ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет програмування та комп'ютерних
і телекомунікаційних систем

Кафедра інженерії програмного забезпечення

Лабораторна робота № 2

з дисципліни «Програмування Інтернет»
на тему:

«Механізм візуалізації обробки контента ASP.NET – Razor. Відправка динамічного контенту браузеру. Створення проекту для демонстрації можливостей та синтаксису Razor»

Виконав:

студент 3 курсу, групи ІПЗ-17-1 _____ В.В.Охота
(підпис) (Ініціали, прізвище)

Перевірив: _____ О.М. Яшина
(підпис)

Хмельницький – 2020 р.

Мета. Вивчити основні можливості та синтаксис механізма візуалізації обробки контента ASP.NET – Razor.

Завдання. Варіант 10. Розробити Web -додаток ASP.NET MVC 4. Використати просту модель предметної області «Аптека». Виконати необхідні розрахунки.

Хід роботи

Для демонстрації можливостей та синтаксису Razor створимо в Visual Studio новий проект ASP.NET MVC 4 Веб -додаток ASP.NET MVC 4 і виберемо для нього шаблон Empty. Додамо в папку Models файл класу на ім'я Product.cs з контентом:

```
public class Product
    {
        public int ProductID { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public decimal Price { get; set; }
        public string Category { set; get; }
    }
```

Створюємо контролер HomeController з наступним кодом

```
Product myProduct = new Product
        {
            ProductID = 1,
            Name = "Автомобільна аптечка",
            Description = "Бінт 5м, Перекись водню 250мг , йод 100мг , пластирі 50шт , жгут
1шт , ношпа 2 пласт. , ношатирний спирт 50мл.",
            Category = "Скомплектоані ліки",
            Price = 275M
        };
        public ActionResult Index()
        {
            return View(myProduct);
        }
        public ActionResult NameAndPrice()
        {
            return View(myProduct);
        }
        public ActionResult DemoExpression()
        {
            ViewBag.ProductCount = 1;
            ViewBag.ExpressShip = true;
            ViewBag.ApplyDiscount = false;
            ViewBag.Supplier = null;
            return View(myProduct);
        }
        public ActionResult DemoArray()
        {
            Product[] array = {
        new Product {Name = "Ношпа", Price = 118.00M, Category = "Знеболююча", ProductID =
2},
        new Product {Name = "Аспірин", Price = 50.50M, Category = "Проти температури",
ProductID =3},
        new Product {Name = "Стрепсилс", Price = 220.20M, Category = "Від болю у горлі",
ProductID = 4},
        new Product {Name = "Грипаут", Price = 34.95M, Category = "Від грипу",ProductID = 5}
 };
            return View(array);
        }
```

Ми визначили метод дії на ім'я Index, в якому створили об'єкт Product і заповнили його властивості. Об'єкт Product передається методу View, тому він використовується в якості моделі, коли представлення візуалізується. При виклику методу View ім'я файлу представлення не вказується, тому буде вибрано стандартне представлення для методу дії - Index.

Компонування - це спеціалізована форма представлень. Виклик методу @RenderBody() вставляє в розмітку компонування контент представлення, зазначеного методом дії. Інший вираз Razor у компонуванні звертається до властивості по імені Title в об'єкті ViewBag, щоб встановити контент елемента title. Будь-які елементи в компонуванні будуть застосовуватися до будь-якого представлення, яке використовує цю компоновку, і саме тому компонування є, по суті, шаблоном. Щоб продемонструвати, як це працює, в лістингу створимо компонування _basikLayout

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
</head>
<body>
    <div>
        <h1>Аптека <a href="https://localhost:44373/">Головна сторінка</a>  <a
href="https://localhost:44373/Home/NameAndPrice">Автомобільна аптечка</a>   <a
href="https://localhost:44373/Home/DemoArray">Ціни на ліки</a></h1>
        <div style="padding: 20px; border: solid medium black; font-size: 20pt">
            @RenderBody()

        </div>
        <h2> <a href="https://moz.gov.ua/">МОЗ</a></h2>
    </div>
</body>
</html>
```

Тут була додана пара елементів заголовка і застосовані стилі CSS до елементу div, що містить вираз @RenderBody. Це зроблено просто для пояснення, який контент надходить з компонування, а який - з представлення.
Демонстрація поділюваних компоновок. Додамо до контролера Home новий метод дії на ім'я NameAndPrice.

```
@model Lab_2_1.Models.Product
@{
    ViewBag.Title = "NameAndPrice";
    Layout = "~/Views/_BasicLayout.cshtml";
}
<h2>Назва та ціна </h2>>
Назва продукту @Model.Name його ціна $ @Model.Price
<a href="https://localhost:44373/Home/DemoExpression">Детальніше</a>
```

Створюємо представлення DemoExpression.cshtml

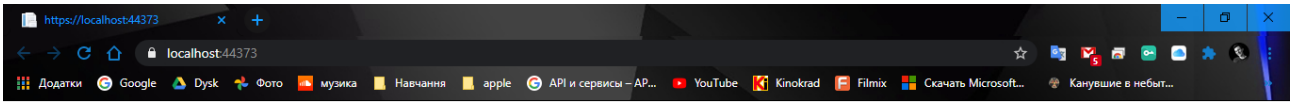
```
@model Lab_2_1.Models.Product
@{
    ViewBag.Title = "DemoExpression";
    Layout = "~/Views/_BasicLayout.cshtml";
}
<table>
    <thead>
        <tr><th>Автомобільна аптечка</th>
        <tr><th>Властивість</th><th>Значення</th></tr>
    </thead>
    <tbody>
        <tr><td>Назва</td><td>@Model.Name</td></tr>
        <tr><td>Ціна</td><td>@Model.Price</td></tr>
        <tr><td>Склад</td><td>@Model.Description</td></tr>
```

```
        <tr><td>Категорія</td><td>@Model.Category</td></tr>
        <tr>
            <td>Запас</td>

            <td>
                @switch ((int)ViewBag.ProductCount)
                {
                    case 0:
                        @: Немає в наявності
                        break;
                    case 1:
                        <b>
                            Низький запас (@ViewBag.ProductCount)
                        </b>
                        break;
                    default:
                        @ViewBag.ProductCount
                        break;
                }
            </td>
        </tr>
    </tbody>
</table>
```

Та представлення DemoArray.cshtml

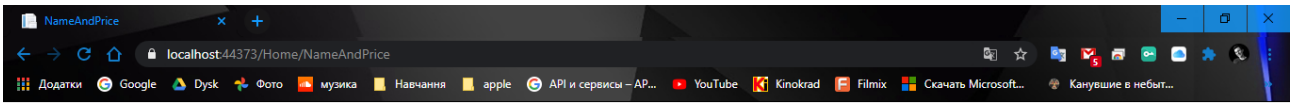
```
@using Lab_2_1.Models
@model Product[]
@{
    ViewBag.Title = "DemoArray";
    Layout = "~/Views/_BasicLayout.cshtml";
}
@if (Model.Length > 0)
{
    <table>
        <thead><tr><th>Назва</th><th>Ціна</th><th>Дія</th></tr></thead>
        <tbody>
            @foreach (Product p in Model)
            {
                <tr>
                    <td>@p.Name</td>
                    <td>$@p.Price</td>
                    <td>@p.Category</td>
                </tr>
            }
        </tbody>
    </table>
}
else
{
    <h2>No product data</h2>



}
```

Скріншоти роботи даного ПЗ



Аптека Головна сторінка Автомобільна аптечка Ціни на ліки

МОЗ

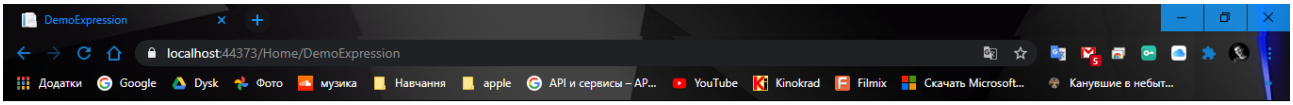


Аптека Головна сторінка Автомобільна аптечка Ціни на ліки

Назва та ціна

> Назва продукту Автомобільна аптечка його ціна $ 275 Детальніше

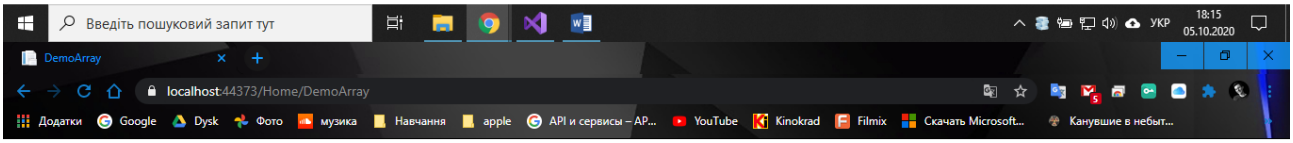
МОЗ



Аптека Головна сторінка Автомобільна аптечка Ціни на ліки

Автомобільна аптечка	
Властивість	**Значення**
Назва	Автомобільна аптечка
Ціна	275
Склад	Бінт 5м, Перекись водню 250мг , йод 100мг , пластирі 50шт , жгут 1шт , ношпа 2 пласт. , ношатирний спирт 50мл.
Категорія	Скомплектоані ліки
Запас	**Низький запас (1)**

МОЗ



Аптека Головна сторінка Автомобільна аптечка Ціни на ліки

Назва	Ціна	Дія
Ношпа	$118,00	Знеболююча
Аспірин	$50,50	Проти температури
Стрепсилс	$220,20	Від болю у горлі
Грипаут	$34,95	Від грипу

МОЗ

Контрольні питання

1. **Використання оператора @model**
   Оператори Razor починаються з символу @. У цьому випадку оператор @model оголошує тип об'єкта моделі, який буде 30 передаватиметься представленню з методу дії.

2. **Читання значення властивості в представленні**
   Оператор Razor дозволяє посилатися на методи, поля і властивості об'єкта моделі представлення з допомогою @Model, як показано в лістингу 2.4, де демонструється просте доповнення до представлення Index.

3. **Що таке Компонування ?**

Робота з компонуваннями. Ось ще один вираз Razor з файлу представлення Index.cshtml: @{ Layout = null; } 31 Це приклад блоку коду Razor, який дозволяє включати в представлення оператори C#. Блок коду відкривається за допомогою @ { і закривається за допомогою }, а утримуючі в ньому оператори оцінюються при візуалізації представлення.

4. **Виклик методу @RenderBody()**

```
<!DOCTYPE html>
<html>
32
<head>
<meta name="viewport" content="width=device-width" />
<title>@ViewBag.Title</title>
</head>
<body>
<div>
 @RenderBody()
 </div>
</body>
```

5. **Додавання елементів в компоновку**

```
<!DOCTYPE html>
<html>
<head>
 <meta name="viewport" content="width=device-width" />
 <title>@ViewBag.Title</title>
</head>
<body>
 <div>
 <h1>Product Information</h1>
 <div style="padding: 20px; border: solid medium black; font-size: 20pt">
 @RenderBody()
 </div>
 <h2>Visit <a href="http://apress.com">Apress</a></h2>
 </div>
</body>
</html>
```

Тут була додана пара елементів заголовка і застосовані стилі CSS до елементу div, що містить вираз @RenderBody. Це зроблено просто для пояснення, який контент надходить з компонування, а який - з представлення.

6. **Переваги застосування компонування**

Застосування компоновок дає багато переваг. Вони дозволяють спростити представлення (як було показано в лістингу 2.7), дають можливість створювати спільну HTML - розмітку, яку можна застосовувати до множини представлень, і полегшують супровід, поскільки загальну HTML - розмітку можна змінити в одному місці, знаючи, що зміни будуть застосовані всюди, де ця компоновка використовується

7. **Представлення _ViewStart.cshtml**

Використання файлу запуску представлення. Залишилася ще одна невелика проблема, з якою необхідно розібратися – ми повинні вказати файл компонування для застосування в кожному представленні. Це означає, що в разі перейменування файлу

компоновки понадобиться знайти всі посилання на нього представлень і внести зміну; такий процес загрожує помилками і суперечить основній концепції інфраструктури MVC - легкості супроводу. Вирішити зазначену проблему можна з використанням файлу запуску представлення. При візуалізації представлення інфраструктура MVC шукає файл з ім'ям ViewStart.cshtml. Контент цього файлу буде трактуватися так, якби він розміщений в самому файлі представлення, і цю можливість можна застосовувати для автоматичної установки 34 значення властивості Layout. Щоб створити файл запуску представлення, додайте в папку Views новий файл представлення, вкажіть _ViewStart.cshtml.

8. **Додавання до контролера нового методу дії**

Демонстрація поділюваних компоновок. Додамо до контролера Home новий метод дії на ім'я NameAndPrice. Визначення цього методу наведено в лістингу 2.10, що містить змінений контент файлу /Controllers/HomeController.cs.

Лістинг 2.10. Додавання до контролера Home нового методу дії

```
using System;
using System.Collections.Generic;
35
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Razor.Models;
namespace Razor.Controllers
{
 public class HomeController : Controller
 {
 //
 // GET: /Home/
 Product myProduct = new Product
 {
 ProductID = 1,
 Name = "Kayak",
 Description = "A boat for one person",
 Category = "Watersports",
 Price = 275M
 };
 public ActionResult Index()
 {
 return View(myProduct);
 }
 public ActionResult NameAndPrice()
 {
 return View(myProduct);
 }
 }
}
```

9. **Створення файла компонування**

Це приклад блоку коду Razor, який дозволяє включати в представлення оператори C#. Блок коду відкривається за допомогою @ { і закривається за допомогою }, а утримуючі в ньому оператори оцінюються при візуалізації представлення. Показаний вище блок коду встановлює значення властивості Layout в null. Результатом установки властивості Layout в null є повідомлення інфраструктури MVC про те, що наше представлення є самодостатнім, і воно буде візуалізувати весь свій контент,

який необхідно повернути клієнту. Щоб створити компоновку, клацніть правою кнопкою миші на папці Views у вікні Solution Explorer, виберіть у контекстному меню пункт Add -> New Item (Додати -> Новий елемент) і вкажіть шаблон MVC 4 Layout Page (Razor).

10. Підключення файла компонування

```
<!DOCTYPE html>
<html>
32
<head>
 <meta name="viewport" content="width=device-width" />
 <title>@ViewBag.Title</title>
</head>
<body>
 <div>
 @RenderBody()
 </div>
</body>
</html>
```

11. Можливості використання компонування

Компонування - це спеціалізована форма представлень. Виклик методу @RenderBody() вставляє в розмітку компонування контент представлення, зазначеного методом дії. Інший вираз Razor у компонуванні звертається до властивості по імені Title в об'єкті ViewBag, щоб встановити контент елемента title. Будь-які елементи в компонуванні будуть застосовуватися до будь-якого представлення, яке використовує цю компоновку, і саме тому компонування є, по суті, шаблоном.

12. Ролі, виконувані методом дії та представленням

Компонент	Що робить	Що не робить
Метод дії	Передає представленням об'єкт моделі представлення	Передає представленням сформатованні дані
Представлення	Використовує об'єкт моделі передставлення для відображення контенту користувачеві	Змінює будь-який аспект об'єкта моделі представлення

13. Вираз @Model для отримання значення необхідних властивостей

У представленні використовується Razor - вираз @Model для отримання значення необхідних властивостей: ... The product name is @Model.Name and it costs $ @Model.Price

14. Використання базових виразів Razor для вставки значень даних в HTML-розмітку

```
@model Razor.Models.Product
@{
 ViewBag.Title = "DemoExpression";
}
<table>
 <thead>
 <tr><th>Property</th><th>Value</th></tr>
 </thead>
 <tbody>
 <tr><td>Name</td><td>@Model.Name</td></tr>
```

```
<tr><td>Price</td><td>@Model.Price</td></tr>
<tr><td>Stock Level</td><td>@ViewBag.ProductCount</td></tr>
</tbody>
</table>
```

У цьому прикладі створена проста HTML -таблиця, комірки якої заповнюються властивостями з об'єкта моделі та об'єкта ViewBag. На рисунку 2.11 можна бачити результат запуску програми та переходу по URL типу /Home/DemoExpression.

15. Символи @:

Символи @: запобігають обробку механізмом Razor цього рядка як оператора C#.

16. Використання оператора if в представленні Razor

```
@model Razor.Models.Product
@{
 ViewBag.Title = "DemoExpression";
 Layout = "~/Views/_BasicLayout.cshtml";
}
<table>
 <thead>
 <tr><th>Property</th><th>Value</th></tr>
 </thead>
 <tbody>
 <tr><td>Name</td><td>@Model.Name</td></tr>
 <tr><td>Price</td><td>@Model.Price</td></tr>
 <tr>
 <td>Stock Level</td>
 <td>
 @if (ViewBag.ProductCount == 0) {
 @:Out of Stock
 } else if (ViewBag.ProductCount == 1) {
 <b>Low Stock (@ViewBag.ProductCount)</b>
 } else {
 @ViewBag.ProductCount
 }
 </td>
 </tr>
 </tbody>
</table>
```

Цей умовний оператор видає ті ж самі результати, що й оператор switch.

17. Масиви і колекцій при розробці програми MVC

Перерахування масивів і колекцій. При розробці програми MVC часто необхідно виконувати перерахування контента масиву або іншого різновиду колекції об'єктів з генерацією докладної інформації для кожного об'єкта. Щоб продемонструвати, як це робиться, визначили в контролері Home новий метод дії на ім'я DemoArray (Лістинг 2.18).

Лістинг 2.18. Метод дії DemoArray

```
public ActionResult DemoArray()
{
Product[] array = {
new Product {Name = "Kayak", Price = 275M},
```

```
new Product {Name = "Lifejacket", Price = 48.95M},
new Product {Name = "Soccer ball", Price = 19.50M},
new Product {Name = "Corner flag", Price = 34.95M}
};
return View(array);
}
```

18. Ручна установка типу моделі представлення

Цей метод дії створює об'єкт Product [], який містить прості значеня даних, і передає його методом View для візуалізації з використанням стандартного представлення. При створенні представлення середовище Visual Studio не пропонує варіанти для масивів і колекцій, тому деталі необхідного типу доведеться вводити вручну в діалоговому вікні Add View із зазначенням Razor.Models.Product [ ] як типу моделі представлення (рис.2.14). Контент файлу представлення DemoArray.cshtml приведений в лістингу 2.19 - він включає додавання, призначені для візуалізації користувачеві деталей елементів масиву.
Лістинг 2.19. Контент файлу DemoArray.cshtml

19. Генерація елементів із застосуванням оператора foreach

Тут за допомогою оператора @if варіюється контент на основі довжини оброблюваного масиву, а за допомогою виразу @foreach виконується перерахування контенту масиву з генерацією рядки HTML - таблиці для кожного елемента масиву. У результаті генерується елемент h2, якщо масив порожній, і по одному рядку HTML - таблиці для кожного елемента масиву в іншому випадку.

20. Робота з просторами імен

Робота з просторами імен. В останньому прикладі для посилання на Product в циклі foreach використовується повністю певне ім'я: ... @foreach (Razor.Models.Product p in Model) { ... Привести в порядок представлення можна за рахунок застосування виразу @using, щоб забезпечити для представлення контекст певного простору імен, як це робиться для звичайного класу C# - @using Razor.Models