ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет програмування та комп’ютерних

і телекомунікаційних систем

Кафедра інженерії програмного забезпечення

**Лабораторна робота № 1**

з дисципліни «Програмування Інтернет»

на тему:

**«Створення примітивного додатка на ASP.NET MVC 4»**

**Виконав:**

студент 3 курсу, групи ІПЗ-17-1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В.Охота

(підпис) (Ініціали, прізвище)

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

(підпис)

Хмельницький – 2020 р.

**Тема.** Створення примітивного додатка на ASP.NET MVC 4

**Мета.** Створити примітивний додаток на MVC 4, який покликаний дати деяке початкове розуміння роботи MVC. Як шаблон проекту використовується Empty і проектується міні-додаток з нуля.

**Завдання.** Варіант 11. Розробити web – додаток, що буде представляти примітивну емуляцію салону по продажу автомобілів, в якому ми зможемо переглядати список автомобілів і оформляти покупку.

**Хід роботи**

1. Відкриваємо Visual Studio, переходимо в діалогове вікно створення проекту (меню File -> New Project…). Переходимо на вкладку Web і вибираємо шаблон ASP.NET NF Виберемо шаблон MVC та створюємо проект.
2. Створимо модель нашої ПО.

Для нашого завдання ми можемо виділити дві області даних: інформація про автомобіль та інформація. Створимо два класа в папці «Models»:

**Клас Car**:

namespace Car\_dealership.Models

{

public class Сars

{

// ID автомобіля

public int Id { get; set; }

// назва автомобіля

public string Name { get; set; }

// марка автомобіля

public string Brand { get; set; }

// ціна

public int Price { get; set; }

}

}

**Клас Purchase** (для оформлення покупки):

public class Purchase

{

// ID покупки

public int PurchaseId { get; set; }

// ім’я і прізвище покупця

public string Person { get; set; }

// адреса покупця

public string Address { get; set; }

// ID книги

public int BookId { get; set; }

// дата покупки

public DateTime Date { get; set; }

}

}

Далі у наш проекти ми імпортуємо з магазину бібліотек «EntityFramework», та створюємо новий клас CarsContext, він потрібний для полегшеної праці з базою даних

**Клас CarsContext:**

using System.Data.Entity;

namespace Car\_dealership.Models

{

public class CarsContext : DbContext

{

public DbSet<Сars> Cars { get; set; }

public DbSet<Purchase> Purchases { get; set; }

}

}

Щоб створити контекст, нам треба наслідувати новий клас від класу *DbContext*. Властивості типу public *DbSet <Book> Books {get; set; }* Допомагають отримувати набір даних певного типу (в даному випадку типу *Book*). У результаті EntityFramework сам візьме на себе відповідальність по створенню бази даних. Тобто в даному випадку ми будемо використовувати так званий підхід Code First - у нас є моделі, і по них фреймворк буде створювати таблиці в базі даних.

Нарешті, вкажемо рядок підключення. Відкриємо файл web.config, знайдемо секцію configSections і після неї вставимо секцію connectionStrings:

<connectionStrings>

<add name="CarsContext" connectionString="Data Source =(LocalDB)\v11.0;AttachDbFilename='|DataDirectory|\Carsstore.mdf';Integrated Security=True" providerName="System.Data.SqlClient" />

</connectionStrings>

**Створення контролера і представлень.** Оскільки з моделями ми завершили роботу, перейдемо до найбільш важливої частини додатки - до контролера. Для контролерів в проекті вже є папка *Controllers*.

Створюємо контролер HomeController він потрібний для пов’язання моделі та користувацького інтерфейсу

public class HomeController : Controller

{

// GET: Home

Models.CarsContext db = new Models.CarsContext();

public ActionResult Index()

{

//Отримка даних з БД

IEnumerable<Models.Сars> cars = db.Cars;

//Передача всіх об'єктів в динамічну властивість Cars в ViewBag

ViewBag.Cars = cars;

//Повертаємо приставлення

return View();

}

}

}

**Дані для моделей за замовчуванням.** Ми можемо вже зараз запустити наш додаток, однак крім рядка із заголовками в таблиці ми нічого не побачимо, тому що у нас немає даних. Дану задачу можливо вирішити двома способами: додати в базу даних MSSQL деякі дані, або ініціалізувати початкові дані для моделей в коді. Виберемо другий варіант. Для цього в папку Models додамо новий клас CarDbInitializer і додамо в нього наступний код:

public class CarDbInitializer:DropCreateDatabaseAlways<CarsContext>

{

protected override void Seed(CarsContext db)

{

db.Cars.Add(new Сars { Name = "X5", Brand = "BMW", Price = 78968 });

db.Cars.Add(new Сars { Name = "C Class", Brand = "Mercedes Benz", Price = 90000 });

db.Cars.Add(new Сars { Name = "Vivaro", Brand = "Opel", Price = 45000 });

base.Seed(db);

}

}

}

Даним класом ми створюємо три об’єкта і додаємо їх в базу даних. Щоб цей клас запрацював, нам треба не тільки його оголосити, але і запустити. Всі початкові налаштування і конфігурації у нас зберігаються у файлі Global.asax. Відкриємо його і додаємо в нього в метод Application\_Start, який відпрацьовує при старті програми, наступний рядок:

Database.SetInitializer(new CarDbInitializer());

Створимо метод Buy, який буде відповідати за обробку введення користувача при оформленні покупки. Додамо в контролер HomeController наступні два методи:

[HttpGet]

public ActionResult Buy(int id) {

ViewBag.CarsID = id;

return View();

}

[HttpPost]

public string Buy(Purchase purchase)

{

purchase.Date = DateTime.Now;

// Додаємо інформацію про покупку в базу даних

db.Purchases.Add(purchase);

// Зберігаємо в БД всі зміни

db.SaveChanges();

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

}

Та створимо найпростіший frontend (Візіолізацію нашого проекту).

Зробимо стилізацію додатка:

body {

font-size: 13px;

font-family: Verdana,Arial,Arial, Helvetica, Sans-Serif;

background-color: f7f7fa;

padding-left: 40px;

}

nav{

display:block;

}

.menu{

padding-left:10px;

}

.menu ul {

list-style:none;

}

.menu li {

display:inline;

}

.menu a:hover{

color:aqua;

}

table{

vertical-align:middle;

text-align:left;

}

.header{

font-weight:bold;

}

td{

padding-right:10px;

}

input{

width: 150px;

}

Оскільки ми маємо кілька сторінок нашого проекту і вони поині бути ідентично оформлені, то створимо Майстра-сторінок:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>@ViewBag.Title</title>

<link href="@Url.Content("~/Content/Site.css")" rel="stylesheet"

type="text/css" />

</head>

<body>

<nav>

<ul class="menu">

<li>@Html.ActionLink("Головна", "Index", "Home")</li>

</ul>

</nav>

@RenderBody()

</body>

</html>

Далі напишемо код для відображення головної сторінки та оформлення покупки:

Головна сторінка:

@{

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<div>

<h3>Автомагазин</h3>

<table>

<tr>

<td><p>Модель</p></td>

<td><p>Марка</p></td>

<td><p>Цiна</p></td>

<td></td>

</tr>

@foreach (var b in ViewBag.Cars)

{

<tr>

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

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

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

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

</tr>

}

</table>

</div>

Оформлення покупки:

@{

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<div>

<h3>Форма оформлення покупки</h3>

<form method="post">

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

<table>

<tr>

<td><p>Введіть своє ім'я </p></td>

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

</td> </tr> <tr>

<td><p>Введіть адресу :</p></td>

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

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

</td><td></td></tr>

</table>

</form>

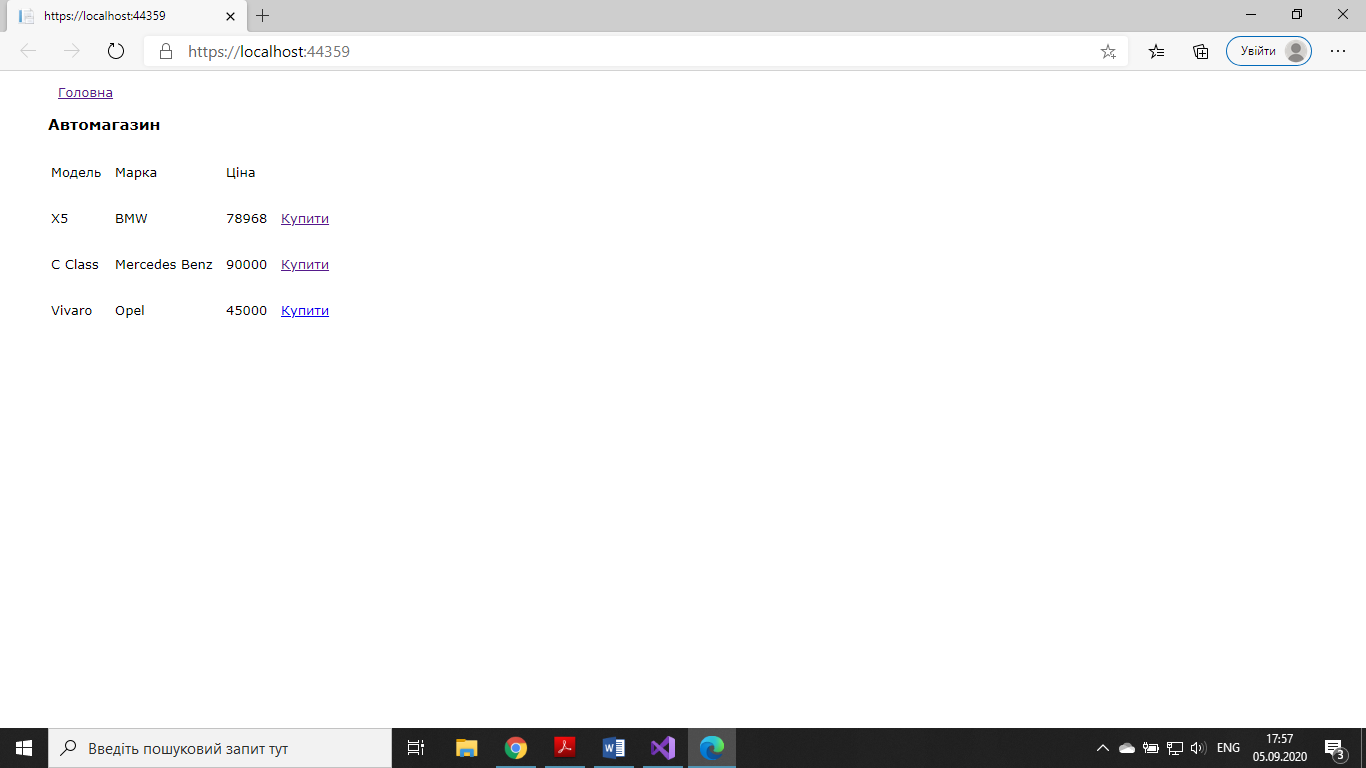
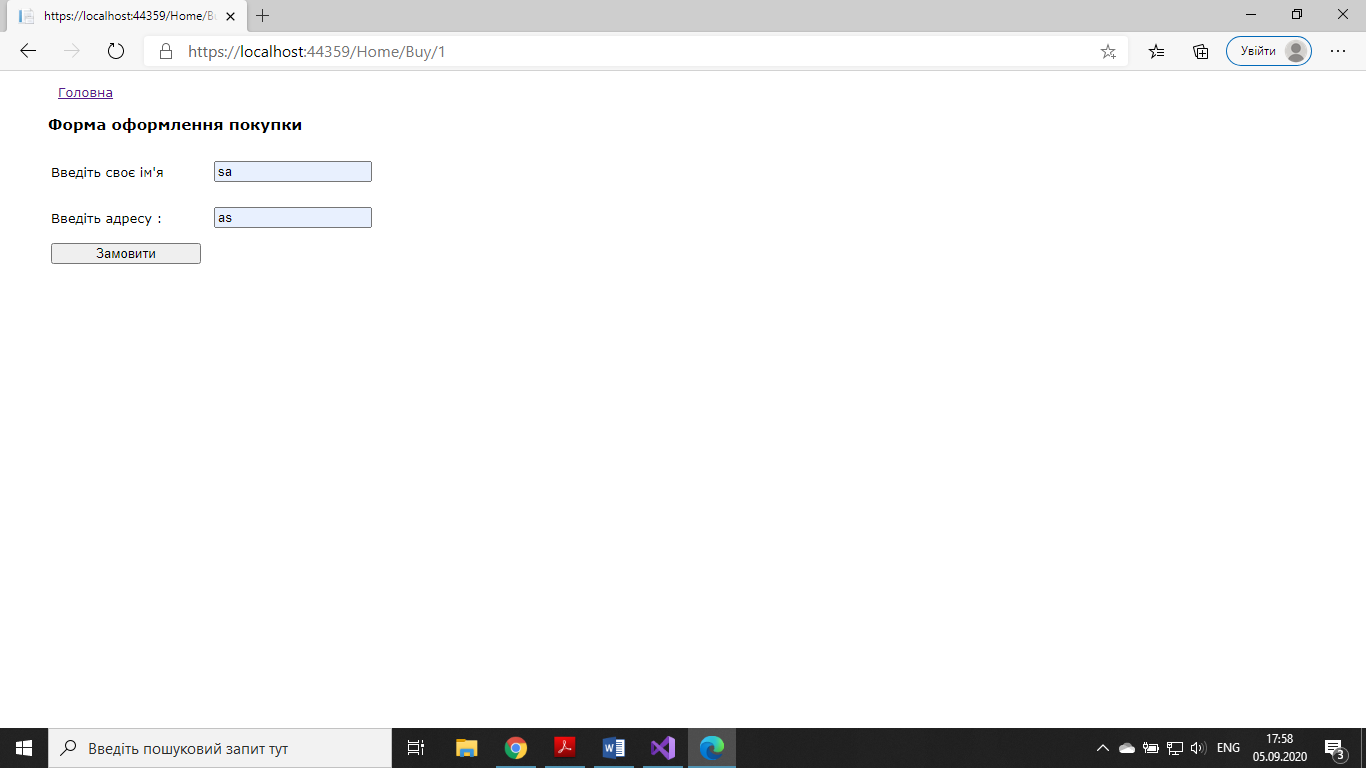
</div>

Рисунок 1 Головна сторінка

Рисунок 2 Форма оформлення покупки

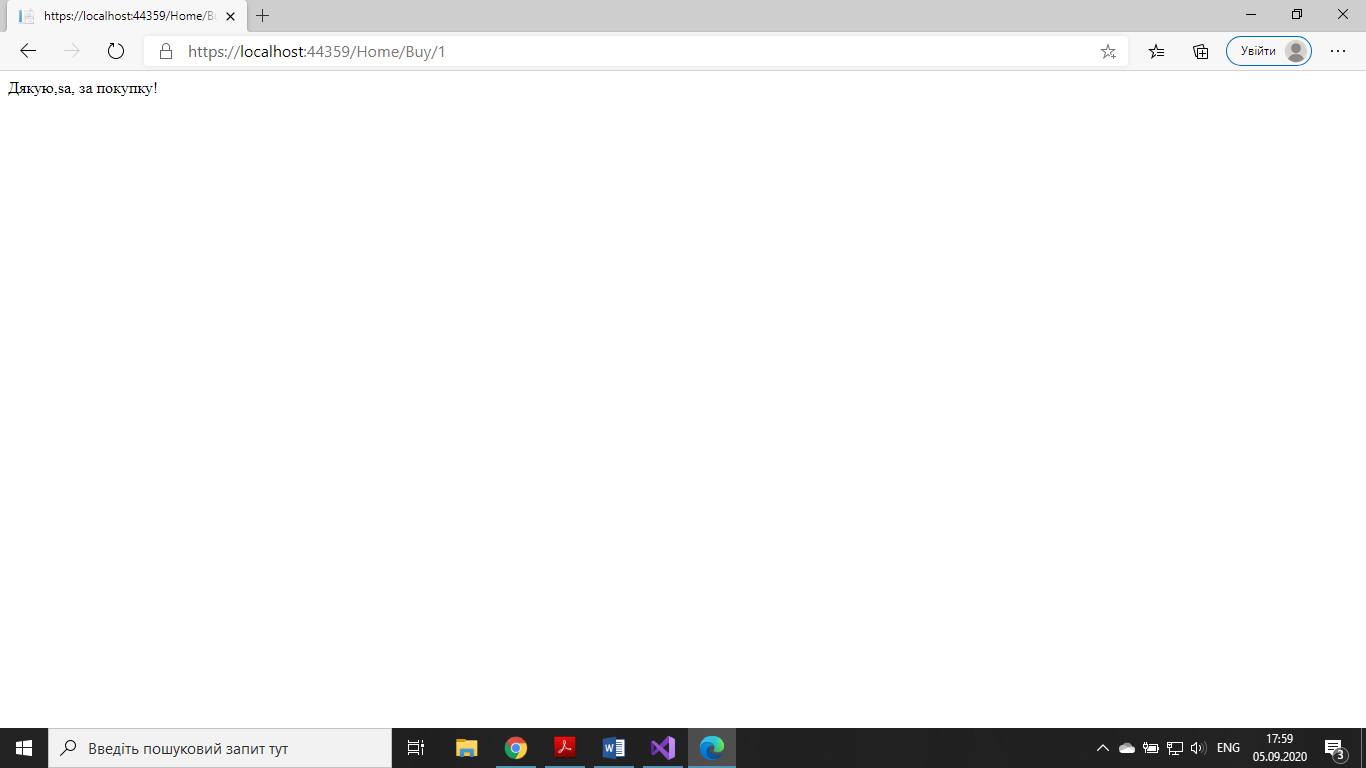


Рисунок 3 Відповідь проекту після покупки

**Контрольні запитання.**

1. **Поняття технології MVC.**

ASP.NET MVC є фреймворком для розробки від Microsoft, який поєднує в собі ефективність і акуратність архітектури MVC, найсучасніші ідеї та методи гнучкої розробки та кращі властивості існуючої платформи ASP.NET. Це альтернатива традиційним ASP.NET Web Forms, яка забезпечує істотну перевагу для всіх, крім найпростіших і тривіальних, проектів веб-розробки. Незважаючи на те, що Microsoft розробляв інструменти та фреймворки для веб-розробки протягом вже досить тривалого періоду, ASP.NET MVC стала проривом, оскільки, на відміну від попередніх розробок, робить наголос на чистий код, концепцію поділу і тестування.

1. **Що таке шаблон MVC**

Шаблон MVC, що лежить в основі нової платформи, передбачає взаємодію трьох компонентів: контролера (controller), моделі (model) і представлення (view).

1. **Які функції виконує Контролер**

Контролер (controller) представляє клас, з якого власне і починається робота програми. Цей клас забезпечує зв'язок між моделлю і представленням. Отримуючи дані, що вводяться користувачем, контролер виходячи з внутрішньої логіки при необхідності звертається до моделі і генерує відповідне представлення.

1. **Які функції виконує Представлення**

Представлення (view) - це власне візуальна частина або користувальницький інтерфейс додатка - наприклад, html-сторінка, через яку користувач, що зайшов на сайт, взаємодіє з веб-додатком.

1. **Які функції виконує Модель**

Модель (model) являє набір класів, що описують логіку використовуваних даних.

1. **Перерахуйте шаблони додатки MVC та їх призначення**

Internet Application. Даний шаблон представляє деяку базову функціональність, наприклад, управління обліковими записами, яку буде містити новий додаток.

Intranet Application. Даний шаблон схожий на шаблон Internet Application. Відмінність полягає в тому, що в Intranet Application для управління обліковими записами використовується аутентифікація Windows, а не система ASP.NET Membership.

Empty. Цей шаблон представляє абсолютно чистий додаток без будь-яких файлів і структури папок.

Basic. Цей шаблон також представляє порожній проект. Правда, він містить базові каталоги, CSS, інфраструктуру додатки MVC.

Mobile Application. Це новинка MVC 4, призначена для створення веб-додатків для мобільних платформ.

Web API. Призначений для створення сервісів і для роботи з фреймворком Web API.

Single Page Application. Шаблон, призначений для створення 6 сайтів-односторінкових.

Facebook Application. Призначений для створення веб-додатків під соціальну мережу Facebook.

1. **Структура проекту MVC**

App\_Data. У цій папці зберігаються всі необхідні файли і ресурси, наприклад, бази даних, що використовуються додатком. Після розгортання програми лише безпосередньо додаток може працювати з цією папкою, доступ простих користувачів в цю папку заборонений.

Файл Global.asax і папка App\_Start. У MVC 4 була додана папка App\_Start. Вона включає весь функціонал конфігурації програми, який в попередніх версіях містився у файлі Global.asax, а тепер перенесений в набір статичних класів, що викликаються в Global.asax. Ці статичні класи містять деяку логіку ініціалізації програми, виконуються при запуску.

Файл Web.config. Файл конфігурації програми, який знаходиться в кореневій теці програми

Content. Містить деякі допоміжні файли, які не включають код на С# або javascript, і які розгортаються разом з додатком. Зокрема, тут можуть розміщуватися файли стилів CSS. Так, в цій папці ви побачите файл Site.css, який містить стилі додатка, а також папку з темами, що включає стилі CSS та зображення для певних тем.

Controllers. Містить контролери - класи, що відповідають за роботу програми. За замовчуванням тут знаходяться два контролера - HomeController і AccountController.

Папки Images і Scripts. Папки Images і Scripts містять відповідно зображення і скрипти на JavaScript, використовувані в додатку. За замовчуванням ці папки вже містять файли, зокрема, в папку Scripts вже поміщені файли бібліотеки jQuery.

Models. Містить моделі, використовувані додатком. За замовчуванням тут визначена одна модель - AccountModel, яка представляє окремий обліковий запис.

Views. Тут розміщуються представлення. Представлення згруповані по папках, кожна з яких відповідає одному контролеру. Після отримання та обробки запиту контролер, відправляє одне з цих представлень, заповнених деякими даними, клієнту. Крім того, є папка загальних для контролерів представлень - папка Shared\

1. **Роль бібліотеки EntityFramework**

Створення контексту даних і EntityFramework. Для більш зручного доступу до даних рекомендується використовувати фреймворк Entity Framework, хоча його використання необов'язково і цілком залежить від уподобань розробника. Цей фреймворк дозволяє абстрагуватися від структури конкретної бази даних і вести всі операції з даними через модель.

1. **Призначення контексту класа**

Він потрібен для полегшення доступу до БД на основі деякої моделі.

1. **Рядок підключення ConnectionStrings**

<connectionStrings>

<add name="CarsContext" connectionString="Data Source =(LocalDB)\v11.0;AttachDbFilename='|DataDirectory|\Carsstore.mdf';Integrated Security=True" providerName="System.Data.SqlClient" />

</connectionStrings> 12

Тим самим ми визначаємо шлях до бази даних, яка потім буде створюватися.

1. **Призначення об'єкта ViewBag**

Об'єкт ViewBag є таким об'єктом, який передається в представлення. І ми можемо визначити в цьому об'єкті будь-яку змінну і передати їй деяке значення, а потім в представленні витягти це значення.

1. **Основи маршрутизації**

Щоб викликати контролер HomeController або відправити йому запит, нам треба вказати в рядку запиту його ім'я - Home. Крім того, після імені контролера нам треба через слеш вказати дію або метод контролера, до якого відправляється запит. За замовчуванням при запуску проекту або при зверненні до сайту система MVC буде викликати дію Index контролера HomeController, якщо ми не вкажемо інший маршрут за замовчуванням в параметрах маршрутизації. Шлях /Home/Buy/@b.Id означає, що ми будемо звертатися до методу Buy контролера HomeController.

1. **Дані для моделей за замовчуванням**

Ми можемо вже зараз запустити наш додаток, однак крім рядка із заголовками в таблиці ми нічого не побачимо, тому що у нас немає даних. Дану задачу можливо вирішити двома способами: додати в базу даних MSSQL деякі дані, або ініціалізувати початкові дані для моделей в коді.

1. **Призначення файла Global.asax**

Файл Global.asax і папка App\_Start. У MVC 4 була додана папка App\_Start. Вона включає весь функціонал конфігурації програми, який в попередніх версіях містився у файлі Global.asax, а тепер перенесений в набір статичних класів, що викликаються в Global.asax. Ці статичні класи містять деяку логіку ініціалізації програми, виконуються при запуску.

1. **Призначення, підключення файла Site.css**

файл Site.css, який містить стилі додатка, а також папку з темами, що включає стилі CSS та зображення для певних тем.

<head>

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

<link type="text/css" rel="stylesheet" href="../../Content/Site.css" />

</head>

1. **Призначення, підключення майстер сторінки**

Майстер-сторінка задає єдиний шаблон для інших представлень. Отже, створимо майстер-сторінку. По суті це просте представлення. Спочатку додамо в папку Views проекту нову папку Shared.

Layout = "~/Views/Shared/\_Layout.cshtml";

1. **Призначення секції RenderBody ()**

Наприкінці йде метод RenderBody (), при виклику ресурсу в цю частину майстер-сторінки і будуть підставлятися вміст інших представлень.