ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет програмування та комп'ютерних
і телекомунікаційних систем

Кафедра інженерії програмного забезпечення

# Лабораторна робота № 3

з дисципліни «Програмування Інтернет»
на тему:

**«Моделі з простою структурою. Визначення таблиць. Створення логіки додатка. Виконання операцій з простими моделями - редагування, видалення, додавання.»**

**Виконав:**

студент 3 курсу, групи ІПЗ-18-1 _____ В.В.Охота
(підпис) (Ініціали, прізвище)

**Перевірив:** _____ О.М. Яшина
(підпис)

Хмельницький – 2020 р.

**Мета.** Отримати навики розробки Web - додатків з простими моделями..

**Завдання.** Варіант 10..

## Хід роботи

### Створюємо моделі

Модель Car

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace car.Models
{
    public class Car
    {
        public int Id { get; set; }
        public string Company { get; set; }
        public string Model { get; set; }
        public int Price { get; set; }
    }
}
```

Модель CarContext

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Web;

namespace car.Models
{
    public class CarContext : DbContext
    {
        public DbSet<Car> Cars { get; set; }
        public DbSet<Purchase> Purchases { get; set; }
    }
}
```

Модель Purchase

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace car.Models
{
    public class Purchase
    {
        public int PurchaseId { get; set; }
        public string Person { get; set; }
        public string PurchaseDate { get; set; }
        public string CarId { get; set; }
        public int ShopId { get; set; }
    }
}
```

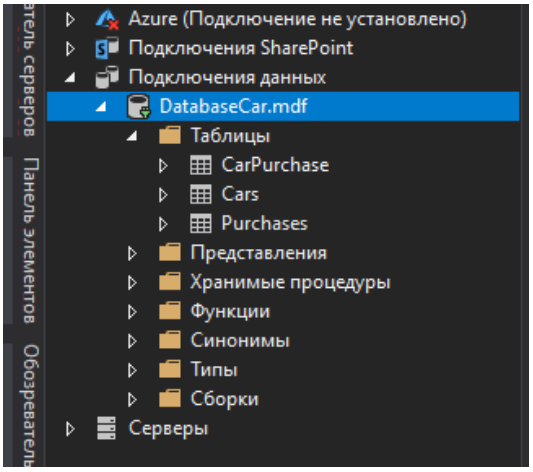
### Створюємо контролер

```
using car.Models;
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
```

```
using System.Web;
using System.Web.Mvc;

namespace DTPReg.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        CarContext db = new CarContext();
        public ActionResult Index()
        {
            return View(db.Purchases);
        }
        public ActionResult MobileView(int id)
        {
            var purch = db.Purchases.Find(id);
            if (purch != null)
            {
                return View(purch);
            }
            return RedirectToAction("Index");
        }

        [HttpGet]
        public ActionResult Edit(int? id)
        {
            if (id == null)
            {
                return HttpNotFound();
            }
            Purchase purch = db.Purchases.Find(id);
            if (purch == null)
            {
                return HttpNotFound();
            }
            return View(purch);
        }
        [HttpPost]
        public ActionResult Edit(Purchase purch)
        {
            db.Entry(purch).State = EntityState.Modified;
            db.SaveChanges();
            return RedirectToAction("Index");
        }

        [HttpGet]
        public ActionResult Create()
        {
            return View();
        }
        [HttpPost]
        public ActionResult Create(Purchase purch)
        {
            db.Purchases.Add(purch);
            db.SaveChanges();
            return RedirectToAction("Index");
        }

        [HttpGet]
        public ActionResult Delete(int? id)
        {
            if (id == null)
            {
                return HttpNotFound();
            }
            Purchase purch = db.Purchases.Find(id);
            if (purch == null)
            {
```

```
                return HttpNotFound();
            }
            return View(purch);
        }
        [HttpPost, ActionName("Delete")]
        public ActionResult DeleteConfirmed(int? id)
        {
            if (id == null)
            {
                return HttpNotFound();
            }
            Purchase purch = db.Purchases.Find(id);
            if (purch == null)
            {
                return HttpNotFound();
            }
            db.Purchases.Remove(purch);
            db.SaveChanges();
            return RedirectToAction("Index");
        }
    }
}
```

## Створюємо БД




## Створюємо Представлення

### Представлення Index

```
@model IEnumerable<car.Models.Purchase>
@{
    ViewBag.Title = "Автомагазин";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<div>
    <h3>Замовлення</h3>
    <table>
        <tr class="header">
            <td><p>Номер замовлення  </p></td>
            <td><p>  ПІП  </p></td>
            <td><p>  Дата замовлення  </p></td>
            <td><p>  Модель автомобіля  </p></td>
            <td><p>  Номер автосалону  </p></td>
            <td></td>
        </tr>
        @foreach (car.Models.Purchase b in Model)
```

```
        {
            <tr>
                <td><p>"@b.PurchaseId"  </p></td>
                <td><p>  "@b.Person"  </p></td>
                <td><p>  "@b.PurchaseDate"  </p></td>
                <td><p>  "@b.CarId"  </p></td>
                <td><p>  "@b.ShopId"  </p></td>
                <td><p><a href="/Home/Edit/@b.PurchaseId">Редагувати  </a></p></td>
                <td><p><a href="/Home/Delete/@b.PurchaseId">Видалити  </a></p></td>
            </tr>
        }
    </table>
</div>
```

## Представлення Create

```
model car.Models.Purchase
@{
    ViewBag.Title = "Створити замовлення";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<h2>Нове замовлення</h2>
@using (Html.BeginForm())
{
    <b>@Html.LabelFor(model => model.Person, "ПІП")</b>
    <br />
    @Html.EditorFor(model => model.Person)
    <br />
    <br />
    <b>@Html.LabelFor(model => model.PurchaseDate, "Дата замовлення")</b>
    <br />
    @Html.EditorFor(model => model.PurchaseDate)
    <br />
    <br />
    <b>@Html.LabelFor(model => model.CarId, "Модель Авто")</b>
    <br />
    @Html.EditorFor(model => model.CarId)
    <br />
    <br />
    <b>@Html.LabelFor(model => model.ShopId, "Номер автосалону")</b>
    <br />
    @Html.EditorFor(model => model.ShopId)
    <br />
    <br />
    <input type="submit" value="Додати" />
}
```

## Представлення Delete

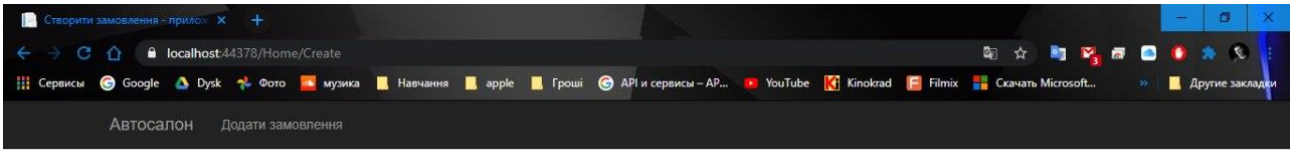
```
@{
    ViewBag.Title = "Видалити замовлення";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
@model car.Models.Purchase
<h2>Видалення замовлення</h2>
<dl>
    <dt><b>ПІП</b></dt>
    <dd>
        @Html.DisplayFor(model => model.Person)
    </dd>
    <dt><b>Дата замовлення</b></dt>
    <dd>
        @Html.DisplayFor(model => model.PurchaseDate)
    </dd>
```

```
    <dt><b>Код авто</b></dt>
    <dd>
        @Html.DisplayFor(model => model.CarId)
    </dd>
    <dt><b>Номер салону</b></dt>
    <dd>
        @Html.DisplayFor(model => model.ShopId)
    </dd>
</dl>
@using (Html.BeginForm())
{
    <input type="submit" value="Видалити" />
}
```

## Представлення Edit

```
@{
    ViewBag.Title = "Редагувати замовлення";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
@model car.Models.Purchase
<h2> Замовлення №@Model.PurchaseId</h2>
@using (Html.BeginForm("Edit", "Home", FormMethod.Post))
{
    <fieldset>
        @Html.HiddenFor(m => m.PurchaseId)
        <p>
            <b>@Html.LabelFor(m => m.Person, "ПІП")</b>
            <br />
            @Html.EditorFor(m => m.Person)
        </p>
        <p>
            <b>@Html.LabelFor(m => m.PurchaseDate, "Дата замовлення")</b>
            <br />
            @Html.EditorFor(m => m.PurchaseDate)
        </p>
        <p>
            <b>@Html.LabelFor(m => m.CarId, "Код телефону")</b>
            <br />
            @Html.EditorFor(m => m.CarId)
        </p>
        <p>
            <b>@Html.LabelFor(m => m.ShopId, "Номер салону")</b>
            <br />
            @Html.EditorFor(m => m.ShopId)
        </p>
        <p><input type="submit" value="Відправити" /></p>
    </fieldset>
}
```

## Скріншоти виконання роботи коду




Автосалон Додати замовлення

## Нове замовлення

**ПІП**

Охота В.В.

**Дата замовлення**

30.10.2020

**Модель Авто**

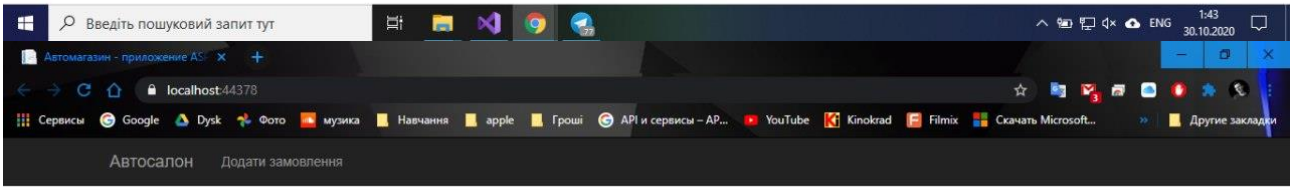
Tesla Model S

**Номер автосалону**

1

Додати



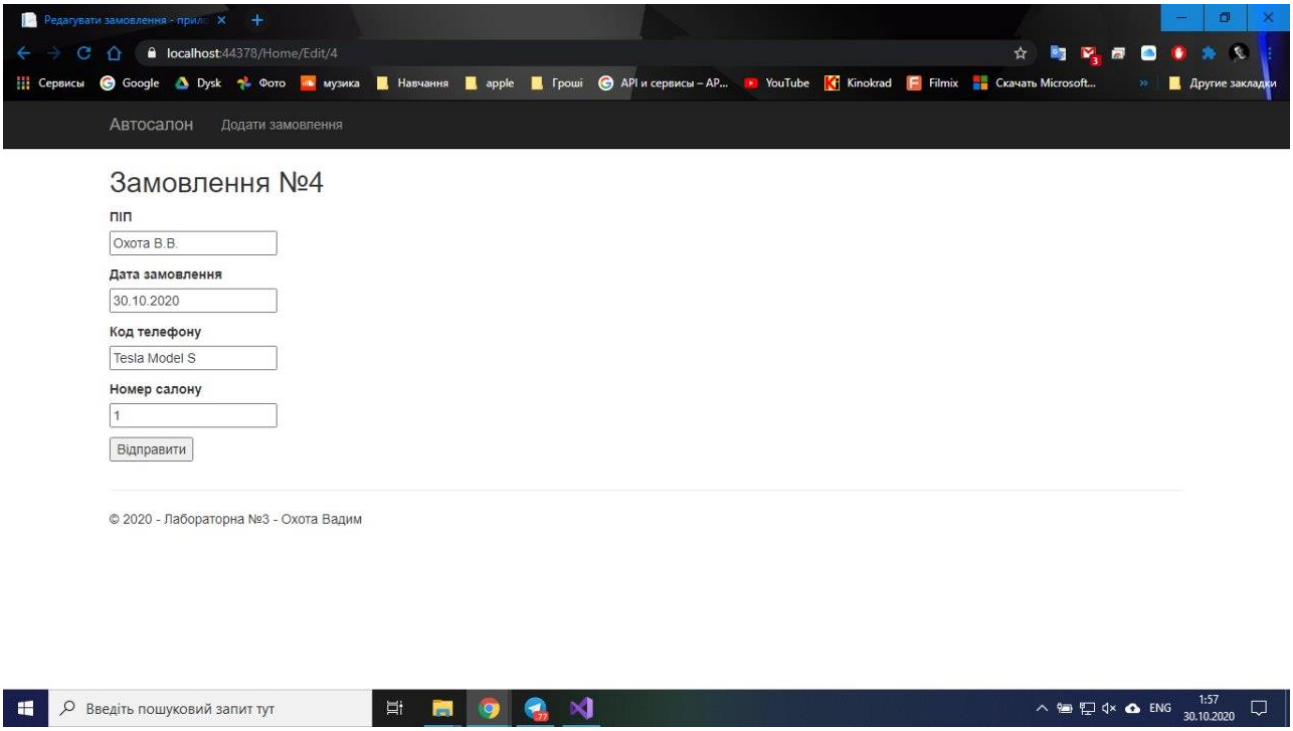



Автосалон Додати замовлення

## Замовлення

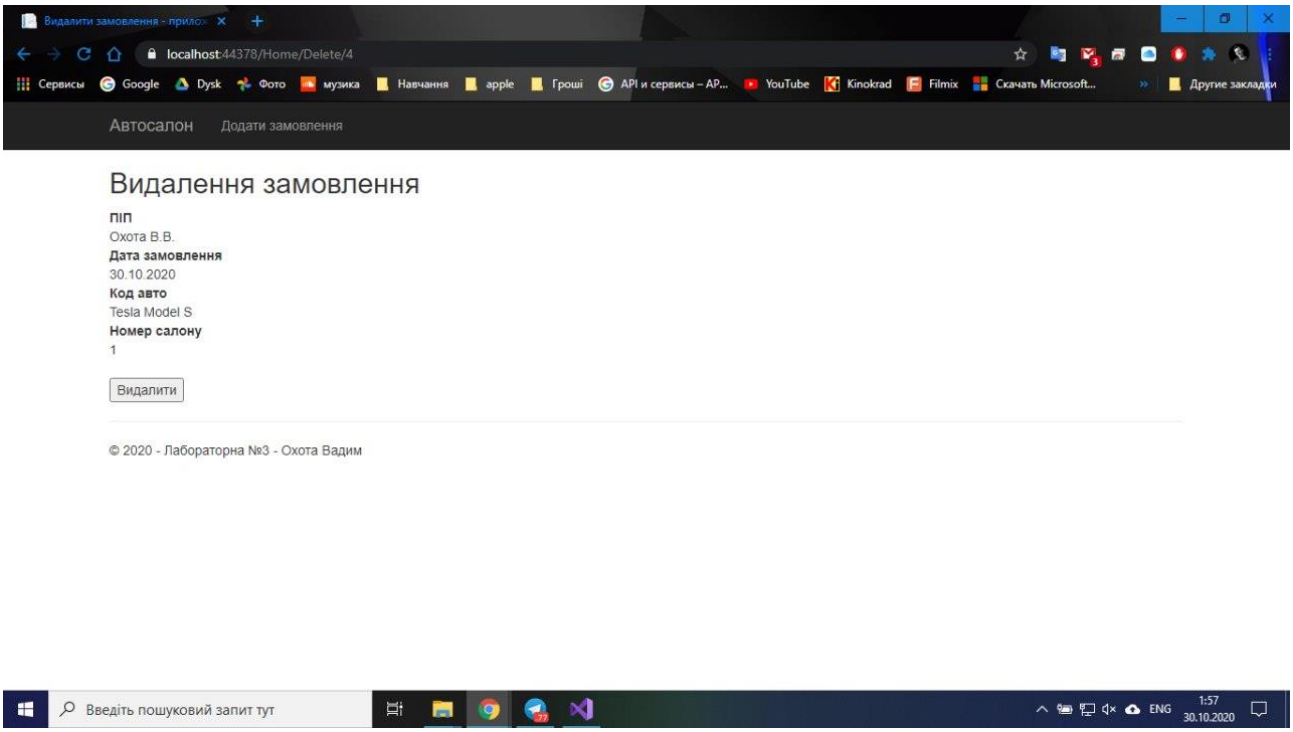
| Номер замовлення | ПІП | Дата замовлення | Модель автомобіля | Номер автосалону | | | |
|---|---|---|---|---|---|---|---|
| "4" | "Охота В.В." | "30.10.2020" | "Tesla Model S" | "1" | Перегляд | Редагувати | Видалити |















## Контрольні питання

1. **Моделі в ASP.NET MVC 4**

   Моделі описують логіку даних

2. **Взаємодія з базою даних**

   Щоб взаємодіяти з БД, нам потрібен клас контексту даних, нехай це буде наступний клас.
   По-друге, визначимо рядок підключення до БД.

3. **Контекст даних в ASP.NET MVC 4**

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

4. **Шаблонні хелпери**

Шаблонні хелпери. Фреймворк MVC має шаблонні хелпери. Вони більш гнучкі в порівнянні з html-хелперами, так як в цьому випадку нам не треба думати, який нам треба створити елемент розмітки і який для цього вибрати хелпер. Ми просто повідомляємо шаблонному хелперу, яку властивість моделі ми хочемо використовувати, а фреймворк вже сам вибирає, який html-елемент згенерувати, виходячи з типу властивості і його метаданих

5. **Шаблонні хелпери для певної моделі**

- Display - створює елемент розмітки, який доступний тільки для читання, для зазначеного властивості моделі: Html.Display ("Name");

- DisplayFor - строго типізований аналог хелпера Display: Html.DisplayFor (e => e.Name);

- Editor - створює елемент розмітки, який доступний для редагування, для зазначеної властивості моделі: Html.Editor ("Name"); 52

- EditorFor - строго типізований аналог хелпера Editor: Html.EditorFor (e => e.Name);

- DisplayText - створює вираз для зазначеного властивості моделі у вигляді простого рядка: Html.DisplayText ("Name");

- DisplayTextFor - строго типізований аналог хелпера DisplayText: Html.DisplayTextFor (e => e.Name). Крім даних шаблонів, які використовуються для окремої властивості моделі, є ще кілька шаблонів, які дозволяють згенерувати разом всі поля для певної моделі:

- DisplayForModel - створює поля для читання для всіх властивостей моделі: Html.DisplayForModel ();

- EditorForModel - створює поля для редагування для всіх властивостей моделі: Html.EditorForModel ()

6. **Процес створення БД і підключення до неї**

Ми можемо створити базу даних рівнозначним чином і на сервері. Після цього база даних додається в проект, і ми можемо побачити її в папці App_Data. Тепер в оглядачі баз даних (вікно Database Explorer) ми можемо підключитися до неї і створити таблиці, які будуть зберігати дані. Розкриємо вузол Bookstore.mdf і знайдемо вузол Tables. Натиснемо на цей вузол правою кнопкою миші і в меню виберемо пункт Add New Table. І перед нами з'явиться вікно, в якому нам треба визначити назви і типи стовпців нової таблиці. За угодами про найменуваннях таблиці при роботі з Entity Framework повинні відповідати імені моделі.

7. **Визначення рядка підключення**

connectionStrings.

Тим самим ми визначаємо шлях до бази даних, яка потім буде створюватися. Вираз | DataDirectory | представляє заступник, який вказує, що база даних буде створюватися в проекті в папці App_Data.

8. **Файл Web.config**

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

9. **Використання підстановки | DataDirectory |**

   Вираз | DataDirectory | представляє заступник, який вказує, що база даних буде створюватися в проекті в папці App_Data

10. **Редагування моделі**

    Редагування моделі. Розглянемо, як зробити саму логіку редагування моделі. Нехай в деякій дії контролера ми отримуємо об'єкт моделі по Id і виводимо її поля для редагування в представленні:

    ```
    [HttpGet]
    public ActionResult EditBook(int? id)
    { if (id == null) {
     return HttpNotFound();
    }
    Book book = db.Books.Find(id);
     if (book == null) { return HttpNotFound(); } 59 return View(book); }
    ```

11. **Видалення моделі**

    Видалення моделі. Тепер найважливіша частина - видалення моделі. Навіть не в плані реалізації, скільки в плані безпеки. Додамо просту дію, яке видаляє модель з бази даних:

    ```
    public ActionResult Delete(int id) {
     Book b = db.Books.Find(id); 63 if (b != null) {
     db.Books.Remove(b); db.SaveChanges()
    ; } return RedirectToAction("Index");
    }
    ```

12. **Вставка моделі**

    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.

13. **Хелпер Html.HiddenFor**

    Приховує поле відображення

14. **Get і POST-запити**

    GET запрос используется чтобы получить данные а POST чтобы отправить

15. **Рядок db.Entry(book).State = EntityState.Modified**

За допомогою рядка db.Entry(book).State = EntityState.Modified; ми вказуємо, що об'єкт book існує вже в базі даних, і для нього треба внести в базу змінене значення, а не створювати новий запис. Після чого перенаправлення на головну сторінку.

**16. Рядок db.Entry (book) .State = EntityState.Added**

Використовується для створення моделі.

**17. Рядок db.Entry(b).State = EntityState.Deleted**

Використовується для видалення моделі.

**18. Get і POST-запити при видалені моделі**

Get має вразливість потрібно спрямувати через POST. Таким чином, ми відійдемо від вразливості GETзапиту

**19. Шаблони формування**

Шаблони формування. Оскільки часто розробники змушені створювати представлення для одних і тих же дій: додавання, зміни, видалення і перегляду записів з БД, то команда розробників MVC впровадила таку корисну функцію, як шаблони формування (scaffolding templates). Ці шаблони дозволяють для заданої моделі і контексту даних сформувати всю необхідну розмітку для представлень і контролера, за допомогою яких можна управляти записами у БД. Для коректного застосування шаблонів формування необхідно, щоб використовувалася одна з зв'язок MVC 4+ EntityFramework 5 або MVC 5 + Entity Framework 6.