ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет програмування та комп'ютерних
і телекомунікаційних систем

Кафедра інженерії програмного забезпечення

Лабораторна робота № 4

з дисципліни «Програмування Інтернет»
на тему:

«Моделі зі складною структурою. Визначення таблиць. Створення логіки додатка. Виконання операцій з моделями - редагування, видалення, додавання. Відношення один до багатьох, багато до багатьох.»

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 зі складними моделями. При цьому необхідно враховувати значення навігаційної властивості, наявне у складній моделі також відношення один до багатьох, багато до багатьох.

Завдання. Варіант 10..

Хід роботи

Додаток 1: «Салон по продажу автомобілів»

1. Створюємо Моделі

1.1 Модель Car

```
public class Car
    {

            public int Id { get; set; }
            public string Name { get; set; }
            public int Age { get; set; }
            public string Prise { get; set; }
            public int? SalonId { get; set; }
            public Salon Salon { get; set; }

    }
```

1.2 Модель CarContext

```
    public class CarContext : DbContext
    {
        public DbSet<Car> Cars { get; set; }
        public DbSet<Salon> Salons { get; set; }
    }
```

1.3 Модель Salon

```
public class Salon
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Mist { get; set; }
        public IEnumerable<Car> Cars { get; set; }
    }
```

2. Створюємо контролер

```
public class HomeController : Controller
    {
        //
        // GET: /Home/
        CarContext db = new CarContext();
        // Виводимо всі автомобілі
        public ActionResult Index()
        {
            var cars = db.Cars.Include(p => p.Salon);
            return View(cars.ToList());
        }

        public ActionResult SalonDetails(int? id)
        {
            if (id == null)
            {
                return HttpNotFound();
```

```
            }
            Salon salon = db.Salons.Find(id);
            if (salon == null)
            {
                return HttpNotFound();
            }
            salon.Cars = db.Cars.Where(m => m.SalonId == salon.Id);
            return View(salon);
        }

        public ActionResult SpisSal()
        {
            return View(db.Salons);
        }


        [HttpGet]
        public ActionResult Create()
        {
            //Формуємо список салонів для передачі в представлення
            SelectList salons = new SelectList(db.Salons, "Id", "Name");
            ViewBag.Salons = salons;
            return View();
        }
        [HttpPost]
        public ActionResult Create(Car car)
        {
            // Додаємо гравця в таблицю
            db.Cars.Add(car);
            db.SaveChanges();
            // перенаправляємо на головну сторінку
            return RedirectToAction("Index");
        }

        [HttpGet]
        public ActionResult Edit(int? id)
        {
            if (id == null)
            {
                return HttpNotFound();
            }
            // Знаходимо в БД автомобіля
            Car car = db.Cars.Find(id);
            if (car != null)
            {
                // Створюємо список салонів для передачі в представлення
                SelectList salons = new SelectList(db.Salons, "Id", "Name",
car.SalonId);
                ViewBag.Salons = salons;
                return View(car);
            }
            return RedirectToAction("Index");
        }
        [HttpPost]
        public ActionResult Edit(Car car)
        {
            db.Entry(car).State = EntityState.Modified;
            db.SaveChanges();
            return RedirectToAction("Index");
        }


        [HttpGet]
        public ActionResult Delete(int? id)
        {
            if (id == null)
            {
                return HttpNotFound();
```

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

    }
```

3. Створюємо базу даних DatabaseCar

3.1 Створюємо таблицю Cars з відповідними налаштуваннями

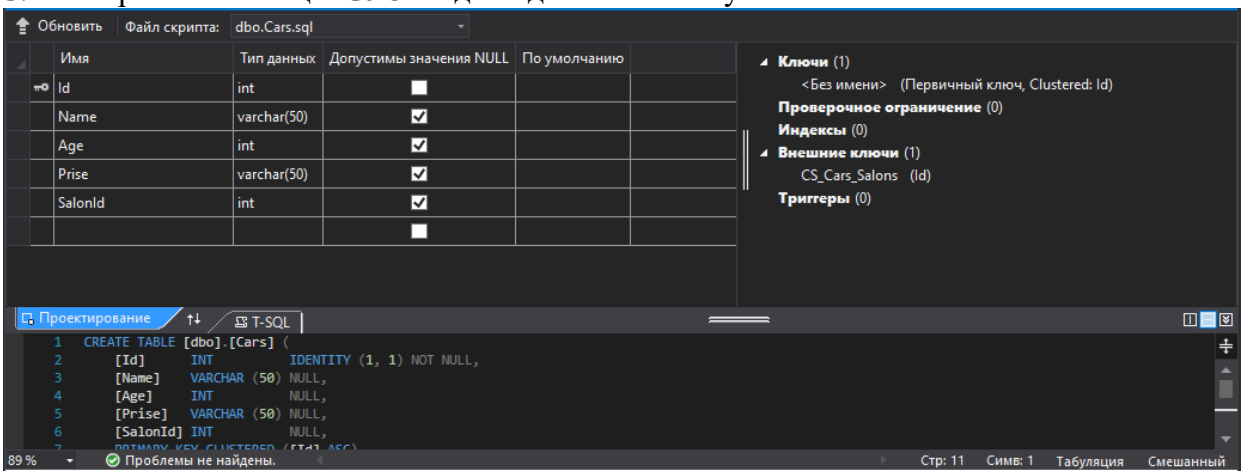

Рисунок 1 Налаштування таблиці Cars

3.2 Створюємо таблицю Salon з відповідними налаштуваннями

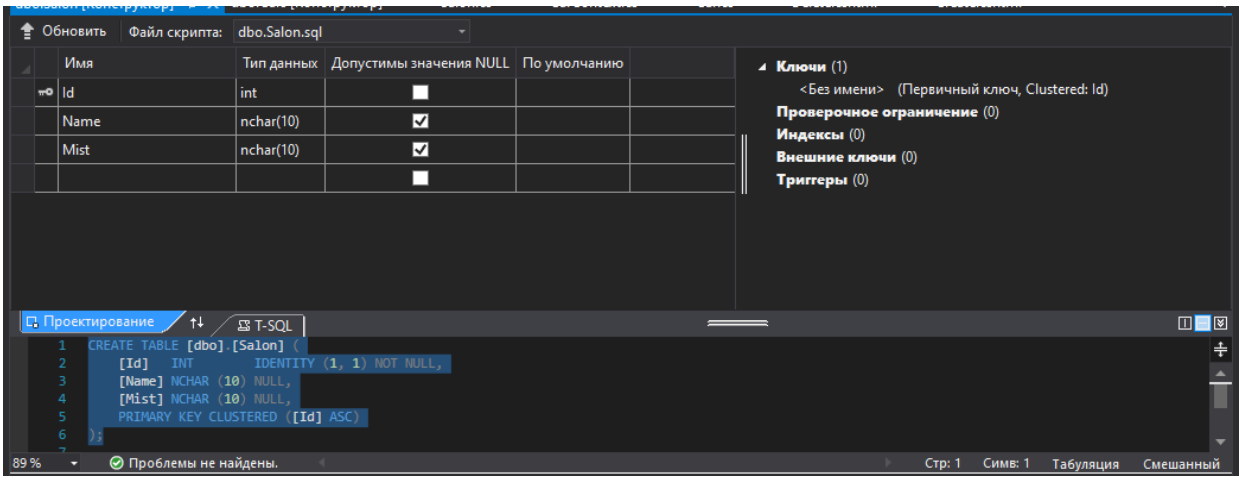

Рисунок 2 Налаштування таблиці Salon

3.3 Створюємо таблицю Salons з відповідними налаштуваннями

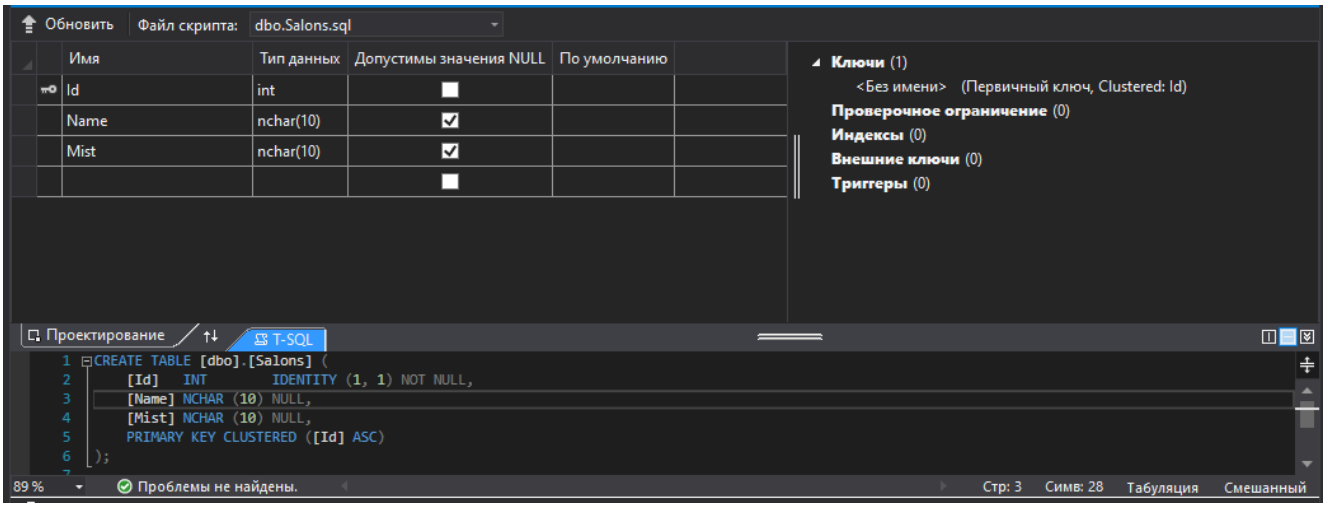

Рисунок 3 Налаштування таблиці Salons

4. Долучаємо рядок з'єднання в файл Web.config

```
<connectionStrings>
        <add name="CarContext" connectionString="Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename='|DataDirectory|\DatabaseCar.mdf';Integra
ted Security=True" providerName="System.Data.SqlClient" />
    </connectionStrings>
```

5. Створюємо представлення

5.1Index

```
@model IEnumerable<CarShop.Models.Car>
@{
    ViewBag.Title = "Каталог Автомобілів";
}
<h2>Каталог Автомобілів</h2>
<table>
    <tr>
        <th>Марка і модель  </th>
        <th>  Дата виробництва  </th>
        <th>  Ціна $  </th>
        <th>  Салон  </th>
        <th></th>
    </tr>
    @foreach (var item in Model)
    {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Name)"
            </td>
            <td>
                  "@Html.DisplayFor(modelItem => item.Age)"
            </td>
            <td>
                  "@Html.DisplayFor(modelItem => item.Prise)"
            </td>
            <td>
                  "@Html.DisplayFor(modelItem => item.Salon.Name)"
            </td>
            <td>
                @Html.ActionLink("Редагувати", "Edit", new { id = item.Id }) |
                @Html.ActionLink("Видалити", "Delete", new { id = item.Id })
```

```
            </td>
        </tr>
    }
</table>
```

5.2 SpisSal

```
@model IEnumerable<CarShop.Models.Salon>
@{
    ViewBag.Title = "Список Салонів";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<div>
    <h3>Список Салонів</h3>
    <table>
        <tr class="header">
            <td><p>Назва салону</p></td>
            <td></td>
        </tr>
        @foreach (CarShop.Models.Salon b in Model)
        {
            <tr>
                <td><p>@b.Name</p></td>
                <td><p><a href="/Home/SalonDetails/@b.Id">Детальніше</a></p></td>
            </tr>
        }
    </table>
</div>
```

5.3 SalonDeteils

```
@using CarShop.Models
@model Salon
@{
    ViewBag.Title = "Салон " + @Model.Name;
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<div>
    <h4>Салон @Model.Name</h4>
    <hr />
    <dl>
        <dt>Назва</dt>
        <dd>
            @Html.DisplayFor(model => model.Name)
        </dd>
        <dt>Місто</dt>
        <dd>
            @Html.DisplayFor(model => model.Mist)
        </dd>
        <dt>Автомобілі</dt>
        <dd>
            <ul>
                @foreach (Car car in Model.Cars)
                {
                    <li>@car.Name (@car.Prise)</li>
                }
            </ul>
        </dd>
    </dl>
</div>
```

5.4 Create

```
@model CarShop.Models.Car
@{
```

```
    ViewBag.Title = " Додавання Автомобіля ";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<h2> Додавання нового автомобіля </h2>
@using (Html.BeginForm())
{
    <fieldset>
        <legend>Автомобіль</legend>
        <p>
            Марка та модель <br />
            @Html.EditorFor(model => model.Name)
        </p>
        <p>
            Рік виробництва <br />
            @Html.EditorFor(model => model.Age)
        </p>
        <p>
            Ціна <br />
            @Html.EditorFor(model => model.Prise)
        </p>
        <p>
            Салон <br />
            @Html.DropDownListFor(model => model.SalonId,
            ViewBag.Salons as SelectList)
        </p>
        <p>
            <input type="submit" value="Добавити Автомобіль " />
        </p>
    </fieldset>
}
```

5.5 Edit

```
@model CarShop.Models.Car
@{
    ViewBag.Title = "Edit";
}
<h2>Редагування автомобіля </h2>
@using (Html.BeginForm())
{
    <fieldset>
        <legend>Автомобіль</legend>
        @Html.HiddenFor(model => model.Id)
        <p>
            Марка та модель <br />
            @Html.EditorFor(model => model.Name)
        </p>
        <p>
            Рік виробництва <br />
            @Html.EditorFor(model => model.Age)
        </p>
        <p>
            Ціна <br />
            @Html.EditorFor(model => model.Prise)
        </p>
        <p>
            Салон <br />
            @Html.DropDownListFor(model => model.SalonId,
           ViewBag.Salons as SelectList)
        </p>
        <p>
            <input type="submit" value="Зберегти" />
        </p>
    </fieldset>
}
```

5.6 Delete

```
@{
    ViewBag.Title = "Видалити автомобіль";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
@model CarShop.Models.Car
<h2>Видалення Автомобіля</h2>
<dl>
    <dt><b>Марка та модель</b></dt>
    <dd>
       @Html.DisplayFor(model => model.Name)
    </dd>
    <dt><b>Рік виробництва</b></dt>
    <dd>
        @Html.DisplayFor(model => model.Age)
    </dd>
    <dt><b>Ціна</b></dt>
    <dd>
        @Html.DisplayFor(model => model.Prise)
    </dd>
</dl>
@using (Html.BeginForm())
{
    <input type="submit" value="Видалити" />
}
```

Результат виконання додатку

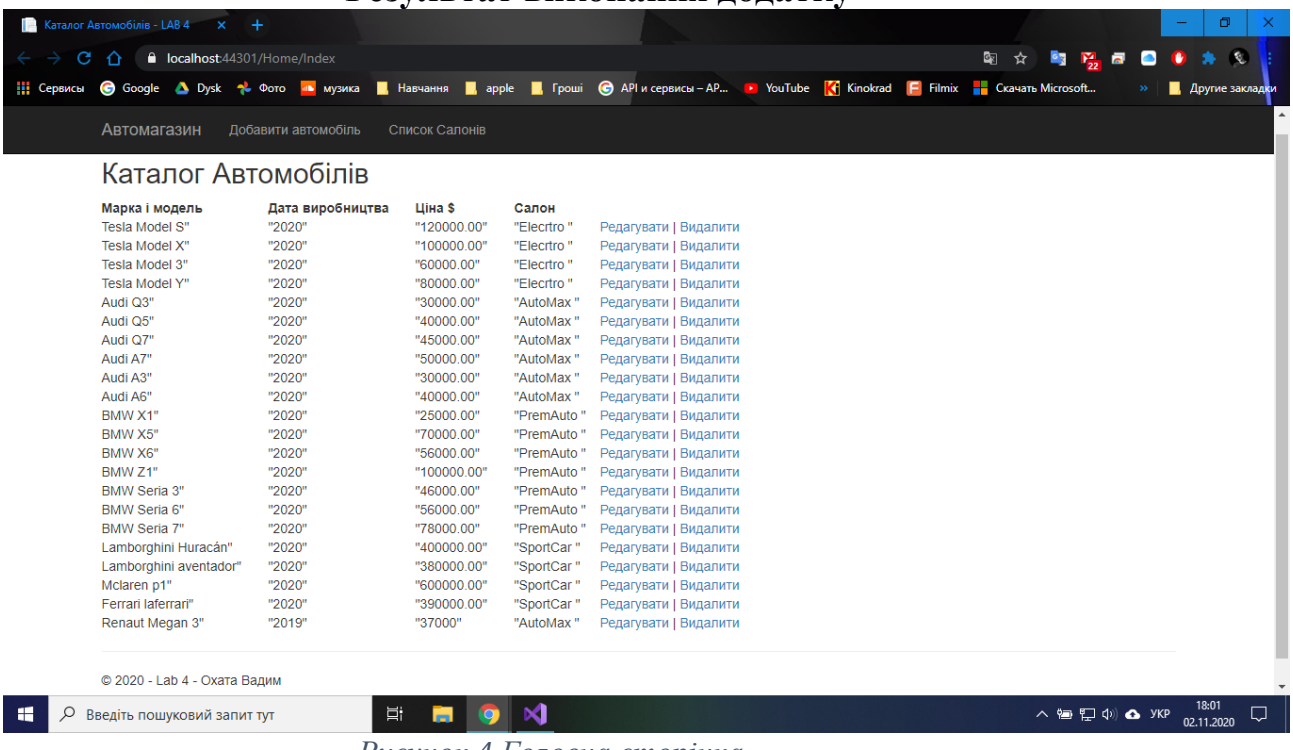

Рисунок 4 Головна сторінка

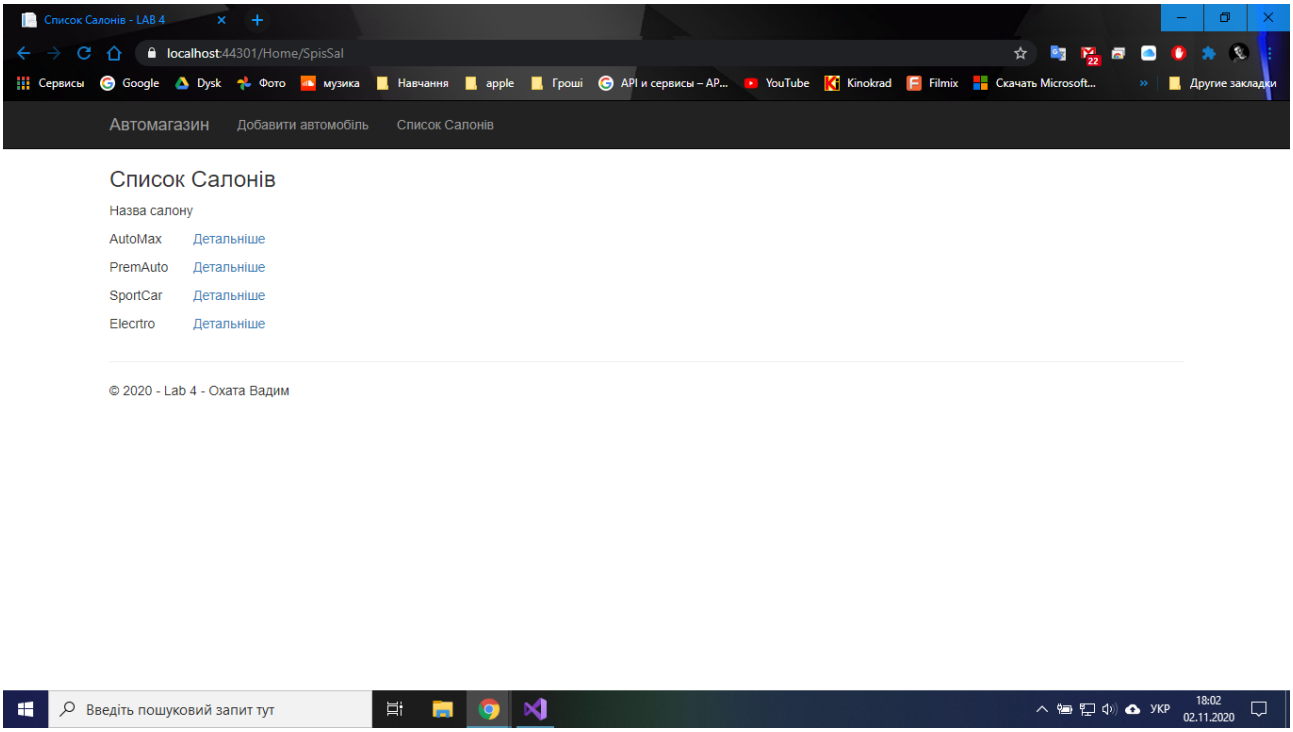

Рисунок 5 Список салонів

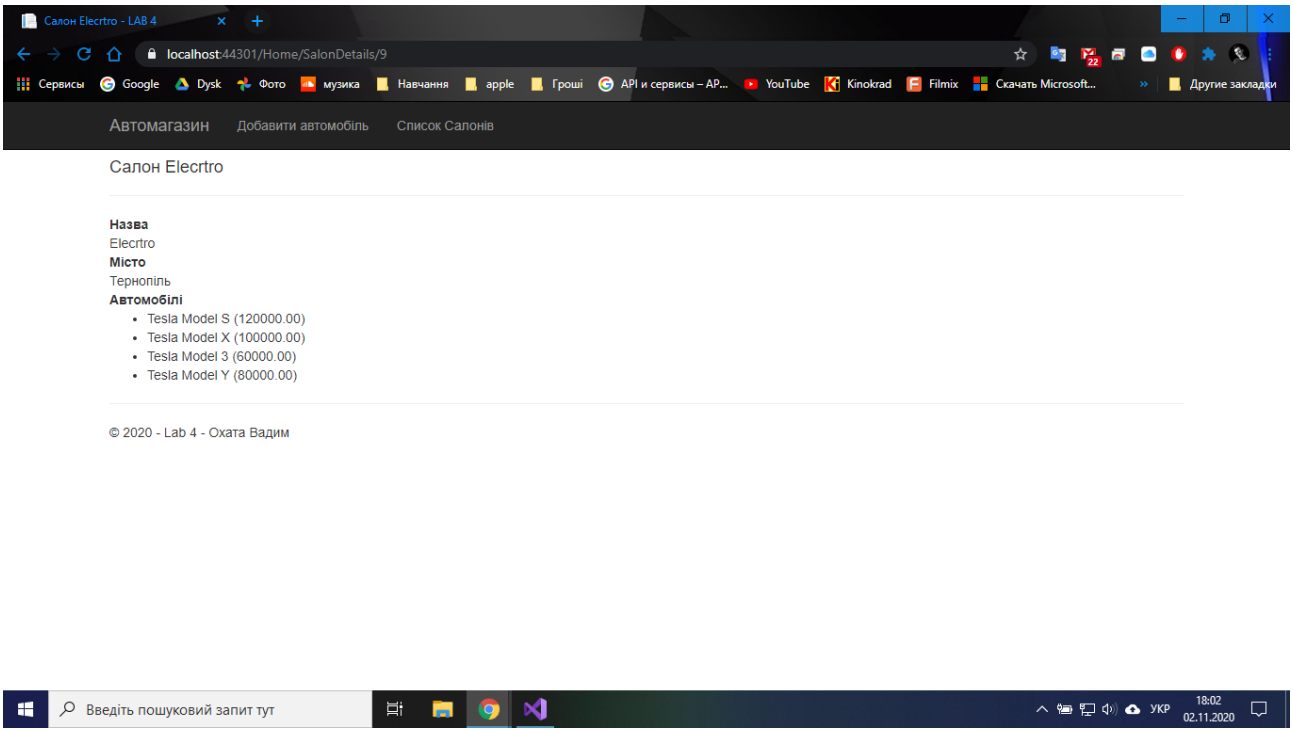

Рисунок 6 Детальніше за салон

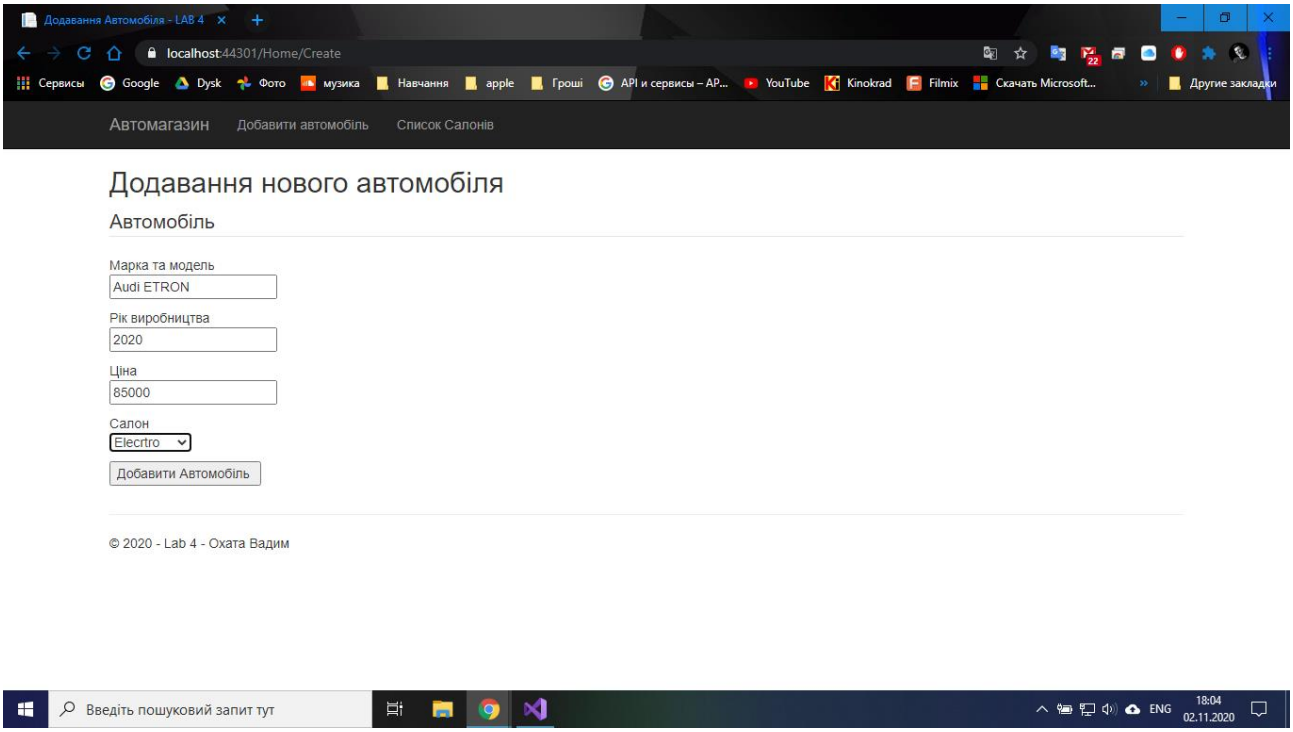

Рисунок 7 Додавання автомобіля

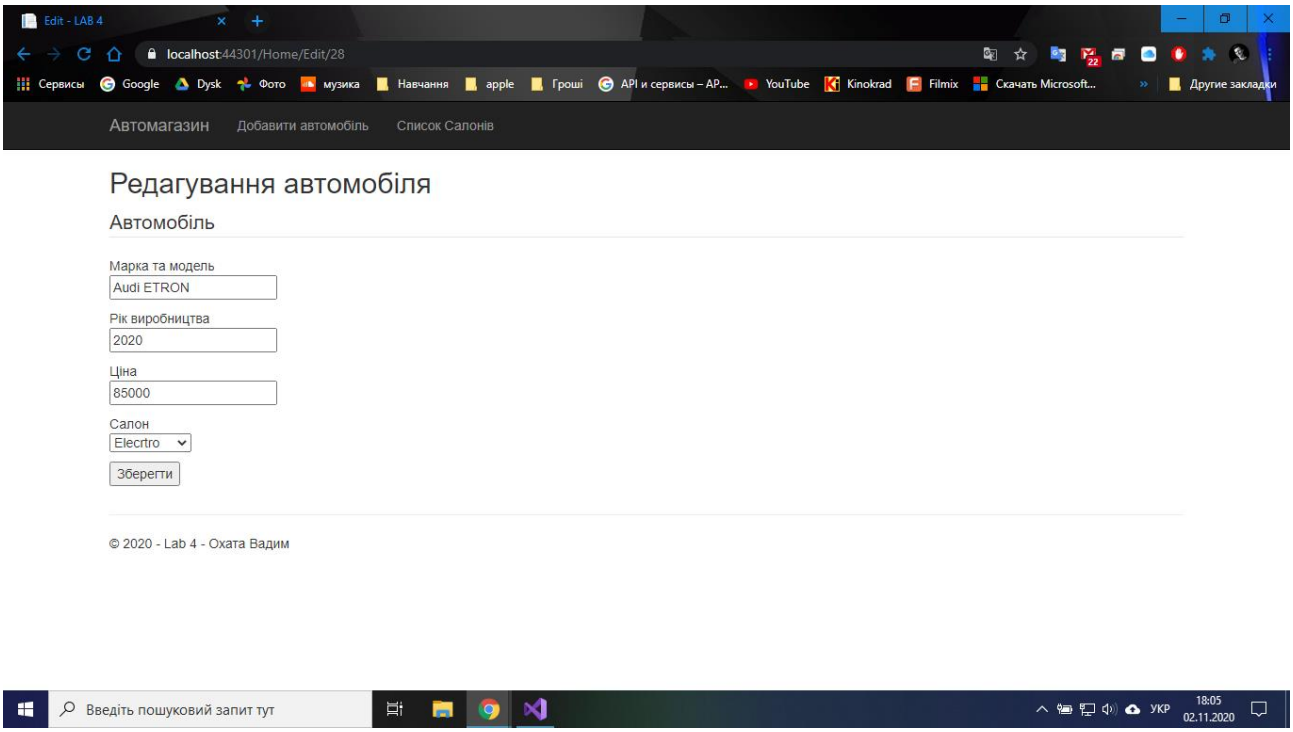

Рисунок 8 Редагування автомобіля

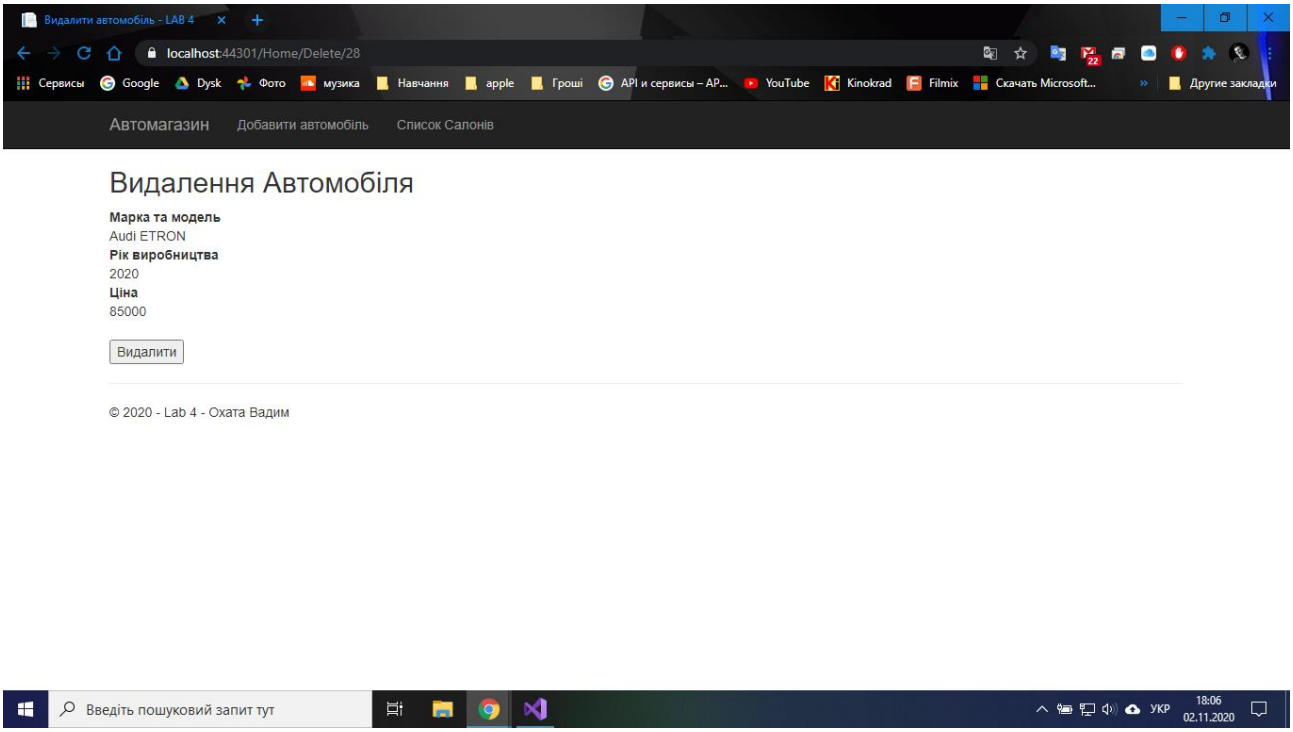

Рисунок 9 Видалення автомобіля

Додаток 2: Моделі зі складною структурою. Відношення багато до багатьох «Університет»

1. Створюємо Моделі

1.1 Course

```
public class Course
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public virtual ICollection<Student> Students { get; set; }

        public Course()
        {
            Students = new List<Student>();
        }


    }
```

1.2 CourseDbInitializer

```
public class CourseDbInitializer : DropCreateDatabaseAlways<StudentsContext>
    {
        protected override void Seed(StudentsContext context)
        {
            Student s1 = new Student { Id = 1, Name = "Егор", Surname = "Иванов" };
            Student s2 = new Student
            {
                Id = 2,
                Name = "Мария",
                Surname = "Васильева"
            };
            Student s3 = new Student { Id = 3, Name = "Олег", Surname = "Кузнецов" };
            Student s4 = new Student { Id = 4, Name = "Ольга", Surname = "Петрова" };
            context.Students.Add(s1);
```

```
            context.Students.Add(s2);
            context.Students.Add(s3);
            context.Students.Add(s4);
            Course c1 = new Course
            {
                Id = 1,
                Name = "Операционные системы",
                Students = new List<Student>() { s1, s2, s3 }
            };
            Course c2 = new Course
            {
                Id = 2,
                Name = "Алгоритмы и структуры данных",
                Students = new List<Student>() { s2, s4 }
            };
            Course c3 = new Course
            {
                Id = 3,
                Name = "Основы HTML и CSS",
                Students = new List<Student>() { s3, s4, s1 }
            };
            context.Courses.Add(c1);
            context.Courses.Add(c2);
            context.Courses.Add(c3);
            base.Seed(context);
        }
    }
```

1.3 Student

```
public class Student
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Surname { get; set; }
        public virtual ICollection<Course> Courses { get; set; }
        public Student()
        {
            Courses = new List<Course>();
        }

    }
```

1.4 StudentsContext

```
public class StudentsContext : DbContext
    {
        public DbSet<Student> Students { get; set; }
        public DbSet<Course> Courses { get; set; }
        public StudentsContext() : base("DefaultConnection")
        { }
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Course>().HasMany(c => c.Students)
            .WithMany(s => s.Courses)
            .Map(t => t.MapLeftKey("CourseId")
            .MapRightKey("StudentId")
            .ToTable("CourseStudent"));
        }
    }
```

2. Створюємо Контролер

```
public class HomeController : Controller
    {
        private StudentsContext db = new StudentsContext();
        public ActionResult Index()
        {
```

```
    var cars = db.Students.Include(p => p.Courses);
    return View(cars.ToList());
}
public ActionResult Details(int id = 0)
{
    Student student = db.Students.Find(id);
    if (student == null)
    {
        return HttpNotFound();
    }
    return View(student);
}
protected override void Dispose(bool disposing)
{
    db.Dispose();
    base.Dispose(disposing);
}

public ActionResult Edit(int id = 0)
{
    Student student = db.Students.Find(id);
    if (student == null)
    {
        return HttpNotFound();
    }
    ViewBag.Courses = db.Courses.ToList();
    return View(student);
}
[HttpPost]
public ActionResult Edit(Student student, int[] selectedCourses)
{
    Student newStudent = db.Students.Find(student.Id);
    newStudent.Name = student.Name;
    newStudent.Surname = student.Surname;

    newStudent.Courses.Clear();
    if (selectedCourses != null)
    {
        //получаем выбранные курсы
        foreach (var c in db.Courses.Where(co =>
       selectedCourses.Contains(co.Id)))
        {
            newStudent.Courses.Add(c);
        }
    }
    db.Entry(newStudent).State = EntityState.Modified;
    db.SaveChanges();
    return RedirectToAction("Index");
}
[HttpGet]
public ActionResult Create()
{
    //Формуємо список салонів для передачі в представлення
    SelectList courses = new SelectList(db.Courses, "Id", "Name");
    ViewBag.Courses = courses;
    return View();
}
[HttpPost]
public ActionResult Create(Student student)
{
    // Додаємо гравця в таблицю
    db.Students.Add(student);
    db.SaveChanges();
    // перенаправляємо на головну сторінку
    return RedirectToAction("Index");
}

[HttpGet]
```

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


    }
```

3. Створюємо Представлення

3.1 Index

```
@model IEnumerable<_2._2.Models.Student>
@{
    ViewBag.Title = "Каталог Автомобілів";
}
<h2>Студенти</h2>
<table>
    <tr>
        <th>Ім'я  </th>
        <th>  Прізвище  </th>
        <th></th>
    </tr>
    @foreach (var item in Model)
    {
        <tr>
            <td>

                  "@Html.DisplayFor(modelItem => item.Name)"
            </td>
            <td>
                  "@Html.DisplayFor(modelItem => item.Surname)"
            </td>
            <td>
                @Html.ActionLink("Детальніше", "Details", new { id = item.Id }) |
                @Html.ActionLink("Редагувати", "Edit", new { id = item.Id }) |
                @Html.ActionLink("Відрахувати", "Delete", new { id = item.Id })
            </td>
        </tr>
    }
</table>
```

3.2 Create

```
@model _2._2.Models.Student
@{
    ViewBag.Title = " Зарахувати студента ";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<h2> Зарахувати студента  </h2>
@using (Html.BeginForm())
{

    <legend>Студент</legend>
    <p>
        Ім'я <br />
        @Html.EditorFor(model => model.Name)
    </p>
    <p>
        Прізвище <br />
        @Html.EditorFor(model => model.Surname)
    </p>

    <input type="submit" value="Зарахувати " />

}
```

3.3 Delete

```
@{
    ViewBag.Title = "Видалити автомобіль";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
@model _2._2.Models.Student
<h2>Відрахуати студента</h2>
<dl>
    <dt><b>Ім'я</b></dt>
    <dd>
        @Html.DisplayFor(model => model.Name)
    </dd>
    <dt><b>Прізвище</b></dt>
    <dd>
        @Html.DisplayFor(model => model.Surname)
    </dd>
</dl>
@using (Html.BeginForm())
{
    <input type="submit" value="Відрахуати" />
}
```

3.4 Details

```
@using _2._2.Models
@model Student
@{
    ViewBag.Title = "Details";
}
<fieldset>
    <legend>Информация о студенте</legend>
    <div class="display-label"><b>Имя</b></div>
    <div class="display-field">
        @Html.DisplayFor(model => model.Name)
    </div>
    <div class="display-label"><b>Фамилия</b></div>
    <div class="display-field">
        @Html.DisplayFor(model => model.Surname)
    </div>
    <div class="display-label"><b>Курсы</b></div>
    <ul>
        @foreach (Course c in Model.Courses)
```

```
        {
            <li>@c.Name</li>
        }
    </ul>
</fieldset>
```

3.5 Edit

```
@using _2._2.Models
@model Student
@{
    ViewBag.Title = "Edit";
}
@using (Html.BeginForm())
{
    <fieldset>
        <legend>Студент</legend>
        @Html.HiddenFor(model => model.Id)
        <div class="editor-label"><b>Имя</b></div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Name)
        </div>
        <div class="editor-label"><b>Фамилия</b></div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Surname)
        </div>
        <div class="editor-label"><b>Курсы</b></div>
        @foreach (Course c in ViewBag.Courses)
        {
            <input type="checkbox" name="selectedCourses" value="@c.Id"
                   @(Model.Courses.Contains(c) ? "checked=\"checked\"" : "") />@c.Name
<br />
            }
        <p>
            <input type="submit" value="Сохранить" />
        </p>
    </fieldset>
}
```

Результат виконання додатку

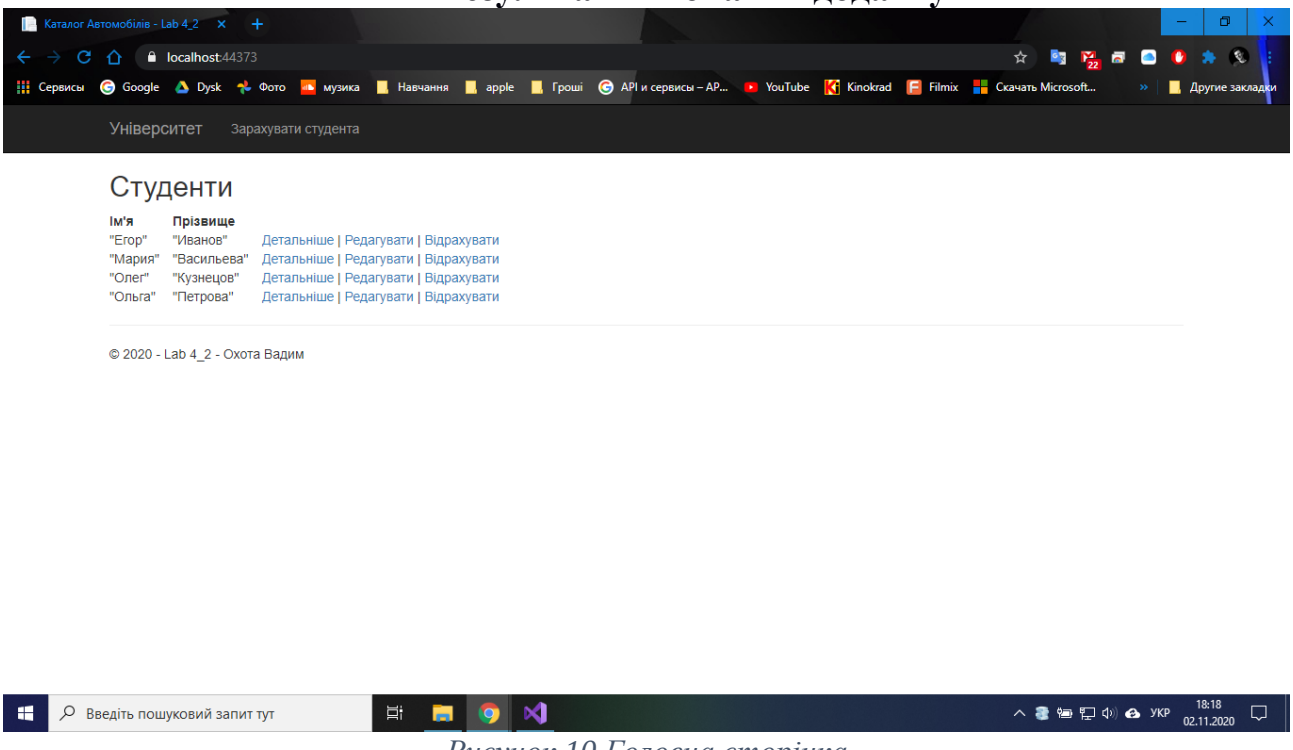

Рисунок 10 Головна сторінка

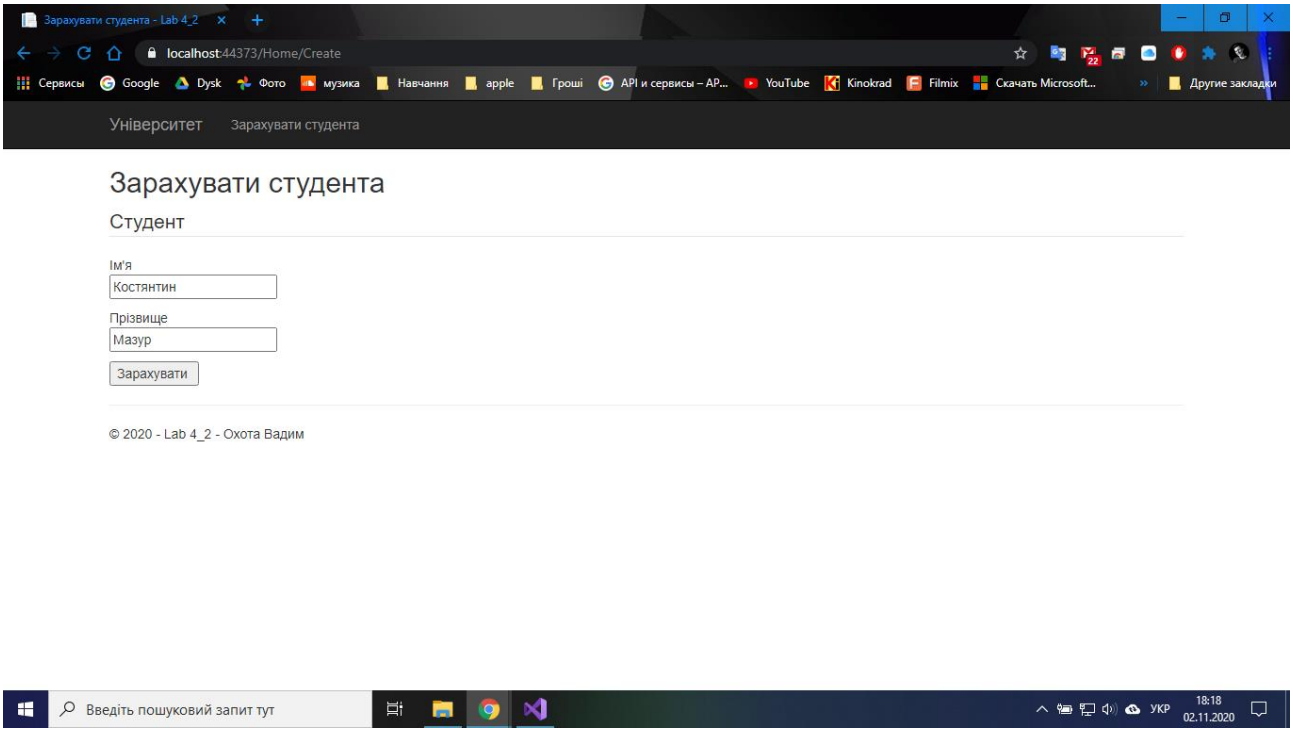

Рисунок 11 Зарахування студента

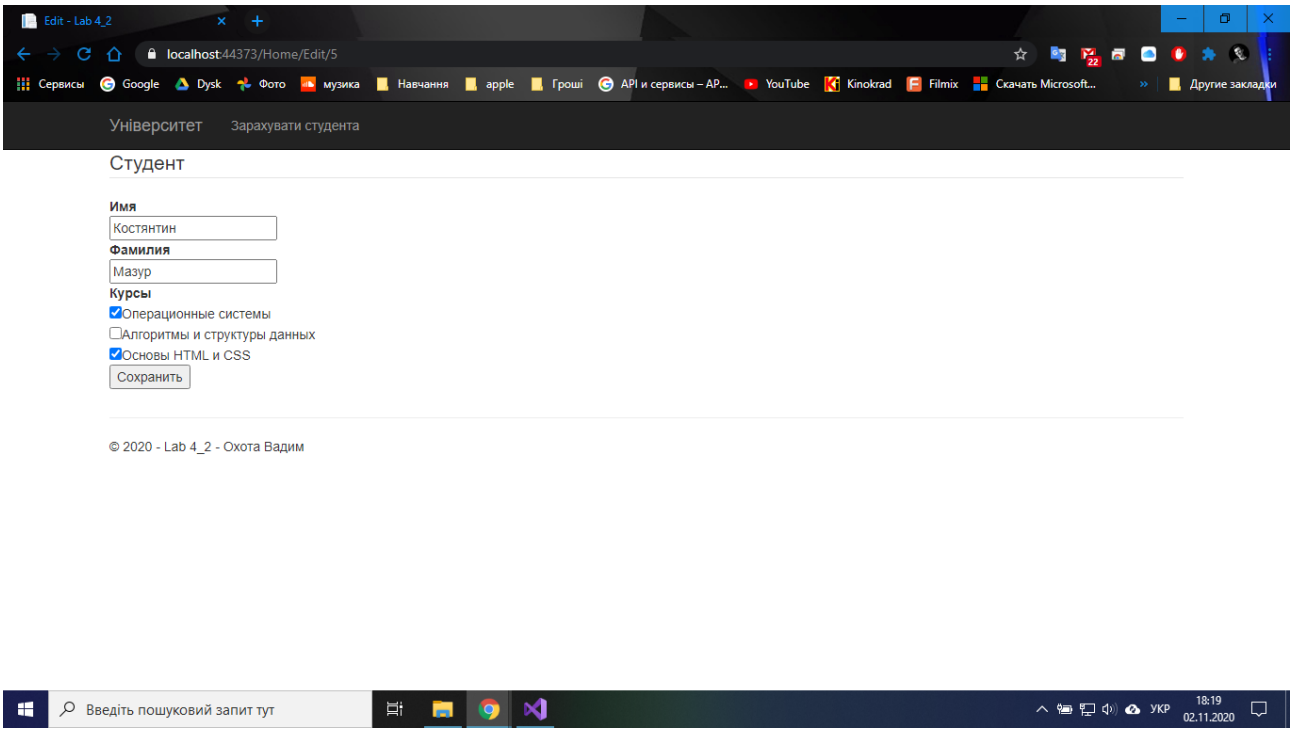

Рисунок 12 Редагування даних

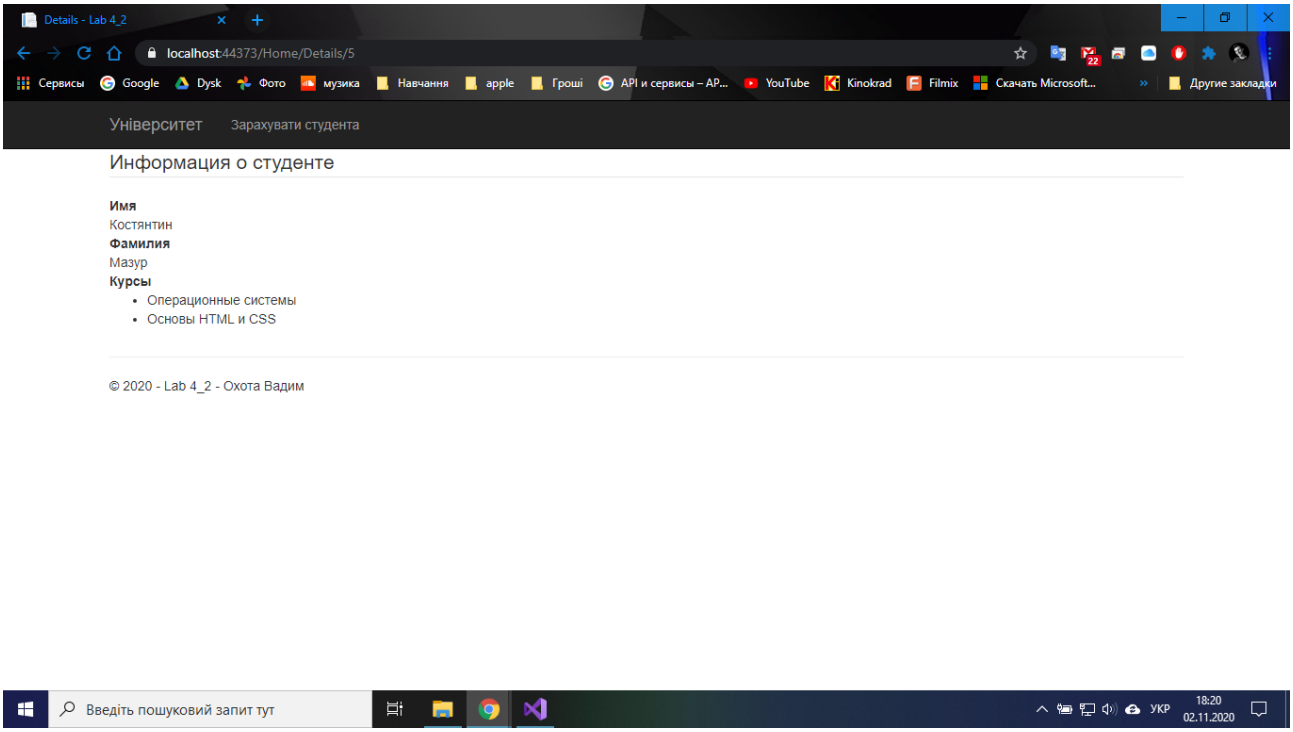

Рисунок 13 Детальніше про студента

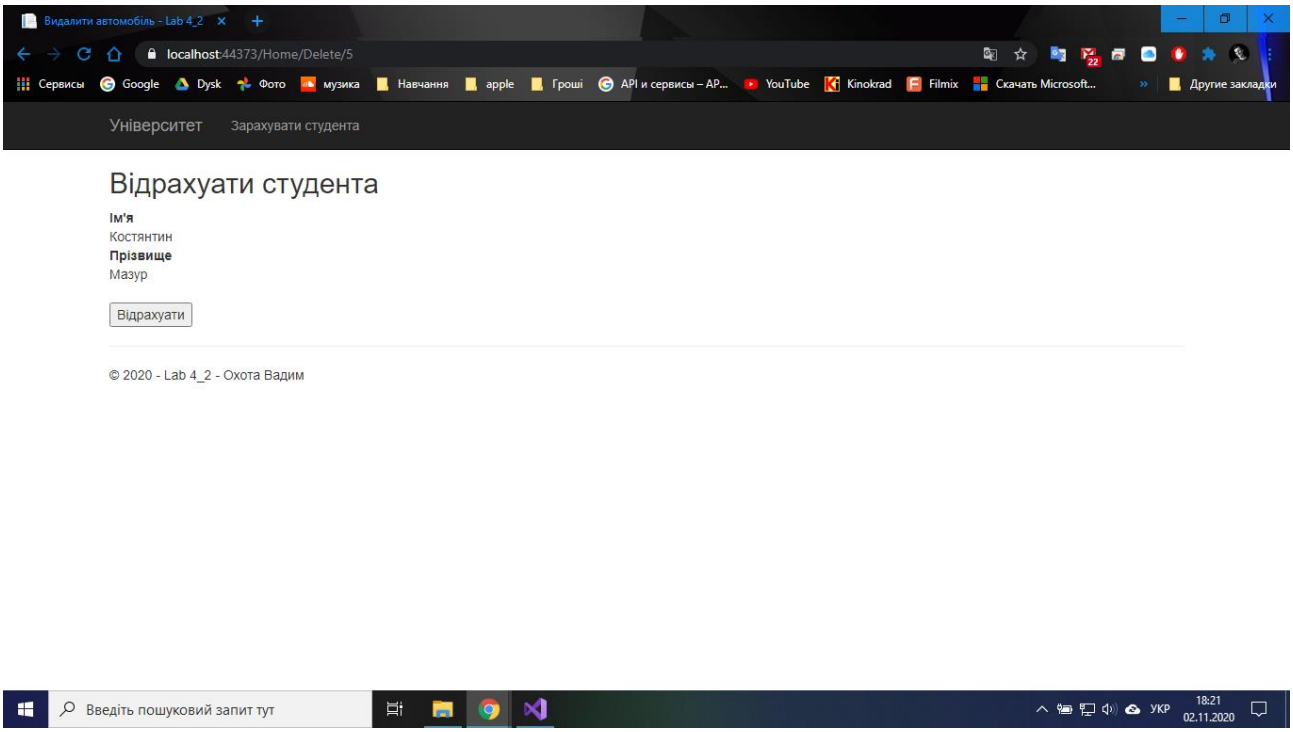

Рисунок 14 Відрахування студента

Контрольні питання

1. **Призначення контролера**
   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

2. **Звернення до контролера з веб-браузера**
   Щоб звернутися контролера з веб-браузера, нам треба в адресному рядку набрати адрес_сайта / Імя_контроллера / Действіе_контроллера. Так, за запитом адрес_сайта /

Home / Index система маршрутизації за замовчуванням викличе метод Index контролера HomeController для обробки вхідного запиту.

3. **Методи дій**
   Методи дій (action methods) представляють такі методи контролера, які обробляють запити за певним URL.

4. **Звернення до методу дій з веб-браузера**
   Метод GetVoid представляє цілком реальна дія контролера, до якого можна звертатися з адресного рядка браузера, передавати параметри, і який може містити складну логіку обробки запиту. Тільки після звернення до цього методу користувач побачить у своєму веб-браузері одну порожнечу, так як метод нічого не повертає.

5. **Опціональний параметр**
   Наприклад, за замовчуванням в проекті MVC визначається наступний маршрут: Контролер/Метод/id. Останній параметр є опціональним. І завдяки цьому ми можемо передати параметр id і так: Home/Buy/2.

6. **Значення параметрів за замовчуванням**
   Наприклад зідно з налаштуваннями за замовчуванням, якщо представлення не вказано явним чином, то в якості представлення буде використовуватися те, ім'я якого співпадає з іменем дій контролера. Наприклад, вищевизначена дія Index за замовчуванням буде проводити пошук представлення Index.cshtml у папці /Views/Home/.

7. **Отримання даних з контексту запиту**
   public string Square(int a=10, int h=3)
   { double s = a*h/2; return "
   Площа трикутника з основою " + a +
   " та висотою " + h + " рівна " + s + "
   "; }
   В цьому випадку при запиті сторінки можна вказати тільки один параметр чи взагалі не вказувати(Home/Square?h=5). Отримання даних з контексту запиту Крім того, можна отримати параметри, і не тільки параметри, але й інші дані, пов'язані із запитом, з об'єктів контексту запиту. Нам доступні такі об'єкти контексту: Request, Response, RoutedData, HttpContext и Server.

8. **Об'єкт Request**
   Об'єкт Request містить колекцію Params, яка зберігає всі параметри, передані в запити.

9. **Клас ActionResult**
   ActionResult представляє собою абстрактний клас, в якому визначено один метод ExecuteResult, що перевизначається в наслідуваних класах

10. **Створення власних результатів дій**
    BookStore.Util; і додамо новий метод:
    public ActionResult GetHtml()
    { return new HtmlResult("Привіт світ! ");
    }
    І звернувшись до цього методу з браузера, наприклад, Home/GetHtml, ми отримаємо html-сторінку. Хоча даний приклад досить примітивний, але в цілому він демонструє, як працюють класи результатів дій.

11. **Навігаційна властивість**

Властивість Team називається навігаційним властивістю - при отриманні даних про гравця вона автоматично буде отримувати дані з БД. Однак для цього нам треба також встановити зовнішній ключ.

12. Звичайна властивість

Звичайна повинна приймати одне з наступних варіантів імені:
1. Ім'я_навігаційної_властивості + Ім'я ключа із зв'язаної таблиці - в нашому випадку ім'я навігаційного властивості Team, а ключа з моделі Team - Id, тому в нашому випадку нам треба назвати властивість TeamId, що власне і було зроблено у вищенаведеному коді.
2. Ім'я _класу_зв'язаної_таблиці + Ім'я ключа із зв'язаної таблиці - в нашому випадку клас Team, а ключа з моделі Team - Id , тому знову ж в цьому випадку виходить TeamId.

13. Зовнішній ключ в моделі зі складною структурою

Крім того, тут ми задаємо зовнішній ключ - властивість TeamId тепер буде посилатися на властивість Id з таблиці Teams. Щоб задати зовнішній ключ, ми додаємо в панелі SQL внизу під дизайнером таблиці наступний рядок:
CONSTRAINT [FK_Players_Teams] FOREIGN KEY ([TeamId]) REFERENCES [Teams] ([Id]) ON DELETE SET NULL

14. Метод Include

За допомогою методу Include фреймворк підвантажує для кожного гравця також і команду, асоційовану з певним гравцем. А при виведенні моделі в представлення Index.cshtml фреймворк буде виводити для кожного гравця назву команди

15. Рядок зєднання connectionStrings

<connectionStrings>
<add name="BookContext2" connectionString="Data Source=(LocalDB)
\v11.0;AttachDbFilename='|DataDirectory|\Bookstore2.mdf';Integrated
Security=True" providerName="System.Data.SqlClient" />
</connectionStrings>
</configuration>

16. Призначення властивості Players в моделі Team

В моделі Team властивість Players, призначення якої зберігати пов'язаних з командою гравців. Використаємо її. Наприклад, виведемо всі дані про команду, в тому числі про її гравцях.

17. Створення моделі

Додаємо код у контролер

```
public ActionResult Create()
 {
     //Формуємо список салонів для передачі в представлення
     SelectList courses = new SelectList(db.Courses, "Id", "Name");
     ViewBag.Courses = courses;
     return View();
 }
 [HttpPost]
 public ActionResult Create(Student student)
 {
     // Додаємо гравця в таблицю
     db.Students.Add(student);
     db.SaveChanges();
     // перенаправляємо на головну сторінку
     return RedirectToAction("Index");
        }
```

Створюємо представлення

```
@model _2._2.Models.Student
@{
    ViewBag.Title = " Зарахувати студента ";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<h2> Зарахувати студента  </h2>
@using (Html.BeginForm())
{

    <legend>Студент</legend>
    <p>
        Ім'я <br />
        @Html.EditorFor(model => model.Name)
    </p>
    <p>
        Прізвище <br />
        @Html.EditorFor(model => model.Surname)
    </p>

    <input type="submit" value="Зарахувати " />

}
```

18. Видалення моделі

Додаємо код у контролер

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

Створюємо представлення

```
@{
    ViewBag.Title = "Видалити автомобіль";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
@model _2._2.Models.Student
<h2>Відрахуати студента</h2>
<dl>
    <dt><b>Ім'я</b></dt>
```

```
    <dd>
        @Html.DisplayFor(model => model.Name)
    </dd>
    <dt><b>Прізвище</b></dt>
    <dd>
        @Html.DisplayFor(model => model.Surname)
    </dd>
</dl>
@using (Html.BeginForm())
{
    <input type="submit" value="Відрахуати" />
}
```

19. Редагування моделі

Додаємо код у контролер

```
public ActionResult Edit(int id = 0)
 {
     Student student = db.Students.Find(id);
     if (student == null)
     {
         return HttpNotFound();
     }
     ViewBag.Courses = db.Courses.ToList();
     return View(student);
 }
 [HttpPost]
 public ActionResult Edit(Student student, int[] selectedCourses)
 {
     Student newStudent = db.Students.Find(student.Id);
     newStudent.Name = student.Name;
     newStudent.Surname = student.Surname;

     newStudent.Courses.Clear();
     if (selectedCourses != null)
     {
         //получаем выбранные курсы
         foreach (var c in db.Courses.Where(co =>
        selectedCourses.Contains(co.Id)))
         {
             newStudent.Courses.Add(c);
         }
     }
     db.Entry(newStudent).State = EntityState.Modified;
     db.SaveChanges();
     return RedirectToAction("Index");
 }
```

Створюємо представлення

```
@using _2._2.Models
@model Student
@{
    ViewBag.Title = "Edit";
}
@using (Html.BeginForm())
{
    <fieldset>
        <legend>Студент</legend>
        @Html.HiddenFor(model => model.Id)
        <div class="editor-label"><b>Имя</b></div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Name)
        </div>
        <div class="editor-label"><b>Фамилия</b></div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Surname)
        </div>
        <div class="editor-label"><b>Курсы</b></div>
        @foreach (Course c in ViewBag.Courses)
```

```
        {
            <input type="checkbox" name="selectedCourses" value="@c.Id"
                   @(Model.Courses.Contains(c) ? "checked=\"checked\"" : "")
/>@c.Name <br />
            }
        <p>
            <input type="submit" value="Сохранить" />
        </p>
    </fieldset>
}
```

20. Призначення класу SelectList

Перший варіант дії Create обробляє Get-запит і видає представлення, передаючи в нього об'єкт SelectList - список всіх команд.

21. Віртуальні властивості - Students і Courses

Моделі досить прості за винятком віртуальних властивостей - Students і Courses - завдяки цим властивостям і відбуватиметься зв'язок багатодо-багатьох.

22. Призначення public StudentsContext() : base("DefaultConnection") { }

public StudentsContext (): base ("DefaultConnection"). У створюваній базі даних всі дані про студентів будуть зберігатися в таблиці Students, а дані про університетські курси - у таблиці Courses

23. Призначення Database.SetInitializer(new CourseDbInitializer());

Як забезпечується зв'язок між курсами і студентами: ми просто додаємо набір створених студентів в колекцію Students для кожного курсу. І щоб все це запрацювало, додамо в файл Global.asax.cs в метод Application_Start наступний рядок: Database.SetInitializer(new CourseDbInitializer());