ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет програмування та комп’ютерних

і телекомунікаційних систем

Кафедра інженерії програмного забезпечення

**Лабораторна робота №** 2

з дисципліни «Програмування в інтернет»

на тему:

«Механізм візуалізації обробки контента ASP.NET – Razor.»

**Виконав:**

студент 1 курсу, групи ІПЗс-20-1, варіант 12 Д.А.Ямборко

**Перевірив:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.М. Яшина

(підпис)

Хмельницький – 2020 р.

ОСНОВИ РОБОТИ З VISUAL STUDIO .NET. СТВОРЕННЯ WINDOWS-ДОДАТКІВ НА МОВІ C#

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

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

Завдання

12. Розробити Web -додаток ASP.NET MVC 4. Використати просту модель предметної області «Мобільні телефони». Виконати необхідні розрахунки.

Код Програми:

Phone.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Laba1.Models

{

public class Phone

{

public int Id { get; set; }

public string Mark { get; set; }

public string Model { get; set; }

public string Ozu { get; set; }

public int Price { get; set; }

}

}

Purchase.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Laba1.Models

{

public class Purchase

{

public int PurchaseId { get; set; }

public string Person { get; set; }

public string Address { get; set; }

public int productId { get; set; }

public DateTime Date { get; set; }

}

}

HomeController.cs

using Laba1.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace Laba1.Controllers

{

public class HomeController : Controller

{

ApplicationDbContext context = new ApplicationDbContext();

public ActionResult Index()

{

IEnumerable<Phone> phones = context.Phones;

ViewBag.Phones = phones;

return View();

}

public ActionResult About()

{

ViewBag.Message = "Your application description page.";

return View();

}

public ActionResult Contact()

{

ViewBag.Message = "Your contact page.";

return View();

}

}

}

BuyController.cs

using Laba1.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace Laba1.Controllers

{

public class BuyController : Controller

{

ApplicationDbContext context = new ApplicationDbContext();

// GET: Buy

[HttpGet]

public ActionResult Buy(int id)

{

ViewBag.ProductId = id;

Phone phone = context.Phones.Find(id);

if (phone != null)

{

return View(phone);

}

return View("Buy");

}

[HttpPost]

public string Buy(Purchase purchase)

{

purchase.Date = DateTime.Now;

context.Purchases.Add(purchase);

context.SaveChanges();

return "Спасибо," + purchase.Person + ", за покупку!";

}

}

}

Buy.cshtml

@{

ViewBag.Title = "Buy";

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>Покупка</title>

</head>

<body>

<div>

<h3>Форма оформлення покупки</h3>

<form method="post" action="">

<input type="hidden" value="@ViewBag.ProductId" name="ProductId" />

<table>

<tr>

<td><p>Введіть ім'я </p></td>

<td><input type="text" name="Person" /> </td>

</tr>

<tr>

<td><p>Введите адрессу :</p></td>

<td>

<input type="text" name="Address" />

</td>

</tr>

<tr><td><input type="submit" value="Відправити" /> </td><td></td></tr>

</table>

</form>

</div>

</body>

</html>

Index.cshtml

@{

ViewBag.Title = "Home Page";

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>Книжный магазин</title>

</head>

<body>

<div>

<h3>Распродажа книг</h3>

<table>

<tr>

<td><p>Название книги</p></td>

<td><p>Автор</p></td>

<td><p>Цена</p></td>

<td><p>Цена</p></td>

<td></td>

</tr>

@foreach (var b in ViewBag.Phones)

{

<tr>

<td><p>@b.Mark</p></td>

<td><p>@b.Model</p></td>

<td><p>@b.Ozu</p></td>

<td><p>@b.Price</p></td>

<td><p><a href="/Buy/Buy/@b.Id">Купить</a></p></td>

</tr>

}

</table>

</div>

</body>

</html>

GIT repo: <https://github.com/alsowentafk/1_semestr_knu/tree/master/PI/ASP_DIma-1/LR-2>

Результати роботи:

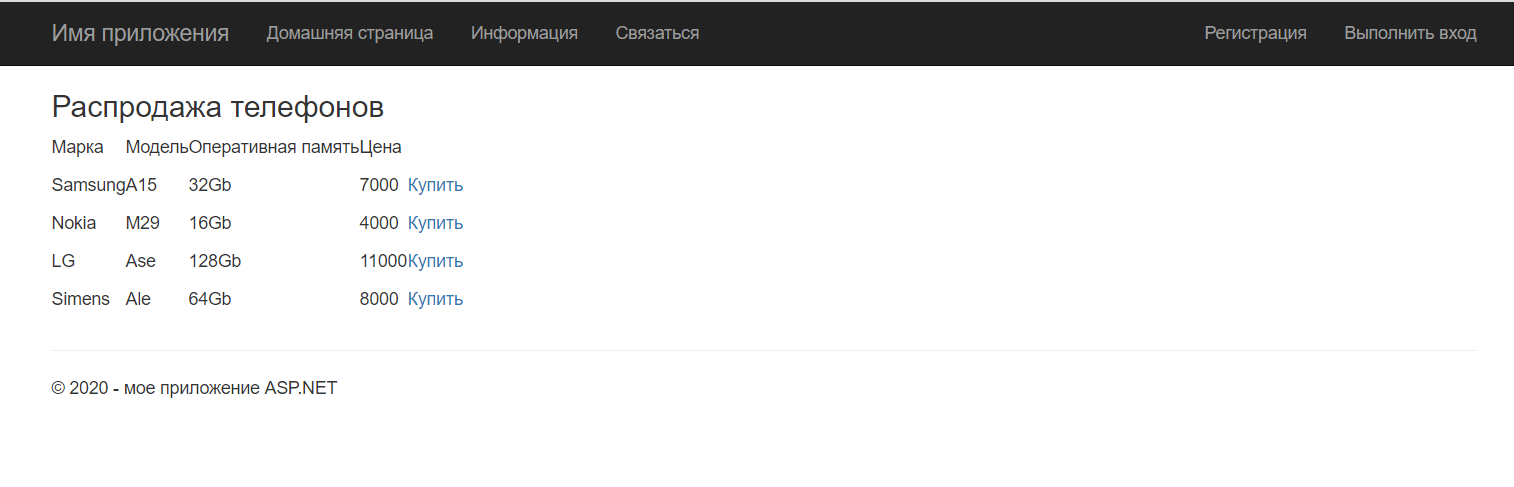


Рис 1.-Головна сторінка програми

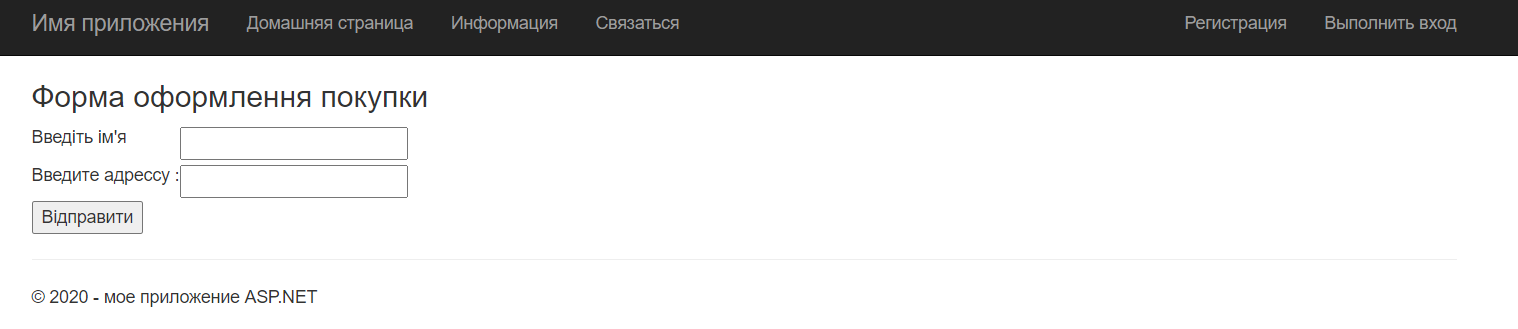


Рис 2.-Форма оформлення покупки

Контрольні питання:

1. Оператори Razor починають з символу "@". У цьому випадку оператор @model оголошує про тип об'єкта моделі, який буде передано до представлення з методу дії Це дозволяє посилатися на методи, поля та властивості об'єкта моделі перегляду за допомогою @Model, як показано в наведеному нижче прикладі, де показано просте доповнення до перегляду Index.cshtml:
2. Читання значення властивості в представленні

Оператори Razor починаються з символу @. У цьому випадку оператор @model оголошує тип об'єкта моделі, який буде 30 передаватиметься представленню з методу дії. Це дозволяє посилатися на методи, поля і властивості об'єкта моделі представлення з допомогою @Model.

Контролер одержує вхідні дані й перетворює їх на команди для моделі чи вигляду.

1. 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.Які функції виконує Представлення

Представлення може являти собою будь-яке представлення інформації, одержуване на виході, наприклад графік чи діаграму. Одночасно можуть співіснувати кілька виглядів (представлень) однієї і тієї ж інформаці

1. Виклик методу @RenderBody()

Виклик методу @RenderBody() вставляє в розмітку компонування контент представлення, зазначеного методом дії.

Перерахуйте шаблони додатки MVC та їх призначення

Пустий проект, ASP.Net MVC Веб-додаток,SST

1. 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.

1. Переваги застосування компонування

Компонування, по суті, є шаблоном і це допомогає нам звертатись до кожного іншого представлення безпосередньо в тому, якому ми знаходимось, так як в @RenderBody ми звертаємось в Layout до всіх представлень, щоб Layout весь час формував header і footer.

1. Сторінка, \_ViewStart.cshtml, призначена для більш гнучкого налаштування подань. Код для цієї сторінки запускається в коді будь-якого з подань в одному каталозі. Цей файл послідовно застосовується до кожного подання в одному каталозі.  
     
   Під вільний час створення проекту [ASP.NET](https://ssl.microsofttranslator.com/bv.aspx?ref=TAns&from=&to=uk&a=ASP.NET) MVC 3/4 в шаблоні Basic або Інтернет, каталог переглядів вже має файл \_ViewStart.cshtml. Цей файл ідентифікує головну сторінку, яка використовується за промовчанням. При створенні нового проекту [ASP.NET](https://ssl.microsofttranslator.com/bv.aspx?ref=TAns&from=&to=uk&a=ASP.NET) MVC 3/4 на порожній шаблон, нам потрібно додати нове подання \_ViewStart.cshtml в папку Перегляди і видалити весь раніше автоматично згенерований текст
2. Додавання до контролера нового методу дії

Для додавання нового методу потрібно перейти в контролер, додати туди клас який має мати тип ActionResult, після чого задати йому параметри і в фігурних дужках описати його дії.

1. Клацніть правою кнопкою миші на методі NameAndPrice в редакторі і виберіть в контекстному меню пункт Add View (Додати представлення), щоб відкрити діалогове вікно Add View (Додавання представлення). Відзначте прапорець Create a strongly – typed view (Створити строго типізоване представлення) і виберіть клас Product в списку Model class (Клас моделі). Відзначте прапорець Use a layout or master page (Використовувати компоновку або майстер - сторінку ).

Створення представлення, яке використовує компоновку Зверніть увагу на текст під прапорцем Use a layout or master page. Він говорить, що ви повинні залишити поле порожнім, якщо використовуване представлення було вказане у файлі запуску представлення. Якщо тепер клацнути на кнопці Add (Додати), представлення буде створено без оператора C#, що встановлює значення властивості Layout. Ми збираємося явно вказати представлення, тому клацніть на кнопці з трикрапкою (...) праворуч від текстового поля. Середовище Visual Studio відобразить діалогове вікно, яке дозволяє вибрати файл компонування.

1. Підключення файла компонування

Застосування компонування. Щоб застосувати компоновку до представлення, потрібно всього лише встановити значення властивості Layout.

1. Будь-які елементи в компонуванні будуть застосовуватися до будь-якого представлення, яке використовує цю компоновку, і саме тому компонування є, по суті, шаблоном.
2. Ролі, виконувані методом дії та представленням

Метод дії виконує роль зв’язку моделей з представленням і описує всі функції які мають виконуватись в представленнях.

Представлення виводить всю інформацію, яка сформована в контролерах і формує зовнішній вигляд сторінки.

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

Вставка значень даних. Найпростіше, що можна робити за допомогою виразу Razor - це вставка значень даних в розмітку. Це здійснюється з використанням вираження @Model, дозволяє посилатися на властивості і методи, які визначені об'єктом моделі представлення, або вираз @ViewBag для посилання на властивості, визначенні динамічно за допомогою засобу ViewBag. Додамо в контролер Home новий метод дії на ім'я DemoExpressions, який передає дані представленням із застосуванням об'єкта моделі та об'єкта ViewBag.

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

Оператори Razor починаються з символу @. Наприклад для виводу елементів моделі потрібно звернутись до моделі. Для цього потрібно ввести @model Проект.Model.Модель.

1. Робата зобластями контенту, які позначені за допомогою символу @. При використанні елементів коду С# в представленнях ми використовували знак @. Призначення, підключення файла Site.css

Задає css стилі для роботи представленнь

1. Використання оператора if в представленні Razor

Для викорстання оператора if потрібно в коді написати @if(Перевірка чого){Що виконується при істині} else(якщо потрібно) {Що виконується при хибній}.

1. 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.
2. Ручна установка типу моделі 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

1. 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.

Робота з просторами імен. В останньому прикладі для посилання на Product в циклі foreach використовується повністю певне ім'я:

1. 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)