліворуч на пів дюйма. І, нарешті, у нас будуть абзаци ще й третього виду - такі ж, як перші, але написані похилими буквами. Рішення цього завдання може виглядати, наприклад, так:

```
<STYLE TYPE="text/css">
<! ---i
P { font-size: 20pt; color: blue}
P.m { margin-left: 0.5in}
P.i     { font-style: italic}
---i>
</STYLE>
<P>
Це звичайний абзац.
<P CLASS=m>
Це зрушений абзац.
<P CLASS=i>
Це похилий абзац.
```

Подивіться, як це виглядає.

Зрозуміло, так можна оформляти не тільки спеціальні абзаци, але й спеціальні заголовки, розділи документа й т.д.

Посилання

Особливий випадок представляють посилання. Взагалі можна визначити стиль для тега **<A>**, і посилання будуть виглядати так, як ми хочемо. Однак, існують ще й посилання, на яких ми вже побували, і активні посилання. Для них спеціально визначені класи: A:link - посилання, A:visited - посилання, на якій уже побували й A:active - посилання активні в цей момент. Наприклад, зробимо так, щоб посилання

на нашій сторінці були висотою в 20 пунктів, перекреслені. Ті посилання на яких ми не були - зелені. Ті, на яких були - червоні.

Рішення

```
<STYLE TYPE="text/css">
<! ---i
A:link { font-size: 20pt; color: green; text-decoration: line-through } A:visited { font-size: 20pt; color: red; text-decoration: line-through }
---i>
</STYLE>
<A HREF="fictive">Фіктивне посилання, на ньому ми ще не були</A>
<P>
<A HREF="http://www.botik.ru/~ks/ip/scripts/listany.cgi?file=lec2.html&code=win">На цьому посиланні ми вже були</A>
```

Спадкування

HTML документ має ієрархічну структуру. Тіло документа укладене в теги <**BODY**>**...**</**BODY**>. Усередині документа можуть бути розділи, укладені в теги <**DIV**>**...**</**DIV**>. Усередині розділів можуть бути ще розділи й т.д.

Ідея спадкування полягає в тому, що всі формати, які Ви визначите для <**BODY**>, будуть успадковуватися всіма розділами. Ви можете перевизначити в розділі деякі формати, а всі інші він успадкує від <**BODY**>.

Таким же чином, розділ, вкладений в інший розділ, успадковує формати розділу що його охватує, але може щось перевизначити для себе.

Розглянемо приклад:

```
<DIV STYLE=" font-size: 16pt; color=black;">
Це перший розділ. Чорні букви розміром в 16 пунктів.
<DIV STYLE=" text-decoration: underline; margin-left: 1cm">
Це другий розділ усередині першого. Усе успадковано від першого,
але букви підкреслені й всі сдвинуто на 1 сантиметр ліворуч від першого розділу.
<DIV STYLE="color: blue; margin-left: 1cm">
Це третій розділ усередині другого. Усе успадковано від другого,
але букви блакитні й всі сдвинуто на 1 сантиметр ліворуч від другого розділу.
</DIV>
Це триває другий розділ (третій закінчився)
</DIV>
Це триває перший розділ (другий закінчився)
</DIV>
```

вкладені розділи успадковують форматування батьків і можуть додавати своє (а якщо треба, то й змінювати батьківське). Однак, як тільки вкладений розділ завершився й триває батьківський, всі його формати відновлюються.

У цьому й полягає зміст спадкування стильового оформлення в **CSS**. Наприклад, використання нового тега **<SPAN>**.

Багато з людей люблять виділяти товстим фломастером місця, що сподобалися їм, у газетах і книгах. При цьому текст на зафарбованій ділянці продовжує проступати, але фон змінює колір. Спробуємо використовувати такий прийом виділення головних думок. Оскільки ми виділяємо не в газеті, скористаємося своєю перевагою й, ще трохи зробимо товстішим шрифт у виділених ділянках. Рішення може бути таким:

```
<STYLE TYPE="text/css">
<!--
SPAN.NB {background: yellow;font-weight: 700}
-->
</STYLE>
<BR>
По некоторым оценкам, благодаря WWW может быть поставлено под угрозу существование правительств. С изобретением технологии, обеспечивающей
<SPAN CLASS="NB">истинную демократию</SPAN>, в отличие от
<SPAN CLASS="NB">существующей представительской демократии</SPAN>,
весь процесс управления может существенно, если не сказать коренным образом, измениться.
<P ALIGN=RIGHT>Том Армстронг
```

Властивості

Дотепер ми користувалися властивостями стилів такими як font-**size**, **text-decoration**, **color** і ін. ніяк не визначаючи які взагалі властивості бувають. Насправді їх дуже багато.

На практиці варто обережно користуватися цією таблицею. Далеко не все працює скрізь, а дещо не працює просто ніде. Проте це - стандарт, і те, що не працює зараз, заробить у нових версіях браузерів.

Тепер, коли Ви розумієте, що таке стилі, як і навіщо їх використовувати, Ви можете просто брати з таблиці властивості й уписувати їх у свої визначення стилів. Особливих проблем *(крім несумісності браузерів)* виникати не повинно.

.

ПРАКТИЧНА РОБОТА № 6 ЗАСТОСУВАННЯ JAVASCRIPT ПРИ СТВОРЕННІ WEB-СТОРІНОК

Введення/виведення в JavaScript

Будь-яка мова програмування немислима без операторів виведення. JavaScript не є виключенням. Виведення даних на екран може відбуватися різними способами. При цьому оператори виводу оптимізовані для найбільш зручного їхнього використання. Найбільш простим є застосування оператора Alert (). Аргументом оператора може бути будь-який рядковий вираз. Якщо аргумент має нерядковий тип, то він переводиться в рядковий. Результатом виконання оператора Alert є виведення на екран діалогового вікна, умістом якого є значення виразу аргументу. При цьому діалогове вікно буде очікувати натискання користувачем кнопки ОК. Тільки після виконання цієї дії виконання програми й відображення сторінки буде продовжено. Виведення за допомогою вікна оператора Alert досить зручно використовувати для контролю значень змінних на тому або іншому етапі виконання програми, тобто при налагодженні. Приведемо код HTML-Документа, що приводить до появи такого вікна.

Виведення на екран з використанням вікна оператора Alert

```
<HTML>
<ПОЗНАЧКА content="text/html; charset= windows-1251" http~equiv= Content-Type>
<BODY>
<SCRIPT>
mas2 = new Array ("A", "B", "C"); dd = mas2.join ("->");
alert(dd);
</SCRIPT>
</BODY>
</HTML>
```

Як вправу рекомендуємо перевірити результат роботи перерахованих у попередньому пункті операторів JavaScript.

Слід зазначити, що функція Alert () є методом об'єкта window, що описує поточне вікно браузера. Тому синтаксично більш коректно викликати цю функцію в такий спосіб: window.alert("Текст повідомлення").

Іншим способом виводу інформації на екран є виведення у тіло документа. Організовується він за допомогою оператора write про, що є методом об'єкта document, що описує поточний документ, завантажений у дане вікно.

Оператор document.writeln() відрізняється від оператора document.write() тим, що переносить позицію виводу на новий рядок. Вивід тексту відбувається з поточними атрибутами, які мають місце на момент виклику того або іншого оператора виводу. Вираження, що є аргументом оператора виводу, може містити будь-яку строкову константу, а також містити в собі різні теги HTML. При виводі подібного вираження ці теги будуть інтерпретуватися відповідним чином. Все це дозволяє будувати HTML-Код на лету, залежно від тих або інших параметрів.

Іноді при виводі на екран засобами JavaScript виникають проблеми з кодуванням російського шрифту. Щоб ці проблеми не турбували вас, необхідно, щоб у настроюваннях браузера була обрана опція автоматичного визначення кодування документа.

Вивід у тіло документа засобами JavaScript ;

```
<HTML>
<ПОЗНАЧКА content="text/html; charset= windows-1251" http-equiv= Content-Type>
<BODY bgcolor= "#aa88aa" >
<H2> Це вивід засобами HTML <BR>
<SCRIPT>
document.write ("А це працює оператор Write ()"); document.writeln ("</H2>
<H3> <FONT color=#fffffff> " + "<center> Поміняємо стиль шрифту </center> </H3>
<BR> ");
</SCRIPT>
!!!!!!!!!!!!!
</FONT>
<SCRIPT>
document.writeln (" Ми можемо навіть вставити картинку: <BR>");
document.writeln (" <img src= s37b.jpg> ");</SCRIPT>
</BODY>
</HTML>
```

Розглянемо тепер оператори введення. JavaScript надає нам кілька способів організації введення. Перший - використання методу Prompt об'єкта window. Він має наступний синтаксис:

```
d = window.promt ("Текст повідомлення", "Значення_за_замовчанням");
```

У результаті виконання такої команди на екрані з'явиться вікно запиту, де користувачеві буде виведене запрошення на введення, що втримується у виразі "Текст повідомлення". Після вводу введене значення привласнюється змінній d. Якщо користувач не ввів нічого, то d буде привласнене значення виразу "Значення_за_замовчанням". Це значення буде виведено у вікні запиту й підсвічено так, що, нажавши кнопку ОК, користувач введе це значення, а, натиснувши будь-яку іншу кнопку, може приступитися до вводу своєї інформації . Останній вираз є необов'язковим елементом синтаксису оператора Promt.

Введення за допомогою оператора Promt

```
<HTML>
<ПОЗНАЧКА content="text/html; charset= windows-1251" http-equiv= Content-Type>
<BODY>
<SCRIPT>
var d = "Моя";
c=prompt("Уведіть слово", "Земля"); s =d+" " +C;
alert (s) ;
</SCRIPT>
```

</BODY>
</HTML>

Ввод значень булевского типу зручніше за все здійснювати за допомогою оператора window.confirm, що має синтаксис: b = confirm ("Питання");

У результаті виконання такої команди на екрані з'явиться вікно із заданим питанням і двома кнопками . Залежно від натискання користувачем тої або іншої кнопки змінна b одержить або значення true (кнопка ОК), або false (кнопка Cancel, у русифікованої ОС Windows - Скасування).

Керування потоком обчислень в JavaScript

У спадщину від мови C++ JavaScript дісталися наступні оператори, що реалізують основні алгоритми керування потоком обчислень (flow control): оператор циклу з кінцевим числом повторень, оператор циклу while, оператор розгалуження. Розглянемо їхній синтаксис.

Оператор розгалуження реалізує вибір тої або іншої послідовності дій залежно від умови (умовний оператор):

if (умова) { Послідовність 1 } else { Послідовність 2 }

Вираження "умова" повинне бути булевского типу. Це може бути комбінація операторів відносини або результат дії оператора confirm. Оператор циклу з кінцевим числом повторень повторює певну послідовність дій задане число раз.

for ("вираження", "умова", "операція") { Послідовність дій. }

Тут необхідне використання целочисленної змінної, початковим значенням якої буде "вираження". Цикл буде повторюватися, поки буде щирим "умова". При цьому при кожній ітерації циклу над змінної-лічильником буде виконуватися дія "операція".

Приклад обчислення суми елементів деякого масиву:

3=0

for (i=0; i<5; i++) { s+=mas[i]; }

Цикл while виконує деяку послідовність дій доти, поки вірно деяка умова.

Синтаксис циклу наступний:

while ("умова") { Послідовність дій }

Через те, що перевірка умови передує виконанню послідовності дій, цикл while одержав назву циклу із предусловием.

Для примусового виходу із циклів використовується команда break. Для переходу до наступної ітерації циклу (дострокового виконання послідовності дій усередині циклу) використовується оператор continue.

Керування вікнами перегляду

Засоби контролю за відображенням сторінок в JavaScript доповнені командами, що дозволяють управляти як вікнами браузера, так і їхнім умістом. Команда document. clear очищає поточне вікно. Це може придатися при виводі даних у тіло документа. Команда window.close закриває поточне вікно браузера. Команда Window.open (ur 1, місце розташування, атрибути) відкриває Документ за адресою uri у вікні або фреймі, заданому у вираженні "місце розташування".

Параметри вікна описані у вираженні "Атрибути". Дана команда широко використовується для відкриття супутньому основному вікну вікон з рекламою.

(Приклади скрипт коду):

Занесення у вибране:

Додати у вибране

Лічильник посилань

<script language="JavaScript">

<! ---i

var now = new Date() fixDate(now)

now.setTime(now.getTime() + 365 * 24 * 60 * 60 * 1000) var visits = getCookie("counter")

if (!visits)

visits = 1

else

visits = parseInt(visits) + 1

setCookie("counter", visits, now) Ви були тут " + visits + " раз(а).")

// ---i>

</script> Навігаційне меню

<Form><Input Type="hidden" Name="select value">

<Select Name="sel" Size="1" OnChange="top.location.href = this.options[this.selectedIndex].value;">

<Option selected value=#>Посилання</Option>

<Option value="http://www.ваш_сайт.ru">Посилання 1</Option>

<Option value="http://www.ваш_сайт.ru">Посилання 2</Option>

<Option value="http://www.ваш_сайт.ru">Посилання 3</Option>

<Option value="http://www.ваш_сайт.ru">Посилання 4</Option>

</Select></Form> Організація переходу

<SCRIPT> Str = "OK \n "+

"Скасування \n\n "+ "OK->1.html \n "+ "Cancel->2.html "; if (confitm(str))

{ location.href = "1.html" }

else { location.href = "2.html" } </SCRIPT>

Виділення напису. Текст повільно переливається кольорами.

<SCRIPT LANGUAGE="JavaScript">

<! ---i Begin

function setupStrobe() {

text = " Плавно миготливий текст"; var x=navigator.appVersion; y=x.substring(0,4); if(y>=4)strobeEffect();

}

var isNav=(navigator.appName.indexOf("Netscape")!=-1); var colors=new Array(

"FFFFFF","FFFFFF","FFFFFF","FFFFFF","FFFFFF","FFFFFF",

"FFFFFF","F9F9F9","F1F1F1","E9E9E9","E1E1E1","D9D9D9",

"D1D1D1","C9C9C9","C1C1C1","B9B9B9","B1B1B1","A9A9A9",

"A1A1A1","999999","919191","898989","818181","797979",

"717171","696969","616161","595959","515151","494949",

```
"414141","393939","313131","292929","212121","191919",
"111111","090909","000000") a=0,b=1;
function strobeEffect() { color=colors[a];
aa="<font color="+color+">" + text + "</font>" if(isNav) {
document.object1.document.write(aa); document.object1.document.close();
}
else object1.innerHTML=aa; a+=b;
if (a==38) b-b-=2; if (a==0) b+=2;
xx=setTimeout("strobeEffect()",10);
}
// End ---i>
</SCRIPT>
</HEAD>
<BODY  onLoad="setupStrobe()">
<div id="object1"></div> Інформація користувача
<font size="2">* Небагато інформації про вас:<br>
<script language="JavaScript"> var name = navigator.appName;
var vers = navigator.appVersion;
var code = navigator.appCodeName; var where = document.referrer;
var platform = navigator.platform; document.write('<Dd>Броузер: ' + name +
'<Dd>Версія броузера: ' + vers + '<Dd>Кодова назва броузера: ' + code +
'<Dd>Ви зайшли з: ' + where + '<Dd>Платформа: ' + platform);
</script>
Дата останнього оновлення сторінки
<SCRIPT LANGUAGE="JavaScript">
var m = "Останнє відновлення " + document.lastModified; var p = m.length-8;
document.writeln("<center>"); document.write(m.substring(p+8, 0));
document.writeln("</center>");
</SCRIPT>
Виведення дати
<script language="JavaScript">
<! ---i
time=new Date(); month=(time.getMonth() + 1); date=time.getDate();
year=time.getYear();
if (month < 10) {month = "0" + month } if (date < 10) {date = "0" + date }
datastr=( date + " / "  + month +" / " + year )
---i>
</script>
</head>

<body>
<center><font face="Arial" size="3" color="#DC5912"><b>
<script language="JavaScript">
<! ---i
```

```
document.write(datastr);
---i>
</script>  Стартова сторінка
<A class=wmenu onclick="this.style.behavior='url(#default#homepage)';
this.setHomePage('http://www.ваш_сайт.ru');return false;"
href="http://newwave.com.ru/#"> <FONT size=2><B>Зробити стартової</B><FONT
size=+0></A>
Спливаюче вікно
<Script Language="JavaScript"> Artel=window.open("frame11.htm","Artel",
"Width=500, Height=160, Toolbar=0, Location=0", "Status=0, Menubar=0, Scrollbars=0,
Resizable=0")
</Script>

Рядок станів, що біжить
<Script Language=JavaScript> var scrollCounter = 0;
var scrollText = "Ваш текст"; var scrollDelay = 70;
var i = 0;
while (i ++ < 80)
scrollText = " " + scrollText; function Scroller()
{window.status = scrollText.substring(scrollCounter++, scrollText.length);
if (scrollCounter == scrollText.length) scrollCounter = 0;
setTimeout("Scroller()", scrollDelay);}
Scroller();
</Script>
Показ випадкового баннера
<Html>
<Head>
<Script Language="JavaScript"> var imagesarray = new Array( "banner1.jpg",
"banner2.jpg", "banner3.jpg");
var commentsarray = new Array( "Alt - banner1",
"Alt - banner2", "Alt - banner3");
</Script>
</Head>
<Body>
<Script Language="JavaScript">
var los = Math.floor(Math.random() * imagesarray.length)
document.write ("<Img Src="'+imagesarray[los]+'"
Alt="'+commentsarray[los]+'">");
</Script>
</Body>
</Html>
```

ПРАКТИЧНА РОБОТА № 7 ЗАСТОСУВАННЯ JAVASCRIPT ПРИ СТВОРЕННІ WEB FORM

Мета: Познайомитися із синтаксисом, основними елементами мови JavaScript.

Знати: Структуру HTML документа, основні елементи мови JavaScript.

Вміти: Компонувати текстову інформацію при створенні гіпертекстових документів з використанням простих текстових редакторів.

Теоретичні відомості (JavaScript і форми HTML)

Об'єкт **form**

Як створити кілька форм? Справа в тому, що кожна форма буде мати своє власне ім'я (для цього використовується значення атрибута name елемента <form>). А виходить, одержати доступ до елементів форми буде нескладно. Спочатку даємо ім'я (name="MyForm"), потім одержуємо доступ до будь-яких елементів:

```
var name = document.myForm.custName.value
```

Відомо, що в цього об'єкта є властивість forms, що представляє собою масив форм, що втримуються в поточному документі. Наприклад, якщо припустити, що форма myForm з попереднього приклада є першою на сторінці, то до неї можна одержати доступ у такий спосіб:

```
var name = document.forms[0].custName.value
```

Іноді буває необхідно організувати автоматичний доступ до форм. Наприклад, такий спосіб:

```
for (x=0; x<3; x=x+1) {var formName[x] = document.forms[x] name}
```

За допомогою такого циклу for можна усі імена форм записати в масив, тим самим організувавши незалежний доступ до кожної з них (formName[x]).

Всі об'єкти форми, незалежно від того, яким чином здійснюється доступ до них, мають свої набори властивостей, як і документ, вікно. Властивості об'єкта form, зокрема, що випливають:

- *action.* URL. По суті, такий, як атрибут action елемента <form>.
- *method.* Такий, як атрибут method елемента <form>.
- *name.* Ім'я форми (такий, як атрибут name елемента <form>).
- *length.* Число елементів input, textarea й select у формі.
- *target.* Цільове вікно або фрейм.
- *elements.* Масив, у якому містяться всі елементи input, textarea і select Об'єкт form має, свої методи, серед яких reset ( ), submit ().

Обробка помилок на формі за допомогою JavaScript

JavaScript і його обробники подій ідеально підходять для перевірки даних, що вводяться користувачами у формах. Під час набору даних скрипт може непомітно перевіряти коректність кодування, числа символів і т.д.

Почнемо з невеликого приклада. У лістингу закодована ціла сторінка, що дозволяє її відвідувачеві вводити дані. Скрипт здійснює перевірку правильності вводу поштового індексу.

Лістинг *Перевірка даних у формах за допомогою JavaScript*

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
<title>Checking the Zip</title>
<meta http-equiv=" Content-Script-Type" content="text/javascript">
<script>
<! ---i
function zipCheck(){
var zipStr = custForm.zipCode.value;
if (zipStr == "") {
alert("Please enter a five digit number for your Zip code"); return(-1);
}
if (zipStr.length != 5) {
alert ("Your Zip code entry should be 5 digits"); return(-1);
}
return(0);
}
function checkSend()
{
var passCheck = zipCheck(); if (passCheck == -1) { return;
}
else { custForm.submit();
}
}
// end hiding ---i>
</script>
</head>
<body>
<h1>Please fill out the following form:</h1>
<form action=" cgi-bin/address.pl" method="post" name="custForm">
<pre>
Name: <input type="text" size="20" name="name">
Address: <input type="text" size="50" name="address">
City: <input type="text" size="30" name="city">
State: <input type="text" size="2" name="state">
Zip: <input type="text" size="5" name ="zipCode"
onBlur = "zipCheck()">
Email: <input type="text" size="40" Name="email">
<input type="button" value="Send It" onClick = "checkSend()">
</form>
</body>
</html>
```

Обробка форми починається при виникненні події onBlur, при втраті фокуса поля вводу індексу, або при натисканні клавіші Tab, або за допомогою миші. Оброблювач запускає функцію zipCheck, що перевіряє уведений поштовий індекс на коректність. Якщо поле залишилося порожнім, видається віконце з нагадуванням

про те, що це поле варто заповнити. Це робиться шляхом перевірки значення рядкової змінної:

```
if (zipStr = "") {
```

Якщо введено більше або менше 6 цифр, також видається попередження.

```
if (zipStr.length != 5) {
```

Якщо користувач зробив все правильно, триває нормальна робота. Користувач може проігнорувати попередження або змінити його, але так, що воно залишиться некоректним. Для цього випадку передбачений повторний виклик zipCheck з функції checkSend:

```
var passCheck = zipCheck( ):
```

І якщо індекс як і раніше залишився неправильним, функція zipCheck поверне значення -1, що перевіряється за допомогою checkSend:

```
if (passCheck = -1) {  return;  }
```

Тому, якщо й цього разу індекс введений невірно, то функція checkSend поверне користувача на продовження заповнення форми. Загалом, вийшов багато східчастий захист від неправильного введення даних. Якщо користувач нарешті набрав на клавіатурі шість цифр, форма підтверджується.

Перевірка могла б бути як завгодно складною. Наприклад, можна заборонити використання буквених символів у цьому полі. Це можна зробити, застосувавши метод charAt( ) об'єкта String, що ретельно перевірить кожну позицію й установить, чи дійсно в ній перебуває цифра від 0 до 9.

Розширена функція zipCheck( ) може виглядати в такий спосіб:

```
function zipCheck()
{
var zipStr = custForm.zipCode.value;

if (zipStr == "")
{
alert("Please enter a five digit number for your Zip code"); return(-1);
}
if (zipStr.length != 6)
{
alert ("Your Zip code entry should be 6 digits"); return(-1);
}
for (x=0; x < 6; x++)
{
if ((zipStr.charAt(x) < "0") || (zipStr.charAt(x) > "9"))
{
alert("All characters in the Zip code should be numbers."); return(-1);
}
}
return(0);
}
```

Перевірка починається із циклу for, щоб програма могла зробити необхідну кількість ітерацій, охопивши всю довжину поля вводу:

```
for (x=0; x < 5; x++) {
```

Використовуючи цикл for, можна послідовно перебрати всі значення масиву zipStr з номерами від 0 до 5.

У циклі перебуває умовний вираз if, він й перевіряє, чи є символи цифровими:

```
if ((zipStr.charAt(x) < "0") || (zipStr.charAt(x) > "9")) {
```

Знак || означає логічне «або». Тобто якщо виконується хоча б одна з умов, то весь вираз приймає значення «істина», і з'являється віконце з попередженням:

```
alert("Індекс повинен складатися тільки із цифр!"); return(-l);
```

Якщо ж вираз if приймає значення «неправда», то команди, що втримуються всередині, пропускаються, і передбачається, що користувачеві вдалося ввести все правильно.

JavaScript на клієнтській машині

Одною з переваг JavaScript є те, що ця мова вбудовується прямо у веб-сторінку й вам не потрібно піклуватися про CGI-скрипти. У деяких випадках з подивом можна виявити, що зовсім і не потрібний доступ до каталогу CGI сервера. У наступному лістингу показаний приклад електронної форми і її обробного механізму.

Лістинг Customer Survey Form

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml 1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Customer Survey Form</title>
<script>
<! ---i
function processform () { var newline = "\n";
var result_str = "";
var Form1 = document.form1;
result_str += Form1.first_name.value + " " + Form1.last_name.value + newline; result_str += Form1.email.value + newline;

for (x = 0; x < Form1.where.length; x++) { if (Form1.where[x].checked) {
result_str += Form1.where[x].value + newline; break;
}
}
if (Form1.desktop.checked) result_str += "desktop computers" + newline;
if (Form1.notebook.checked) result_str += "notebook computers" + newline; if (Form1.peripherals.checked) result_str += "peripherals" + newline;
if (Form1.software.checked) result_str += "software" + newline; document.form2.results.value = result_str;
return;
}
```

```
// ---i>
</script>
</head>
<body>
<h1>Web Site Survey</h1>
<form name="form1" id="form1">
<table cellpadding="5">
<tr>
<td>First name:</td>
<td><input type="text"
name="first_name"
size="40" maxlength="40" /></td>
</tr>
<tr>
<td>Last
name:</td>
<td><input type="text"
name="last_name"
size="40" maxlength="40" /></td>
</tr>
<tr>
<td> E-mail address: </td>
<td><input type="text" name="email" size="40" maxlength="40" /></td>
</tr>
</table>
<p>
<b>Where you heard about us:</b>
</p>
<input type="radio" name="where" value="Web" checked="checked">
Web Search or Link</input><br />
<input type="radio" name="where" value="Advertisement">
Radio or TV Ad</input><br />
<input type="radio" name="where" value="Press Mention">Article or press
mention</input><br />
<input type="radio" name="where" value="Other"> Other </input><br />
</p>
<p>
<b>What products would you like more information about? (check all that
apply)</b><br/>
<input type="checkbox" name="desktop"> desktop computers
<input type="checkbox" name="notebook"> notebook computers
<input type="checkbox" name="peripherals"> peripherals
<input type="checkbox" name="software"> software
</p>
```

```
<button name="submit" type="button" onclick="processform ()">
<span style=" font-family: Arial, Helvetica; font-variant: small-caps; font-size: 12pt"> Submit Survey</span>
</button>
<button name="reset" type="reset">
<span style=" font-family: Arial, Helvetica; font-variant: small-caps; font-size: 12pt"> Clear Page
</span>
</button>
</form>
<hr/>
<form name="form2" id="form2" action="mailto:survey@fakecorp.net">
<p>
Check the entries below for accuracy. If they're accurate, then enter a comment on the last line (if desired) and click Send It to send the form via e-mail.
</p>
<textarea name="results" cols="40" rows="10">
</textarea>
<button name="submit" type="submit">
<span style=" font-family: Arial, Helvetica; font-variant: small-caps; font-size: 12pt"> Send It!</span>
</button>
</form>
</body>
</html>
```

Скрипт бере значення, введені користувачем у першій формі, і поміщає їх у текстовому вигляді в другу форму. При натисканні на кнопку SEND IT! (Відправити!) вся накопичена інформація відправляється по електронній пошті.

Цей приклад показує, як важливо одержувати коректні значення прапорців і перемикачів. У лістингу демонструються два способи, якими це можна зробити. Для початку звернемося до перемикачів. Нам потрібно визначити, яке положення селективної кнопки обрано, тоді ми зможемо працювати з відповідними даними.

Значення перемикачів зберігаються в спеціальних масивах, а отже, ми можемо одержати доступ до них за допомогою циклу:

```
for (x = 0; x < Form1.where.length; x++)
{ if (Form1.where[x].checked)
{
result_str += Form1.where[x].value + newline;break;
}
}
```

У свою чергу, всі перемикачі теж зберігаються в масиві об'єктів, імена яких визначаються за допомогою атрибута name, що вказується при їхньому створенні. У нашому прикладі name="where". З кожним екземпляром where можна

використовувати властивість checked для того, щоб визначити, у якому положенні перебуває перемикач. Коли ми знайдемо той об'єкт where, властивість checked якого дорівнює true, це буде означати, що саме його value нам і потрібно.

Тому фактично наведений раніше цикл займається визначенням положення перемикача. Коли він його знаходить, цикл for завершується (командою break), а зі значенням виробляються певні дії. У цьому випадку воно додається до рядка result_str.

Друге, що показано в цьому прикладі, - це робота з наборами прапорців. І тут теж потрібно перевіряти стан властивості checked. Але, оскільки кожний прапорець має своє власне ім'я (name), то немає необхідності зберігати їх у якомусь масиві або використовувати цикл для їхнього перебору. До кожного прапорця потрібен індивідуальний підхід, у кожного з них особисто потрібно з'ясувати, чи є він checked. Якщо це так, то використовується його значення:

```
if (Form1.desktop.checked)  result_str += "desktop computers" + newline;
if (Form1.notebook.checked)  result_str += "notebook computers" + newline;
if (Form1.peripherals.checked) result_str += "peripherals" + newline;
if (Form1.software.checked) result_str += "software" + newline.
```

СПИСОК ЛІТЕРАТУРИ

1. Прищепа М.М., Погребняк В.П. Мікроелектроніка. Частина І. Елементи електроніки. – Київ: Вища школа, 2004. – 431 с.

2. Закалик Л.У., Ткачук Р.А. Основи мікроелектроніки. - Тернопіль, 1998. - 380 с.

3. Хоружний В.А., Письмецький В.О. Функціональна мікроелектроніка, опто- та акустоелектроніка. - Харків, 1995. - 186 с.

4. Сенько В.І., Панасенко М.В., Сенько Є.В. Електроніка і мікросхемотехніка. - Т.1. Елементна база електронних пристроїв. - Київ: Обереги, 2000. - 300 с.

5. Стахів П.Г., Коруд В.І., Гамола О.Є. Основи електроніки: функціональні елементи та їх застосування. - Львів: Новий світ-2000, 2003. - 128 с.

6. Радіотехніка: Енциклопедичний навчальний довідник: Навч. Посіб. / за ред. Ю.Л.Мазора, Є.А.Мачуського, В.І. Правди. – Київ: Вища школа, 1999. – 838 с.

7. Мікроелектроніка і наноелектроніка. Вступ до спеціальності / Ю. М. Поплавко, О. В. Борисов, В. І. Ільченко та ін. – Київ: НТУУ «КПІ», 2010. – 160 с