

**SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM  
MAROSVÁSÁRHELYI KAR,  
INFORMATIKA SZAK**



**SAPIENTIA**  
ERDÉLYI MAGYAR  
TUDOMÁNYEGYETEM



**CarSpotter**

**Szoftver rendszerek projekt**

**Témavezető:**  
Dr. Szántó Zoltán

**Végzős hallgatók:**  
Biró Edina-Anita  
Demeter Dávid Levente  
Rigmányi Loránd

**2024**

# Tartalomjegyzék

<b>1. Bevezető</b>	<b>3</b>
<b>2. Projekt célja</b>	<b>4</b>
<b>3. Követelmény specifikációk</b>	<b>5</b>
3.1. Felhasználói követelmények. Use case diagram. . . . .	5
3.2. Rendszerkövetelmények . . . . .	6
3.2.1. Funkcionális követelmények . . . . .	6
3.2.2. Nem-funkcionális követelmények . . . . .	7
<b>4. Tervezés</b>	<b>8</b>
4.1. Architektúra. Komponens diagram . . . . .	8
4.2. Modulok leírása . . . . .	9
4.2.1. Kliens oldal . . . . .	9
4.2.2. Szerver oldal . . . . .	12
4.2.3. Adatbázis . . . . .	13
4.2.4. Adatbegyűjtés . . . . .	13
4.3. Osztálydiagramok. Adatbázis terv. . . . .	13
4.4. Verzió követő és projekt menedzsment . . . . .	15
<b>5. Alkalmazás működése</b>	<b>17</b>
5.1. Üdvözlőképernyő (Splash Screen) . . . . .	18
5.2. Bejelentkezés. Regisztráció . . . . .	18
5.3. Home képernyő . . . . .	19
5.4. Profil képernyő . . . . .	19
5.5. Friends (Követett felhasználók) képernyő . . . . .	20
5.6. Keresés (Search) képernyő . . . . .	21
5.7. Kedvencek (Favourite) képernyő . . . . .	22
5.8. Chatbot . . . . .	22
<b>6. Összefoglaló</b>	<b>24</b>
6.1. További fejlesztési lehetőségek . . . . .	24
<b>Ábrák jegyzéke</b>	<b>25</b>
<b>Irodalomjegyzék</b>	<b>26</b>

# 1. fejezet

## Bevezető

Egy új autó megvásárlása sosem egyszerű folyamat. Érdemes alaposan átgondolni, hogy melyik a számunkra megfelelő választás figyelembe véve olyan szempontokat, mint a vásárlásra szánt keret vagy hogy mire fogjuk leginkább használni az autót: nem mindegy, hogy többnyire csak a városban fogunk közlekedni, vagy gyakran teszünk meg hosszabb utat.

Manapság már rengeteg lehetőség áll a rendelkezésünkre, ha új autót szeretnénk vásárolni. Számos weboldal és applikáció létezik, ahol eladó autókat találhatunk. Sajnos új autó vásárlásakor sokan válnak átverések áldozatává, ugyanis sok az olyan rosszindulatú eladó, aki hasznott húz a vásárlók tudatlanságából. Ezért különösen fontos, hogy vásárlás előtt győződjünk meg arról, hogy tényleg azt kapjuk a pénzünkért, amit eladni próbálnak nekünk.

Erre a legjobb módszer egy biztos forrás, legyen az egy ismerős, aki eladással foglalkozik, vagy egy megbízható weboldal. Egy ilyen népszerű weboldal az Autovit, ahol minden lényeges információt meg lehet találni az eladásra szánt autóról. Ezek közül talán a legfontosabb az alvázszám(VIN), ami alapján azonosítani lehet az autót. Ennek birtokában ellenőrizni lehet a szervizelési előzményeit és a megtett kilóméterek változását is, így elkerülve az esetleges átveréseket.

Sajnos, egyes eladók rossz híre miatt, egyre nehezebb autó dealerként a vásárlók bizalmát elnyerni és jó hírnévre szert tenni, ami biztosíthatja a vállalkozás sikerét.

A CarSpotter alkalmazás a vásárlók és dealerek dolgát egyaránt próbálja megkönnyíteni, ehhez Autovit oldalról dolgoz fel információkat, könnyen elérhetővé téve ezeket a potenciális vásárlók vagy eladók számára. Lehetővé teszi minden az oldalon regisztrált dealer követését, tevékenységük monitorizálását, valamint erről lényegretörő statisztikákat készít. Továbbá a dealerként regisztrált felhasználóknak lehetőségük van a saját tevékenységüket, költségeiket és profitukat nyomon követni.

## 2. fejezet

### Projekt célja

A projekt célja, egy olyan applikáció fejlesztése, amely lehetővé teszi az <https://www.autovit.ro/> weboldalról származó adatok feldolgozását, statisztikák készítését, valamint megkönnyíti a felhasználók és az autokereskedők (dealerek) közötti interakciót. A hangsúly a dealerek keresésén, a kedvencek számon tartásán, más felhasználók követésén és az autók adatainak feldolgozásán van. A másik cél pedig az volt hogy egy

könnyű hozzáférést biztosítsunk az autovit.ro oldalhoz. Ezt egy Python script segítségével oldottuk meg, amely az oldalról automatikusan letölti és feldolgozza az adatokat. A diagrammok, illetve statisztikai ábrák segítenek a felhasználóknak a könnyed megérteben, illetve elemzésben. Elemezni a hónapos, vagy akár hetes statisztikákat az autók eladásával kapcsolatosan. A felhasználók közötti interakció is a célok közé tartozott, egy másik felhasználót követve látni lehet az általa kedvencként megjelölt dealereket.

Reményeink szerint ezek alapján a végeredmény egy olyan alkalmazás lesz, amely nemcsak megkönnyíti az autovit.ro weboldal elérését, hanem kibővíti annak funkcionálisait, és új lehetőségeket biztosít mind az autóvásárlók, mind a dealerek számára.

Tehát összegezve, egy olyan applikációt szerettünk volna megvalósítani, amely könnyebb hozzáférést biztosít az autovit.ro oldalon regisztrált dealerekhez és ezek autóihoz, valamint ezeket az adatokat releváns módon dolgozza fel, ezáltal segítve más kereskedők munkáját és a vásárlók döntésének meghozását egyaránt.

#### Felhasználóbarát élmény kialakítása:

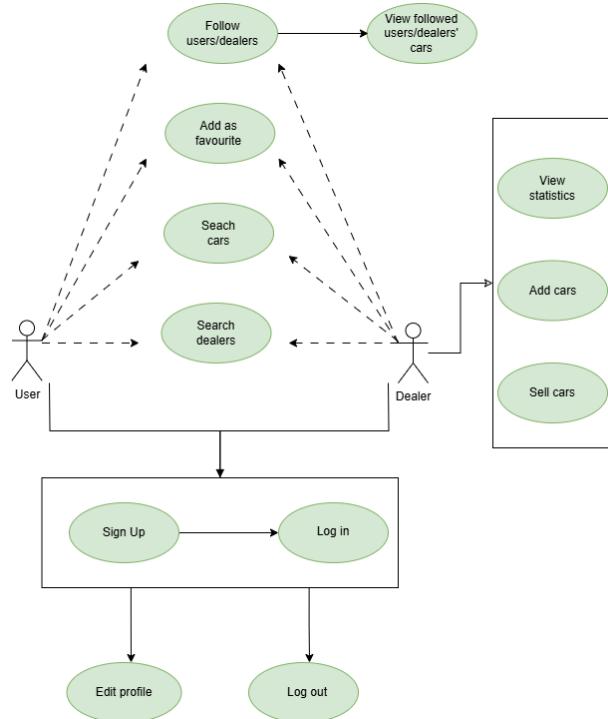
Az applikáció megtervezése és kivitelezése során törekedtünk a felhasználói élmény maximalizálására, hogy az alkalmazás intuitív, esztétikus és könnyen használható legyen. A felhasználó könnyen tudjon navigálni és keresni, valamint az alkalmazás felületén gyorsan és egyszerűen megtalálja a számára releváns információkat.

### 3. fejezet

## Követelmény specifikációk

### 3.1. Felhasználói követelmények. Use case diagram.

Kétféle felhasználó lehetséges: user és dealer, mint ahogy a 3.1 -es ábrán láthatjuk.



3.1. ábra. Use case diagram

User: Regisztrálnia kell a szükséges felhasználói adatokkal. Hogyha már létezik egy felhasználói fiokja, akkor be kell lépnie. Belépés után érhetőek el számára a további funkcionálisok: profil editelése, hogyha aktualizálni vagy kicserélni szeretne saját adatokat. Továbbá tud keresni dealereket vagy autókat. Kedvencekbe tud adni autókat, illetve tud követni más user vagy dealert. Hogyha követ egy bizonyos user/dealer akkor annak megtekintheti az autót. Nem utolsó sorban ki tud jelentkezni az fiokjából.

**Dealer:** Ugyanazok a funkcionálitások elérhetőek a dealerek számára, mint a sima userek számára, csak még kiegészítődik a következőkkel: autók hozzáadása(amik eladásra vannak szánva), statisztikák megnézése, ami a saját eladásokra és kocsikra vonatkozik, illetve tud eladni kocsikat.

## **3.2. Rendszerkövetelmények**

A számítógépes szoftverek hatékony működéséhez bizonyos hardveres és szoftveres feltételek megléte szükséges. Ezeket a feltételeket rendszerkövetelményeknek nevezzük, melyek két fő kategóriába sorolhatók: funkcionális követelmények, amelyek a szoftver konkrét működésére vonatkoznak, és nem funkcionális követelmények, amelyek a rendszer általános jellemzőit határozzák meg.

### **3.2.1. Funkcionális követelmények**

- **Felhasználókezelés:**
  - Az alkalmazás lehetővé teszi a felhasználók számára, hogy regisztráljanak, megadva nevüket, email címüket, telefonszámukat és jelszavukat.
  - A sikeres bejelentkezés után a felhasználók hozzáférhetnek saját profiljukhoz, ahol megtekinthetik vagy szerkeszthetik személyes adataikat, illetve profilképet állíthatnak be galériából.
- **Adatok letöltése és feldolgozása:**
  - Az alkalmazás Python script segítségével automatikusan letölти az autovit.ro weboldal adatait, beleértve a dealerek és az általuk feltöltött autók adatait.
  - A rendszer feldolgozza az adatokat, és statisztikai elemzéseket, diagramokat generál, amelyek segítik a felhasználókat az autók piaci trendjeinek megértésében.
- **Kedvencek kezelése:**
  - A felhasználók lehetőséget kapnak kedvenc dealerek és autók megjelölésére.
  - A kedvencek listája megtekinthető, vagy bármikor törölhető.
  - A felhasználók követhetnek más felhasználókat, és megtekinthetik az általuk kedvencként megjelölt dealereket és autókat.
- **Elemzések és statisztikák:**
  - Az alkalmazás diagramokat és statisztikai ábrákat kínál, amelyek hetekre vagy hónapokra lebontva mutatják az autók eladásával kapcsolatos adatokat, segítve ezzel a döntéshozást.

### **3.2.2. Nem-funkcionális követelmények**

- **Felhasználói élmény:**
  - Az alkalmazás felülete intuitív és könnyen használható, hogy minden felhasználó, függetlenül tapasztalati szintjétől, gyorsan eligazodhasson benne.
  - Az alkalmazás felülete esztétikus, modern kialakítású, és elkerüli a túlzsfolt elemeket, hogy tiszta megjelenítést biztosítson.
- **Skálázhatóság:**
  - A rendszer képes kezelní a növekvő számú felhasználó és adatmennyiségek esetén is stabilan működni, biztosítva a zökkenőmentes élményt.
- **Biztonság:**
  - A felhasználók autentifikációja JSON Web Token (JWT) használatával történik, biztosítva a biztonságos és megbízható belépést.
  - A felhasználók csak a szerepkörüknek megfelelő adatokhoz és funkciókhoz férhetnek hozzá, ezáltal biztosítva a megfelelő hozzáférés-ellenőrzést és adatvédelmet.
  - Az alkalmazás biztosítja, hogy a felhasználók számára elérhető adatok naponta automatikusan frissülnek, így mindenkor a legfrissebb információk állnak rendelkezésre.
- **Hardver:**
  - Az alkalmazás futtatásához az alapvető rendszerkövetelmény az okostelefon, legalább 2 GB RAM-mal és 1 GHz-es processzorral. A felhasználói élmény optimalizálása érdekében ajánlott egy gyorsabb processzor és legalább 4 GB RAM.
  - Android 4.1 (Jelly Bean) vagy újabb.
  - iOS 12.0 vagy újabb.
- **Hálózati kapcsolat:**
  - Stabil internetkapcsolattal kell rendelkeznie az alkalmazás gyors és gördülékeny működése érdekében.

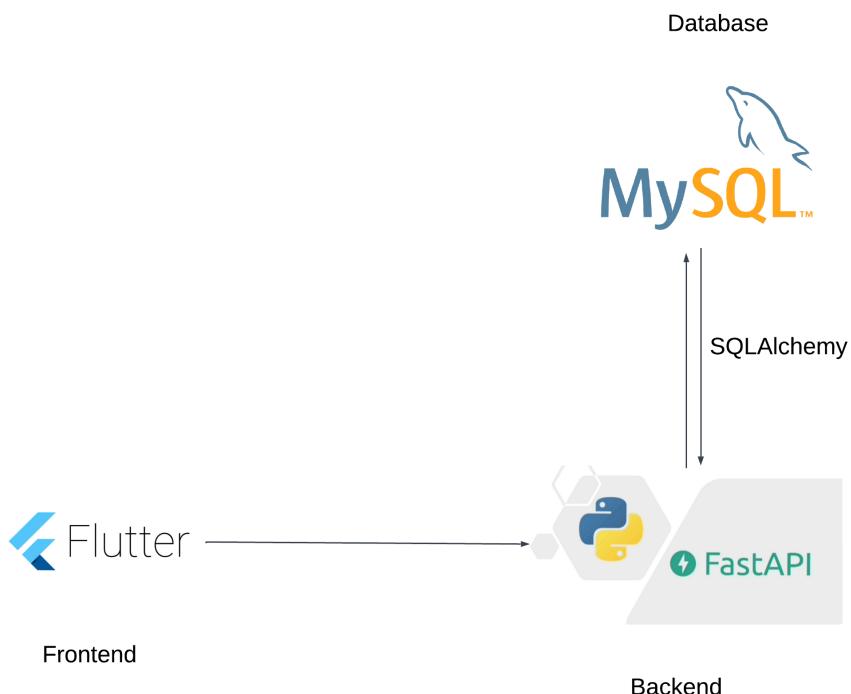
# 4. fejezet

## Tervezés

### 4.1. Architektúra. Komponens diagram

Az alkalmazásunk architektúrája a egy modern fejlesztési technológiákra épít, amelyek lehetővé teszik a rugalmas és hatékony fejlesztést. A gyors válaszidő, stabilitás és hatékony fejlesztést választottuk fő szempontonként. Az alábbiakban részletesen bemutatom a kliens és szerver oldali fejlesztést.

A 4.1 -es ábrán látható a komponens diagramunk.



4.1. ábra. Komponens diagram

Az alkalmazás a következő főbb részekből épül fel:

- Szerver oldal: Python FastAPI

- Kliens oldal: Flutter mobilalkalmazás
- Adatbázis: MySQL
- Adatok begyűjtése: Python BeautifulSoup könyvtár

## 4.2. Modulok leírása

### 4.2.1. Kliens oldal

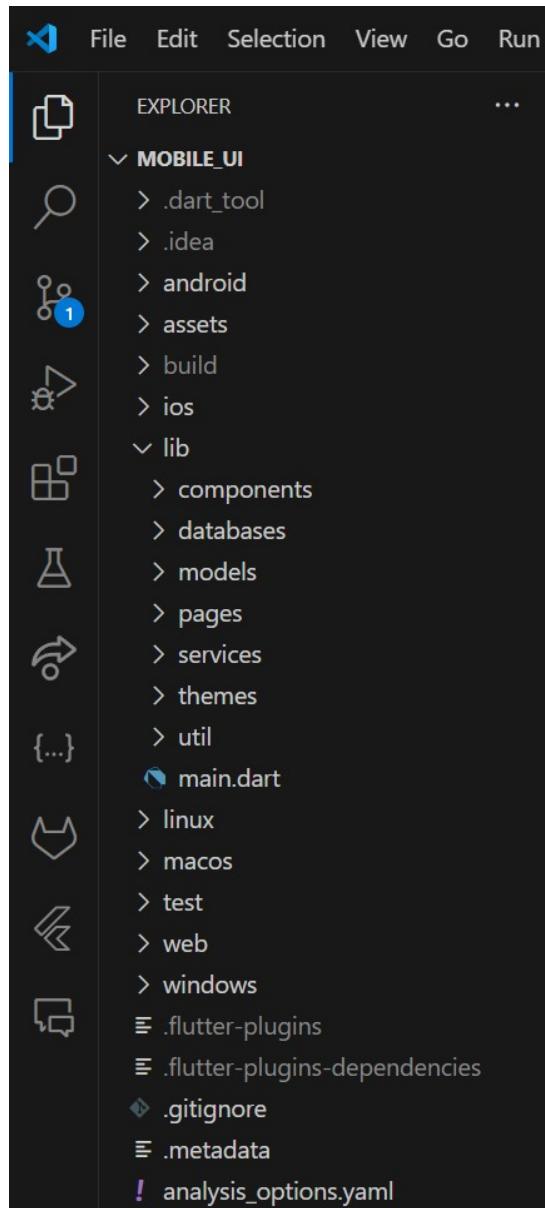
A kliens oldali fejlesztés során a Flutter keretrendszer választottuk, hogy egyetlen kódalap segítségével több platformon is futtathassuk az alkalmazást.

Miért választottuk a Fluttert?

- Multiplatform támogatás: A Flutter lehetőséget biztosít számunkra, hogy egyetlen kódot használjunk Android és iOS alkalmazások fejlesztésére, illetve a webes verzió is könnyen implementálható.
- Magas teljesítmény: Mivel a Flutter natív kódra fordítja a widgeteket, a felhasználói felület gyors és sima működését biztosítja.
- Hot reload funkció: Ez a funkció lehetővé teszi, hogy a fejlesztők valós időben lássák az alkalmazásuk változásait, így a tesztelési és hibakeresési folyamat is jelentősen felgyorsul.

A mobilalkalmazást a következő részekre osztottuk(lásd 4.2 -es ábra):

- Services: Itt találhatóak azok a függvényhívások és modulok, amelyek a backenddel való kommunikációért felelősek. Tartalmazza a különböző autókon végzett műveleteket megvalósító függvényeket és az autentikáció logikát is.
- Databases: Lokális adatbázisok a Flutter Isar könyvtárának segítségével, ezek szolgálnak az autókhoz adott megjegyzések, illetve a különboző események(eladás, vétel) eltárolására.
- Models: A models mappában tároljuk az alkalmazás model osztályait, amelyek a backend és frontend közötti adatcserét teszik lehetővé. Az API által visszaadott Json adatokat a válaszok feldolgozása után átalakítjuk model osztályokká, amelyek egyszerűsítik az adatok kezelhetőségét a frontend logikájában. A model osztályok segítségével biztosítjuk, hogy az adatokat jól strukturált formában, típusellenőrzéssel és átlátható módon tudjuk kezelní.
- Components: a components mappában olyan UI elemeket megvalósító osztályok vannak, amiket szerte az alkalmazásban használunk az adatok megjelenítésére. Ezekből épülnek fel a különböző oldalak.

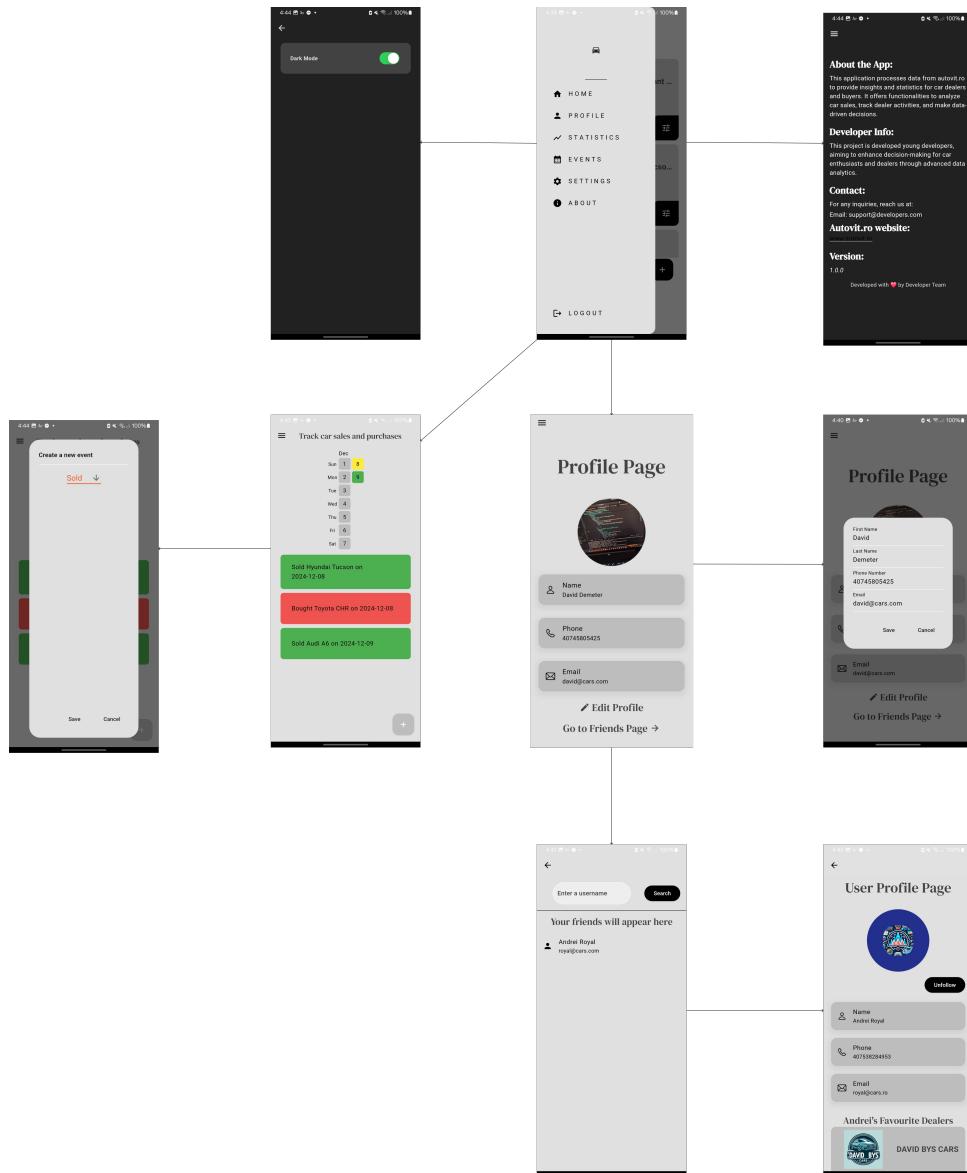


4.2. ábra. Folder Structure

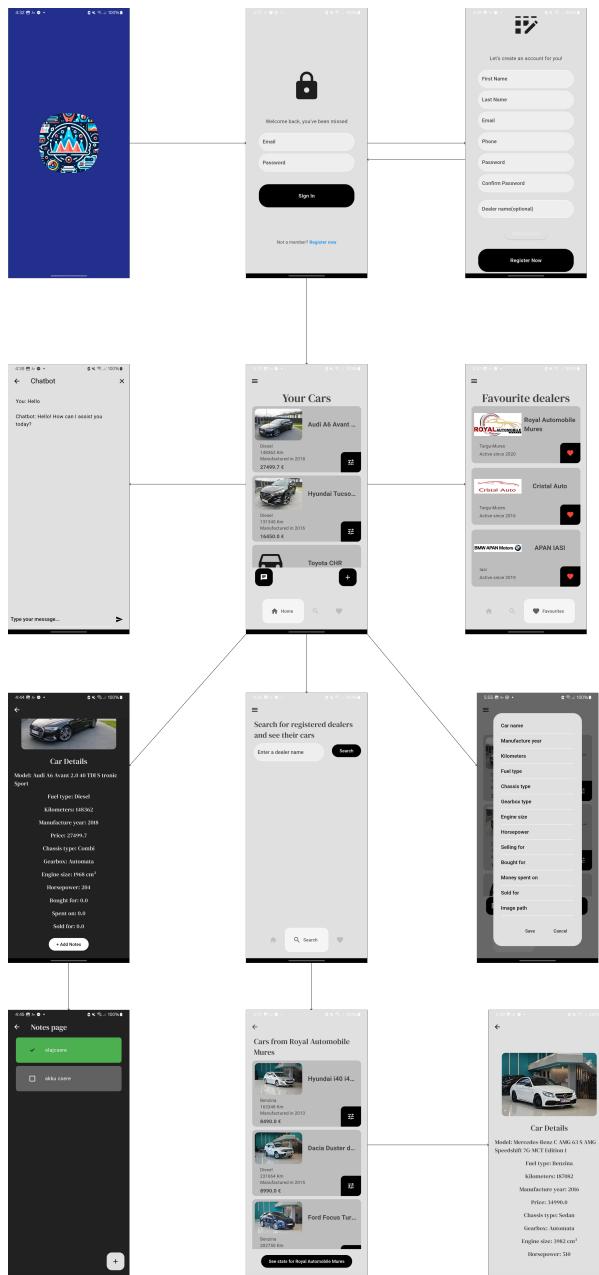
- Pages: a pages mappában találhatóak az alkalmazást alkotó oldalak. A 4.3-es illetve 4.4-as ábrákon láthatóak a screenek és az ezek között megvalósuló navigáció. A navigációt a Flutter Navigator osztályának segítségével könnyen meg tudtuk valósítani. Ez egy stack-ben tárolja el a megnyitott oldalakat, a stack tartalma a push és pop műveletek következtében változik, és ezzel egy időben a megjelenített oldal is.
- Themes: a themes mappában két témát definiáltunk, hogy az alkalmazás kinézete minél testreszabhatóbb legyen. Ez a két téma az alapértelmezett Light mode, ezt le lehet cserélni Dark mode-ra. Ehhez a Flutter Provider osztályát használtuk fel, amiből a számunkra szükséges ThemeProvider osztályt származtattuk. Ez tartalmaz egy függvényt, amely a két téma közötti váltásért felelős. Ezt a függvényt

a settings oldalon található gomb hívja meg és teszi így lehetővé a témák közötti váltást.

- Util: a util mappa tartalmazza az Event osztályhoz tartozó segédfüggvényeket, amelyek szükségesek az event oldalon megjelenített heatmap előkészítéséhez.



**4.3. ábra.** User Interface



**4.4. ábra.** User Interface

### 4.2.2. Szerver oldal

A szerver oldali fejlesztést Python nyelven valósítottuk meg, FastAPI keretrendszer használva. A FastAPI egy modern, gyors keretrendszer, skálázható és könnyen integrálható más rendszerekkel.

Miért választottuk a FastAPI-t?

- Teljesítmény: A FastAPI az egyik leggyorsabb Python keretrendszer, mivel az alapja az ASGI (Asynchronous Server Gateway Interface) standard, amely lehetővé teszi a gyors aszinkron válaszokat.

- Típusellenőrzés és auto-generált dokumentáció: A FastAPI típusellenőrzést végez a bemeneti adatokat illetően, ami jelentősen csökkenti a hibák lehetőségét.
- Széleskörű integráció: A FastAPI könnyen integrálható más rendszerekkel, adatbázisokkal és külső szolgáltatásokkal, ami lehetővé teszi a projekt rugalmasságát.

#### 4.2.3. Adatbázis

Adatbázisként a MySQL-t választottuk, mivel egy hatékony relációs adatbázis kezelő rendszer. Az adatbázisban tároljuk az alkalmazás összes adatát, a felhasználók adatait, kocsik adatait, a felhasználó bejelentkezés eszközeit és adatait és egyéb információkat. Az SQLAlchemy ORM (Object-Relational Mapping) segítségével biztosítjuk az adatbázis és a Python backend közötti kommunikációt, amely megkönnyíti a relációk kezelését és az adatok lekérdezését.

Miért választottuk a MySQL-t?

- Skálázhatóság és megbízhatóság: A MySQL jól skálázható, és megbízhatóan kezeli az adatokat, így ideális választás a projekt számára.

#### 4.2.4. Adatbegyűjtés

A valós adatok elnyerését websracpingel oldottuk meg, amely segítségével strukturált adatokat gyűjtünk a weboldalról. Ez magában foglalja az oldal tartalmának lekérdezését, feldolgozását és tárolását további felhasználás céljából. A webscrapingnél gyorsa adatkinyerést szerettünk volna elérni ezért a Beautiful Soup könyvtárat alkalmaztuk.

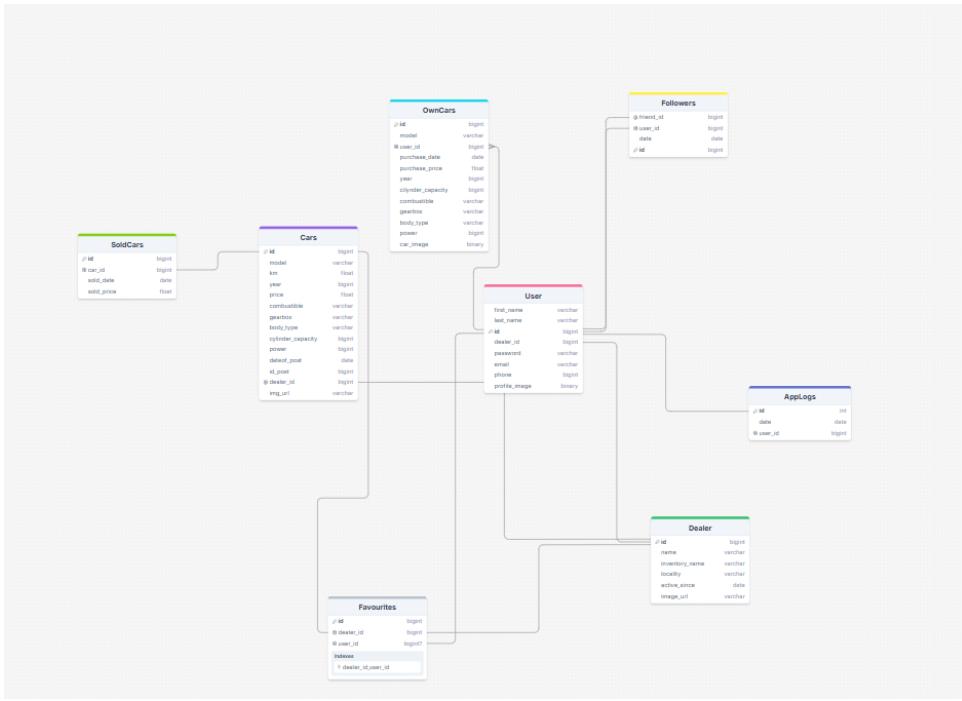
A Beautiful Soup egy népszerű Python könyvtár, amelyet weboldalak HTML és XML tartalmának feldolgozására, elemzésére és adatkinyerésére használnak. Ez a könyvtár egy könnyen használható eszköztárat biztosít a web scraping feladathoz, és egyszerűbbé teszi a weboldal szerkezetének bejárását és az adatok kinyerését.

### 4.3. Osztálydiagramok. Adatbázis terv.

Az adatbázisnál MySQLre esett a választásunk, mivel egy relációs adatbázis kezelő rendszerre volt szükségünk. Az adatbázis táblákat SqlDrawban vizualizáltuk, ami a 4.5-es ábrán látható.

A következő tábláink vannak:

- **Cars:** A feltöltött autók adatai.
- **SoldCars:** Az eladott autók nyilvántartása.
- **OwnCars:** A dealer által birtokolt autók adatai.



4.5. ábra. Adatbázis

- **AppLogs**: Az alkalmazás naplózott eseményei.
- **Favourites**: A felhasználók kedvencnek jelölt autói.
- **User**: A felhasználóról tárolt adatok.
- **Dealer**: A dealerről tárolt adatok.
- **Followers**: A követés eseményében résztvevő felek és adatok eltárolása.

A Cars, Dealer illetve a User táblákban kép tárolására van lehetőség, amelyet URL formájában tárolunk el. A SoldCars táblában tároljuk az eladott autókat, amelyeknek van egy eladási ára és dátuma. Az eladási adatok segítenek a statisztikák készítésében és a pontos statisztikai adatok kiszámolására

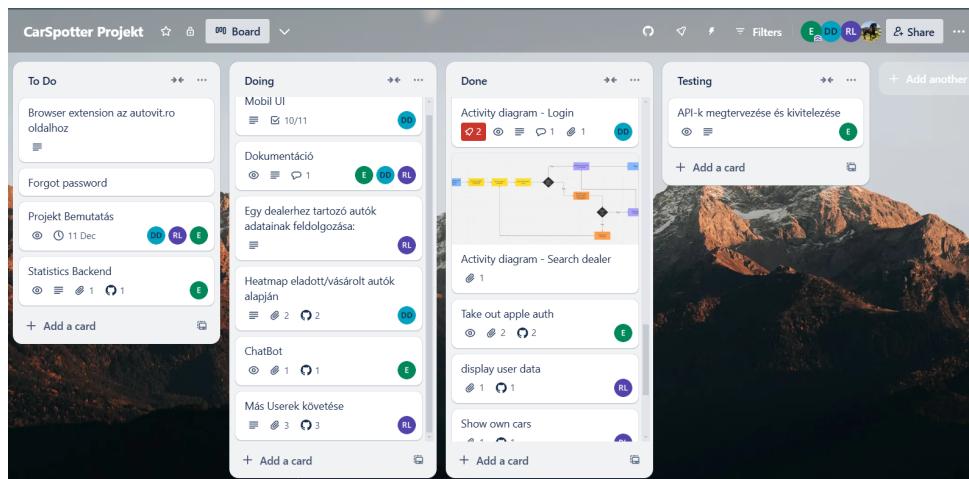
Az OwnCars táblában tárolódnak el a userek a saját autóinak és ennek megvételének adatai: mikor vette és mennyiért, ez ismét segít a statisztika megvalósítására, illetve hogy melyik autót vette meg: carId és hogy melyik felhasználóhoz tartozik: userId.

A favourites táblában tárolódik el, hogy melyik felhasználónak kik a kedvenc dealerei, ugyanis a felhasználónak lehetősége van kedvencként hozzáadni az applikációban az általuk kedvenceknek ítélt dealereket, így nyomon lehet követni az aktivitásukat. A Dealer táblában megjelenik az inventory name, amely elkülöníti a sima felhasználót egy dealer-től, tehát egy szervezetet vagy egy nevet tud megadni, amelyet ő képvisel. Az active since mező pedig mutatja a dealer régiségett, tehát hogy mióta aktivál a weboldalon/applikációban.

## 4.4. Verzió követő és projekt menedzsment

Csapatmunkában fontosnak találtuk, hogy együttműködjünk, megfelelően kommunáljunk egymással és a taskokat vezessük, amit Trello segítségével valósítottunk meg. Nagyon hasznos volt a taskok nyomon követése érdekében, illetve a munka szervezésében.

A To Do oszlopban sorakoztattuk fel a megoldásra váró feladatokat, megbeszéltük, hogy ki mit fog csinálni és mindenki hozzáadta saját magát a taskhoz. A Testing oszlopban azok a taskok kerültek be, amit már elvégeztünk, viszont a jobb verziók és kiemenet érdekében teszteltünk. A Doing oszlopan szerepeltek a jelenlegi taskok, amiknél épp dolgoztunk, és miután befejeztük az egyik taskot és mergeltük a developer branchre, a task átkerült a Done oszlopba. A 4.6-as ábrán látható az általunk menedzselt Trello board.



4.6. ábra. Trello

A verziókövetés és a kód megosztásra GitHubot használtunk. A GitHub repositorynk a 4.7-es ábrán látható. A verzió kezelés kulcsfontosságú volt a projekt fejlesztése során, mivel 3 személy dolgozott a projekten csapatként, ezért egy olyan platformot kellett keressünk, ahol könnyedén tudunk közösen dolgozni.

Két fő branchünk volt: developer és main. Több mellék branchen oldottuk meg a taskokat, amiket mergeltünk miután kész lett, a developer branchre. Példa mellék branchekre: statistics(statisticska oldal kivitelezése, mind a frontend, mind a backend része, heatmap, display-user-data, stb...), ami a 4.8-ös ábrán látható. A fejlesztés során a developer branchre pusholtuk fel a kódjainkat, amit majd a main branchre helyeztünk el, miután a projekt kiteljesedett.

A branchetket issuekkal kapcsoltuk össze, ugyanis mindegyik taskhoz tartozott egy issue, amelynek megvolt a neki megfelelő branchje. Az issueknak igyekeztünk beszédes nevet adni, hogy egyértelmű legyen, hogy mit kéne tartalmazzon a megadott issue. Az issuek listája a 4.9-as ábrán megfigyelhető.

```

diff --git a/src/api/controllers/dealer_controller.py b/src/api/controllers/dealer_controller.pyc
@@ -2,6 +2,7 @@ From fastapi import APIRouter, Depends, HTTPException
@@ -3,2 +3,3 @@ from sqlalchemy.orm import Session
@@ -4,4 +4,5 @@ from aplications.dealer_service import insert_cars_by_dealer
@@ -5,6 +5,7 @@ from db.database import get_db
@@ -6,7 +6,8 @@ router = APIRouter()
@@ -7,8 +7,13 @@ @router.get("/dealer/{dealer_name}/cars")
@@ -8,13 +14,3 @@ def add_cars(dealer_name: str, db: Session = Depends(get_db)):
    raise HTTPException(status_code=400, detail=f"An error occurred: {str(e)}")
except Exception as e:
    raise HTTPException(status_code=400, detail=f"An error occurred: {str(e)}")

@@ -14,3 +14,33 @@ def get_cars_by_dealer(dealer_name: str, db: Session = Depends(get_db)):
    if len(result) == 0:
        raise HTTPException(status_code=404, detail=f"Dealer '{dealer_name}' not found.")
    else:
        cars = db.query(Car).filter_by(dealer_id=result[0].id).all()

@@ -15,15 +35,16 @@ if not cars:
    raise HTTPException(status_code=404, detail=f"No cars found for dealer '{dealer_name}'.")

@@ -16,17 +36,18 @@ return [
@@ -17,18 +37,19 @@     {
@@ -18,19 +38,20 @@         "id": car.id,
@@ -19,19 +39,20 @@         "model": car.model,
@@ -20,19 +40,20 @@         "year": car.year,
@@ -21,19 +41,20 @@     }
@@ -22,19 +42,20 @@ ]

```

## 4.7. ábra. GitHub

Label	Issue Count
statistics	1
Following other Users	1
Heatmap	1
Modifying own car details	1
Adding new own car	1
Take out apple auth	1
Backend	1

## 4.8. ábra. Branches

Label	Issue Count
statistics	1
Following other Users	1
Heatmap	1
Modifying own car details	1
Adding new own car	1
Take out apple auth	1
Backend	1

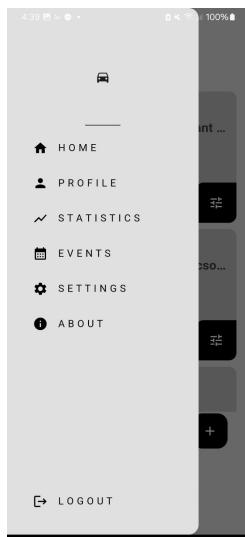
## 4.9. ábra. Issues

## 5. fejezet

# Alkalmazás működése

Az alkalmazásunkban törekedtünk a könnyű, esztétikus felhasználói élmény kialakításában. Intuitív használat volt a célunk. minden funkció könnyen elérhető, a gombok és navigációs elemek jól láthatóak és érthetők. A navigáció egyik legfőbb eleme a sidebar(oldalsáv), amely által navigálni tudunk az alkalmazás különböző részeihez anélkül, hogy az alsó navigációs sávot túlzsúfolnánk.

Az alkalmazásban található sidebar a következő oldalakhoz biztosít közvetlen hozzáférést: home, profile, statisztika, settings, about és az alkalmazásból való kijelentkezés is itt valósul meg.



5.1. ábra. SideBar

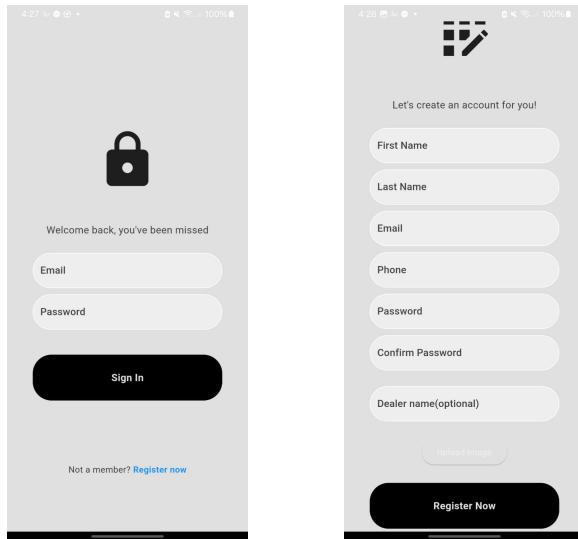
A navigációt a screenek között a 4.3-as és 4.4-es ábrán lehet megtekinteni. A 5.1-es ábrán szemléltettük a sidebarból való elérési utakat, míg a 5.2-es ábrán az autentifikációtól kezdődően a screenek kapcsolódását.

## 5.1. Üdvözlőképernyő (Splash Screen)

Alkalmazás indításakor az első screen ami megjelenik, a Splash Screen. Megjelenik alkalmazásunk logója, ami pár másodperc után autamatikusan átvált az autentifikációhoz.

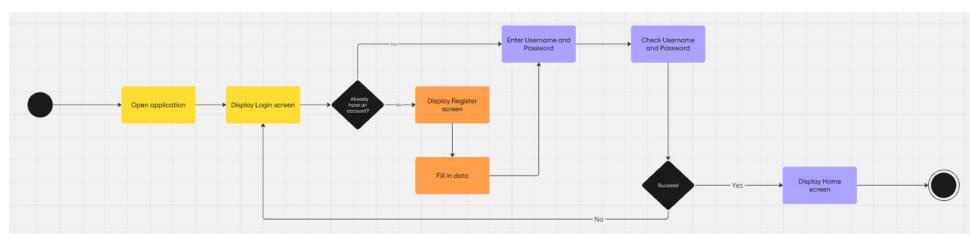
## 5.2. Bejelentkezés. Regisztráció

Az autentifikáció első része: felhasználói fiok létrehozása ahol alapvető adatokat kell megadni: keresztnév, családnév, email, telefonszám, jelszó, jelszó ismétlése, dealer name(opcionális, csak dealerek esetében használandó, ami egy érvényes név kell legyen), illetve képfeltöltésre van lehetőség a felhasználó galériájából. Hogyan a felhasználó már rendelkezik egy felhasználói fiokkal, akkor a be kell jelentkeznie megadván az email címét és jelszavát. A bejelentkezés után a főképernyőre navigál a felhasználó.



5.2. ábra. Authentification

A 5.3-as ábrán látható a bejelentkezés/regisztráció folyamatának diagramja. Új felhasználóknak lehetőségük van regisztrálni, majd megadva az email címüket és jelszavukat be tudnak jelentkezni. Ha valamelyik ezek közül nem helyes, újból meg kell adni. Végül a felhasználó át lesz irányítva a home képernyőre.

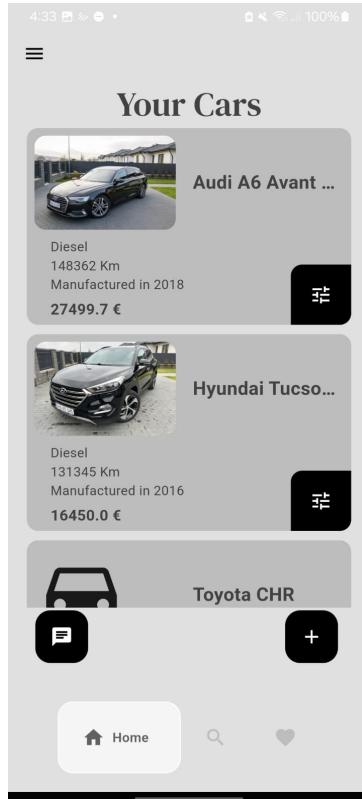


5.3. ábra. Login Activity diagram

### 5.3. Home képernyő

A home képernyőn(5.4-es ábra), amennyiben a user egyben dealer is, megjelennek a felhasználó saját autói. Ezeket tudja módosítani, vagy akár kitörölni. A + gombra nyomva új autót tud hozzáadni a listához, pontosítva ennek tulajdonságait. A bal alsó sarokban helyet kapott egy chatbot gomb, ami a chatbot screenre vezet. Az autót tartalmazó doboz jobb alsó sarkában levő gombra nyomva az adott autót eladttnak minősítheti, ami így átkerül a Sold Cars listába.

A képernyő alján látható a navigation bar is, aminek segítségével a többi oldalra navigálhat a felhasználó.



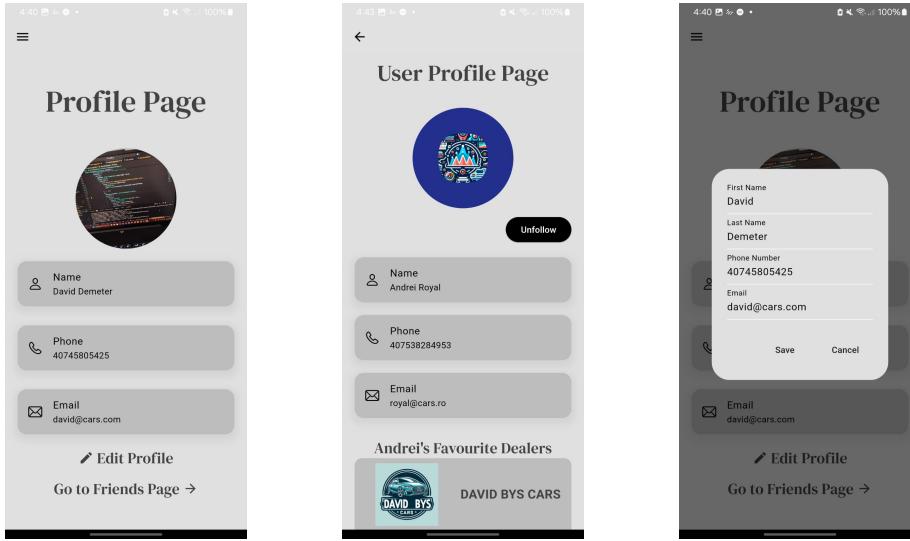
5.4. ábra. Home Screen

### 5.4. Profil képernyő

A profil képernyőn a felhasználó megtekintheti személyes adatait:

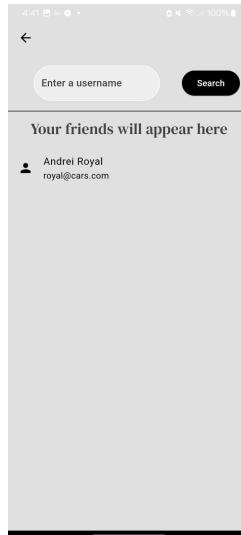
- Név
- Telefonszám
- Email
- Profilkép

Ezeket az adatokat módosítani is tudja. Lehetséges más userek keresése is a friends oldalra navigálva. Itt egy listában megjelennek a felhasználó által követett más felhasználók. Mindezek profiljait részletesebben is megtekintheti a friend oldalon. Itt láthatóvá válnak az adott felhasználó által követett dealerek is, így a bejelentkezett felhasználó megismerhet új dealereket is, akiket hozzá is adhat a kedvencekhez. A profil adatokat tartalmazó képernyők az 5.5-ös ábrán láthatóak, míg a userek keresése a 5.6-os ábrán történik.



5.5. ábra. Profile Screens

## 5.5. Friends (Követett felhasználók) képernyő

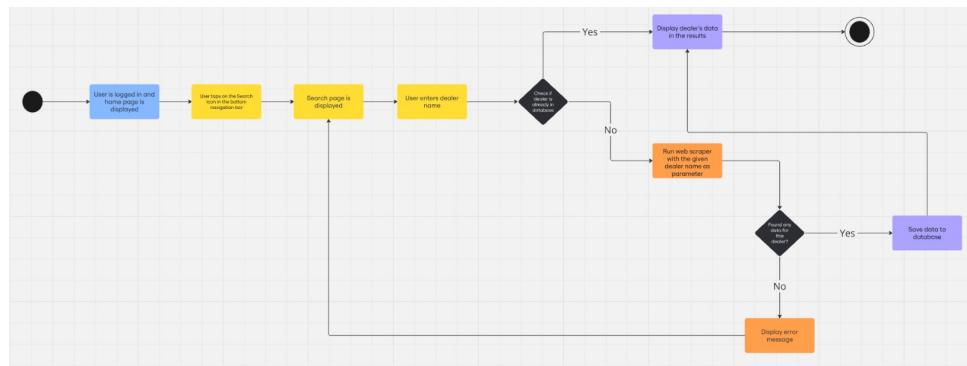


5.6. ábra. Friend Screen

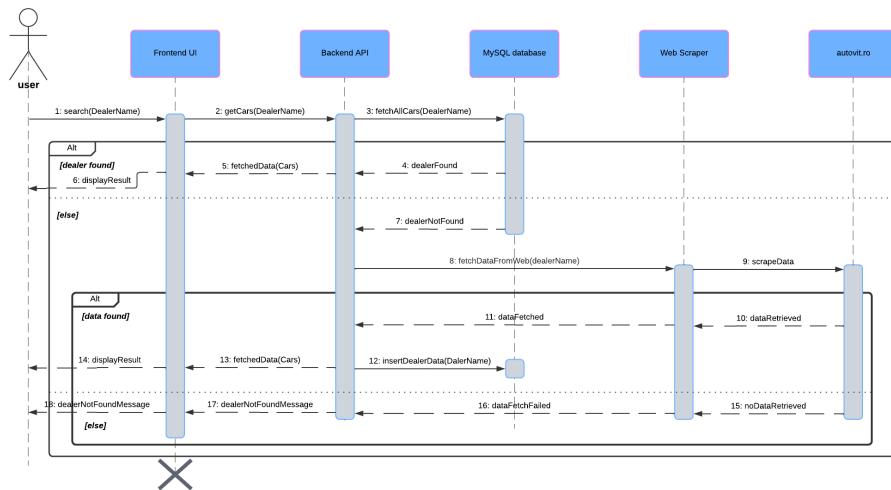
Hogyha a felhasználó követ más felhasználókat, akkor ezek a Friend Screenben jelennek meg(lásd 5.6). Ezen a screenen is lehetőség van keresni felhasználó után név szerint.

## 5.6. Keresés (Search) képernyő

Fontos funkcionálitást valósít meg a search képernyő, ahol a felhasználó dealereket tud keresni név szerint. Amennyiben egy az autovit.ro oldalon is regisztrált dealer, megjelenik eredményként egy doboz a dealer fontosabb adataival: név, mióta aktív és helység. Erre nyomva a felhasználó megtekintheti ennek a dealernek az autót, illetve a szív ikonra nyomva a dealer bekerül a kedvencek közé, ahol később is látható marad. Így a felhasználó láthatja ha új autó jelenik meg vagy ha elad autót a dealer. A keresés folyamatát a 5.7-es és 5.8-as ábrák írják le részletesen.



5.7. ábra. Search Activity diagram



5.8. ábra. Sequence diagram

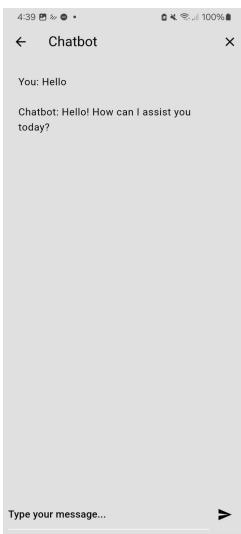
## 5.7. Kedvencek (Favourite) képernyő

Tartalmazza a bejelentkezett felhasználó által kedvencnek megjelölt dealereket, ahogy az a 5.9-es ábrán is látható. Minden dealer doboz tartalmaz egy listát a dealerhez tartozó autókkal, illetve minden autó dobozhoz tartozik egy details screen további részletekkel.



5.9. ábra. Favourites Screen

## 5.8. Chatbot



5.10. ábra. ChatBot Screen

A home screenen a chat gombra kattintva lehet előhívni a chatbot screenet, ahol a felhasználó válaszokat kaphat a felmerülő kérdéseire, illetve tanácsokat az autó vásárlással vagy eladással kapcsolatban, amint a 5.10-es ábrán látható. Általános információkat is kérdezhet tőle.

A ChatBot OpenAI által biztosított API-t használ a funkciók implementálására. A megírt osztály alapvetően arra szolgál, hogy kapcsolatot létesítsen az OpenAI GPT-3.5-Turbo modelljével, és képes legyen válaszokat generálni a felhasználó által küldött szöveges bemenetekre. Az osztály tartalmaz egy privát apiKey változót, amely az OpenAI API-hitelesítéshez szükséges kulcsot tárolja.

A ChatbotService osztály a HTTP POST kéréseken keresztül kommunikál az OpenAI API-val. A következőképpen valósul meg:

- Bemenetként megkapja a felhasználó által küldött szöveget.
- Kérést indít az API-hoz, amely tartalmazza az üzenetet és a modell nevét.
- Választ fogad, majd kinyeri az OpenAI által generált szöveget.
- Visszaadja a feldolgozott választ a hívó félnek.
- A metódus figyelembe veszi az esetleges hibákat, és gondoskodik azok kezeléséről, például ha a szerver válasza sikertelen vagy formailag hibás.

## **6. fejezet**

# **Összefoglaló**

A CarSpotter alkalmazás egy hatékony megoldást kínál a felhasználóknak az <https://www.autovit.ro/> oldalhoz való adatok hozzáféréséhez, akik eladni, venni szeretnének autókat, akik keresni szeretnének egy adott eladó elérhető autói között vagy statisztikát nézni a saját profitjaikról, eladásaikról. Hatékony, mivel az autópiaci adatokat könnyen tudja elérni és elemezni. Ezen kívül az alkalmazás célja, hogy segítse a kereskedőket is a saját autóik nyomon követésében és elemzésében. Megkönnyíti az információ szerzést, elemzést a felhasználók számára.

### **6.1. További fejlesztési lehetőségek**

- Automatizált ajánlások. Személyre szabott ajánlások a felhasználó keresései, kedvencei és követett felhasználóknak az érdeklődési körei alapján.
- Többnyelvűség, tehát az alkalmazás használása más nyelven is, mint a jelenlegi angol nyelv.
- Integráció közösségi média platformokkal. A felhasználók megoszthatnák kedvenc hirdetéseiket, vagy kapcsolatot létesíthetnek más érdeklődőkkel.
- Lehetőség a dealerek mellett autók hozzáadására is a kedvencekhez

# Ábrák jegyzéke

3.1. Use case diagram . . . . .	5
4.1. Komponens diagram . . . . .	8
4.2. Folder Structure . . . . .	10
4.3. User Interface . . . . .	11
4.4. User Interface . . . . .	12
4.5. Adatbázis . . . . .	14
4.6. Trello . . . . .	15
4.7. GitHub . . . . .	16
4.8. Branches . . . . .	16
4.9. Issues . . . . .	16
5.1. SideBar . . . . .	17
5.2. Authentification . . . . .	18
5.3. Login Activity diagram . . . . .	18
5.4. Home Screen . . . . .	19
5.5. Profile Screens . . . . .	20
5.6. Friend Screen . . . . .	20
5.7. Search Activity diagram . . . . .	21
5.8. Sequence diagram . . . . .	21
5.9. Favourites Screen . . . . .	22
5.10. ChatBot Screen . . . . .	22

# Irodalomjegyzék

- [1] Python Documentation. [Online]. Available: <https://docs.python.org/3/>. Accessed: 2024.
- [2] MySQL Documentation. [Online]. Available: <https://dev.mysql.com/doc/>. Accessed: 2024.
- [3] Flutter Documentation. [Online]. Available: <https://docs.flutter.dev/>. Accessed: 2024.
- [4] Dart Language Tour. [Online]. Available: <https://dart.dev/guides/language/language-tour>. Accessed: 2024.
- [5] Python GPT ChatBot Tutorial. [Online]. Available: <https://www.youtube.com/watch?v=q5HiD5PNuck>. Accessed: 2024.
- [6] BeautifulSoup Documentation. [Online]. Available: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>. Accessed: 2024.
- [7] Flask Documentation. [Online]. Available: <https://flask.palletsprojects.com/>. Accessed: 2024.
- [8] Python GPT ChatBot Tutorial. [Online]. Available: [https://www.youtube.com/watch?v=q5HiD5PNuck&ab\\_channel=TechWithTim](https://www.youtube.com/watch?v=q5HiD5PNuck&ab_channel=TechWithTim). Accessed: 2024.