DSZC Mechwart András Gépipari és Informatikai Technikum
13.C
Szoftverfejlesztő és -tesztelő ágazat

PremiumCars Használtautó-kereskedelmi hálózat fejlesztői dokumentációja.

Vizsgaremek

Készítette
Gál Bálint Szelim

Készítette
Kiss Dániel

2025. március 31.

Tartalomjegyzék

1. fejezet

Bevezetés

A jelen vizsgaremekben bemutatott rendszer egy autókereskedési platform, mely célja, hogy hatékony és felhasználóbarát megoldást kínáljon a használt autók online értékesítésére és kezelésére. A projekt a *PremiumCars* névre hallgat, és lehetőséget biztosít a felhasználók számára autók mentésére, keresésére, valamint a hozzájuk kapcsolódó adatkezelésre, mint például a mentett autók listázása és törlése.

A téma választásának indoklása részben a személyes érdeklődésen alapul, mivel a szoftverfejlesztés, különösen az üzleti alkalmazások fejlesztése iránti vágy mindig is jelen volt. Azonban a projekt iránti érdeklődésem egyik fő motivációja az volt, hogy Magyarországon rendkívül ritka az olyan típusú autókereskedelmi modell, amely online platformot biztosít a használt autók értékesítésére. Ez a gondolat akkor kezdett el formálódni, amikor meglátogattam egy *WeCars* autókereskedőt Japánban. Bár használt autókról volt szó, az ott tapasztalt élmény azt a benyomást keltette bennem, mintha egy vadonatúj autót vásárolnék. Az autókereskedés a minőségi szolgáltatásaival és professzionális hozzáállásával egy olyan vásárlási élményt nyújtott, amely messze túlmutatott a hagyományos használt autópiaci elvárásokon. Ez az élmény és a japán piac hatása inspirálta a projektet.

A projekt tehát elsősorban a japán piacra összpontosít, figyelembe véve a helyi autókereskedelmi trendeket, mint például a *WeCars* és más hasonló cégek működését. Azonban a fejlesztés során figyelembe vettük, hogy a rendszer egy kis átalakítással a magyar piacra is alkalmazható lehet. A japán autópiacra jellemző szolgáltatási szint és vásárlói élmény például inspiráló lehet Magyarország számára is, ahol a használt autók értékesítése gyakran kisebb vállalkozások körében zajlik, és az online platformok még nem elterjedtek.

A szakmai célkitűzés a projektben, hogy egy olyan zökkenőmentes, könnyen kezelhető platformot biztosítson, amely egyszerűsíti az autóvásárlás és -értékesítés folyamatát. A rendszer valós üzleti környezetben alkalmazható megoldásokat céloz, és nemcsak az autókereskedők, hanem a vásárlók számára is értékes lehetőségeket kínál. A téma tehát egy aktuális és gyakorlati probléma megoldására tett kísérlet, amely mind a fejlesztő, mind a felhasználók számára hasznos ta-

pasztalatokat biztosít, és a jövőben akár a magyar autópiacon is sikeres lehet.

2. fejezet

Csapatmunka

2.1. Csapatmunka és Kommunikáció

A projekt fejlesztése során a csapattagok napi szinten együtt dolgoztak, így a személyes kommunikáció volt a legfontosabb eszköz. Emellett online platformokat is használtunk a hatékony munkavégzés és a fejlesztési folyamatok követése érdekében.

Kommunikációs Eszközök

- **Személyes együttműködés** – Mivel napi 8 órát együtt töltöttünk a fejlesztés során, a legtöbb probléma megbeszélésére és megoldására azonnal sor kerülhetett. Ez jelentősen gyorsította a fejlesztési folyamatot.
- **Discord** – Online megbeszélésekhez és képernyőmegosztásos hibakereséshez használtuk, amikor nem volt lehetőség személyes találkozásra.
- **Messenger** – Gyors üzenetváltásokhoz és szervezési kérdésekhez használtuk.
- **JIRA** – A feladatok nyomon követésére és a munkafolyamatok strukturálására alkalmaztuk. A fejlesztési feladatokat kisebb egységekre bontottuk, és a státuszukat folyamatosan frissítettük.

Feladatmegosztás

A csapat két fő fejlesztőből állt, akik egyértelműen meghatározott felelősségi köröket láttak el:

- **Gál Bálint Szelim** – A backend fejlesztéséért, a web API kapcsolatokért és a WPF alkalmazás nagy részéért volt felelős. Az adatbázis-kezelés, az üzleti logika és az API-hívások implementálása az ő feladata volt.

- **Kiss Dániel** – A weboldal frontend fejlesztésének teljes egészéért volt felelős, kivéve az API-hívásokat. Az oldalak struktúrája, a komponensek és a vizuális megjelenés hozzá tartoztak.

A személyes együttműködés kulcsszerepet játszott a projekt sikerességében, mivel a folyamatos kommunikáció és az azonnali problémamegoldás lehetővé tette a hatékony fejlesztést.

3. fejezet

Technológiai választások és fejlesztői eszközök

3.1. Technológiai választások

A projekt fejlesztése során a következő technológiákat választottuk:

- **C# .NET 8 Web API** – A backend fejlesztésére használtuk, mivel modern, stabil és jól támogatott keretrendszer, amely lehetővé teszi a gyors fejlesztést.
- **PostgreSQL** – Az adatbáziskezeléshez ezt választottuk, mivel a MySQL használata macOS alatt az Entity Frameworkkel együtt furcsa hibákat eredményezett, amelyek megoldása nem volt ésszerű.
- **React + TypeScript** – A frontend fejlesztéséhez TypeScriptet használtunk, mivel erősebb típusellenőrzést biztosít, ami csökkenti a hibák számát.
- **WPF (Windows Presentation Foundation)** – Az asztali alkalmazás fejlesztésére, mivel lehetővé teszi az MVVM nélküli gyors fejlesztést és jól integrálható a .NET API-val.

3.2. Fejlesztői eszközök

A fejlesztés során a következő eszközöket használtuk:

- **JetBrains Rider** – A .NET alkalmazás fejlesztésére választottuk, mivel kiváló támogatást nyújt a C# és a .NET környezet számára.
- **Visual Studio 2022** – A WPF fejlesztéshez használtuk, mivel teljes körű támogatást nyújt a .NET technológiákhoz és XAML-hez.

- **WebStorm** – A frontend fejlesztéséhez React és TypeScript támogatás miatt.
- **Visual Studio Code** – Könnyűsúlyú szerkesztőként használtuk gyors kódolási feladatokhoz.
- **Navicat Premium** – Az adatbázis kezeléséhez és vizuális tervezéséhez.
- **pgAdmin 4** – Az adatbázis kezeléséhez és az SQL lekérdezések teszteléséhez.
- **Postman** – Az API végpontok teszteléséhez.
- **Git + GitHub** – Verziókezeléshez és csapatmunkához.

3.3. Külső csomagok és függőségek

A fejlesztés során a következő külső csomagokat használtuk:

- **AutoMapper (13.0.1)** – Az objektumok közötti automatikus leképezéshez az üzleti logikában.
- **Microsoft.AspNetCore.Authentication.JwtBearer (8.0.11)** – A JWT-alapú hitelesítés kezeléséhez.
- **Microsoft.AspNetCore.Identity.EntityFrameworkCore (8.0.11)** – Az Identity kezeléséhez Entity Framework Core segítségével.
- **Microsoft.EntityFrameworkCore.Proxies (9.0.0)** – Az adatbázis entitások lazy loading funkcióinak támogatásához.
- **Microsoft.EntityFrameworkCore.Tools (9.0.0)** – Az Entity Framework migrációk és adatbáziskezelés támogatásához.
- **Microsoft.EntityFrameworkCore (9.0.0)** – Az adatbázis-kezeléshez és az ORM réteg megvalósításához.
- **Npgsql.EntityFrameworkCore.PostgreSQL (9.0.2)** – A PostgreSQL adatbázis Entity Framework Core támogatásához.
- **Swashbuckle.AspNetCore (6.4.0)** – Az API dokumentáció generálásához Swagger segítségével.

3.4. Egyéb eszközök és segédprogramok

A fejlesztés során egyéb eszközöket és saját szkripteket is használtunk a hatékonyabb munkavégzés érdekében:

- **Python szkript tömeges képletöltéshez** – Egy saját Python szkriptet írtunk, amely segített autóhirdetések képeit tömegesen letölteni, hogy az adatbázisban valósághű demóadatokat használhassunk.
- **iTerm2** – A terminálos munkavégzéshez macOS-en.

A fent említett technológiák és eszközök együttes használata biztosította a rendszer stabilitását és skálázhatóságát.

4. fejezet

Telepítés és konfiguráció

4.1. Szerver Beállítások

Ez a szakasz bemutatja a Hasznaltauto-kereskedelmi rendszer szerverének beállítási lépéseit.

4.1.1. Előfeltételek

A projekt szerverének futtatásához az alábbi szoftverekre van szükség:

- **.NET 8** - A szerver és a backend fejlesztéséhez.
- **PostgreSQL** - Az adatbázis kezeléséhez.

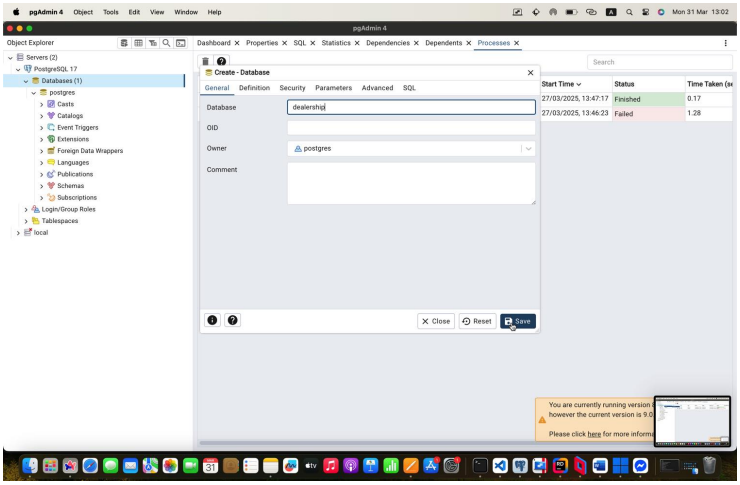
4.1.2. Adatbázis Telepítése

A projekt adatbázisának beállításához importáld a mellékelt adatbázis dump fájlt a PostgreSQL szerveredbe (az új "dealership" adatbázisba):

- Töltsd le a dump fájlt a GitHub oldalunkról: A projekt github oldalda.
- Nyisd meg a pgAdmin4-et, és hozd létre a `dealership` adatbázist.
- Importáld a dump fájlt a `dealership` adatbázisba.

2. Új adatbázis létrehozása

1. Nyissuk meg **pgAdmin4**-et, és jelentkezzünk be. 2. A bal oldali menüben kattintsunk a **Servers** > **PostgreSQL** elemre. 3. Kattintsunk jobb gombbal a **Databases** elemre, majd válasszuk az **Create** > **Database...** opciót. 4. Az új ablakban adjuk meg az adatbázis nevét, például: `dealership`.



4.1. ábra. Új adatbázis létrehozása pgAdmin4-ben

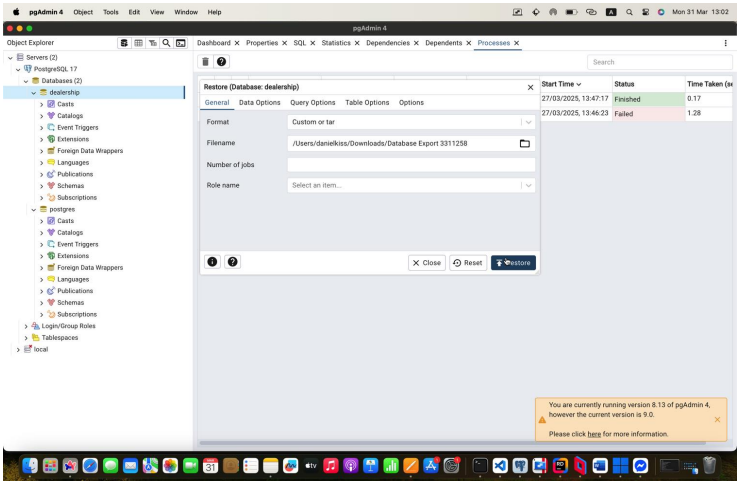
5. A **Save** gombbal hozzuk létre az adatbázist.

3. Adatbázis dump importálása

Az adatbázis feltöltéséhez a már meglévő dump fájl segítségével a következő lépéseket kell követni:

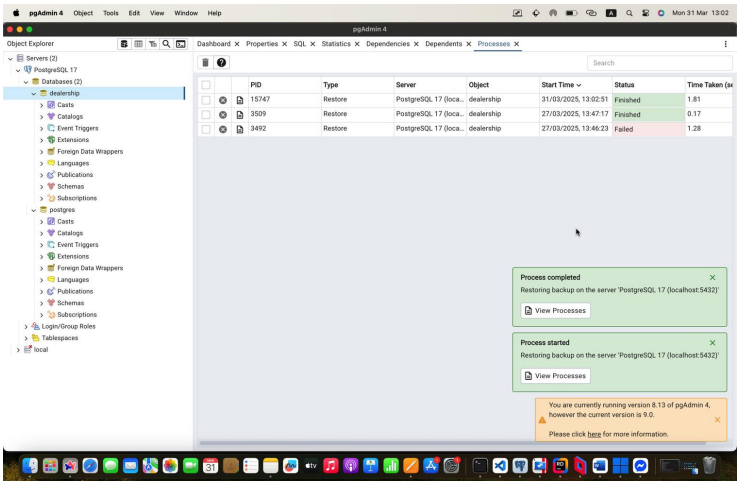
1. Nyissuk meg a pgAdmin4-et, és csatlakozzunk a PostgreSQL szerverhez. 2. A bal oldali menüben bontsuk ki a **Databases** listát, és válasszuk ki az előző lépésben létrehozott `dealership` adatbázist. 3. Kattintsunk jobb gombbal az adatbázisra, majd válasszuk az **Restore...** opciót.
4. A megnyíló ablakban:

- A **Filename** mezőben tallózzuk be az adatbázis dump fájlt (`dealership_dump.sql`).
- A **Format** mezőnél válasszuk a `Custom or tar` opciót, ha a dump fájl bináris formátumú (`.backup` vagy `.tar`).
- Ha a dump fájl SQL formátumú, válasszuk a `Plain` formátumot.



4.2. ábra. Adatbázis visszaállítása (Restore) menü

5. Kattintsunk a **Restore** gombra az importálás elindításához.



4.3. ábra. Importálás befejeződött

4.1.3. Szerver Beállítása

A szerver konfigurálásához módosítanod kell a következő fájlokat:

appSettings.json

A `appSettings.json` fájlban állítsd be a PostgreSQL adatbázis kapcsolatot az alábbi minta szerint:

```
"ConnectionStrings": {
    "PostgreSqlConnection": "Host=localhost;Port=5432;Database=dealership;
    Username=postgres;Password=qweasd"
}
```

4.1.4. NuGet Függőségek Telepítése

A szükséges NuGet függőségek telepítéséhez futtasd a következő parancsot a projekt könyvtárában:

```
dotnet restore
```

4.1.5. Projekt Futtatása

A szerver futtatásához az alábbi parancsot használhatod:

```
dotnet run --urls "https://0.0.0.0:7268;http://0.0.0.0:5085"
```

Ez elindítja a szervert, és biztosítja, hogy HTTPS-en is működjön.

Mivel a 0.0.0.0-s cím minden címre bindeolja, localhost használata is lehetséges ha nem szükséges az hogy virtuális gépen vagy más hálózati eszközről elérhető legyen.

5. fejezet

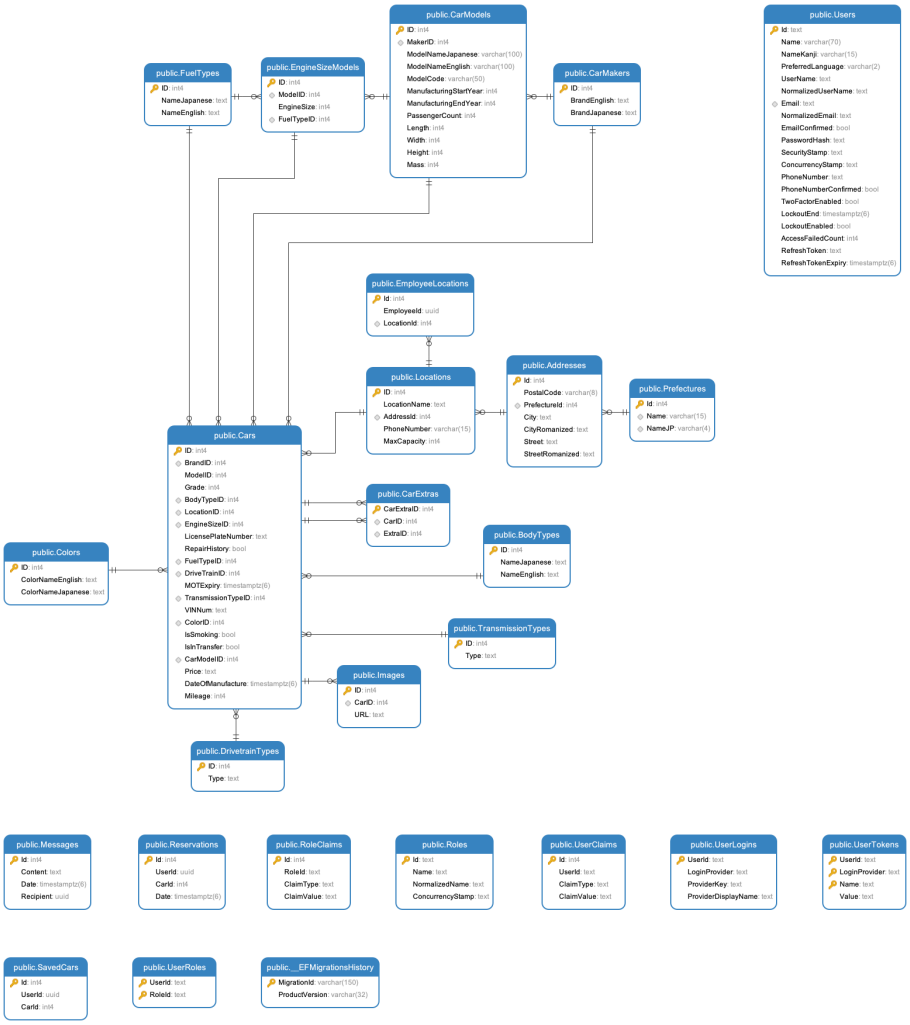
Backend dokumentációja

5.1. Adatbázis felépítése

A rendszer PostgreSQL adatbázist használ az adatok tárolására és kezelésére. Az adatbázis struktúráját az Entity Framework határozza meg, amely automatikusan kezeli a táblák létrehozását és kapcsolatokat az alkalmazásban definiált modellek alapján.

A főbb entitások közé tartoznak a járművek (`Car`), a járműmodellek (`CarModel`), a felhasználók (`User`), a foglalások (`Reservation`), valamint a különböző egyéb kapcsolódó entitások, például a helyszínek (`Location`) és az üzemanyagtípusok (`FuelType`). Az Entity Framework biztosítja az adatintegritást és támogatja a kapcsolatokat az entitások között.

Az adatbázisban található táblák közötti kapcsolatokat az alábbi ábra szemlélteti. Minden kapcsolat az EF által kezelt idegen kulcsokon alapul, amely biztosítja az adatok konzisztenciáját és az üzleti logika megfelelő működését.



5.1. ábra. Az adatbázis sémája és az entitások közötti kapcsolatok

A PostgreSQL mellett való döntés egyik fő oka az volt, hogy a MySQL használata során az Entity Framework MacOS-en furcsa hibákat produkált, amelyek megakadályozták a fejlesztést. Mivel a fejlesztői környezetben a Windows használata nem volt opció, alternatív megoldásra volt szükség. A PostgreSQL ezzel szemben stabilan működött, és semmilyen problémát nem okozott a fejlesztés során.

Mivel az alkalmazás az Entity Framework-öt használja az adatbázis-kezeléshez, a háttérben futó adatbázismotor cseréje viszonylag egyszerű feladat lenne, ha a jövőben mégis szükség lenne rá. Az EF biztosítja a platformfüggetlen működést, lehetővé téve az adatbázis sémájának módosítását anélkül, hogy manuálisan kel-

lene beavatkozni az adatbázisba.

Az adatbázis minden egyes táblájának részletes definíciója a modellek között található meg. Az Entity Framework konfigurálja az adatbázis tábláit, és az inicializálási folyamat során bizonyos táblákat előzetesen feltölt az alapvető működéshez szükséges adatokkal. Például az üzemanyagtípusok (`FuelType`) és a prefektúrák (`Prefecture`) adatai az adatbázis inicializálásakor automatikusan bekerülnek. Ezeket most nem szeretném részletesen kiemelni, hiszen az ApplicationDbContext megtekintésével könnyen fény derül hogy melyek tartalmaznak HasData()-t.

5.2. Fájlstruktúra

A projekt mappastruktúrája az alábbiak szerint van felépítve:

5.2.1. Root könyvtár

A projekt gyökérkönyvtárában találhatóak a projekt legfontosabb fájljai és beállításai:

- **Program.cs**: Az alkalmazás belépési pontja, ahol az indítás és konfigurálás történik.
- **appsettings.json**: Az alkalmazás globális konfigurációs fájlja, amely adatbáziskapcsolatokat és egyéb beállításokat tartalmaz.
- **appsettings.Development.json**: Fejlesztési környezethez szükséges külön konfiguráció.
- **DealershipSystem.http**: A HTTP-kéréseket és válaszokat tartalmazó fájl.

5.2.2. Dependencies

A `Dependencies` mappa tartalmazza az alkalmazás által használt összes külső függőséget:

- **Assemblies**: Az összes szükséges assembly a projekt működéséhez.
- **Analyzers**: Az alkalmazás által használt kód-elemző eszközök.
- **Packages**: A projekt által használt NuGet csomagok.
- **Frameworks**: A projekt által használt keretrendszerek és platformok.

5.2.3. wwwroot

A statikus fájlokat, mint például képeket és egyéb webes erőforrásokat tartalmazza. A példák a következő képeket tartalmazzák:

- `037d874d-d51b-4914-ae53-a847cf749f0f_9572469A30241026W00106.jpg`
- `f363231b-e785-456f-bf59-94c27958880e_9572469A30241026W00107.jpg`

5.2.4. Configurations

Ez a mappa tartalmazza az alkalmazás konfigurációs fájljait:

- **PrefectureConfiguration.cs**: A prefektúrákkal kapcsolatos konfigurációk.

5.2.5. Context

Ez a mappa az adatbázis-kapcsolatok kezeléséért felelős fájlokat tartalmazza:

- **ApplicationDbContext.cs**: Az adatbázis inicializálását és konfigurálását végző fájl.

5.2.6. Controllers

A vezérlők felelősek a HTTP-kérések kezeléséért, valamint az üzleti logika irányításáért. Az alábbi vezérlők találhatók a projektben:

- **CarController.cs**: A járművekkel kapcsolatos kéréseket kezeli, mint például a járművek lekérése, létrehozása, frissítése és törlése.
- **CarMakerController.cs**: A járműgyártókkal kapcsolatos kéréseket kezeli, beleértve a gyártók listázását, hozzáadását és törlését.
- **CarMetadataController.cs**: A járművek metaadataival kapcsolatos műveleteket kezeli, például a kiegészítő adatok és paraméterek kezelését.
- **CarModelController.cs**: A járműmodellekhez tartozó kéréseket kezeli, mint a modellek listázása, hozzáadása és módosítása.
- **EmployeeLocationController.cs**: A dolgozói helyekkel kapcsolatos műveleteket kezeli, például a dolgozók helyeinek frissítését és kezelését.
- **EngineController.cs**: A motorokkal kapcsolatos kéréseket kezeli, például a motorok típusainak és paramétereinek lekérdezését és frissítését.
- **ImageController.cs**: A képekkel kapcsolatos műveleteket kezeli, például a képek feltöltését és lekérését a járművekhez.
- **LocationController.cs**: A helyekkel kapcsolatos kéréseket kezeli, mint például a helyek listázása, hozzáadása és törlése.

- **MessageController.cs**: Az üzenetkezeléssel kapcsolatos műveleteket kezeli, például üzenetek küldését és fogadását.
- **ReservationController.cs**: A foglalásokkal kapcsolatos kéréseket kezeli, például foglalások létrehozását, módosítását és törlését.
- **SavedCarsController.cs**: A mentett járművekkel kapcsolatos műveleteket kezeli, például járművek mentését és törlését.
- **UserController.cs**: A felhasználókkal kapcsolatos kéréseket kezeli, mint például felhasználói adatok lekérése, regisztráció és bejelentkezés.

5.2.7. DTO

A Data Transfer Object (DTO) osztályok az adatokat hordozzák az alkalmazás különböző rétegei között:

- **AddressDTO.cs**: Címadatokat tartalmazó DTO.
- **AdminUserCreateDTO.cs**: Adminisztrátori felhasználó létrehozását tartalmazó DTO.
- **AdminUserUpdateDTO.cs**: Adminisztrátori felhasználó frissítését tartalmazó DTO.
- **BodyTypeDTO.cs**: Karosszériatípus adatokat tartalmazó DTO.
- **CarDTO.cs**: Járműadatokat tartalmazó DTO.
- **CarExtraDTO.cs**: Jármű extrák adatokat tartalmazó DTO.
- **CarMakerDTO.cs**: Járműgyártó adatokat tartalmazó DTO.
- **CarModelDTO.cs**: Járműmodell adatokat tartalmazó DTO.
- **ChangePasswordAdminDTO.cs**: Adminisztrátori jelszó módosítását tartalmazó DTO.
- **ChangePasswordDTO.cs**: Felhasználói jelszó módosítását tartalmazó DTO.
- **ColorDTO.cs**: Színadatokat tartalmazó DTO.
- **CreateBodyTypeDTO.cs**: Új karosszériatípus létrehozását tartalmazó DTO.
- **CreateCarDTO.cs**: Új jármű létrehozását tartalmazó DTO.
- **CreateCarMakerDTO.cs**: Új járműgyártó létrehozását tartalmazó DTO.
- **CreateCarModelDTO.cs**: Új járműmodell létrehozását tartalmazó DTO.

- **CreateEngineDTO.cs**: Új motor létrehozását tartalmazó DTO.
- **CreateReservationDTO.cs**: Új foglalás létrehozását tartalmazó DTO.
- **DealerLoginResponseDTO.cs**: Kereskedői bejelentkezés válaszát tartalmazó DTO.
- **DealerUserDTO.cs**: Kereskedői felhasználói adatokat tartalmazó DTO.
- **DrivetrainTypeDTO.cs**: Hajtáslánc típusokat tartalmazó DTO.
- **EngineSizeModelDTO.cs**: Motor méretmodellek adatokat tartalmazó DTO.
- **ExtraDTO.cs**: Jármű extrák általános adatokat tartalmazó DTO.
- **ExtraTypeDTO.cs**: Extra típusokat tartalmazó DTO.
- **FuelTypeDTO.cs**: Üzemanyagtípus adatokat tartalmazó DTO.
- **ImageDTO.cs**: Képadatokat tartalmazó DTO.
- **ImageUploadDTO.cs**: Képfeltöltési adatokat tartalmazó DTO.
- **LocationDTO.cs**: Helyadatokat tartalmazó DTO.
- **MessageDTO.cs**: Üzenetadatokat tartalmazó DTO.
- **PrefectureDTO.cs**: Prefektúra adatokat tartalmazó DTO.
- **ReservationDTO.cs**: Foglalási adatokat tartalmazó DTO.
- **TransmissionTypeDTO.cs**: Sebességváltó típusokat tartalmazó DTO.
- **UpdateCarMakerDTO.cs**: Járműgyártó frissítését tartalmazó DTO.
- **UpdateCarModelDTO.cs**: Járműmodell frissítését tartalmazó DTO.
- **UpdateEngineDTO.cs**: Motor frissítését tartalmazó DTO.
- **UpdateReservationDTO.cs**: Foglalás frissítését tartalmazó DTO.
- **UserDTO.cs**: Felhasználói adatokat tartalmazó DTO.
- **UserLoginDTO.cs**: Felhasználói bejelentkezési adatokat tartalmazó DTO.
- **UserRegisterDTO.cs**: Felhasználói regisztrációs adatokat tartalmazó DTO.

5.2.8. Helpers

Ez a mappa segédosztályokat tartalmaz, amelyek az alkalmazás különböző aspektusait segítik:

- **AuthSettings.cs**: Az alkalmazás hitelesítési beállításait tartalmazó fájl.
- **SeedRoles.cs**: Az IdentityRole-hoz a szerepkörök feltőltése

5.2.9. Interfaces

A projekt interfészeit tartalmazó fájlok, amelyek a szolgáltatások közötti kommunikációt biztosítják:

- **ICarMakerService.cs**: Járműgyártó szolgáltatás interfésze.
- **ICarModelService.cs**: Járműmodell szolgáltatás interfésze.
- **IEngineSizeModelService.cs**: Motorméret-modellek kezelési interfésze.
- **IMessageService.cs**: Üzenetek kezelési interfésze.
- **IReservationService.cs**: Foglalások kezelési interfésze.
- **ISavedCarService.cs**: Mentett járművek kezelési interfésze.
- **IUserService.cs**: Felhasználói szolgáltatás interfésze.

5.2.10. Mappings

A `MappingProfile.cs` fájl tartalmazza az összes adatátviteli mappa (DTO) és az adatmodellek közötti leképezéseket.

5.2.11. Migrations

A migrációs fájlokat tartalmazó mappa, amelyek lehetővé teszik az adatbázis sémájának módosítását. Ez az EntityFramework-höz tartozik.

5.2.12. Models

A projekt adatmodelljei, amelyek az adatbázis entitásait képviselik:

- **Address.cs**: A cím adatmodellje, amely a cím adatait tárolja, beleértve a postal kódot, prefektúrát, várost és utcát.
  - **Id**: A cím egyedi azonosítója.
  - **PostalCode**: A címhez tartozó irányítószám. (Max 8 karakter hosszú)
  - **PrefectureId**: A címhez tartozó prefektúra azonosítója.
  - **Prefecture**: A címhez tartozó prefektúra (navigációs tulajdonság).
  - **City**: A város neve japánul.
  - **CityRomanized**: A város neve romanizált (latin betűs) formában.
  - **Street**: Az utca neve.
  - **StreetRomanized**: Az utca neve romanizált formában.
- **BodyType.cs**: A jármű karosszéria típusát tároló adatmodell. Ez első migrációnál automatikusan feltöltődik az adatbázisba.

  - **ID**: A karosszéria típus egyedi azonosítója. (Kötelező)
  - **NameJapanese**: A karosszéria típus neve japánul.
  - **NameEnglish**: A karosszéria típus neve angolul.

- **Car.cs**: Jármű adatmodellje.
  - **ID**: A jármű egyedi azonosítója (primary key).
  - **BrandID**: A jármű gyártójának azonosítója.
  - **ModelID**: A jármű modelljének azonosítója.
  - **Grade**: A jármű besorolása.
  - **BodyTypeID**: A jármű karosszéria típusának azonosítója.
  - **LocationID**: A jármű helyének azonosítója.
  - **EngineSizeID**: A jármű motor méretének azonosítója.
  - **LicensePlateNumber**: A jármű rendszáma.
  - **RepairHistory**: A jármű javítási előzményeinek megléte.
  - **FuelTypeID**: A jármű üzemanyag típusának azonosítója.
  - **DriveTrainID**: A jármű hajtásláncának azonosítója.
  - **MOTExpiry**: A jármű műszaki vizsgájának lejárata.
  - **TransmissionTypeID**: A jármű váltójának típusának azonosítója.
  - **VINNum**: A jármű alvázszáma (VIN).
  - **ColorID**: A jármű színének azonosítója.
  - **IsSmoking**: A jármű dohányzástól mentes-e.
  - **Mileage**: A jármű futásteljesítménye.
  - **IsInTransfer**: A jármű telephely közötti mozgatásban van-e.
  - **Price**: A jármű ára.
  - **Navigation Properties**:
    * **Brand**: A jármű gyártója (CarMaker).
    * **CarModel**: A jármű modellje (CarModel).
    * **BodyType**: A jármű karosszéria típusa (BodyType).
    * **Location**: A jármű helye (Location).
    * **EngineSize**: A jármű motorjának mérete (EngineSizeModel).
    * **FuelType**: A jármű üzemanyagtípusa (FuelType).
    * **DriveTrain**: A jármű hajtáslánca (DrivetrainType).
    * **TransmissionType**: A jármű váltó típusa (TransmissionType).
    * **Color**: A jármű színe (Color).
    * **CarExtras**: A jármű extra felszereltségei (CarExtra).
    * **Images**: A járműhöz tartozó képek (Image).

  - **DateOfManufacture**: A jármű gyártási dátuma.

- **CarExtra.cs**: Jármű extrák összekapcsolását biztosító adatmodell. Jelenleg nincs implementálva.

- **CarMaker.cs**: A járműgyártót tároló adatmodell.

  - **ID**: A járműgyártó egyedi azonosítója.
  - **BrandEnglish**: A járműgyártó neve angolul.
  - **BrandJapanese**: A járműgyártó neve japánul.

  - **ID**: A gyártó azonosítója.
  - **BrandEnglish**: A gyártó neve angol nyelven.
  - **BrandJapanese**: A gyártó neve japán nyelven.

- **Color.cs**: A színadatokat tároló adatmodell.

  - **ID**: A szín egyedi azonosítója (primary key).
  - **ColorNameEnglish**: A szín neve angolul.
  - **ColorNameJapanese**: A szín neve japánul.
  - **Cars**: Kapcsolódó járművek gyűjteménye (Navigation property), amelyek a színhez tartoznak.

- **EmployeeLocation.cs**: Alkalmazott helyadatmodellje.

  - **Id**: Az alkalmazott helyének egyedi azonosítója (primary key).
  - **EmployeeId**: Az alkalmazott azonosítója (GUID, a felhasználói azonosító, amely a string típusú IdentityUser-tól származik).
  - **LocationId**: Az alkalmazott hely azonosítója (külső kulcs a Location táblára).
  - **Navigation Property**:
    * **Location**: Az alkalmazott helye (Location). A hely a **LocationId** segítségével van kapcsolva.

- **EngineSizeModel.cs**: Motor méret modell adatmodellje.

  - **ID**: A motor méret modell egyedi azonosítója (primary key).
  - **ModelID**: A kapcsolódó járműmodell azonosítója (külső kulcs a CarModel táblára).
  - **EngineSize**: A motor mérete (300 és 10000 között, érvénytelen méret esetén hibát jelez).
  - **Navigation Properties**:
    * **FuelType**: A motor üzemanyag típusa.

        * **CarModel**: A járműmodell, amelyhez a motor méret tartozik (a **ModelID** segítségével kapcsolódik). Az **JsonIgnore** attribútum miatt nem kerül bele a JSON-ba.

- **Extra.cs**: Az extra típusokat és azok nevét tároló adatmodell, amely kapcsolódik az **ExtraType** táblához.
- **FuelType.cs**: Járművek üzemanyag típusait tároló adatmodell, mely előre be van töltve az adatbázisba.
  - **ID**: Az üzemanyag típus egyedi azonosítója (primary key).
  - **NameJapanese**: Az üzemanyag típus neve japánul.
  - **NameEnglish**: Az üzemanyag típus neve angolul.
- **Grade.cs**: A járművek osztályozására szolgáló enumeráció.
  - **A**: Legjobb minőség.
  - **B**: Kiváló minőség.
  - **C**: Jó minőség.
  - **D**: Megfelelő minőség.
  - **E**: Elégtelen minőség.
- **Image.cs**: Kép adatmodell, amely a járműhöz tartozó képeket tárolja.
  - **ID**: A kép egyedi azonosítója (primary key).
  - **CarID**: A kapcsolódó jármű egyedi azonosítója (külső kulcs).
  - **URL**: A kép elérhetősége (URL).
  - **Car**: A kapcsolódó jármű.
- **Location.cs**: A helyszín adatmodell, amely tartalmazza a helyszín nevét, címét és egyéb részleteit.
  - **ID**: A helyszín egyedi azonosítója (primary key).
  - **LocationName**: A helyszín neve.
  - **AddressId**: A kapcsolódó cím egyedi azonosítója (külső kulcs).
  - **Address**: A helyszín címét tároló adatmodell.
  - **MaxCapacity**: A helyszín maximális kapacitása.
  - **PhoneNumber**: A helyszín telefonszáma.
- **Message.cs**: Üzenet adatmodell, amely az üzenetek tartalmát, dátumát és címzettjét tartalmazza.
  - **Id**: Az üzenet egyedi azonosítója.
  - **Content**: Az üzenet tartalma.

  - **Date**: Az üzenet dátuma.
  - **Recipient**: A címzett GUID azonosítója.

- **Prefecture.cs**: Prefektúra adatmodell, amely Japán 47 prefektúráját tartalmazza.
  - **Id**: A prefektúra egyedi azonosítója (primary key).
  - **Name**: A prefektúra neve.
  - **NameJP**: A prefektúra neve japánul.

- **Reservation.cs**: Foglalás adatmodell, amely tartalmazza a járműhöz kapcsolódó foglalásokat.
  - **Id**: A foglalás egyedi azonosítója.
  - **UserId**: A felhasználó azonosítója, aki a foglalást tette.
  - **CarId**: A foglalt jármű azonosítója.
  - **Date**: A foglalás dátuma.

- **SavedCar.cs**: Jármű mentési adatmodell, amely tárolja a felhasználók által mentett járműveket.
  - **Id**: A mentett jármű egyedi azonosítója.
  - **UserId**: A felhasználó azonosítója, aki a járművet mentette.
  - **CarId**: A mentett jármű azonosítója.

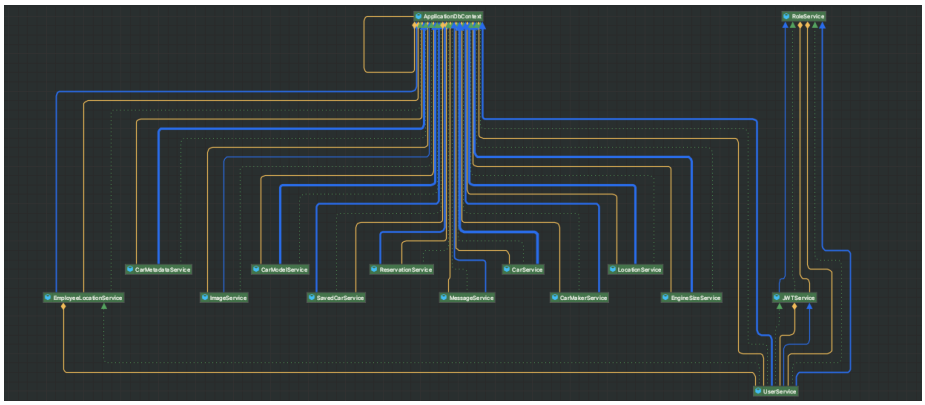
- **TransmissionType.cs**: A járművek sebességváltó típusait tároló adatmodell.
  - **ID**: A sebességváltó típus egyedi azonosítója.
  - **Type**: A sebességváltó típusa (pl. kézi, automatikus).

- **User.cs**: Felhasználó adatmodell, amely az IdentityUser kiterjesztéseként tárolja a felhasználói adatokat.
  - **Name**: A felhasználó neve.
  - **NameKanji**: A felhasználó neve kanjival (ha van).
  - **PreferredLanguage**: A felhasználó preferált nyelve.
  - **RefreshToken**: A felhasználó frissítő tokenje.
  - **RefreshTokenExpiry**: A frissítő token lejárati ideje.

5.2.13. Services

Szolgáltatások, amelyek az alkalmazás üzleti logikáját tartalmazzák:

- **CarMakerService.cs**: Járműgyártók kezelésére szolgáló osztály, amely lehetővé teszi a gyártók létrehozását, lekérését, frissítését és törlését. Az AutoMapper könyvtár segítségével biztosítja az adatok átvitelét a modellek és DTO-k között.
- **CarMakerService.cs**: Járműgyártókkal kapcsolatos üzleti logika.
- **UserService.cs**: Felhasználókkal kapcsolatos üzleti logika.
- **ReservationService.cs**: Foglalásokkal kapcsolatos üzleti logika.



5.2. ábra. A diagram a szolgáltatások közötti osztályfüggőségek ábrázolásáról.

5.3. Osztályok

5.3.1. CarMakerService osztály

Az `CarMakerService` osztály felelős a járműgyártók kezeléséért. Ez az osztály az alábbi funkciókat biztosítja:

- **CreateNewMakerAsync**: Új autógyártó létrehozása, ha még nem létezik. Az adatok a DTO (Data Transfer Object) és az entitás között történő átvitelét az AutoMapper könyvtár biztosítja.
- **GetMakersAsync**: Az összes autógyártó lekérése.
- **GetMakerByIdAsync**: Az autógyártó lekérése ID alapján.
- **UpdateMakerByIdAsync**: Egy meglévő autógyártó frissítése ID alapján.

- **DeleteMakerByIdAsync**: Autógyártó törlése ID alapján.

Az osztály metódusai alább találhatók C# kóddal és kommentekkel:

5.3.2. CarMakerService implementációja

CreateNewMakerAsync metódus

```
1      /// <summary>
2      /// Creates a new car maker in the database if it doesn'
           t already exist.
3      /// If the maker already exists, returns null.
4      /// </summary>
5      /// <param name="createCarMakerDto">The DTO containing
           car maker data to create.</param>
6      /// <returns>A CarMakerDTO if creation was successful,
           or null if the maker already exists.</returns>
7      public async Task<CarMakerDTO?> CreateNewMakerAsync(
           CreateCarMakerDTO createCarMakerDto)
```

GetMakersAsync metódus

```
1      /// <summary>
2      /// Retrieves all car makers from the database.
3      /// </summary>
4      /// <returns>A list of CarMakerDTOs representing all car
            makers.</returns>
5      public async Task<List<CarMakerDTO>> GetMakersAsync()
```

GetMakerByIdAsync metódus

```
1      /// <summary>
2      /// Retrieves a specific car maker by its ID.
3      /// </summary>
4      /// <param name="id">The ID of the car maker to retrieve
           .</param>
5      /// <returns>A CarMakerDTO representing the car maker,
           or null if not found.</returns>
6      public async Task<CarMakerDTO?> GetMakerByIdAsync(int id
           )
```

UpdateMakerByIdAsync metódus

```
1      /// <summary>
2      /// Updates an existing car maker's data in the database
            by its ID.
```

```
3       /// </summary>
4       /// <param name="updateCarMakerDto">The DTO containing
            the updated car maker data.</param>
5       /// <returns>The updated CarMakerDTO, or null if the
            maker was not found.</returns>
6       public async Task<CarMakerDTO?> UpdateMakerByIdAsync(
            UpdateCarMakerDTO updateCarMakerDto)
```

DeleteMakerByIdAsync metódus

```
1       /// <summary>
2       /// Deletes a car maker by its ID from the database.
3       /// </summary>
4       /// <param name="id">The ID of the car maker to delete
            .</param>
5       /// <returns>True if deletion was successful, false if
            the maker was not found.</returns>
6       public async Task<bool> DeleteMakerByIdAsync(int id)
```

Összegzés

A `CarMakerService` osztály egy átfogó megoldást kínál az autógyártók kezelésére, lehetővé téve a felhasználók számára, hogy könnyedén létrehozzanak, lekérdezzenek, frissítsenek és töröljenek autógyártókat. Az AutoMapper könyvtár használatával az adatok átvitele a DTO-k és az entitások között hatékonyan történik, biztosítva a tiszta és karbantartható kódot.

5.3.3. CarMetadataService osztály

Az `CarMetadataService` osztály felelős a járművek különböző metaadatai, mint például a karosszéria típusok, sebességváltó típusok, üzemanyag típusok, hajtáslánc típusok és színek kezeléséért. Az osztály az alábbi funkciókat biztosítja:

- **CarMetadataService.cs**: Járműmetaadatok kezelésére szolgáló osztály, amely lehetővé teszi a karosszéria típusok, sebességváltó típusok, üzemanyag típusok, hajtáslánc típusok és színek létrehozását, lekérését.
- **CreateCarBodyTypeAsync**: Új karosszéria típus létrehozása, ha még nem létezik. Az adatok a DTO (Data Transfer Object) és az entitás között történő átvitelét az AutoMapper könyvtár biztosítja.
- **GetBodyTypesAsync**: Az összes karosszéria típus lekérése.
- **GetBodyTypeByIdAsync**: A karosszéria típusok lekérése ID alapján.
- **GetTransmissionTypesAsync**: Az összes sebességváltó típus lekérése.

- **GetTransmissionTypeByIdAsync**: Sebességváltó típus lekérése ID alapján.
- **GetFuelTypesAsync**: Az összes üzemanyag típus lekérése.
- **GetFuelTypeByIdAsync**: Üzemanyag típus lekérése ID alapján.
- **GetDrivetrainTypesAsync**: Az összes hajtáslánc típus lekérése.
- **GetDrivetrainTypeByIdAsync**: Hajtáslánc típus lekérése ID alapján.
- **GetColorTypesAsync**: Az összes szín típus lekérése.
- **GetColorByIdAsync**: Szín típus lekérése ID alapján.

5.3.4. CarMakerService implementációja

CreateCarBodyTypeAsync Metódus

```
1      /// <summary>
2      /// Creates a new car body type if it does not already
           exist.
3      /// </summary>
4      /// <param name="bodyTypeDto">The DTO containing body
           type details.</param>
5      /// <returns>A tuple containing the body type entity and
            a boolean indicating if it was newly created.</
           returns>
6      public async Task<(BodyType bodyType, bool isNew)>
           CreateCarBodyTypeAsync(CreateBodyTypeDTO bodyTypeDto)
```

GetBodyTypesAsync Metódus

```
1      /// <summary>
2      /// Retrieves a list of all body types.
3      /// </summary>
4      /// <returns>A list of body type DTOs.</returns>
5      public async Task<List<BodyTypeDTO>> GetBodyTypesAsync()
```

GetBodyTypeByIdAsync Metódus

```
1      /// <summary>
2      /// Retrieves a body type by its ID.
3      /// </summary>
4      /// <param name="id">The ID of the body type.</param>
5      /// <returns>The body type DTO if found, otherwise null
           .</returns>
6      public async Task<BodyTypeDTO?> GetBodyTypeByIdAsync(int
            id)
```

GetTransmissionTypesAsync Metódus

```
1       /// <summary>
2       /// Retrieves a list of all transmission types.
3       /// </summary>
4       /// <returns>A list of transmission type DTOs.</returns>
5       public async Task<List<TransmissionTypeDTO>>
            GetTransmissionTypesAsync()
```

GetTransmissionTypeByIdAsync Metódus

```
1       /// <summary>
2       /// Retrieves a transmission type by its ID.
3       /// </summary>
4       /// <param name="id">The ID of the transmission type.</
            param>
5       /// <returns>The transmission type DTO if found,
            otherwise null.</returns>
6       public async Task<BodyTypeDTO?>
            GetTransmissionTypeByIdAsync(int id)
```

GetFuelTypesAsync Metódus

```
1       /// <summary>
2       /// Retrieves a list of all fuel types.
3       /// </summary>
4       /// <returns>A list of fuel type DTOs.</returns>
5       public async Task<List<FuelTypeDTO>> GetFuelTypesAsync()
```

GetFuelTypeByIdAsync Metódus

```
1       /// <summary>
2       /// Retrieves a fuel type by its ID.
3       /// </summary>
4       /// <param name="id">The ID of the fuel type.</param>
5       /// <returns>The fuel type DTO if found, otherwise null
            .</returns>
6       public async Task<FuelTypeDTO?> GetFuelTypeByIdAsync(int
             id)
```

GetDrivetrainTypesAsync Metódus

```
1       /// <summary>
2       /// Retrieves a list of all drivetrain types.
3       /// </summary>
```

```
4       /// <returns>A list of drivetrain type DTOs.</returns>
5       public async Task<List<DrivetrainTypeDTO>>
            GetDrivetrainTypesAsync()
```

GetDrivetrainTypeByIdAsync Metódus

```
1       /// <summary>
2       /// Retrieves a drivetrain type by its ID.
3       /// </summary>
4       /// <param name="id">The ID of the drivetrain type.</
            param>
5       /// <returns>The drivetrain type DTO if found, otherwise
             null.</returns>
6       public async Task<DrivetrainTypeDTO?>
            GetDrivetrainTypeByIdAsync(int id)
```

GetColorTypesAsync Metódus

```
1       /// <summary>
2       /// Retrieves a list of all color types.
3       /// </summary>
4       /// <returns>A list of color type DTOs.</returns>
5       public async Task<List<ColorDTO>> GetColorTypesAsync()
```

GetColorByIdAsync Metódus

```
1       /// <summary>
2       /// Retrieves a color by its ID.
3       /// </summary>
4       /// <param name="id">The ID of the color.</param>
5       /// <returns>The color DTO if found, otherwise null.</
            returns>
6       public async Task<ColorDTO?> GetColorByIdAsync(int id)
```

Összegzés

A `CarService` osztály egy átfogó megoldást kínál a járművek kezelésére, lehetővé téve a felhasználók számára, hogy könnyedén létrehozzanak, lekérdezzenek, frissítsenek és töröljenek járműveket. Az AutoMapper könyvtár használatával az adatok átvitele a DTO-k és az entitások között hatékonyan történik, biztosítva a tiszta és karbantartható kódot.

5.3.5. CarModelService osztály

Az `CarModelService` osztály felelős a járműmodellek kezeléséért, beleértve azok létrehozását, lekérését, frissítését és törlését. Az osztály az alábbi funkciókat biztosítja:

- **CarModelService.cs**: Járműmodellek kezelésére szolgáló osztály, amely lehetővé teszi a járműmodellek létrehozását, lekérését, frissítését és törlését.
- **CreateCarModelAsync**: Új járműmodell létrehozása. Az adatok a DTO (Data Transfer Object) és az entitás között történő átvitelét az AutoMapper könyvtár biztosítja.
- **GetCarModelAsync**: Járműmodell lekérése ID alapján.
- **GetCarModelsFilteredAsync**: Járműmodellek lekérése opcionális paraméterek alapján.
- **GetAllCarModelsAsync**: Az összes járműmodell lekérése.
- **UpdateCarModelAsync**: Meglévő járműmodell frissítése.
- **DeleteCarModelAsync**: Járműmodell törlése ID alapján.
- **GetCarsByMakerIdAsync**: Járműmodellek lekérése a gyártó ID-ja alapján.

CreateCarModelAsync Metódus

```
1      /// <summary>
2      /// Creates a new car model.
3      /// </summary>
4      /// <param name="carModelDTO">The car model data
           transfer object.</param>
5      /// <returns>The created CarModelDTO.</returns>
6      public async Task<CarModelDTO> CreateCarModelAsync(
           CreateCarModelDTO carModelDTO)
```

GetCarModelAsync Metódus

```
1      /// <summary>
2      /// Retrieves a car model by its ID.
3      /// </summary>
4      /// <param name="id">The car model ID.</param>
5      /// <returns>The CarModelDTO if found, otherwise null.</
           returns>
6      public async Task<CarModelDTO?> GetCarModelAsync(int id)
```

GetCarModelsFilteredAsync Metódus

```
1       /// <summary>
2       /// Retrieves car models filtered by optional parameters
            .
3       /// </summary>
4       /// <param name="makerID">The maker ID to filter by.</
            param>
5       /// <param name="startYear">The manufacturing start year
             to filter by.</param>
6       /// <param name="endYear">The manufacturing end year to
            filter by.</param>
7       /// <param name="passengerCount">The passenger count to
            filter by.</param>
8       /// <returns>A list of filtered CarModelDTOs.</returns>
9       public async Task<List<CarModelDTO>>
            GetCarModelsFilteredAsync(int? makerID = null, int?
            startYear = null, int? endYear = null,
10          int? passengerCount = null)
```

GetAllCarModelsAsync Metódus

```
1       /// <summary>
2       /// Retrieves all car models.
3       /// </summary>
4       /// <returns>A list of CarModelDTOs.</returns>
5       public async Task<List<CarModelDTO>>
            GetAllCarModelsAsync()
```

UpdateCarModelAsync Metódus

```
1       /// <summary>
2       /// Updates an existing car model.
3       /// </summary>
4       /// <param name="id">The ID of the car model to update
            .</param>
5       /// <param name="updatedCarModel">The updated car model
            data.</param>
6       /// <returns>The updated CarModelDTO, or null if not
            found.</returns>
7       public async Task<CarModelDTO?> UpdateCarModelAsync(int
            id, UpdateCarModelDTO updatedCarModel)
```

DeleteCarModelAsync Metódus

```
1       /// <summary>
2       /// Deletes a car model by ID.
3       /// </summary>
4       /// <param name="id">The ID of the car model to delete
            .</param>
5       /// <returns>True if deleted, false if not found.</
            returns>
6       public async Task<bool> DeleteCarModelAsync(int id)
```

GetCarsByMakerIdAsync Metódus

```
1       /// <summary>
2       /// Retrieves all car models for a given maker ID.
3       /// </summary>
4       /// <param name="makerId">The maker ID.</param>
5       /// <returns>A list of CarModelDTOs.</returns>
6       public async Task<List<CarModelDTO>>
            GetCarsByMakerIdAsync(int makerId)
```

Összegzés

A `CarModelService` osztály egy átfogó megoldást kínál a járműmodellek kezelésére, lehetővé téve a felhasználók számára, hogy könnyedén létrehozzanak, lekérdezzenek, frissítsenek és töröljenek járműmodelleket. Az AutoMapper könyvtár használatával az adatok átvitele a DTO-k és az entitások között hatékonyan történik, biztosítva a tiszta és karbantartható kódot.

5.3.6. CarService osztály

Az `CarService` osztály felelős a járművekkel kapcsolatos műveletek kezeléséért, beleértve a járművek lekérését, hozzáadását, frissítését és törlését. Az osztály az alábbi funkciókat biztosítja:

- **CarService.cs**: Járművekkel kapcsolatos műveletek kezelésére szolgáló osztály, amely lehetővé teszi a járművek létrehozását, lekérését, frissítését és törlését.
- **GetAllCarsAsync**: Az összes jármű lekérése az adatbázisból.
- **GetCarByIdAsync**: Egy adott jármű lekérése ID alapján.
- **AddCarAsync**: Új jármű hozzáadása az adatbázishoz.
- **DeleteCarAsync**: Jármű törlése ID alapján.
- **EditCarAsync**: Meglévő járműadatok frissítése.

GetAllCarsAsync Metódus

```
1       /// <summary>
2       /// Retrieves all cars from the database.
3       /// </summary>
4       /// <returns>A list of CarDTOs.</returns>
5       public async Task<List<CarDTO>> GetAllCarsAsync()
```

GetCarByIdAsync Metódus

```
1       /// <summary>
2       /// Retrieves a specific car by ID.
3       /// </summary>
4       /// <param name="id">The car ID.</param>
5       /// <returns>The CarDTO if found, otherwise null.</
            returns>
6       public async Task<CarDTO?> GetCarByIdAsync(int id)
```

AddCarAsync Metódus

```
1       /// <summary>
2       /// Adds a new car to the database.
3       /// </summary>
4       /// <param name="createCarDto">The car data to be added
            .</param>
5       /// <returns>The created CarDTO.</returns>
6       public async Task<CarDTO> AddCarAsync(CreateCarDTO
            createCarDto)
```

DeleteCarAsync Metódus

```
1       /// <summary>
2       /// Deletes a car from the database by ID.
3       /// </summary>
4       /// <param name="id">The ID of the car to delete.</param
            >
5       /// <returns>True if deleted, false if not found.</
            returns>
6       public async Task<bool> DeleteCarAsync(int id)
```

EditCarAsync Metódus

```
1       /// <summary>
2       /// Updates an existing car record.
3       /// </summary>
```

```
4       /// <param name="id">The ID of the car to update.</param
            >
5       /// <param name="editCarDto">The updated car data.</
            param>
6       /// <returns>The updated CarDTO, or null if not found.</
            returns>
7       public async Task<CarDTO?> EditCarAsync(int id,
            CreateCarDTO editCarDto)
```

Összegzés

A `CarService` osztály egy átfogó megoldást kínál a járművek kezelésére, lehetővé téve a felhasználók számára, hogy könnyedén létrehozzanak, lekérdezzenek, frissítsenek és töröljenek járműveket. Az AutoMapper könyvtár használatával az adatok átvitele a DTO-k és az entitások között hatékonyan történik, biztosítva a tiszta és karbantartható kódot.

5.3.7. EmployeeLocationService osztály

Az `EmployeeLocationService` osztály felelős az alkalmazottak helyszíneinek kezeléséért. Az osztály az alábbi funkciókat biztosítja:

- **GetAllEmployeeLocationsAsync**: Az összes alkalmazott helyszínének lekérése az adatbázisból.
- **GetEmployeeLocationByEmployeeIdAsync**: Az alkalmazott helyszínének lekérése alkalmazott ID alapján.
- **AddEmployeeLocationAsync**: Új alkalmazott helyszín hozzáadása az adatbázishoz.

5.3.8. EmployeeLocationService implementációja

GetAllEmployeeLocationsAsync Metódus

```
1       /// <summary>
2       /// Retrieves all employee locations from the database.
3       /// </summary>
4       /// <returns>A list of employee locations.</returns>
5       public async Task<List<EmployeeLocation>>
            GetAllEmployeeLocationsAsync()
```

GetEmployeeLocationByEmployeeIdAsync Metódus

```
1       /// <summary>
2       /// Retrieves an employee location by employee ID.
3       /// </summary>
4       /// <param name="employeeId">The employee ID.</param>
5       /// <returns>The employee location if found, otherwise
            null.</returns>
6       public async Task<EmployeeLocation?>
            GetEmployeeLocationByEmployeeIdAsync(Guid employeeId)
```

AddEmployeeLocationAsync Metódus

```
1       /// <summary>
2       /// Adds a new employee location to the database.
3       /// </summary>
4       /// <param name="employeeLocation">The employee location
             data to be added.</param>
5       /// <returns>True if the operation was successful,
            otherwise false.</returns>
6       public async Task<bool> AddEmployeeLocationAsync(
            EmployeeLocation employeeLocation)
```

Összegzés

A `EmployeeLocationService` osztály egy átfogó megoldást kínál az alkalmazottak helyszíneinek kezelésére, lehetővé téve a felhasználók számára, hogy könnyedén lekérdezzék az alkalmazottak helyszíneit és új helyszíneket adjanak hozzá az adatbázishoz. Az aszinkron műveletek biztosítják, hogy az adatbázis lekérdezések és módosítások hatékonyan és gyorsan történjenek.

5.3.9. EngineSizeService osztály

Az `EngineSizeService` osztály felelős a motor méretek kezeléséért. Az osztály az alábbi funkciókat biztosítja:

- **AddEngineSizeAsync**: Új motor méret hozzáadása az adatbázishoz.
- **GetEnginesAsync**: Az összes motor méret lekérése.
- **GetEngineByModelIdAsync**: Motor méretek lekérése autó modell ID alapján.
- **UpdateEngineAsync**: Egy meglévő motor méret frissítése.
- **DeleteEnginesAsync**: Motor méret törlése az adatbázisból.

5.3.10. EngineSizeService implementációja

AddEngineSizeAsync Metódus

```
1       /// <summary>
2       /// Adds a new engine size entry to the database.
3       /// </summary>
4       /// <param name="modelId">The ID of the car model.</
            param>
5       /// <param name="engineSize">The engine size in cubic
            centimeters.</param>
6       /// <param name="fuelTypeId">The ID of the fuel type.</
            param>
7       /// <returns>The added engine size as a DTO.</returns>
8       /// <exception cref="KeyNotFoundException">Thrown if the
             car model is not found.</exception>
9       public async Task<EngineSizeModelDTO> AddEngineSizeAsync
            (int modelId, int engineSize, int fuelTypeId)
```

GetEnginesAsync Metódus

```
1       /// <summary>
2       /// Retrieves all engine sizes from the database.
3       /// </summary>
4       /// <returns>A list of engine size DTOs.</returns>
5       public async Task<List<EngineSizeModelDTO>>
            GetEnginesAsync()
```

GetEngineByModelIdAsync Metódus

```
1       /// <summary>
2       /// Retrieves engine sizes by car model ID.
3       /// </summary>
4       /// <param name="modelId">The car model ID.</param>
5       /// <returns>A collection of engine size DTOs, or null
            if none are found.</returns>
6       public async Task<IEnumerable<EngineSizeModelDTO>?>
            GetEngineByModelIdAsync(int modelId)
```

UpdateEngineAsync Metódus

```
1       /// <summary>
2       /// Updates an existing engine size entry.
3       /// </summary>
4       /// <param name="engineId">The ID of the engine size
            entry.</param>
```

```
5       /// <param name="newEngineSize">The updated engine size
            .</param>
6       /// <param name="fuelTypeId">The new fuel type ID.</
            param>
7       /// <returns>The updated engine size DTO if successful,
            otherwise null.</returns>
8       public async Task<EngineSizeModelDTO?> UpdateEngineAsync
            (int engineId, int newEngineSize, int fuelTypeId)
```

DeleteEnginesAsync Metódus

```
1       /// <summary>
2       /// Deletes an engine size entry from the database.
3       /// </summary>
4       /// <param name="engineId">The ID of the engine size
            entry to delete.</param>
5       /// <returns>True if deletion was successful, otherwise
            false.</returns>
6       public async Task<bool> DeleteEnginesAsync(int engineId)
```

Összegzés

A `EngineSizeService` osztály egy átfogó megoldást kínál a motor méretek kezelésére, lehetővé téve a felhasználók számára, hogy könnyedén hozzáadják, lekérdezzék, frissítsék és töröljék a motor méreteket az adatbázisból. Az AutoMapper könyvtár segítségével az adatok átvitele az entitások és DTO-k között hatékonyan és karbantartható módon történik. Az aszinkron műveletek biztosítják a magas teljesítményt és a hatékony adatkezelést.

5.3.11. ImageService osztály

Az `ImageService` osztály felelős a képek feltöltéséért és lekéréséért a járművekhez. Az osztály az alábbi funkciókat biztosítja:

- **UploadImageAsync**: Kép feltöltése egy adott járműhöz.
- **GetImagesForCarAsync**: Az adott járműhöz tartozó összes kép URL-jének lekérése.

5.3.12. ImageService implementációja

UploadImageAsync Metódus

```
1       /// <summary>
2       /// Uploads an image for a specific car.
3       /// </summary>
```

```
4       /// <param name="imageFile">The image file to upload.</
            param>
5       /// <param name="carId">The ID of the car associated
            with the image.</param>
6       /// <returns>The URL of the uploaded image.</returns>
7       /// <exception cref="ArgumentException">Thrown when the
            provided file is invalid.</exception>
8       public async Task<string> UploadImageAsync(IFormFile
            imageFile, int carId)
```

GetImagesForCarAsync Metódus

```
1       /// <summary>
2       /// Retrieves all image URLs associated with a specific
            car.
3       /// </summary>
4       /// <param name="carId">The ID of the car.</param>
5       /// <returns>A list of image URLs.</returns>
6       public async Task<List<string>> GetImagesForCarAsync(int
             carId)
```

Összegzés

A `ImageService` osztály lehetővé teszi a járművekhez tartozó képek kezelését. Az osztály biztosítja a képek feltöltését az adott járműhöz, a fájlok a szerveren a `wwwroot/uploads/carId` könyvtárban kerülnek tárolásra. Ha a kép már létezik, az URL visszaadásra kerül, anélkül hogy újra feltöltené. Az adatbázisban tárolja a kép URL-jét is. Az osztály emellett képes lekérni az adott járműhöz tartozó összes kép URL-jét.

5.3.13. JWTService osztály

A `JWTService` osztály felelős a JWT alapú hitelesítés kezeléséért. Az osztály biztosítja a felhasználói tokent generáló, a frissítő tokent generáló, és azok kezelését.

- **GenerateTokens**: Generálja a hozzáférési és frissítő tokeneket egy felhasználó számára.
- **GenerateRefreshToken**: Biztonságos frissítő tokent generál.
- **RefreshAccessToken**: Frissíti a hozzáférési tokent egy érvényes frissítő token segítségével.
- **RevokeRefreshToken**: Érvényteleníti a frissítő tokent.
- **GenerateClaims**: Generálja a felhasználói azonosítókat és jogosultságokat.

5.3.14. JWTService implementációja

GenerateTokens Metódus

```
1       /// <summary>
2       /// Generates access and refresh tokens for a user.
3       /// </summary>
4       /// <param name="user">The user for whom the tokens are
            generated.</param>
5       /// <returns>A tuple containing the access token and
            refresh token.</returns>
6       public async Task<(string AccessToken, string
            RefreshToken)> GenerateTokens(User user)
```

GenerateRefreshToken Metódus

```
1       /// <summary>
2       /// Generates a secure refresh token.
3       /// </summary>
4       /// <returns>A base64-encoded refresh token.</returns>
5       private string GenerateRefreshToken()
```

RefreshAccessToken Metódus

```
1       /// <summary>
2       /// Refreshes the access token using a valid refresh
            token.
3       /// </summary>
4       /// <param name="refreshToken">The refresh token.</param
            >
5       /// <returns>A tuple indicating whether the operation
            was successful and the new access token.</returns>
6       public async Task<(bool IsValid, string AccessToken)>
            RefreshAccessToken(string refreshToken)
```

RevokeRefreshToken Metódus

```
1       /// <summary>
2       /// Revokes a refresh token, making it invalid.
3       /// </summary>
4       /// <param name="refreshToken">The refresh token to
            revoke.</param>
5       /// <returns>A boolean indicating whether the operation
            was successful.</returns>
6       public async Task<bool> RevokeRefreshToken(string
            refreshToken)
```

GenerateClaims Metódus

```
1       /// <summary>
2       /// Generates claims for a user.
3       /// </summary>
4       /// <param name="user">The user whose claims are
            generated.</param>
5       /// <returns>A ClaimsIdentity containing user claims.</
            returns>
6       private async Task<ClaimsIdentity> GenerateClaims(User
            user)
```

Összegzés

A `JWTService` osztály lehetővé teszi a felhasználók számára, hogy biztonságosan kezeljék a hozzáférési és frissítő tokeneket, amelyek szükségesek a felhasználók hitelesítéséhez és az API-hoz való hozzáféréshez. A frissítő tokenek 30 napig érvényesek, míg a hozzáférési tokenek 15 percig maradnak érvényben. Az osztály kezeli a frissítő tokenek visszavonását is, és biztosítja, hogy a felhasználói jogosultságok a megfelelő módon legyenek érvényesítve a JWT-k segítségével.

5.3.15. MessageService osztály

A `MessageService` osztály felelős az üzenetek kezeléséért a kereskedelmi rendszerben. Az osztály lehetővé teszi új üzenetek létrehozását, meglévő üzenetek lekérését és törlését.

- **CreateMessageAsync**: Létrehozza az új üzenetet egy adott címzett számára.
- **GetMessagesByUserAsync**: Lekéri a felhasználó összes üzenetét.
- **DeleteMessageAsync**: Törli a megadott üzenetet.
- **GetUserIdFromJwt**: Kinyeri a felhasználó azonosítóját a JWT-ből.

5.3.16. MessageService implementációja

CreateMessageAsync Metódus

```
1       /// <summary>
2       /// Creates a new message asynchronously.
3       /// </summary>
4       /// <param name="content">The content of the message.</
            param>
5       /// <param name="recipient">The recipient's ID.</param>
6       public async Task CreateMessageAsync(string content,
            Guid recipient)
```

GetMessagesByUserAsync Metódus

```
1       /// <summary>
2       /// Gets messages by user ID asynchronously.
3       /// </summary>
4       /// <param name="userId">The user's ID.</param>
5       /// <returns>A list of messages for the specified user
            .</returns>
6       /// <exception cref="UnauthorizedAccessException">Thrown
             if the current user is not authorized to view the
            messages.</exception>
7       public async Task<List<Message>> GetMessagesByUserAsync(
            Guid userId)
```

GetUserIdFromJwt Metódus

```
1       /// <summary>
2       /// Gets the user ID from the JWT.
3       /// </summary>
4       /// <returns>The user ID as a Guid.</returns>
5       /// <exception cref="UnauthorizedAccessException">Thrown
             if the user ID is not found in the JWT.</exception>
6       private Guid GetUserIdFromJwt()
```

DeleteMessageAsync Metódus

```
1       /// <summary>
2       /// Deletes a message asynchronously.
3       /// </summary>
4       /// <param name="messageId">The ID of the message to
            delete.</param>
5       /// <exception cref="KeyNotFoundException">Thrown if the
             message is not found.</exception>
6       /// <exception cref="UnauthorizedAccessException">Thrown
             if the current user is not authorized to delete the
            message.</exception>
7       public async Task DeleteMessageAsync(int messageId)
```

Összegzés

A `MessageService` osztály biztosítja az üzenetek kezelését a rendszerben. A felhasználók számára lehetőséget biztosít új üzenetek létrehozására, meglévő üzenetek lekérdezésére és törlésére. Az osztály a JWT-t használja a felhasználói azonosításra, és biztosítja, hogy csak a jogosult felhasználók férhessenek hozzá és törölhessenek üzeneteket. Az üzenetek törléséhez a felhasználónak először azonosítania kell magát, és biztosítani kell, hogy az üzenet neki legyen címzett.

5.4. Bevezetés

Ez a dokumentáció a `LocationServiceTests` osztály egységtesztjeit tartalmazza, amely a `DealershipSystem` rendszerben a helyszínekkel kapcsolatos funkciók teszteléséért felel. A tesztek az `xUnit`, `Moq` és `AutoMapper` keretrendszereket használják.

5.5. Tesztkörnyezet

5.5.1. Használt technológiák

- xUnit - Egységteszt keretrendszer
- Moq - Mocking könyvtár a függőségek izolálására
- AutoMapper - Objektumtérkép használata a modellek és DTO-k között
- Entity Framework Core - InMemory adatbázis az izolált teszteléshez

5.5.2. Tesztelési infrastruktúra

A tesztek egy `InMemoryDatabase`-t használnak az adatbázis-kezelés szimulálására, amely lehetővé teszi az adatok izolált módon történő kezelését minden tesztesetnél.

5.6. Tesztesetek

5.6.1. CreateLocationAsync - Helyszín létrehozása (prefektúra nem található)

Leírás: A teszt egy helyszín létrehozását szimulálja egy nem létező prefektúrával, és ellenőrzi, hogy a létrehozás eredménye `null`.

Várható kimenet: A prefektúra nem találása esetén a helyszín létrehozása `null`.

Módszer:

```
1 public async Task
    CreateLocationAsync_Returns_Null_When_Prefecture_Not_Found
    ()
```

5.6.2. DeleteLocationAsync - Helyszín törlése (nem található)

Leírás: A teszt egy nem létező helyszín törlését szimulálja, és ellenőrzi, hogy a törlés eredménye `false`.

Várható kimenet: A helyszín nem találása esetén a törlés eredménye `false`.

Módszer:

```
public async Task
    DeleteLocationAsync_Returns_False_When_Not_Found()
```

5.6.3. DeleteLocationAsync - Helyszín törlése (sikeres)

Leírás: A teszt egy létező helyszín törlését szimulálja, és ellenőrzi, hogy a törlés sikeres-e. A visszakapott eredménynek `true`-nak kell lennie.

Várható kimenet: A törlés sikeres, a rendszer `true` értéket ad vissza.

Módszer:

```
public async Task
    DeleteLocationAsync_Returns_True_When_Deleted()
```

5.6.4. GetAllLocationsAsync - Az összes helyszín lekérdezése

Leírás: A teszt az összes helyszín lekérdezését szimulálja, és ellenőrzi, hogy a helyszínek listája helyesen van-e visszaadva.

Várható kimenet: A helyszínek listája helyesen visszaadásra kerül.

Módszer:

```
public async Task
    GetAllLocationsAsync_Returns_List_Of_Locations()
```

5.6.5. GetLocationByIdAsync - Több eset kezelése

Leírás: A teszt több helyszín lekérdezését szimulálja különböző adatokkal, és ellenőrzi, hogy a helyszínek adatai helyesen vannak-e visszaadva.

Várható kimenet: A helyszínek adatai helyesen visszaadásra kerülnek.

Módszer:

```
public async Task
    GetLocationByIdAsync_Handles_Multiple_Cases(int
    locationId, string locationName, string street, string
    city, string cityRomanized, string streetRomanized, int
    maxCapacity, string phoneNumber)
```

GetLocationByIdAsync - Helyszín lekérdezése azonosító alapján (megtalálva - első eset)

Leírás: A teszt egy helyszín lekérdezését szimulálja az azonosító alapján, és ellenőrzi, hogy a helyszín helyesen van-e visszaadva.

Várható kimenet: A helyszín megtalálása esetén a rendszer a helyszín adatait tartalmazó `LocationDto` objektumot ad vissza.

Módszer:

```
public async Task
    GetLocationByIdAsync_Handles_Multiple_Cases(locationId:
    1, locationName: "", street: "", city: "", cityRomanized:
     "Shibuya", streetRomanized: "Shibuya", maxCapacity: 10,
    phoneNumber: "1234567890")
```

GetLocationByIdAsync - Helyszín lekérdezése azonosító alapján (megtalálva - második eset)

Leírás: A teszt egy helyszín lekérdezését szimulálja az azonosító alapján, és ellenőrzi, hogy a helyszín helyesen van-e visszaadva.

Várható kimenet: A helyszín megtalálása esetén a rendszer a helyszín adatait tartalmazó `LocationDto` objektumot ad vissza.

Módszer:

```
public async Task
    GetLocationByIdAsync_Handles_Multiple_Cases(locationId:
    2, locationName: "", street: "", city: "", cityRomanized:
     "", streetRomanized: "", maxCapacity: 20, phoneNumber: "
    9876543210")
```

5.6.6. GetLocationByIdAsync - Helyszín lekérdezése azonosító alapján (megtalálva)

Leírás: A teszt egy helyszín lekérdezését szimulálja az azonosító alapján, és ellenőrzi, hogy a helyszín helyesen van-e visszaadva.

Várható kimenet: A helyszín megtalálása esetén a rendszer a helyszín adatait tartalmazó `LocationDto` objektumot ad vissza.

Módszer:

```
public async Task
    GetLocationByIdAsync_Returns_LocationDto_When_Found()
```

5.6.7. GetLocationByIdAsync - Helyszín lekérdezése azonosító alapján (nem található)

Leírás: A teszt egy helyszín lekérdezését szimulálja egy nem létező azonosító alapján, és ellenőrzi, hogy a lekérdezés eredménye `null`.

Várható kimenet: A helyszín nem találása esetén a rendszer `null` értéket ad vissza.

Módszer:

```
public async Task
    GetLocationByIdAsync_Returns_Null_When_Location_Not_Found
    ()
```

5.6.8. GetLocationByIdAsync - Helyszín lekérdezése azonosító alapján (nem található)

Leírás: A teszt egy helyszín lekérdezését szimulálja egy nem létező azonosító alapján, és ellenőrzi, hogy a lekérdezés eredménye `null`.

Várható kimenet: A helyszín nem találása esetén a rendszer `null` értéket ad vissza.

Módszer:

```
public async Task
    GetLocationByIdAsync_Returns_Null_When_Not_Found_MinusID
    ()
```

5.6.9. UpdateLocationAsync - Helyszín frissítése (nem található)

Leírás: A teszt egy nem létező helyszín frissítését szimulálja, és ellenőrzi, hogy a frissítés eredménye `null`.

Várható kimenet: A helyszín nem találása esetén a frissítés eredménye `null`.

Módszer:

```
public async Task
    UpdateLocationAsync_ReturnsNull_WhenLocationNotFound()
```

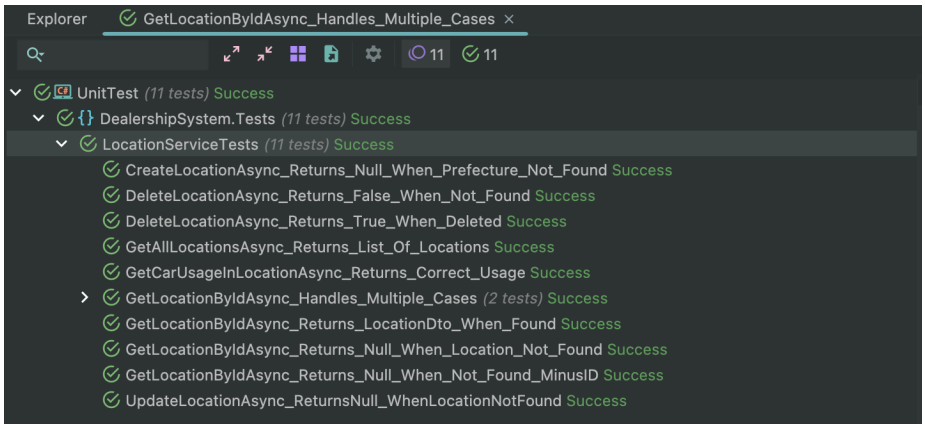
5.6.10. GetCarUsageInLocationAsync - Autók kapacitás telítettsége egy helyszínen

Leírás: A teszt egy helyszín autókapacitását szimulálja, és ellenőrzi, hogy a maximális kapacitás és az aktuális kapacitás helyesen van-e visszaadva.

Várható kimenet: A helyszín maximális kapacitása és az aktuális kapacitás helyesen visszaadásra kerül.

Módszer:

```
public async Task
    GetCarUsageInLocationAsync_Returns_Correct_Usage()
```



5.3. ábra. A lefutott unit tesztek

Az átviteli tesztelésnek az eredményei és kifejtése a WPF, illetve weboldal dokumentációjában történik.

5.7. API Végpontok

5.7.1. Autóval kapcsolatos végpontok

Ez a vezérlő a gépjárművekkel kapcsolatos műveletek kezelését biztosítja, mint például gépjárművek lekérése, létrehozása, frissítése és törlése.

1. GET /api/cars

Leírás: Ez a végpont lekérdezi az összes gépjárművet a rendszerből.
Paraméterek: Nincs.
Válasz:

- **200 OK**: A gépjárművek listája.

2. GET /api/cars/id

Leírás: Ez a végpont egy adott gépjármű adatait kérdezi le az `id` paraméter alapján.
Paraméterek:

- `id`: A lekérdezett gépjármű azonosítója (egész szám).

Válasz:

- **200 OK**: A gépjármű adatai.
- **404 Not Found**: Ha nem található a megadott azonosítóval rendelkező gépjármű.

3. POST /api/cars

Leírás: Ez a végpont lehetővé teszi egy új gépjármű létrehozását a rendszerben.
Paraméterek:

- `CreateCarDTO`: Az új gépjármű adatainak leírása.

Válasz:

- **201 Created**: A gépjármű sikeres létrehozása, a válasz tartalmazza az új gépjármű adatait.
- **400 Bad Request**: Ha a bemeneti adatok érvénytelenek.

4. PUT /api/cars/id

Leírás: Ez a végpont frissíti a megadott azonosítóval rendelkező gépjármű adatait.
Paraméterek:

- `id`: A módosítandó gépjármű azonosítója (egész szám).
- `CreateCarDTO`: Az új gépjármű adatainak leírása.

Válasz:

- **200 OK**: A frissített gépjármű adatai.
- **404 Not Found**: Ha nem található a megadott azonosítóval rendelkező gépjármű.
- **400 Bad Request**: Ha a bemeneti adatok érvénytelenek.

5. DELETE /api/cars/id

Leírás: Ez a végpont törli a megadott azonosítóval rendelkező gépjárművet a rendszerből.
Paraméterek:

- `id`: A törlendő gépjármű azonosítója (egész szám).

Válasz:

- **204 No Content**: A gépjármű sikeres törlése.
- **404 Not Found**: Ha nem található a megadott azonosítóval rendelkező gépjármű.

5.7.2. Autó gyártóval kapcsolatos végpontok

Ez a vezérlő az autógyártókkal kapcsolatos műveletek kezelését biztosítja, mint például az autógyártók lekérése, létrehozása, frissítése és törlése.

1. GET /api/cars/makers

Leírás: Ez a végpont lekérdezi az összes autógyártót a rendszerből.
Paraméterek: Nincs.
Válasz:

- **200 OK**: Az autógyártók listája.

2. GET /api/cars/makers/id

Leírás: Ez a végpont egy adott autógyártó adatait kérdezi le az `id` paraméter alapján.
Paraméterek:

- `id`: A lekérdezett autógyártó azonosítója (egész szám).

Válasz:

- **200 OK**: Az autógyártó adatai.
- **400 Bad Request**: Ha a megadott azonosítóval nem található autógyártó.

3. POST /api/cars/makers

Leírás: Ez a végpont lehetővé teszi egy új autógyártó létrehozását a rendszerben.
Paraméterek:

- `CreateCarMakerDTO`: Az új autógyártó adatainak leírása.

Válasz:

- **200 OK**: A sikeres autógyártó létrehozása, a válasz tartalmazza az új autógyártó adatait.
- **400 Bad Request**: Ha a bemeneti adatok érvénytelenek.
- **403 Forbidden**: Ha a műveletet nem lehet végrehajtani.

4. PUT /api/cars/makers

Leírás: Ez a végpont frissíti a megadott autógyártó adatait.
Paraméterek:

- `UpdateCarMakerDTO`: Az autógyártó frissített adatainak leírása.

Válasz:

- **200 OK**: A frissített autógyártó adatai.
- **400 Bad Request**: Ha a bemeneti adatok érvénytelenek.

5. DELETE /api/cars/makers/id

Leírás: Ez a végpont törli a megadott autógyártót a rendszerből.
Paraméterek:

- `id`: A törlendő autógyártó azonosítója (egész szám).

Válasz:

- **200 OK**: Az autógyártó sikeres törlése.
- **404 Not Found**: Ha nem található a megadott azonosítóval rendelkező autógyártó.

5.7.3. Autó metaadatokkal kapcsolatos végpontok

Ez a vezérlő az autókhoz kapcsolódó metaadatok lekérdezését teszi lehetővé, például a karosszériatípusok, váltótípusok, üzemanyagtípusok, hajtáslánc típusok és színek elérését.

1. GET /api/cars/metadata/bodytypes

Leírás: Lekéri az összes elérhető karosszériatípust.
Paraméterek: Nincs.
Válasz:

- **200 OK**: A karosszériatípusok listája.

2. GET /api/cars/metadata/bodytypes/id

Leírás: Lekéri egy adott karosszériatípus adatait.
Paraméterek:

- `id`: A karosszériatípus azonosítója (egész szám).

Válasz:

- **200 OK**: A kiválasztott karosszériatípus adatai.
- **404 Not Found**: Ha az adott azonosítóval nem található karosszériatípus.

3. GET /api/cars/metadata/transmissionTypes

Leírás: Lekéri az összes elérhető váltótípust.
Paraméterek: Nincs.
Válasz:

- **200 OK**: A váltótípusok listája.

4. GET /api/cars/metadata/transmissionTypes/id

Leírás: Lekéri egy adott váltótípus adatait.
Paraméterek:

- `id`: A váltótípus azonosítója (egész szám).

Válasz:

- **200 OK**: A kiválasztott váltótípus adatai.
- **404 Not Found**: Ha az adott azonosítóval nem található váltótípus.

5. GET /api/cars/metadata/fuelTypes

Leírás: Lekéri az összes elérhető üzemanyagtípust.
Paraméterek: Nincs.
Válasz:

- **200 OK**: Az üzemanyagtípusok listája.

6. GET /api/cars/metadata/fuelTypes/id

Leírás: Lekéri egy adott üzemanyagtípus adatait.
Paraméterek:

- `id`: Az üzemanyagtípus azonosítója (egész szám).

Válasz:

- **200 OK**: A kiválasztott üzemanyagtípus adatai.
- **404 Not Found**: Ha az adott azonosítóval nem található üzemanyagtípus.

7. GET /api/cars/metadata/drivetrainTypes

Leírás: Lekéri az összes elérhető hajtáslánc típust.
Paraméterek: Nincs.
Válasz:

- **200 OK**: A hajtáslánc típusok listája.

8. GET /api/cars/metadata/drivetrainTypes/id

Leírás: Lekéri egy adott hajtáslánc típus adatait.
Paraméterek:

- `id`: A hajtáslánc típus azonosítója (egész szám).

Válasz:

- **200 OK**: A kiválasztott hajtáslánc típus adatai.
- **404 Not Found**: Ha az adott azonosítóval nem található hajtáslánc típus.

9. GET /api/cars/metadata/colors

Leírás: Lekéri az összes elérhető színt.
Paraméterek: Nincs.
Válasz:

- **200 OK**: Az autószínek listája.

10. GET /api/cars/metadata/colors/id

Leírás: Lekéri egy adott szín adatait.
Paraméterek:

- `id`: A szín azonosítója (egész szám).

Válasz:

- **200 OK**: A kiválasztott szín adatai.
- **404 Not Found**: Ha az adott azonosítóval nem található szín.

5.7.4. Autómodellekkel kapcsolatos végpontok

Ez a vezérlő az autómodellek kezeléséhez szükséges végpontokat biztosítja, például az autómodellek listázását, keresését, létrehozását, frissítését és törlését.

1. GET /api/cars/models

Leírás: Lekéri az összes autómodellt.
Paraméterek: Nincs.
Válasz:

- **200 OK**: Az autómodellek listája.

2. POST /api/cars/models

Leírás: Létrehoz egy új autómodellt.
Paraméterek:

- `CreateCarModelDTO`: Az új autómodell adatai (JSON formátumban).

Válasz:

- **201 Created**: A sikeresen létrehozott autómodell adatai.
- **400 Bad Request**: Hibás kérés esetén, például ha a bemenet érvénytelen.

3. GET /api/cars/models/search

Leírás: Autómodellek keresése megadott szűrési feltételek alapján.
Paraméterek:

- `makerID` (opcionális) – A gyártó azonosítója.
- `startYear` (opcionális) – A gyártás kezdőéve.
- `endYear` (opcionális) – A gyártás befejező éve.
- `passengerCount` (opcionális) – Az utasok száma.

Válasz:

- **200 OK**: A keresési feltételeknek megfelelő autómodellek listája.

4. PUT /api/cars/models/id

Leírás: Frissíti a megadott azonosítójú autómodell adatait.
Paraméterek:

- `id` – Az autómodell azonosítója.
- `UpdateCarModelDTO` – Az új adatok JSON formátumban.

Válasz:

- **200 OK**: A sikeresen frissített autómodell adatai.
- **400 Bad Request**: Érvénytelen adatok esetén.
- **404 Not Found**: Ha az adott autómodell nem található.
- **500 Internal Server Error**: Ha egyéb hiba történt a frissítés során.

5. DELETE /api/cars/models/id

Leírás: Törli a megadott azonosítójú autómodellt.
Paraméterek:

- `id` – Az autómodell azonosítója.

Válasz:

- **204 No Content**: Sikeres törlés esetén.
- **404 Not Found**: Ha az autómodell nem található.

6. GET /api/cars/models/maker/makerId

Leírás: Lekéri egy adott gyártóhoz tartozó autómodelleket.
Paraméterek:

- `makerId` – Az autógyártó azonosítója.

Válasz:

- **200 OK**: Az adott gyártóhoz tartozó autómodellek listája.
- **404 Not Found**: Ha a gyártóhoz nem tartozik autómodell.

5.7.5. Alkalmazotti helyszínekkel kapcsolatos végpontok

Ez a vezérlő az alkalmazotti helyszínek kezeléséhez biztosít végpontokat. Az összes végpont adminisztrátori jogosultságot igényel.

1. GET /api/employeeLocations

Leírás: Lekéri az összes alkalmazotti helyszínt.
Jogosultság:

- **Admin** szerepkör szükséges.

Paraméterek: Nincs.
Válasz:

- **200 OK**: Az alkalmazotti helyszínek listája.

2. GET /api/employeeLocations/employeeId

Leírás: Lekéri egy adott alkalmazott helyszínét.
Jogosultság:

- **Admin** szerepkör szükséges.

Paraméterek:

- `employeeId` – Az alkalmazott egyedi azonosítója (UUID).

Válasz:

- **200 OK**: Az alkalmazott helyszíne.
- **404 Not Found**: Ha az alkalmazotti helyszín nem található.

3. POST /api/employeeLocations

Leírás: Hozzáad egy új alkalmazotti helyszínt.
Jogosultság:

- **Admin** szerepkör szükséges.

Paraméterek:

- `EmployeeLocation` – Az új alkalmazotti helyszín adatai (JSON formátumban).

Válasz:

- **201 Created**: A sikeresen létrehozott alkalmazotti helyszín.
- **400 Bad Request**: Ha az adatok érvénytelenek, vagy a helyszín hozzáadása sikertelen.

5.7.6. Motortípusokkal kapcsolatos végpontok

Ez a vezérlő az autókhoz tartozó motoradatok kezelésére szolgál.

1. POST /api/cars/engine

Leírás: Új motortípus létrehozása egy adott autómodellhez.
Paraméterek:

- `CreateEngineDTO` – A motor adatai (JSON formátumban):
  - `ModelID` – Az autómodell azonosítója.
  - `EngineSize` – A motor mérete.
  - `FuelType` – Az üzemanyagtípus.

Válasz:

- **200 OK**: A sikeresen létrehozott motoradatok.
- **400 Bad Request**: Ha az adatok érvénytelenek.

2. GET /api/cars/engine/model/{modelId}

Leírás: Lekéri egy adott autómodellhez tartozó motortípusokat.
Paraméterek:

- `modelId` – Az autómodell egyedi azonosítója.

Válasz:

- **200 OK**: Az adott autómodellhez tartozó motorok listája.
- **404 Not Found**: Ha nincs elérhető motortípus a modellhez.

3. PUT /api/cars/engine/update

Leírás: Egy meglévő motoradat módosítása.
Paraméterek:

- `UpdateEngineDTO` – A módosított motoradatok (JSON formátumban):
  - `ID` – A motor azonosítója.
  - `NewEngineSize` – Az új motor mérete.
  - `FuelTypeID` – Az új üzemanyagtípus azonosítója.

Válasz:

- **200 OK**: A sikeresen módosított motoradatok.
- **400 Bad Request**: Ha az adatok érvénytelenek.
- **404 Not Found**: Ha a motor nem található.

4. DELETE /api/cars/engine/{engineId}

Leírás: Egy adott motoradat törlése.
Paraméterek:

- `engineId` – A törlendő motor egyedi azonosítója.

Válasz:

- **200 OK**: A sikeres törlés.
- **404 Not Found**: Ha a motor nem található.

5.7.7. Képkezeléssel kapcsolatos végpontok

Ez a vezérlő az autókhoz tartozó képek feltöltésére és lekérdezésére szolgál.

1. POST /api/images/upload

Leírás: Egy új kép feltöltése egy adott autóhoz.
Paraméterek:

- `ImageUploadDto` – A feltöltendő kép adatai (multipart form-data):
  - `ImageFile` – A feltöltendő kép fájlja.
  - `CarID` – Az autó azonosítója, amelyhez a kép tartozik.

Válasz:

- **200 OK**: Sikeres feltöltés esetén a visszakapott kép URL-je.
- **400 Bad Request**: Ha a feltöltés sikertelen.

2. GET /api/images/car/{carId}

Leírás: Lekéri egy adott autóhoz tartozó képeket.
Paraméterek:

- `carId` – Az autó egyedi azonosítója.

Válasz:

- **200 OK**: Az adott autóhoz tartozó képek listája.
- **404 Not Found**: Ha az autóhoz nem tartoznak képek.

5.7.8. Telephelyekkel kapcsolatos végpontok

Ez a vezérlő a telephelyek kezelésére szolgál.

1. GET /api/locations

Leírás: Lekéri az összes telephelyet.
Válasz:

- **200 OK**: A telephelyek listája.
- **204 No Content**: Ha nincs elérhető telephely.

2. GET /api/locations/prefectures

Leírás: Lekéri az összes prefektúrát.
Válasz:

- **200 OK**: A prefektúrák listája.

3. GET /api/locations/{id}

Leírás: Lekér egy telephelyet azonosító alapján.
Paraméterek:

- `id` – A telephely azonosítója.

Válasz:

- **200 OK**: A telephely adatai.
- **404 Not Found**: Ha a telephely nem található.

4. POST /api/locations

Leírás: Új telephely létrehozása. Csak adminisztrátorok számára elérhető.
Fejlécek:

- `Authorization: Bearer <token>` – Adminisztrátori jogosultság szükséges.

Törzs:

- `LocationDto` – A létrehozandó telephely adatai.

Válasz:

- **200 OK**: A létrehozott telephely.
- **422 Unprocessable Entity**: Ha a létrehozás sikertelen.

5. PUT /api/locations/update

Leírás: Telephely adatainak frissítése. Csak adminisztrátorok számára elérhető.
Fejlécek:

- `Authorization: Bearer <token>` – Adminisztrátori jogosultság szükséges.

Törzs:

- `LocationDto` – A frissítendő telephely adatai.

Válasz:

- **200 OK**: A frissített telephely.
- **400 Bad Request**: Ha a bemeneti adatok érvénytelenek.
- **404 Not Found**: Ha a telephely nem található.
- **500 Internal Server Error**: Ha belső hiba történik.

6. DELETE /api/locations/{locationId}

Leírás: Egy telephely törlése. Csak adminisztrátorok számára elérhető.
Fejlécek:

- `Authorization: Bearer <token>` – Adminisztrátori jogosultság szükséges.

Paraméterek:

- `locationId` – A törlendő telephely azonosítója.

Válasz:

- **200 OK**: Sikeres törlés.
- **204 No Content**: Ha a telephely nem található.
- **500 Internal Server Error**: Ha belső hiba történt.

7. GET /api/locations/{locationId}/car-usage

Leírás: Lekéri egy telephely autóhasználati adatait.
Paraméterek:

- `locationId` – A telephely azonosítója.

Válasz:

- **200 OK**: Az aktuális és maximális kapacitás adatai.
- **404 Not Found**: Ha a telephely nem található.

5.7.9. Üzenetekkel kapcsolatos végpontok

Ez a vezérlő az üzenetek lekérdezésére és törlésére szolgál.

1. GET /api/messages/{userId}

Leírás: Lekéri egy adott felhasználóhoz tartozó üzeneteket.
Paraméterek:

- `userId` – A felhasználó azonosítója (GUID formátumban).

Válasz:

- **200 OK**: Az üzenetek listája.
- **401 Unauthorized**: Ha a felhasználó nem jogosult az üzenetek megtekintésére.

2. DELETE /api/messages/{messageId}

Leírás: Törli az adott azonosítójú üzenetet.
Paraméterek:

- `messageId` – A törlendő üzenet azonosítója.

Válasz:

- **204 No Content**: Sikeres törlés.
- **404 Not Found**: Ha az üzenet nem található.
- **401 Unauthorized**: Ha a felhasználó nem jogosult az üzenet törlésére.

5.7.10. Foglalásokkal kapcsolatos végpontok

Ez a vezérlő a felhasználók foglalásainak kezelésére szolgál.

1. GET /api/reservations

Leírás: Lekéri a bejelentkezett felhasználó összes foglalását.
Válasz:

- **200 OK**: A felhasználó foglalásainak listája.
- **401 Unauthorized**: Ha a felhasználó nincs bejelentkezve.

2. GET /api/reservations/{id}

Leírás: Lekéri az adott foglalás adatait.
Paraméterek:

- `id` – A foglalás azonosítója.

Válasz:

- **200 OK**: A foglalás adatai.
- **401 Unauthorized**: Ha a felhasználó nincs bejelentkezve.
- **404 Not Found**: Ha a foglalás nem található.

3. POST /api/reservations

Leírás: Új foglalást hoz létre a bejelentkezett felhasználó számára.
Törzs (Body): `CreateReservationDTO` objektumot vár.
Válasz:

- **201 Created**: A létrehozott foglalás adatai.
- **401 Unauthorized**: Ha a felhasználó nincs bejelentkezve.
- **403 Forbidden**: Ha a felhasználónak nincs jogosultsága a foglalás létrehozásához.

4. PUT /api/reservations/{id}

Leírás: Frissíti az adott foglalás adatait.
Paraméterek:

- `id` – A frissítendő foglalás azonosítója.

Törzs (Body): `UpdateReservationDTO` objektumot vár.
Válasz:

- **200 OK**: A frissített foglalás adatai.
- **401 Unauthorized**: Ha a felhasználó nincs bejelentkezve.
- **404 Not Found**: Ha a foglalás nem található.

5. DELETE /api/reservations/{id}

Leírás: Törli az adott foglalást.
Paraméterek:

- `id` – A törlendő foglalás azonosítója.

Válasz:

- **204 No Content**: Sikeres törlés.
- **401 Unauthorized**: Ha a felhasználó nincs bejelentkezve.
- **404 Not Found**: Ha a foglalás nem található.

5.7.11. Mentett autókkal kapcsolatos végpontok

Ez a vezérlő a felhasználók által mentett autók kezelésére szolgál.

1. GET /api/savedcars/{userId}

Leírás: Lekéri a megadott felhasználó által mentett autók azonosítóit.
Paraméterek:

- `userId` – A felhasználó azonosítója (GUID formátumban).

Válasz:

- **200 OK**: A felhasználó által mentett autók azonosítóinak listája.
- **401 Unauthorized**: Ha a felhasználó nem jogosult az adatok lekérésére.
- **500 Internal Server Error**: Ha váratlan hiba történt.

2. POST /api/savedcars/{userId}/save

Leírás: Ment egy autót a felhasználó számára.
Paraméterek:

- `userId` – A felhasználó azonosítója (GUID formátumban).

Törzs (Body):

- Egyetlen `carId` érték (az autó azonosítója, egész szám).

Válasz:

- **204 No Content**: Sikeres mentés.
- **401 Unauthorized**: Ha a felhasználó nem jogosult az autó mentésére.
- **500 Internal Server Error**: Ha váratlan hiba történt.

3. DELETE /api/savedcars/{userId}/remove/{carId}

Leírás: Eltávolít egy mentett autót a felhasználó listájából.
Paraméterek:

- `userId` – A felhasználó azonosítója (GUID formátumban).
- `carId` – Az eltávolítandó autó azonosítója (egész szám).

Válasz:

- **204 No Content**: Sikeres törlés.
- **401 Unauthorized**: Ha a felhasználó nem jogosult a törlésre.
- **500 Internal Server Error**: Ha váratlan hiba történt.

5.7.12. Felhasználókkal kapcsolatos végpontok

Ez a vezérlő a felhasználók kezelésére szolgál.

1. POST /api/users/register

Leírás: Új felhasználó regisztrálása.
Törzs (Body):

- Felhasználói adatok, mint email cím, jelszó, név, telefonszám és preferált nyelv.

Válasz:

- **200 OK**: A felhasználó sikeresen regisztrálva lett.
- **400 Bad Request**: Ha az email cím már foglalt, vagy ha érvénytelen adatokat adtak meg.

2. POST /api/users/login

Leírás: Felhasználó bejelentkezése.
Törzs (Body):

- Felhasználói email cím és jelszó.

Válasz:

- **200 OK**: A felhasználó sikeresen bejelentkezett.
- **401 Unauthorized**: Ha a felhasználói adatok helytelenek.

3. PUT /api/users/{userId}/update

Leírás: A felhasználó adatainak frissítése.
Paraméterek:

- `userId` – A frissíteni kívánt felhasználó azonosítója.

Törzs (Body):

- Frissített felhasználói adatok: név, telefonszám, preferált nyelv.

Válasz:

- **200 OK**: A felhasználói adatok sikeresen frissítve lettek.
- **400 Bad Request**: Ha érvénytelen adatokat adtak meg.
- **401 Unauthorized**: Ha a felhasználó nem jogosult a frissítésre.

4. DELETE /api/users/{userId}

Leírás: Felhasználó törlése.
Paraméterek:

- `userId` – A törölni kívánt felhasználó azonosítója.

Válasz:

- **204 No Content**: A felhasználó sikeresen törölve lett.
- **401 Unauthorized**: Ha a felhasználó nem jogosult a törléshez.
- **500 Internal Server Error**: Ha váratlan hiba történt.

6. fejezet

WPF dokumentációja

6.1. Bevezetés

Ez a rész a WPF kliens alkalmazásának telepítését, konfigurációját és működését mutatja be. Az alkalmazás egy backend szerverhez csatlakozik HTTP API-n keresztül.

6.2. Telepítés és Konfiguráció

6.2.1. Forráskód és Fordítás

A projekt Visual Studio segítségével fordítható. A forráskódban található `HttpClientService.cs` fájlban módosítani kell a backend szerver címét:

```
client = new HttpClient(tokenRefreshHandler)
{
BaseAddress = new Uri("https://192.168.1.100:7268")
};
```

Ezt a megfelelő IP-címre vagy domain névre kell cserélni a helyes működés érdekében.

6.2.2. Függőségek

Az alkalmazás a következő NuGet csomagokat használja:

- Newtonsoft.Json
- System.Net.Http

6.3. Felépítés és Ablakok

Az alkalmazás több ablakból és oldalból áll:

6.3.1. Ablakok (Windows)

- `LoginWindow.xaml`
- `AddCarWindow.xaml`
- `EditCarWindow.xaml`
- `EditCarModelWindow.xaml`
- `EditLocationWindow.xaml`
- `AddLocationWindow.xaml`
- `CreateNewCarModelWindow.xaml`
- `EditUserWindow.xaml`
- `EditCarMakerWindow.xaml`
- `AddUserWindow.xaml`
- `AddEngineWindow.xaml`

6.3.2. Oldalak (Pages)

- `CarModelPage.xaml`
- `CarsPage.xaml`
- `UserPage.xaml`
- `CarMakersPage.xaml`
- `LocationsPage.xaml`

6.4. Első indítás és bejelentkezés

- Indítsd el a PremiumCars alkalmazást.
- Ha a szerver (API) nem fut, a kliens nem fog tudni kapcsolódni, és a következő hiba jelenhet meg: `Could not connect to server`.
- Bejelentkezés admin jogosultságú felhasználóval (felhasználónév/jelszó). Hibás adatok esetén: `Unauthorized` vagy `Invalid credentials`.

6.5. Autók kezelése

6.5.1. Új autó hozzáadása

- Kérés: `POST /api/cars`
- Kötelező adatok:
  - **brand**: autómárka (pl. Toyota, BMW).
  - **model**: a kiválasztott márkához tartozó modell.
  - **engineSize**: motoradat (köbcenti, üzemanyag).
  - **bodyType**: karosszériatípus (SUV, Sedan, Hatchback stb.).
  - **locationId**: a telephely, ahol az autó elérhető.
  - **price**, **licensePlateNumber**, **MOTExpiry**, **VINNum**: az autó egyéb adatai.
- Képfeltöltés: fájlok (jpg, png), max. 5-10 MB/kép.
- Válaszok:
  - Siker: `200 OK` – „Car added successfully!"
  - Hiányzó mező: `400 Bad Request` – hibajelzés a hiányzó mező nevével.

6.5.2. Meglévő autó módosítása

- Kérés: `PUT /api/cars/{id}`
- Adatmódosítások:
  - Ár (**price**) frissítése.
  - Kilométeróra állása (**mileage**) módosítása.
  - Rendszám (**licensePlateNumber**) javítása.
  - Képek cseréje (backend támogatás esetén).
- Válaszok:
  - Siker: `200 OK` – „Autó adatai frissültek!"
  - Nem létező ID: `404 Not Found`.

6.6. Helyszínek kezelése

6.6.1. Új telephely hozzáadása

- Kérés: `POST /api/locations`
- Kötelező adatok:
  - **locationName**, **address** (irányítószám, város, utca, prefektúra).
  - **maxCapacity**: maximális kapacitás.
  - **phoneNumber**: telefonszám.
- Duplikáció ellenőrzése: ha az adott cím már létezik, `400 Bad Request`.

6.6.2. Telephely módosítása

- Kérés: `PUT /api/locations/{id}`
- Az adatok előtöltve jelennek meg szerkesztéskor.

6.7. Motoradatok, Márkák, Modellek szerkesztése

- `POST /api/cars/engine` – új motoradatok felvitele (**engineSize**, **fuelType**).
- `PUT /api/cars/makers` – márkák szerkesztése (**brandEnglish**, **brandJapanese**).
- `PUT /api/cars/models/{id}` – modellek módosítása (**modelNameEnglish**, **manufacturingStartYear**, **passengerCount**).

6.8. Gyakori hibák és megoldások

6.8.1. Szerverelérési problémák

- `Could not connect to server` vagy `404 Not Found` esetén:
  - Ellenőrizd, hogy a szerver fut-e.
  - Győződj meg róla, hogy a hálózati kapcsolat él.
  - Tűzfal beállítások ellenőrzése.

6.8.2. Érvénytelen beviteli adatok

- **Mileage**: csak számokat fogad el. Hiba esetén: `Invalid integer format`.
- **VINNum**: max. 17 karakter lehet. Hosszabb adatnál: `400 Bad Request`.
- **MOTExpiry**: elvárt formátum: `YYYY-MM-DD`.

6.9. Tesztkörnyezet

Az alkalmazás tesztelése az alábbi környezetben történt:

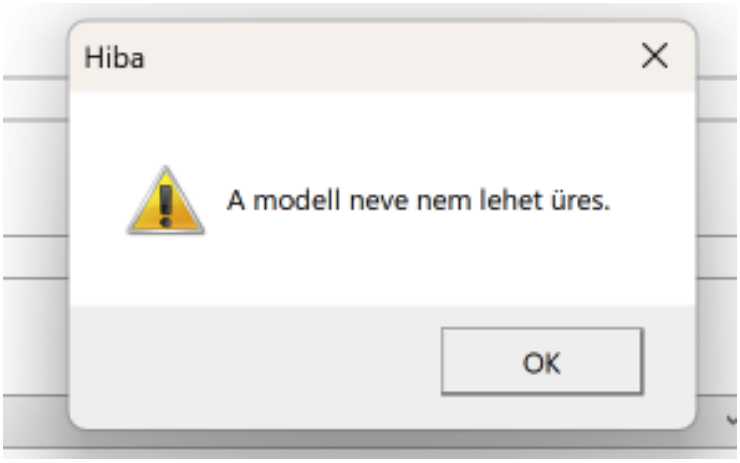
- Hardver: Apple M1 (ARM architektúra)
- Operációs rendszer: Windows 11 ARM, japán nyelvi beállításokkal
- Képernyőfelbontás: 1920x1080
- Böngésző: Google Chrome (legújabb verzió)

Megjegyzés: Noha bizonyos mezők japán nyelvű bemenethez specializáltak, ezek validálása nem történik meg, mivel az túlzottan korlátozó lenne, és nehezítené az egynyelvű tesztelést. Ha egy mező japán karaktereket kér, akkor a helyes eljárás a latin betűs bevitel vagy adott esetben a mező figyelmen kívül hagyása (pl. név Kanji mező, amelyet magyar és angol nyelvű felhasználók jellemzően nem használnak).

6.10. Teszt Esetek

6.10.1. Üres modell név esetén hibaüzenet

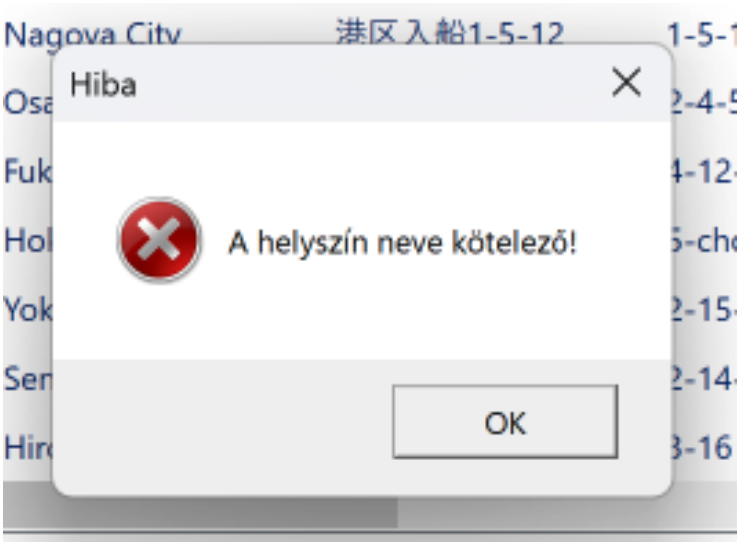
Ha a modell neve üres, a rendszer hibát dob: `Modell neve nem lehet üres`.



6.1. ábra. Hibaüzenet üres modellnév esetén

6.10.2. Üres helyszín név esetén hibaüzenet

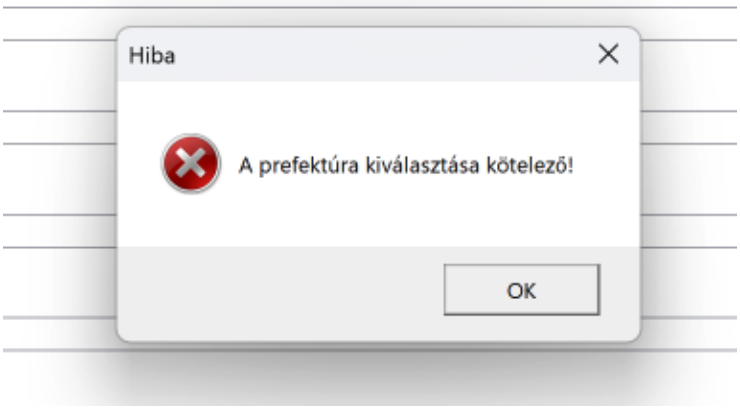
Ha a helyszín neve nincs kitöltve, a rendszer az alábbi hibát dobja: `A helyszín neve kötelező`.



6.2. ábra. Hibaüzenet üres helyszínnév esetén

6.10.3. Nincs kiválasztva prefektúra

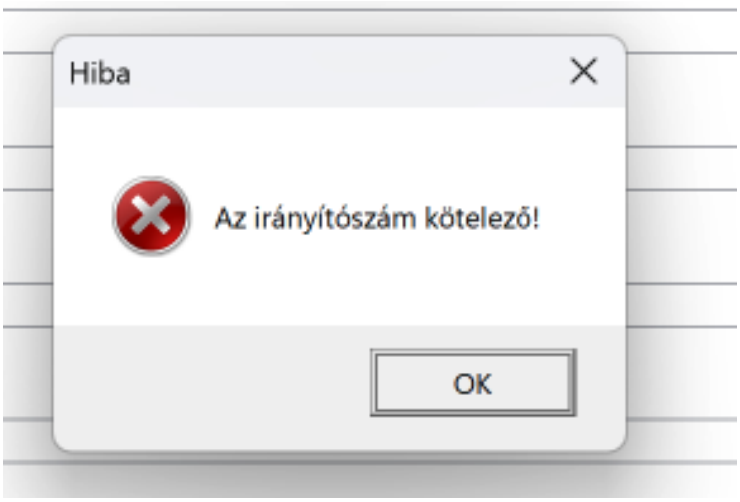
Ha a felhasználó nem választ ki prefektúrát, a következő hibaüzenet jelenik meg: `Prefektúra kiválasztása kötelező`.



6.3. ábra. Hibaüzenet hiányzó prefektúra esetén

6.10.4. Üres irányítószám

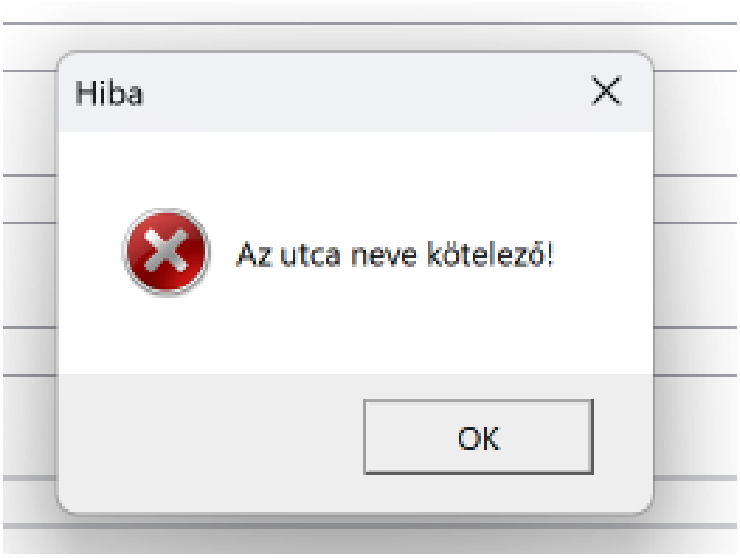
Ha az irányítószám nincs kitöltve, a rendszer az alábbi hibaüzenetet jeleníti meg: `Az irányítószám kitöltése kötelező`.



6.4. ábra. Hibaüzenet üres irányítószám esetén

6.10.5. Üres utcanév esetén hibaüzenet

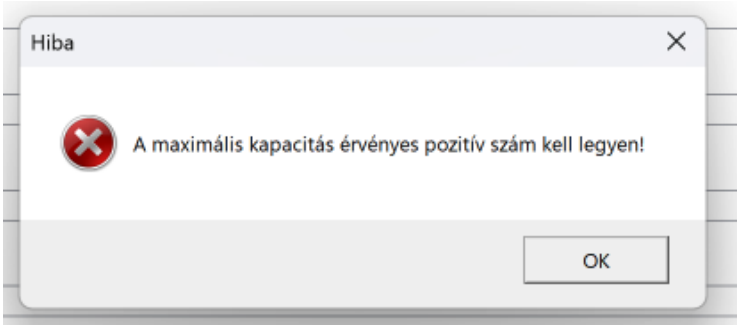
Ha az utca neve nincs kitöltve, a következő hibaüzenet jelenik meg: `Az utca neve kötelező`.



6.5. ábra. Hibaüzenet üres utcanév esetén

6.10.6. Érvénytelen maximális kapacitás

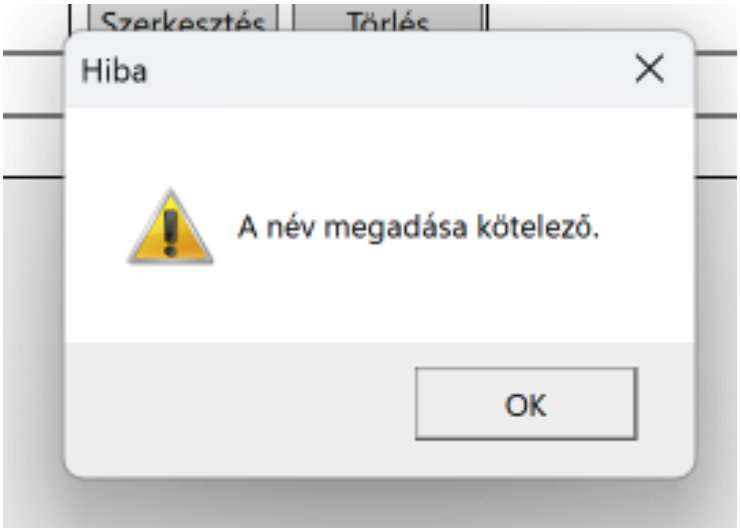
Ha a maximális kapacitás nem pozitív szám, a rendszer a következő hibát dobja: `A maximális kapacitás érvényes pozitív szám kell hogy legyen`.



6.6. ábra. Hibaüzenet érvénytelen kapacitás esetén

6.10.7. Új felhasználó létrehozása név nélkül

Ha az új felhasználó neve nincs megadva, a rendszer az alábbi hibát dobja: `A név megadása kötelező`.



6.7. ábra. Hibaüzenet üres felhasználónév esetén

6.11. További fejlesztési tervek

- **Statisztikák**: autók száma telephelyenként, átlagos értékesítési idő, legnépszerűbb márkák.
- **Felhasználókezelés**: admin jogok kezelése, moderátorok hozzáadása, fiókok felfüggesztése.
- **Riportok és kimutatások**: PDF export, grafikonok, statisztika.

6.12. Backend Kommunikáció

Az alkalmazás a backend API-val való kommunikációhoz a `HttpClientService` osztályt használja, amely biztosítja az authentikációt és a token frissítést.

6.13. Token Frissítés és HTTP Kérések Kezelése

A `WPF` kliens alkalmazás a `HttpClientService` és a `TokenRefreshHttpHandler` osztályokat használja a backend API-val való kommunikáció során. A következő szakasz részletesebben bemutatja ezek működését és az egyes lépések logikáját.

6.13.1. TokenRefreshHttpHandler – Kérések és Token Frissítés Kezelése

A `TokenRefreshHttpHandler` egy egyedi `DelegatingHandler`, amely figyeli a kérések válaszát, és ha az `401 Unauthorized` státuszkódot kap, automatikusan megpróbálja frissíteni a hozzáférési tokent.

Folyamat:

1. **Elsődleges kérés küldése:** Az osztály először naplózza a küldött kérést (`Debug.WriteLine` segítségével). Továbbítja a kérést az alatta lévő `HttpMessageHandler` példánynak. Naplózza a kapott választ (`Debug.WriteLine`).
2. **401 Unauthorized válasz kezelése:** Ha a szerver `401 Unauthorized` válasszal reagál, az azt jelenti, hogy a hozzáférési token lejárt. Naplózza az eseményt: `"Token expired, attempting to refresh..."` Meghívja a `HttpClientService.RefreshTokenIfNeeded()` metódust.
3. **Token frissítés és újrapróbálkozás:** Ha a token frissítés sikeres:
   - Lekéri az új hozzáférési tokent a `HttpClientService.GetAccessToken()` metódussal.
   - Frissíti a kérés fejlécében az `Authorization` mezőt az új tokennel.
   - Ismét elküldi az eredeti kérést.

   Ha a frissítés sikertelen, a válasz változatlanul visszatér.

Példa:

A következő kódrészlet mutatja, hogyan történik a token frissítés és újrapróbálkozás a `401 Unauthorized` státuszkód esetén:

```
if (response.StatusCode == HttpStatusCode.Unauthorized)
{
    Debug.WriteLine("Token expired, attempting to refresh...");

    bool refreshSuccess = await HttpClientService.RefreshTokenIfNeeded();

    if (refreshSuccess)
    {
        var newAccessToken = HttpClientService.GetAccessToken();
        if (!string.IsNullOrEmpty(newAccessToken))
        {
            Debug.WriteLine("Access token refreshed, retrying request with new token.");
            request.Headers.Authorization = new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", newAccessToken);
            response = await base.SendAsync(request, cancellationToken);
        }
        else
        {
            Debug.WriteLine("Failed to get new access token.");
        }
    }
    else
    {
        Debug.WriteLine("Token refresh failed.");
    }
}
```

6.13.2. HttpClientService – HTTP Kliens Kezelése

A `HttpClientService` egy statikus osztály, amely az alábbi funkciókat látja el:

- **HTTP kliens inicializálása** a megfelelő `HttpMessageHandler` lánccal.
- **Süti kezelés** (`CookieContainer`) az autentikációs adatok tárolására.
- **Token frissítési mechanizmus** a `RefreshTokenIfNeeded()` metódussal.

Fontos változók és objektumok:

```
private static readonly CookieContainer cookieContainer;
private static readonly HttpClientHandler handler;
private static readonly HttpClient client;
```

Token frissítés folyamata:

1. **Refresh token kinyerése a sütikből.**
2. **Token frissítési kérelem küldése a szerver `/api/users/refresh` végpontjára.**
3. **Ha a válasz sikeres (`200 OK`), az új tokenek mentése sütiként.**
4. **Ha a frissítés sikertelen, az eredeti válasz visszaadása.**

Kód:

```
public static async Task<bool> RefreshTokenIfNeeded()
{
    var cookies = cookieContainer.GetCookies(client.
        BaseAddress);
    var refreshToken = cookies["RefreshToken"]?.Value;

    if (refreshToken != null)
    {
        var content = new StringContent($"\"{refreshToken}\"
            ", Encoding.UTF8, "application/json");
        HttpResponseMessage response = await client.
            PostAsync("/api/users/refresh", content);

        if (response.IsSuccessStatusCode)
        {
            var responseContent = await response.Content.
                ReadAsStringAsync();
            var newTokens = JsonSerializer.Deserialize<
                TokenResponseDTO>(responseContent, new
                JsonSerializerOptions {
                PropertyNameCaseInsensitive = true });

            if (newTokens?.AccessToken != null)
            {
                AddCookie("AccessToken", newTokens.
                    AccessToken, client.BaseAddress.ToString
                    ());
                AddCookie("RefreshToken", newTokens.
                    RefreshToken, client.BaseAddress.ToString
                    ());
```

```
20                      return true;
21                  }
22              }
23      }
24
25      return false;
26  }
```

6.14. Összegzés

Az alkalmazás a `TokenRefreshHttpHandler` és a `HttpClientService` osztályok segítségével biztosítja, hogy az autentikációs tokenek automatikusan frissüljenek, ha azok lejárnak, anélkül hogy a felhasználónak újra be kellene jelentkeznie. A `TokenRefreshHttpHandler` a HTTP kérések kezelésére szolgál, és ha `401 Unauthorized` válasz érkezik, megpróbálja frissíteni a hozzáférési tokent és újraküldi a kérést. A `HttpClientService` felelős az autentikációs sütik tárolásáért és a tokenek frissítéséért. A kód és a példák bemutatják, hogyan történik a tokenek kezelése és a kérések újraküldése a frissített tokenekkel.

7. fejezet

Frontend

7.1. Telepítési Útmutató

Ez az útmutató bemutatja, hogyan telepítheted és futtathatod a React alapú alkalmazást a fejlesztői környezetedben.

7.2. Előfeltételek

Mielőtt nekiállnál a telepítésnek, győződj meg róla, hogy az alábbi szoftverek telepítve vannak a gépeden:

- **Node.js**: A React alkalmazás futtatásához szükséges. A Node.js letöltéséhez és telepítéséhez látogass el a `https://nodejs.org/` weboldalra.
- **npm (Node Package Manager)**: Az npm alapértelmezés szerint telepítve van a Node.js-sel együtt.

A telepítés után ellenőrizheted, hogy a Node.js és az npm helyesen telepítve van a következő parancsok futtatásával:

```
node -v
npm -v
```

7.3. Alkalmazás Telepítése

Miután a projektet klónoztad, kövesd az alábbi lépéseket:

7.3.1. Navigálj a frontend könyvtárba

A projekt mappájában lépj a frontend könyvtárba:

```
cd frontend
```

7.3.2. Függőségek telepítése

A projekt összes szükséges csomagját és függőségét az npm telepíti a következő parancs futtatásával:

```
npm install
```

Ez letölti a projektben meghatározott összes függőséget a `node_modules` mappába.

7.3.3. IP cím módosítása (ha szükséges)

A backend szerver IP-címét az alkalmazásban található `src/api/axiosInstance.ts` fájlban kell módosítanod, ha nem `localhost`-ot használsz. Keresd meg az alábbi sort:

```
const api = axios.create({
  baseURL: 'http://localhost:5000/api',  % itt módosíthatod az IP címet
});
```

Cseréld ki `localhost` értékét a megfelelő IP címre, ha a backend szerver más gépen vagy IP címen fut.

7.3.4. Futtasd a fejlesztői szervert

A fejlesztés során a következő parancsot használhatod az alkalmazás elindításához a helyi környezetben:

```
npm run dev
```

Ez elindítja a Vite alapú fejlesztői szervert, és elérheted az alkalmazást a böngészőben a `http://localhost:5173` címen.

7.3.5. Frontend Projektstruktúra

A frontend alkalmazás a React keretrendszerre épül, és a Vite fejlesztői környezetet használja. Az alábbiakban bemutatjuk a könyvtárstruktúrát és az egyes fájlok szerepét.

Főkönyvtár (`frontend`)

- `.idea` – IDE (pl. WebStorm) konfigurációs fájljai.
- `node_modules` – A projekt függőségeit tartalmazza (automatikusan generálódik).
- `public` – Statikus fájlokat tartalmaz, például képeket és az `index.html`-t.
- `index.html` – Az alkalmazás belépési pontja.
- `vite.svg` – A Vite logója.

`src` **Könyvtár (Forráskód)**

- `api` – A backend API hívásait kezelő szolgáltatásfájlokat tartalmaz.
  - `axiosInstance.ts` – Az Axios konfigurációját tartalmazza.
  - `carMetadataService.ts` – Autó-specifikus metaadatokat kezel.
  - `carService.ts` – Autókkal kapcsolatos API-hívásokat kezel.
  - `locationService.ts` – Helyszínekkel kapcsolatos API-hívásokat kezel.
  - `messageService.ts` – Üzenetek küldését és fogadását kezeli.
  - `reservationService.ts` – Foglalási műveletek API-hívásai.
  - `savedCarService.ts` – Mentett autókkal kapcsolatos műveletek.
  - `userService.ts` – Felhasználókezelő API-hívások.
- `assets` – Statikus fájlokat, képeket és egyéb erőforrásokat tartalmaz.
  - `Author` – Szerzői információk.
  - `react.svg` – React logó.
- `Components` – Újrafelhasználható UI komponenseket tartalmaz.
  - `Footer.tsx` – Az oldal alján található lábléc komponens.
  - `Header.tsx` – Az oldal felső navigációs sávja.
  - `LoginModal.tsx` – Bejelentkezési modálablak.
  - `MessagesModal.tsx` – Üzenetek megjelenítésére szolgáló modál.
  - `MobileMenu.tsx` – Mobilbarát navigációs menü.
  - `PasswordChangeModal.tsx` – Jelszóváltoztatás modálablak.
  - `PostCarOffer.tsx` – Új autóhirdetés feladását lehetővé tevő komponens.
  - `SavedCarsModal.tsx` – Mentett autók megtekintése.
  - `ScrollToTop.tsx` – Az oldal tetejére ugró funkció.
- `Interfaces` – Az alkalmazásban használt interfészeket tartalmazza.
  - `Message.ts` – Üzenetek struktúráját definiálja.
  - `Reservation.ts` – Foglalásokhoz tartozó adatmodell.
  - `User.ts` – Felhasználói adatok struktúrája.
- `Pages` – Az alkalmazás főoldalait tartalmazza.
  - `CarDetails.tsx` – Egy adott autó részletes adatait tartalmazó oldal.
  - `Cars.tsx` – Az elérhető autók listázása.
  - `Home.tsx` – A kezdőoldal.

  - `LocationPage.tsx` – A telephelyeket bemutató oldal.
  - `Profile.tsx` – A felhasználói profiloldal.
- `Types` – Az alkalmazásban használt típusdefiníciókat tartalmazza.
- `App.tsx` – Az alkalmazás fő komponense.
- `main.tsx` – Az alkalmazás belépési pontja, itt renderelődik az alkalmazás.
- `translations.ts` – A többnyelvűség támogatásához használt fájl.
- `UserContext.tsx` – A felhasználói kontextust biztosító fájl (React Context API).

Konfigurációs és egyéb fájlok

- `vite-env.d.ts` – Vite konfigurációhoz szükséges TypeScript fájl.
- `eslint.config.js` – Az ESLint beállításait tartalmazza.
- `package.json` – Az alkalmazás függőségeit és parancsait tartalmazza.
- `package-lock.json` – A telepített függőségek pontos verzióit rögzíti.
- `README.md` – Dokumentáció és telepítési instrukciók.
- `tsconfig.json` – TypeScript beállításokat tartalmaz.
- `tsconfig.app.json` – Az alkalmazás TypeScript konfigurációja.
- `tsconfig.node.json` – A Node.js-specifikus TypeScript beállítások.
- `vite.config.ts` – A Vite konfigurációs fájlja.

Összegzés: Ez a frontend struktúra egy jól felépített, moduláris React alkalmazást képvisel. A különböző mappák és fájlok felelőssége egyértelműen elkülönül, megkönnyítve a fejlesztést, bővítést és karbantartást.

7.4. Hitelesítés és API konfiguráció

A frontend a backenddel az Axios könyvtár segítségével kommunikál. A hitelesítés cookie-kon keresztül történik, és az `AccessToken` a cookie-ból van lekérdezve és hozzáadva a kérés fejlécéhez.

8. fejezet

Záróösszegzés

A **PremiumCars** projekt célja egy átfogó és hatékony rendszer kifejlesztése volt a japán használt autókereskedelem számára, figyelembe véve az iparági igényeket és sajátos elvárásokat. A rendszer biztosította az autók eladásának, vásárlásának, nyilvántartásának és a kapcsolódó adminisztratív folyamatok egyszerűsítését, beleértve az autók állapotának frissítését, a műszaki vizsgálatok nyomon követését, valamint a felhasználói élmény javítását. A projekt csapatmunka eredménye volt, ahol minden tag hozzájárult a backend, frontend és WPF UI fejlesztéséhez.

Bár a projekt még nem érte el végső formáját, és még számos további funkció és finomhangolás vár rá, az alapok már stabilak és jól működnek. Az alkalmazás képes kiszolgálni a jelenlegi igényeket, és biztosítja a rendszer hosszú távú fejlődéséhez szükséges alapot. A jövőbeni fejlesztések célja, hogy a PremiumCars valódi piaci szereplővé váljon a japán használt autópiacon, és képes legyen versenyképes szolgáltatásokat nyújtani a vásárlók és eladók számára.

A projekt során használt fejlesztési technológiák széles skáláját alkalmaztuk:

- A backend fejlesztése C# 8.0-ban történt, .NET Core környezetben.
- A frontend React és TypeScript segítségével valósult meg, amely modern, dinamikus felhasználói élményt biztosított.
- Az alkalmazás WPF felületet használ a felhasználói interakciók kezelésére, C# 7.3-as verzióval.
- Az adatbázis-kezelés PostgreSQL alapú, amely megbízható és rugalmas tárolást biztosít az alkalmazás számára.

A projekt során a legnagyobb kihívást az üzleti logika megértése és a különböző szolgáltatások integrálása jelentette, különös figyelmet fordítva a japán használt autópiac specifikus igényeire, mint a járművek teljeskörű felújítása, a *shaken* műszaki vizsgálatra nyújtott ajánlataikra, valamint a garanciák és szervizszolgáltatások biztosítása. A japán piac különlegessége, hogy az autókereskedők rengeteg telephelyen és logisztikai központokon keresztül működnek,

amelyek lehetővé teszik az autók gyors szállítását és az országos szintű értékesítést. Ez a logisztikai és szervizelési háttér komoly hatással van a fejlesztett rendszerre, mivel a szolgáltatások minősége és a vásárlói bizalom kulcsfontosságú tényezők az üzleti sikerhez.

A személyes céljaim között szerepelt a japán használt autópiac mélyebb megértése is. Különösen figyeltem arra, hogyan működnek a legnagyobb cégek, mint a WeCars, a Gulliver, a Car Sensor és a Toyota Used Car. Mindezek a vállalatok különböző módon integrálják a szervizszolgáltatásokat és garanciális rendszereket, valamint országos szintű telephelyekkel és logisztikai rendszerekkel rendelkeznek. Bár a japán cégek nem terjeszkedtek nemzetközi piacokra, a több telephelyes működés és az autók közötti szállítás mégis globális gondolkodást igényel a szolgáltatás fejlesztésében.

A projekt során szerzett tapasztalataim segítettek abban, hogy mélyebb megértést nyerjek a fejlesztési környezetek optimalizálásában és az alkalmazások megbízhatóságának javításában. Emellett érdeklődésem a *DevOps* iránt is növekedett, amely a jövőben lehetőséget adhat az alkalmazások gyorsabb fejlesztésére és skálázhatóbb üzemeltetésére, különösen felhőalapú infrastruktúrák és automatizált fejlesztési folyamatok révén.

Összességében a **PremiumCars** projekt nemcsak technikai kihívásokat jelentett, hanem lehetőséget biztosított arra is, hogy a japán piacra vonatkozó speciális ismereteket szereztem, amelyek segítettek a szoftverfejlesztés során figyelembe venni a valós üzleti igényeket. Bár a projekt még csak az első lépéseket tette meg a piacon való megjelenés felé, a kidolgozott alapok már készen állnak a további fejlesztésekre, amelyek lehetővé teszik, hogy a PremiumCars egy sikeres és megbízható szereplővé váljon a japán használt autókereskedelemben.

8.1. Köszönetnyilvánítás

Köszönet illeti:

- Szabó Sándor tanár urat, aki mentorálta a projektet és folyamatosan segítette a csapatot a fejlesztés során.
- A csapattársaimat, akik a projekt különböző részein dolgoztak (backend, frontend, WPF UI).
- A tesztelőket, akik észrevételeikkel finomhangolták a funkciókat (szülők, barátok).