**A blue logo with a arrow and a symbol

Description automatically generated Технически Университет - Варна**

**Курсов проект**

**по:**

**„Технология на софтуерното производство“**

Студент: Теодор Асенов Иванов ФН:21621650 гр: 1а

Съдържание

[Увод 3](#_Toc166499871)

[Общ преглед 3](#_Toc166499872)

[Основни цели: 3](#_Toc166499873)

[Основни функционалности 3](#_Toc166499874)

[Структура на проекта 3](#_Toc166499875)

[Реализация 4](#_Toc166499876)

[Frontend (Angular) 4](#_Toc166499877)

[Общ преглед: 4](#_Toc166499878)

[Главни компоненти: 4](#_Toc166499879)

[Услуги: 5](#_Toc166499880)

[Второстепенни компоненти: 9](#_Toc166499881)

[Backend (C# с ASP.NET Core) 16](#_Toc166499882)

[Общ преглед: 16](#_Toc166499883)

[Модели: 16](#_Toc166499884)

[DTO-та 19](#_Toc166499885)

[Контролери 20](#_Toc166499886)

[База от данни 32](#_Toc166499887)

[Релационен модел: 32](#_Toc166499888)

[Тестване 33](#_Toc166499889)

[Резултат: 33](#_Toc166499890)

# Увод

## Общ преглед

Проектът представлява система за резервация на автомобили, създадена с цел оптимизиране на процесите по продажба и резервация на автомобили, като предоставя дигитален интерфейс, в който потребителите могат да разглеждат наличните автомобили, да се информират за техните характеристики и да правят резервации директно онлайн. Тази система е предназначена за автокъщи, които искат да разширят своите продажбени канали и да подобрят ангажираността на клиентите чрез дигитализацията на процесите.

## Основни цели:

**Представяне на налични автомобили:** Показване на каталог с автомобили, включващ изображения, спецификации и цени, с цел информиране на потенциалните купувачи и улеснение на процеса свързан с вземане на решение.

**Управление на резервациите:** Позволява на клиентите да правят резервации за оглед, което обогатява традиционния процес на покупка на автомобили с удобство и ефективност.

**Оптимизация на процесите по продажби:** Предоставя на продавачите заден панел за управление на автомобилния парк и преглед на предстоящи резервации.

## Основни функционалности

**Каталог с автомобили:** Потребителите могат да разглеждат обширен списък от автомобили, и да виждат детайлни страници със спецификации, снимки и информация за цените.

Регистрация на потребители: Сигурна система за управление на потребители, която позволява на клиентите да се регистрират и да влизат в профилите си.

Система за резервации: Интегрирана функционалност за резервации, която позволява на клиентите да запазят срещи за посещение в автокъщата за оглед.

## Структура на проекта

Цялостният софтуерен продукт представлява Уеб апликация с графичен потребителски интърфейс и сървърна част, обработваща информацията и записите в базата от данни.

# Реализация

## Frontend (Angular)

### Общ преглед:

Frontend частта на системата е разработена използвайки Angular, съвременна платформа за разработка на клиентски приложения. Използва се за създаване на динамичен и отзивчив потребителски интерфейс. Тя комбинира популярни технологии за Frontend разработка – HTML, SCSS(CSS), TypeScript(JavaScript). Отличаващите ѝ характеристика са изнасянето на комплексна логика в клиентската част по лесен, бърз и удобен начин.

### Главни компоненти:

**App Component -** Основният компонент, който служи като контейнер за останалите компоненти.

*import* { Component } *from* '@angular/core';

*import* { Router, RouterLink, RouterOutlet } *from* '@angular/router';

*import* { AuthService } *from* './services/auth.service';

*import* { HttpClientModule } *from* '@angular/common/http';

*import* { CommonModule } *from* '@angular/common';

*import* { NgbModule } *from* '@ng-bootstrap/ng-bootstrap';

@Component({

  selector: 'app-root',

  standalone: true,

  imports: [RouterOutlet, RouterLink, HttpClientModule, CommonModule],

  templateUrl: './app.component.html',

  styleUrl: './app.component.scss',

  providers: [AuthService, Router],

})

*export* *class* AppComponent {

  title = 'frontend';

  currentUser: any = null;

*constructor*(*private* authService: AuthService, *private* router: Router) {

*this*.authService.currentUser.subscribe(user => {

*this*.currentUser = user;

    });

  }

  logout() {

*if*(confirm('Сигурни ли сте, че искате да излезете от вашия профил?')) {

*this*.authService.logout();

    }

*this*.router.navigateByUrl('/');

  }

}

App Routes – Компонент за навигация в уеб приложението:

*import* { Routes } *from* '@angular/router';

*import* { HomeComponent } *from* './components/home/home.component';

*import* { AboutComponent } *from* './components/about/about.component';

*import* { CarsComponent } *from* './components/cars/cars.component';

*import* { ContactComponent } *from* './components/contact/contact.component';

*import* { RegisterComponent } *from* './components/register/register.component';

*import* { LoginComponent } *from* './components/login/login.component';

*import* { OrderComponent } *from* './components/order/order.component';

*import* { PanelComponent } *from* './components/panel/panel.component';

*export* *const* routes: Routes = [

    {path: '', component: HomeComponent},

    {path: 'about', component: AboutComponent},

    {path: 'cars', component: CarsComponent},

    {path: 'contact', component: ContactComponent},

    {path: 'register', component: RegisterComponent},

    {path: 'login', component: LoginComponent},

    {path: 'orders', component: OrderComponent},

    {path: 'panel', component: PanelComponent},

];

### Услуги:

Услугите (Services) в Angular, представляват компоненти от програмата, които имат специфичната задача да комуникират със сървъра от Front-end частта или да извършват операции за четене/запис в браузъра (cookies, local storage). В тях е енкапсулирана логиката за изпращането и получаването на отговор в JSON формат. Програмата съдържа следните услуги:

**Auth Service** – услуга за действия свързани с потребители: Вход, Регистрация, Извличане на потребители и др.

**Car Service** – услуга за десйствия свързани с автомобили, модели и марки и поръчки: Четене, Запис и др.

**Local Storage Service** – услуга за записването и извличането на информация в локалното пространство в браузъра. Тя бива използвана за да запази потребителя в неговият акаунт.

*// src/app/auth.service.ts*

*import* { Injectable } *from* '@angular/core';

*import* { HttpClient } *from* '@angular/common/http';

*import* { BehaviorSubject, Observable, throwError } *from* 'rxjs';

*import* { catchError, map } *from* 'rxjs/operators';

*import* { LocalStorageService } *from* './localstorage.service';

@Injectable({

  providedIn: 'root'

})

*export* *class* AuthService {

*private* baseUrl = 'https://localhost:7165/api';

*private* currentUserSubject: BehaviorSubject<any>;

*public* currentUser: Observable<any>;

*constructor*(*private* http: HttpClient, *private* localStorageService: LocalStorageService) {

*this*.currentUserSubject = new BehaviorSubject<any>(JSON.parse(*this*.localStorageService.getItem('user') ?? 'null'));

*this*.currentUser = *this*.currentUserSubject.asObservable();

  }

*public* *get* currentUserValue(): any {

*return* *this*.currentUserSubject.value;

  }

  register(firstname: string, lastname:string, username: string, email: string, phone: string, password: string) {

*return* *this*.http.post<any>(`${*this*.baseUrl}/User/register`, { firstname, lastname,username, email, phone, password })

      .pipe(

        catchError(err => {

*return* throwError(() => new Error('Грешка при регистрация! Моля, опитайте по-късно!'));

        }),

        map(user => {

*return* user;

        })

      );

  }

  login(username: string, password: string) {

*return* *this*.http.post<any>(`${*this*.baseUrl}/User/login`, { username, password })

      .pipe(

        catchError(err => {

*return* throwError(() => new Error('Грешка при влизане!'));

        }),

        map(res => {

*this*.localStorageService.setItem('user', JSON.stringify(res));

*this*.currentUserSubject.next(res);

*return* res;

        })

      );

  }

  logout() {

*this*.localStorageService.removeItem('user');

*this*.currentUserSubject.next(null);

  }

}

*// src/app/services/car.service.ts*

*import* { HttpClient } *from* '@angular/common/http';

*import* { Injectable } *from* '@angular/core';

*import* { Observable } *from* 'rxjs';

@Injectable({

  providedIn: 'root'

})

*export* *class* CarService {

*private* baseUrl = 'https://localhost:7165/api';

*constructor*(*private* http: HttpClient) {}

  getCars(): Observable<any[]> {

*return* *this*.http.get<any[]>(`${*this*.baseUrl}/Cars`);

  }

  getCarBrands(): Observable<any[]> {

*return* *this*.http.get<any[]>(`${*this*.baseUrl}/CarBrand`);

  }

  getCarModels(): Observable<any[]> {

*return* *this*.http.get<any[]>(`${*this*.baseUrl}/CarModel`);

  }

  createCarBrand(carBrand: any): Observable<any> {

*return* *this*.http.post(`${*this*.baseUrl}/CarBrand`, carBrand);

  }

  createCarModel(carModel: any): Observable<any> {

*return* *this*.http.post(`${*this*.baseUrl}/CarModel`, carModel);

  }

  createCarWithImage(formData: FormData): Observable<any> {

*return* *this*.http.post(`${*this*.baseUrl}/Cars/createwithimage`, formData);

  }

  orderCars(userId: number, orderItems: any[]): Observable<any> {

*const* orderData = { userId, orderItems };

*return* *this*.http.post(`${*this*.baseUrl}/Order`, orderData);

  }

  getOrders(userId: number): Observable<any[]> {

*return* *this*.http.get<any[]>(`${*this*.baseUrl}/Order?userId=${userId}`);

  }

  getAllOrders(): Observable<any[]> {

*return* *this*.http.get<any[]>(`${*this*.baseUrl}/Order/All`);

  }

}

*import* { DOCUMENT } *from* '@angular/common';

*import* { Inject, Injectable, PLATFORM\_ID } *from* '@angular/core';

*import* { isPlatformBrowser } *from* '@angular/common';

@Injectable({

  providedIn: 'root',

})

*export* *class* LocalStorageService {

*private* localStorage: Storage|null = null;

*constructor*(@Inject(DOCUMENT) *private* document: Document, @Inject(PLATFORM\_ID) *private* platformId: Object) {

*if* (isPlatformBrowser(*this*.platformId)) {

*this*.localStorage = *this*.document.defaultView!.localStorage;

    }

  }

  setItem(key: string, value: string): void {

*if* (*this*.localStorage) {

*this*.localStorage.setItem(key, value);

    }

  }

*// Get a value from local storage*

  getItem(key: string): string | null {

*if* (*this*.localStorage) {

*return* *this*.localStorage.getItem(key);

    }

*return* null;

  }

  removeItem(key: string): void {

*if* (*this*.localStorage) {

*this*.localStorage.removeItem(key);

    }

  }

  clear(): void {

*if* (*this*.localStorage) {

*this*.localStorage.clear();

    }

  }

}

### Второстепенни компоненти:

**About Component –** Компонент със статична информация за автокъщата.

**Cars Component –** Компонент съдържащ страницата със списък с автомобили. Той извлича резултатите от базата данни, чрез Http заявки. Съответно при конкретни действия от потребителя биват изпратени заявки към сървъра.

**Contact Component** – Компонент със статична информация за автокъщата – телефон, електронна поща.

**Home Component** -Компонент със статична информация, представляващ увод в приложението. Съдържа хипервръзки към други компоненти.

**Login Component –** Компонент, чрез който потребителите могат да влязат в своите акаунти.

**Register Component –** Компонент, чрез който потребителите могат да се регистрират.

**Order Component –** Компонент за преглед на направени поръчки.

**Panel Component –** Компонент, представляващ админски панел за добавянето на автомобили, марки и модели.

*import* { Component, OnInit } *from* '@angular/core';

*import* { CarService } *from* '../../services/car.service';

*import* { HttpClient, HttpClientModule } *from* '@angular/common/http';

*import* { CommonModule } *from* '@angular/common';

*import* { AuthService } *from* '../../services/auth.service';

@Component({

  selector: 'app-cars',

  templateUrl: './cars.component.html',

  imports:[HttpClientModule, CommonModule] ,

  standalone: true,

  providers: [CarService],

  styleUrl: './cars.component.scss'

})

*export* *class* CarsComponent *implements* OnInit {

  cars: any[] = [];

  error: string = '';

  currentUser: any = {};

  selectedCars: any[] = [];

*constructor*(*private* carService: CarService, *private* authService: AuthService) {}

  ngOnInit() {

*this*.carService.getCars().subscribe({

      next: (cars) => {

*this*.cars = cars;

      },

      error: (err) => *this*.error = 'Failed to load cars'

    });

*this*.authService.currentUser.subscribe(user => {

*this*.currentUser = user;

    });

  }

  toggleCarSelection(car: any) {

*const* index = *this*.selectedCars.findIndex(c => c.carId === car.id);

*if* (index > -1) {

*this*.selectedCars.splice(index, 1);

    } *else* {

*this*.selectedCars.push({ carId: car.id, price: car.price });

    }

  }

    isCarSelected(carId: number): boolean {

*return* *this*.selectedCars.some(car => car.carId === carId);

    }

    orderCars() {

*if* (*this*.selectedCars.length) {

*this*.carService.orderCars(*this*.currentUser.id, *this*.selectedCars).subscribe({

          next: () => {

            alert('Успешно резервирани автомобили за оглед! Екипът ни ще се свърже с вас скоро!');

*this*.selectedCars = [];

          },

          error: (err) => alert('Грешка при резервирането на автомобили')

        });

      } *else* {

        alert('Грешка! Няма избрани автомобили!');

      }

    }

}

*import* { Component } *from* '@angular/core';

*import* { HttpClient } *from* '@angular/common/http';

*import* { catchError, throwError } *from* 'rxjs';

*import* { Router } *from* '@angular/router';

*import* { FormsModule, ReactiveFormsModule } *from* '@angular/forms';

*import* { CommonModule } *from* '@angular/common';

*import* { LocalStorageService } *from* '../../services/localstorage.service';

*import* { AuthService } *from* '../../services/auth.service';

@Component({

  selector: 'app-login',

  templateUrl: './login.component.html',

  standalone: true,

  imports: [CommonModule, FormsModule]

})

*export* *class* LoginComponent {

  username: string = '';

  password: string = '';

  error: string = '';

  success: string = '';

*constructor*(*private* http: HttpClient, *private* router: Router, *private* localStorageService: LocalStorageService, *private* authService: AuthService) {}

  login() {

*this*.error = '';

*this*.success = '';

*this*.authService.login(*this*.username, *this*.password).subscribe({

      next: (res) => {

*this*.success = 'Успешен вход!';

        setTimeout(() => {

*this*.router.navigate(['/']);

        }, 1000);

      },

      error: (err) => {

*this*.error = 'Грешка при вход!';

      }

    });

  }

}

*import* { CommonModule } *from* '@angular/common';

*import* { HttpClient, HttpClientModule } *from* '@angular/common/http';

*import* { Component } *from* '@angular/core';

*import* { CarService } *from* '../../services/car.service';

*import* { AuthService } *from* '../../services/auth.service';

*import* { BulgarianDatePipe } *from* '../../pipes/bulgarian-date.pipe';

@Component({

  selector: 'app-order',

  standalone: true,

  imports: [CommonModule, BulgarianDatePipe],

  templateUrl: './order.component.html',

  styleUrl: './order.component.scss',

  providers: [HttpClient, CarService, AuthService],

})

*export* *class* OrderComponent {

  orders: any[] = [];

  error: string = '';

  currentUser: any = {};

*constructor*(*private* http: HttpClient, *private* carService: CarService, *private* authService: AuthService) {}

  ngOnInit() {

*this*.authService.currentUser.subscribe(user => {

*this*.currentUser = user;

    });

*if*(*this*.currentUser?.role?.name === 'Admin') {

*this*.carService.getAllOrders().subscribe({

        next: (orders) => *this*.orders = orders,

        error: (err) => *this*.error = 'Грешка при зареждането на резервациите!'

      })

    }

*else* {

*this*.carService.getOrders(*this*.currentUser.id).subscribe({

        next: (orders) => *this*.orders = orders,

        error: (err) => *this*.error = 'Грешка при зареждането на резервациите!'

      })

    }

  }

}

*import* { Component, OnInit } *from* '@angular/core';

*import* { FormBuilder, FormGroup, FormsModule, ReactiveFormsModule } *from* '@angular/forms';

*import* { CarService } *from* '../../services/car.service';

*import* { CommonModule } *from* '@angular/common';

@Component({

  selector: 'app-panel',

  templateUrl: './panel.component.html',

  styleUrls: ['./panel.component.scss'],

  imports: [FormsModule, ReactiveFormsModule, CommonModule],

  standalone: true,

  providers: [CarService]

})

*export* *class* PanelComponent *implements* OnInit {

  carBrandForm: FormGroup;

  carModelForm: FormGroup;

  carForm: FormGroup;

  carBrands: any[] = [];

  carModels: any[] = [];

  selectedFile: File | null = null;

*constructor*(*private* fb: FormBuilder, *private* carService: CarService) {

*this*.carBrandForm = *this*.fb.group({

      name: ''

    });

*this*.carModelForm = *this*.fb.group({

      name: '',

      carBrandId: ''

    });

*this*.carForm = *this*.fb.group({

      color: '',

      year: '',

      kilometers: '',

      price: '',

      carModelId: '',

      image: [''],

    });

  }

  onFileSelected(event: any): void {

*if* (event.target.files.length > 0) {

*this*.selectedFile = event.target.files[0];

    }

  }

  ngOnInit(): void {

*this*.loadCarBrands();

*this*.loadCarModels();

  }

  loadCarBrands() {

*this*.carService.getCarBrands().subscribe({

      next: (brands) => *this*.carBrands = brands,

      error: (err) => console.error('Грешка при зареждането на марките!', err)

    });

  }

  loadCarModels() {

*this*.carService.getCarModels().subscribe({

      next: (models) => *this*.carModels = models,

      error: (err) => console.error('Грешка при зареждането на моделите!', err)

    });

  }

  createCarBrand(): void {

*this*.carService.createCarBrand(*this*.carBrandForm.value).subscribe({

      next: () => {

        alert('Успешно добавена марка!');

*this*.loadCarBrands();

*this*.carBrandForm.reset();

      },

      error: (err) => alert('Грешка при добавянето на марка!')

    });

  }

  createCarModel(): void {

*this*.carService.createCarModel(*this*.carModelForm.value).subscribe({

      next: () => {

        alert('Успешно добавен модел!');

*this*.loadCarModels();

*this*.carModelForm.reset();

      },

      error: (err) => alert('Грешка при добавянето на модел!')

    });

  }

  uploadCar(): void {

*const* formData = new FormData();

    formData.append('color', *this*.carForm.value.color);

    formData.append('year', *this*.carForm.value.year);

    formData.append('kilometers', *this*.carForm.value.kilometers);

    formData.append('price', *this*.carForm.value.price);

    formData.append('carModelId', *this*.carForm.value.carModelId);

    formData.append('file', *this*.selectedFile ?? '');

*this*.carService.createCarWithImage(formData).subscribe({

      next: () => {

        alert('Успешно добавен автомобил!');

*this*.carForm.reset();

      },

      error: (err) => alert('Грешка при добавянето на автомобил')

    });

  }

}

*// src/app/register/register.component.ts*

*import* { Component } *from* '@angular/core';

*import* { Router } *from* '@angular/router';

*import* { AuthService } *from* '../../services/auth.service';

*import* { CommonModule } *from* '@angular/common';

*import* { FormsModule } *from* '@angular/forms';

@Component({

  selector: 'app-register',

  templateUrl: './register.component.html',

  standalone: true,

  imports: [CommonModule, FormsModule]

})

*export* *class* RegisterComponent {

  firstname: string = '';

  lastname: string = '';

  username: string = '';

  email: string = '';

  password: string = '';

  phone: string = '';

  error: string = '';

  success: string = '';

*constructor*(*private* authService: AuthService, *private* router: Router) {}

  register() {

*this*.error = '';

*this*.success = '';

*this*.authService.register(*this*.firstname, *this*.lastname, *this*.username, *this*.email, *this*.phone, *this*.password).subscribe({

      next: (res) => {

*this*.success = 'Успешно създаден профил!';

        setTimeout(() => {

*this*.router.navigate(['/login']);

        }, 2000);

      },

      error: (err) => {

*this*.error = 'Грешка при регистрация!';

      }

    });

  }

}

## Backend (C# с ASP.NET Core)

### Общ преглед:

Backend системата е изградена с ASP.NET Core, мощна и гъвкава платформа за разработка на интернет приложения и API услуги. Базира се на MVC (Model, View, Controller), което позволява лесно манипулиране на база от данни с програмен код.

### Модели:

Моделите представляват шаблонен код за това, как ще изглеждат таблиците в базата от данни. В тях могат да се дефинират полета и релационни връзки.

**A screenshot of a computer

Description automatically generated**

CarSalesContext – Общ файл дефиниращ всички модели.

Car – Автомобил

CarBrand – Автомобилна марка

CarModel – Авотмобилен модел

Order – Поръчка

OrderItem – Автомобили в поръчка

Role – Потребителска роля

User – Потребител

using Microsoft.EntityFrameworkCore;

namespace backend.Models

{

public class CarSalesContext : DbContext

{

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

public DbSet<User> Users { get; set; }

public DbSet<Car> Cars { get; set; }

public DbSet<CarBrand> CarBrands { get; set; }

public DbSet<CarModel> CarModels { get; set; }

public DbSet<Order> Orders { get; set; }

public DbSet<OrderItem> OrderItems { get; set; }

public DbSet<Role> Roles { get; set; }

}

}

namespace backend.Models

{

public class Car

{

public int Id { get; set; }

public string Color { get; set; }

public int Year { get; set; }

public decimal Price { get; set; }

public int CarModelId { get; set; }

public string? ImageUrl { get; set; }

public int Kilometers { get; set; }

public CarModel CarModel { get; set; }

}

}

namespace backend.Models

{

public class CarBrand

{

public int Id { get; set; }

public string Name { get; set; }

}

}

namespace backend.Models

{

public class CarModel

{

public int Id { get; set; }

public string Name { get; set; }

public int CarBrandId { get; set; }

public CarBrand CarBrand { get; set; }

}

}

namespace backend.Models

{

public class Order

{

public int Id { get; set; }

public DateTime OrderDate { get; set; }

public int UserId { get; set; }

public User User { get; set; }

public List<OrderItem> OrderItems { get; set; } = new List<OrderItem>();

}

}

namespace backend.Models

{

public class OrderItem

{

public int Id { get; set; }

public int OrderId { get; set; }

public int CarId { get; set; }

public Car Car { get; set; }

public decimal Price { get; set; }

}

}

namespace backend.Models

{

public class Role

{

public int Id { get; set; }

public string Name { get; set; }

}

}

namespace backend.Models

{

public class User

{

public int Id { get; set; }

public string Firstname { get; set; }

public string Lastname { get; set; }

public string Username { get; set; }

public string Email { get; set; }

public string Phone { get; set; }

public int RoleId { get; set; }

public Role Role { get; set; }

public string Password { get; set; }

}

}

### DTO-та

DTO-тата (Data Transfer Object) се използват за за входна и изходна информация в комуникацията от API услугата. Те представляват шаблон на предадената или върнатата информация от приложението.

A screenshot of a computer program

Description automatically generated

namespace backend.DTO

{

public class CarInputDTO

{

public string Color { get; set; }

public int Year { get; set; }

public decimal Price { get; set; }

public int Kilometers { get; set; }

public int CarModelId { get; set; }

}

}

namespace backend.DTO

{

public class CarModelDTO

{

public string Name { get; set; }

public int CarBrandId { get; set; }

}

}

namespace backend.DTO

{

public class LoginDTO

{

public string Username { get; set; }

public string Password { get; set; }

}

}

namespace backend.DTO

{

public class OrderDTO

{

public int UserId { get; set; }

public List<OrderItemDTO> OrderItems { get; set; }

}

}

namespace backend.DTO

{

public class OrderItemDTO

{

public int CarId { get; set; }

public decimal Price { get; set; }

}

}

namespace backend.DTO

{

public class RegisterDTO

{

public required string Firstname { get; set; }

public required string Lastname { get; set; }

public required string Username { get; set; }

public required string Email { get; set; }

public required string Password { get; set; }

public required string Phone { get; set; }

}

}

### Контролери

Контролерите представляват най-ключовата част от приложението. Чрез тях се извършва комуникацията. Те са отговорни за изпълнението на действието, след получаване на заявка от потребител, чрез API услугата. Самите те дефинират крайните точки (endpoints) за комуникация. В тях присъстват основните **CRUD** операции (Create, Read, Update, Delete)

A screenshot of a computer

Description automatically generated

**CarBrandController** – основни CRUD операции за модела Car Brand

**CarModelController** – основни CRUD операции за модела Car Model

**CarsController –** основни CRUD операции за модела Cars и допълнителна логика за манипулирането на изображения в приложението.

**OrderController –** основни CRUD операции за поръчки и бизнес логиката за запис и четене на поръчки спрямо потребителя и неговата роля.

**UserController –** контролер, съдържащ операциите регистрация,вход, достъп до потребител и др.

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using backend.Models;

using System.Linq;

using System.Threading.Tasks;

namespace backend.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class CarBrandController : ControllerBase

{

private readonly CarSalesContext \_context;

public CarBrandController(CarSalesContext context)

{

\_context = context;

}

// GET: api/CarBrand

[HttpGet]

public async Task<ActionResult<IEnumerable<CarBrand>>> GetCarBrands()

{

return await \_context.CarBrands.ToListAsync();

}

// GET: api/CarBrand/5

[HttpGet("{id}")]

public async Task<ActionResult<CarBrand>> GetCarBrand(int id)

{

var carBrand = await \_context.CarBrands.FindAsync(id);

if (carBrand == null)

{

return NotFound();

}

return carBrand;

}

// POST: api/CarBrand

[HttpPost]

public async Task<ActionResult<CarBrand>> PostCarBrand(CarBrand carBrand)

{

\_context.CarBrands.Add(carBrand);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetCarBrand", new { id = carBrand.Id }, carBrand);

}

// PUT: api/CarBrand/5

[HttpPut("{id}")]

public async Task<IActionResult> PutCarBrand(int id, CarBrand carBrand)

{

if (id != carBrand.Id)

{

return BadRequest();

}

\_context.Entry(carBrand).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!CarBrandExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return NoContent();

}

// DELETE: api/CarBrand/5

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteCarBrand(int id)

{

var carBrand = await \_context.CarBrands.FindAsync(id);

if (carBrand == null)

{

return NotFound();

}

\_context.CarBrands.Remove(carBrand);

await \_context.SaveChangesAsync();

return NoContent();

}

private bool CarBrandExists(int id)

{

return \_context.CarBrands.Any(e => e.Id == id);

}

}

}

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using backend.Models;

using System.Linq;

using System.Threading.Tasks;

using backend.DTO;

namespace backend.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class CarModelController : ControllerBase

{

private readonly CarSalesContext \_context;

public CarModelController(CarSalesContext context)

{

\_context = context;

}

// GET: api/CarModel

[HttpGet]

public async Task<ActionResult<IEnumerable<CarModel>>> GetCarModels()

{

return await \_context.CarModels.Include(cm => cm.CarBrand).ToListAsync();

}

// GET: api/CarModel/5

[HttpGet("{id}")]

public async Task<ActionResult<CarModel>> GetCarModel(int id)

{

var carModel = await \_context.CarModels.Include(cm => cm.CarBrand).FirstOrDefaultAsync(cm => cm.Id == id);

if (carModel == null)

{

return NotFound();

}

return carModel;

}

// POST: api/CarModel

[HttpPost]

public async Task<ActionResult<CarModel>> PostCarModel([FromBody] CarModelDTO carModelDTO)

{

if (!\_context.CarBrands.Any(cb => cb.Id == carModelDTO.CarBrandId))

{

return BadRequest("Invalid Car Brand ID");

}

var carModel = new CarModel

{

Name = carModelDTO.Name,

CarBrandId = carModelDTO.CarBrandId

};

\_context.CarModels.Add(carModel);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetCarModel", new { id = carModel.Id }, carModel);

}

// PUT: api/CarModel/5

[HttpPut("{id}")]

public async Task<IActionResult> PutCarModel(int id, CarModel carModel)

{

if (id != carModel.Id)

{

return BadRequest();

}

\_context.Entry(carModel).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!CarModelExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return NoContent();

}

// DELETE: api/CarModel/5

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteCarModel(int id)

{

var carModel = await \_context.CarModels.FindAsync(id);

if (carModel == null)

{

return NotFound();

}

\_context.CarModels.Remove(carModel);

await \_context.SaveChangesAsync();

return NoContent();

}

private bool CarModelExists(int id)

{

return \_context.CarModels.Any(e => e.Id == id);

}

}

}

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using backend.Models;

using System.Collections.Generic;

using System.Threading.Tasks;

using backend.Models;

using backend.DTO;

namespace backend.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class CarsController : ControllerBase

{

private readonly CarSalesContext \_context;

public CarsController(CarSalesContext context)

{

\_context = context;

}

// GET: api/Cars

[HttpGet]

public async Task<ActionResult<IEnumerable<Car>>> GetCars()

{

var cars = await \_context.Cars

.Include(c => c.CarModel)

.ThenInclude(cm => cm.CarBrand)

.ToListAsync();

return cars;

}

// GET: api/Cars/5

[HttpGet("{id}")]

public async Task<ActionResult<Car>> GetCar(int id)

{

var car = await \_context.Cars

.Include(c => c.CarModel)

.ThenInclude(cm => cm.CarBrand)

.FirstOrDefaultAsync(c => c.Id == id);

if (car == null)

{

return NotFound();

}

return car;

}

// POST: api/Cars

[HttpPost]

public async Task<ActionResult<Car>> PostCar(Car car)

{

\_context.Cars.Add(car);

await \_context.SaveChangesAsync();

return CreatedAtAction(nameof(GetCar), new { id = car.Id }, car);

}

// PUT: api/Cars/5

[HttpPut("{id}")]

public async Task<IActionResult> PutCar(int id, Car car)

{

if (id != car.Id)

{

return BadRequest();

}

\_context.Entry(car).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!\_context.Cars.Any(e => e.Id == id))

{

return NotFound();

}

else

{

throw;

}

}

return NoContent();

}

// DELETE: api/Cars/5

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteCar(int id)

{

var car = await \_context.Cars.FindAsync(id);

if (car == null)

{

return NotFound();

}

\_context.Cars.Remove(car);

await \_context.SaveChangesAsync();

return NoContent();

}

[HttpPost]

[Route("upload")]

public async Task<IActionResult> UploadCarImage([FromForm] int carId, [FromForm] IFormFile file)

{

var car = await \_context.Cars.FindAsync(carId);

if (car == null)

{

return NotFound();

}

var path = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot/images", file.FileName);

using (var stream = new FileStream(path, FileMode.Create))

{

await file.CopyToAsync(stream);

}

car.ImageUrl = $"images/{file.FileName}";

await \_context.SaveChangesAsync();

return Ok(new { path = car.ImageUrl });

}

[HttpPost]

[Route("createwithimage")]

public async Task<ActionResult<Car>> CreateCarWithImage([FromForm] CarInputDTO carInput, [FromForm] IFormFile file)

{

if (file == null || carInput == null)

{

return BadRequest("Invalid car data or file.");

}

var imagePath = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot/images", file.FileName);

using (var stream = new FileStream(imagePath, FileMode.Create))

{

await file.CopyToAsync(stream);

}

var car = new Car

{

Color = carInput.Color,

Year = carInput.Year,

Price = carInput.Price,

CarModelId = carInput.CarModelId,

Kilometers = carInput.Kilometers,

ImageUrl = $"images/{file.FileName}"

};

\_context.Cars.Add(car);

await \_context.SaveChangesAsync();

return CreatedAtAction(nameof(GetCar), new { id = car.Id }, car);

}

}

}

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using backend.Models;

using System.Collections.Generic;

using System.Threading.Tasks;

using backend.Models;

using backend.DTO;

namespace backend.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class OrderController : ControllerBase

{

private readonly CarSalesContext \_context;

public OrderController(CarSalesContext context)

{

\_context = context;

}

// GET: api/Users/All

[HttpGet]

[Route("All")]

public async Task<ActionResult<IEnumerable<Order>>> GetAllOrders()

{

var orders = await \_context.Orders

.Include(o => o.OrderItems)

.ThenInclude(oi => oi.Car)

.ThenInclude(c => c.CarModel)

.ThenInclude(cm => cm.CarBrand)

.Include(u => u.User)

.ToListAsync();

return orders;

}

// GET: api/Users

[HttpGet]

public async Task<ActionResult<IEnumerable<Order>>> GetOrders(int userId)

{

var orders = await \_context.Orders

.Where(o => o.UserId == userId)

.Include(o => o.OrderItems)

.ThenInclude(oi => oi.Car)

.ThenInclude(c => c.CarModel)

.ThenInclude(cm => cm.CarBrand)

.ToListAsync();

return orders;

}

// GET: api/Orders/5

[HttpGet("{id}")]

public async Task<ActionResult<Order>> GetOrder(int id)

{

var order = await \_context.Orders.FindAsync(id);

if (order == null)

{

return NotFound();

}

return order;

}

// POST: api/Orders

[HttpPost]

public async Task<ActionResult<Order>> PostOrder([FromBody] OrderDTO orderDto)

{

if (!\_context.Users.Any(u => u.Id == orderDto.UserId))

{

return BadRequest("Invalid User ID");

}

var order = new Order

{

OrderDate = DateTime.Now,

UserId = orderDto.UserId,

OrderItems = orderDto.OrderItems.Select(oi => new OrderItem

{

CarId = oi.CarId,

Price = oi.Price

}).ToList()

};

\_context.Orders.Add(order);

await \_context.SaveChangesAsync();

return CreatedAtAction(nameof(GetOrder), new { id = order.Id }, order);

}

// PUT: api/Orders/5

[HttpPut("{id}")]

public async Task<IActionResult> PutOrder(int id, Order order)

{

if (id != order.Id)

{

return BadRequest();

}

\_context.Entry(order).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!\_context.Cars.Any(e => e.Id == id))

{

return NotFound();

}

else

{

throw;

}

}

return NoContent();

}

// DELETE: api/Orders/5

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteOrder(int id)

{

var order = await \_context.Cars.FindAsync(id);

if (order == null)

{

return NotFound();

}

\_context.Cars.Remove(order);

await \_context.SaveChangesAsync();

return NoContent();

}

}

}

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using backend.Models;

using System.Collections.Generic;

using System.Threading.Tasks;

using backend.Models;

using backend.DTO;

namespace backend.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class UserController : ControllerBase

{

private readonly CarSalesContext \_context;

public UserController(CarSalesContext context)

{

\_context = context;

}

// GET: api/Users

[HttpGet]

public async Task<ActionResult<IEnumerable<User>>> GetUsers()

{

return await \_context.Users.ToListAsync();

}

// GET: api/Users/5

[HttpGet("{id}")]

public async Task<ActionResult<User>> GetUser(int id)

{

var user = await \_context.Users

.Include(u => u.Role)

.FirstOrDefaultAsync(u => u.Id == id);

if (user == null)

{

return NotFound();

}

user.Password = null;

return user;

}

// POST: api/Users

[HttpPost]

public async Task<ActionResult<User>> PostCar(User user)

{

\_context.Users.Add(user);

await \_context.SaveChangesAsync();

return CreatedAtAction(nameof(GetUser), new { id = user.Id }, user);

}

// PUT: api/Users/5

[HttpPut("{id}")]

public async Task<IActionResult> PutUser(int id, User user)

{

if (id != user.Id)

{

return BadRequest();

}

\_context.Entry(user).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!\_context.Cars.Any(e => e.Id == id))

{

return NotFound();

}

else

{

throw;

}

}

return NoContent();

}

// DELETE: api/Users/5

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteUser(int id)

{

var user = await \_context.Cars.FindAsync(id);

if (user == null)

{

return NotFound();

}

\_context.Cars.Remove(user);

await \_context.SaveChangesAsync();

return NoContent();

}

// POST: api/Users/register

[HttpPost("register")]

public async Task<ActionResult<User>> Register([FromBody] RegisterDTO registerDTO)

{

if (\_context.Users.Any(u => u.Username == registerDTO.Username))

{

return BadRequest("Username already exists.");

}

var defaultRole = await \_context.Roles.FirstOrDefaultAsync(r => r.Name == "User");

if (defaultRole == null)

{

return BadRequest("Default role not found.");

}

var user = new User

{

Firstname = registerDTO.Firstname,

Lastname = registerDTO.Lastname,

Username = registerDTO.Username,

Email = registerDTO.Email,

Password = registerDTO.Password,

Phone = registerDTO.Phone,

RoleId = defaultRole.Id

};

\_context.Users.Add(user);

await \_context.SaveChangesAsync();

user.Password = null;

return CreatedAtAction(nameof(GetUser), new { id = user.Id }, user);

}

// POST: api/Users/login

[HttpPost("login")]

public async Task<ActionResult<User>> Login([FromBody] LoginDTO loginDTO)

{

var user = await \_context.Users

.Include(u => u.Role)

.FirstOrDefaultAsync(u => u.Username == loginDTO.Username && u.Password == loginDTO.Password);

if (user == null)

{

return Unauthorized("Invalid username or password.");

}

user.Password = null;

return Ok(user);

}

}

}

Цялостната API услуга е комбинирана с документация за използването ѝ, чрез автоматично генерираща се Swagger уеб страница: A screenshot of a computer

Description automatically generated

## База от данни

За база от данни е избран лесно съвместимият с C# ASP.NET Core: MySQL.

Базата от данни се генерира и обновява, чрез прости команди, поради присъствието на шаблонните модели. При промяна единствено трябва да се създаде *миграция* – файл, описващ промените в структурата на базата от данни, чрез командния ред. След това трябва да се изпълни:





Това осигурява лесно и гъвкаво използване, създаване и обновяване на базата от данни.

Тя може да бъде прегледана, чрез всеки клиент, поддържащ MySQL. За разработка е използван XAMPP:

A screenshot of a computer

Description automatically generated

### Релационен модел:

**A screenshot of a computer

Description automatically generated**

# Тестване

Тестването на системата е важен етап от разработката, който позволява проверката на функционалностите и потребителския интерфейс на приложението. Това тестване включва следните ключови аспекти:

Тестване на функционалност: Преглеждане на всички функционалности на приложението.

Тестване на потребителски интерфейс: Проверка дали потребителският интерфейс е лесен и удобен за иползване

Тестване на изключения: Въвеждане на грешни входни данни, за да се провери обработката им от приложението.

Тестване на сценарии за употреба: Изпълнение на различни потребителски сценарии, с цел постигане на желан краен резултат.

## Резултат:

Системата функционира успешно и крайните резултати са постигнати.