***Лабораторна робота №4***

**Тема: CRUD-операції в EF Core. Валідація даних.**

**Мета:** набути навичок роботи з ORM, навчитися реалізовувати операції створення, читання, оновлення та видалення даних з використанням EF Core, набути навичок роботи з механізмами валідації даних.

**Завдання 1: Використовуючи підхід Code First, створити модель, яка разом з існуючою моделлю, утворюватиме зв’язок один-до-багатьох.**

Для цього створимо нову модель Station. Її суть та завдання-зберігати інформацію про станції, де можно отримати/забронювати транспортний засіб.  
  
Лістінг моделі Station:  
namespace VehicleRentalService.Models

{

public class Station

{

public long StationId { get; set; }

public string Address { get; set; }

public ICollection<Vehicle> Vehicles { get; set; }

}

}

Зв’язок один до багатьох реалізується у стрічці

public ICollection<Vehicle> Vehicles { get; set; }

Це можно пояснити наступним чином: станції зберігають в собі інформацію про ТЗ, які зараз знаходяться на станції. Позволено, щоби на станції зберігалось декілька автівок.

**Завдання 2. Реалізувати CRUD (Сreate, Read, Update, Delete) операції для обох моделей.**

Далі йдуть лістинги всіх файлів, пов’язанних із паттерном репозиторій  
  
Лістінг файлу ServiceDbContext:

using Microsoft.EntityFrameworkCore;

namespace VehicleRentalService.Models

{

public class ServiceDbContext : DbContext

{

public ServiceDbContext(DbContextOptions<ServiceDbContext> options) : base(options){}

public DbSet<Vehicle> Vehicles => Set<Vehicle>();

public DbSet<Car> Cars => Set<Car>();

public DbSet<Scooter> Scooters => Set<Scooter>();

public DbSet<Bike> Bikes => Set<Bike>();

public DbSet<News> News => Set<News>();

public DbSet<Trip> Trips => Set<Trip>();

public DbSet<Station> Stations => Set<Station>();

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Vehicle>().ToTable("Vehicle");

base.OnModelCreating(modelBuilder);

}

}

}

Лістінг файлу IServiceRepository:

namespace VehicleRentalService.Models

{

public interface IServiceRepository

{

IQueryable<T> GetAll<T>() where T : class;

T? GetById<T>(long id) where T : class;

void Create<T>(T entity) where T : class;

void Update<T>(T entity) where T : class;

void Delete<T>(T entity) where T : class;

Vehicle? FindById(VehicleType vehicleType, long id);

Task<Vehicle?> FindByIdAsync(long id);

}

}

Лістінг фалу EFSerivceRepository:

using Microsoft.EntityFrameworkCore;

namespace VehicleRentalService.Models

{

public class EFServiceRepository : IServiceRepository

{

private ServiceDbContext context;

public EFServiceRepository(ServiceDbContext ctx)

{

context = ctx;

}

public IQueryable<T> GetAll<T>() where T : class

{

return context.Set<T>();

}

public T? GetById<T>(long id) where T : class

{

return context.Set<T>().Find(id);

}

public void Create<T>(T entity) where T : class

{

context.Set<T>().Add(entity);

context.SaveChanges();

}

public void Update<T>(T entity) where T : class

{

context.Set<T>().Update(entity);

context.SaveChanges();

}

public void Delete<T>(T entity) where T : class

{

context.Set<T>().Remove(entity);

context.SaveChanges();

}

public Vehicle? FindById(VehicleType vehicleType, long id)

{

Vehicle vehicle = null;

switch (vehicleType)

{

case VehicleType.Car:

vehicle = context.Cars.FirstOrDefault(v => v.VehicleId == id);

break;

case VehicleType.Bike:

vehicle = context.Bikes.FirstOrDefault(v => v.VehicleId == id);

break;

case VehicleType.Scooter:

vehicle = context.Scooters.FirstOrDefault(v => v.VehicleId == id);

break;

}

return vehicle;

}

public async Task<Vehicle?> FindByIdAsync(long id)

{

return await context.Vehicles.FirstOrDefaultAsync(v => v.VehicleId == id);

}

}

}

Далі перейдемо до самої реалізації CRUD:  
  
**CREATE:**

Для цього було створено окремий View Component з назвою AdminVehiclePanel

Лістінг приведено нижче:

@model VehicleRentalService.Models.ViewModels.VehicleCreateViewModel

@if (TempData["FormErrors"] != null)

{

<div class="alert alert-danger">@TempData["FormErrors"]</div>

}

<form asp-controller="Home" asp-action="AddVehicle" method="post">

<input asp-for="Name" placeholder="Vehicle Name" required />

<input asp-for="Description" placeholder="Vehicle Description" />

<input asp-for="PricePerHour" type="number" step="0.01" min="0.01" max="10000" required />

<input type="hidden" asp-for="IsAvailable" value="true" />

<div>

<input asp-for="VehicleType" name="VehicleType" id="vt1" type="radio" value="Car" checked /> Car

<input asp-for="VehicleType" name="VehicleType" id="vt2" type="radio" value="Bike" /> Bike

<input asp-for="VehicleType" name="VehicleType" id="vt3" type="radio" value="Scooter" /> Scooter

</div>

<div id="car-fields" style="display:none">

<input asp-for="MaxFuel" type="number" placeholder="Fuel capacity" />

<select asp-for="Transmission">

<option value="Manual">Manual</option>

<option value="Automatic">Automatic</option>

<option value="CVT">CVT</option>

<**option** **value**="SemiAutomatic">SemiAutomatic</**option**>

</**select**>

<**input** **asp-for**="CurrentFuel" **type**="hidden" **value**="100" />

</div>

<div id="bike-fields" style="display:none"></div>

<div id="scooter-fields" style="display:none">

<**input** **asp-for**="Charge" **type**="hidden" **value**="100" />

</div>

<button type="submit">Add Vehicle</button>

</**form**>

<script>

document.addEventListener('DOMContentLoaded', () => {

const maxFuelInput = document.getElementById('MaxFuel');

const currentFuelInput = document.getElementById('CurrentFuel');

if (maxFuelInput && currentFuelInput) {

maxFuelInput.addEventListener('input', () => {

const value = parseFloat(maxFuelInput.value);

currentFuelInput.value = isNaN(value) ? 0 : value;

});

}

const radios = document.querySelectorAll('input[name="VehicleType"]');

const sections = {

Car: document.getElementById('car-fields'),

Bike: document.getElementById('bike-fields'),

Scooter: document.getElementById('scooter-fields')

};

function updateFields() {

const selected = document.querySelector('input[name="VehicleType"]:checked').value;

for (const key in sections) {

sections[key].style.display = (key === selected) ? 'block' : 'none';

}

}

radios.forEach(radio => {

radio.addEventListener('change', updateFields);

});

updateFields();

});

</script>

Як можна побачити, форма використовує метод AddVehicle із HomeController.

Розглянемо його далі

[HttpPost]

public IActionResult AddVehicle(VehicleCreateViewModel model)

{

if (!ModelState.IsValid)

{

TempData["Error"] = "Validation failed. Please check your input.";

return RedirectToAction("Index", "Home");

}

Vehicle vehicle;

switch (model.VehicleType)

{

case "Car":

if (!model.MaxFuel.HasValue || !model.CurrentFuel.HasValue || !model.Transmission.HasValue)

{

TempData["Error"] = "All fields for Car are required.";

return RedirectToAction("Index", "Home");

}

vehicle = new Car

{

Name = model.Name,

Description = model.Description,

PricePerHour = model.PricePerHour,

IsAvailable = model.IsAvailable,

MaxFuel = model.MaxFuel.Value,

CurrentFuel = model.CurrentFuel.Value,

Transmission = model.Transmission.Value

};

break;

case "Bike":

vehicle = new Bike

{

Name = model.Name,

Description = model.Description,

PricePerHour = model.PricePerHour,

IsAvailable = model.IsAvailable

};

break;

case "Scooter":

vehicle = new Scooter

{

Name = model.Name,

Description = model.Description,

PricePerHour = model.PricePerHour,

IsAvailable = model.IsAvailable,

Charge = model.Charge ?? 100

};

break;

default:

TempData["Error"] = "Unknown vehicle type.";

return RedirectToAction("Index", "Home");

}

\_repository.Create(vehicle);

TempData["Success"] = $"Vehicle '{vehicle.Name}' successfully created.";

Console.WriteLine($"Created vehicle: {vehicle.Name}, Type: {model.VehicleType}");

return RedirectToAction("Index", "Home");

}

Таким чином виконується додавання.

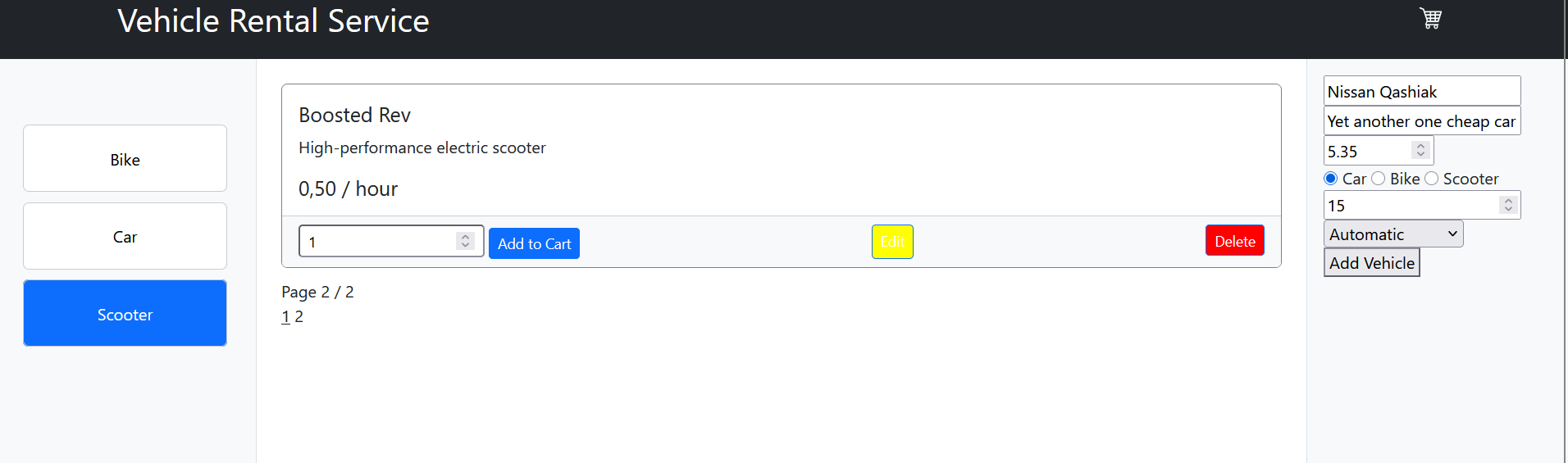


Рис.1.Додавання нового ТЗ у адмін панелі

**UPDATE:**

Для цього, як зображено на Рис.1. можна побачити додані кнопки Edit та Delete у блоці ТЗ.

Для цього функціонала нам потрібна саме кнопка Edit.   
  
Лістінг кнопки Edit:

<**a** **asp-controller**="Edit"

**asp-action**="Index"

**asp-route-id**="@v.VehicleId"

class="btn btn-primary btn-sm"

style="background-color: yellow; color: white;">

Edit

</**a**>

Ця кнопка переводить нас на сторінку Edit.

Розгянемо EditController та представлення Edit:

Лістінг файлу EditController:  
using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using VehicleRentalService.Models;

namespace VehicleRentalService.Controllers

{

public class EditController : Controller

{

private IServiceRepository \_repository;

public EditController(IServiceRepository repository)

{

\_repository = repository;

}

[HttpGet("{id:long}")]

public async Task<IActionResult> Index(long id)

{

var vehicle= await \_repository.FindByIdAsync(id);

if (vehicle == null)

return NotFound();

return View(vehicle);

}

[HttpPost]

[ValidateAntiForgeryToken]

public IActionResult EditVehicle(Vehicle vehicle)

{

if(!ModelState.IsValid)

return View(vehicle);

\_repository.Update(vehicle);

return RedirectToAction("Home");

}

}

}

Як ми можемо побачити метод Index перевіряє правильність даних, та заповнює поля у представленні. В свою чергу метод EditVehicle оновлено

Лістінг представлення Edit:

@model VehicleRentalService.Models.Vehicle

@{

ViewData["Title"] = "Edit";

}

<**form** **asp-action**="EditVehicle" method="post">

<div class="form-group">

<**label** **asp-for**="Name"></**label**>

<**input** **asp-for**="Name" class="form-control" />

<**span** **asp-validation-for**="Name" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Description"></**label**>

<**input** **asp-for**="Description" class="form-control" />

</div>

<div class="form-group">

<**label** **asp-for**="PricePerHour"></**label**>

<**input** **asp-for**="PricePerHour" class="form-control" />

<**span** **asp-validation-for**="PricePerHour" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Location"></**label**>

<**input** **asp-for**="Location" class="form-control" />

</div>

<**input** **type**="hidden" **asp-for**="VehicleId" />

<button type="submit" class="btn btn-primary">Save</button>

</**form**>

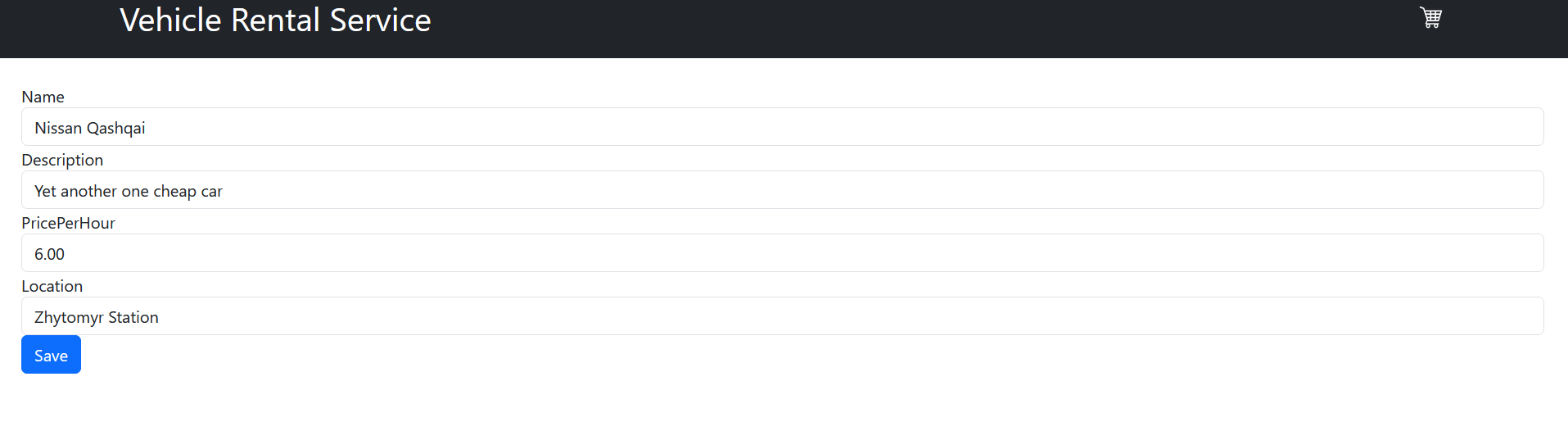


Рис.2.Редагування ТЗ

**DELETE:**

За можливість видалення відповідає метод DeleteVehicle контролера Home.

Лістінг методу DeleteVehicle:

public async Task<IActionResult> DeleteVehicle(long id)

{

var vehicle = await \_repository.FindByIdAsync(id);

if (vehicle == null)

{

return NotFound();

}

\_repository.Delete(vehicle);

return RedirectToAction("Index", "Home");

}

Лістінг кнопки Delete:  
<**a** **asp-controller**="Home"

**asp-action**="DeleteVehicle"

**asp-route-id**="@v.VehicleId"

class="btn btn-primary btn-sm"

style="background-color:red">

Delete

</**a**>

**Завдання 3. Додати серверну валідацію для перевірки введених користувачем даних перед виконанням CRUD-операцій.**

Оновлена модель Vehicle:

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace VehicleRentalService.Models

{

public abstract class Vehicle

{

public long? VehicleId { get; set; }

[Required(ErrorMessage = "Please enter a vehicle name")]

public string Name { get; set; } = string.Empty;

public string? Description { get; set; } = string.Empty;

[NotMapped]

public decimal PricePerMinute { get; set; } //Old

[Required]

[Range(0.01, double.MaxValue, ErrorMessage = "Please enter a positive price")]

[Column(TypeName = "decimal(8, 2)")]

public decimal PricePerHour { get; set; }

public bool IsAvailable { get; set; }

public string? Location { get; set; }

}

}

Висновки: під час виконання Л/Р 4 я навчився навичок роботи з ORM, навчився реалізовувати операції створення, читання, оновлення та видалення даних з використанням EF Core, набув навичок роботи з механізмами валідації даних.