

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



SAPIENTIA
ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM

Path Planner: Városi útvonaltervező alkalmazás a
tömegközlekedés optimalizálásához

DIPLOMADOLGOZAT

Témavezető:
Dr. Antal Margit,
Egyetemi docens

Végzős hallgató:
Kacsó Róbert

2023

UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,
SPECIALIZAREA INFORMATICĂ



UNIVERSITATEA
SAPIENTIA

Path Planner: Aplicație pentru planificarea rutelor urbane în
vederea optimizării transportului public

LUCRARE DE DIPLOMĂ

Coordonator științific:
Dr. Antal Margit,
Conferențiar universitar

Absolvent:
Kacsó Róbert

2023

**SAPIENTIA HUNGARIAN UNIVERSITY OF
TRANSYLVANIA
FACULTY OF TECHNICAL AND HUMAN SCIENCES
COMPUTER SCIENCE SPECIALIZATION**



SAPIENTIA
HUNGARIAN UNIVERSITY
OF TRANSYLVANIA

Path Planner: Urban route planner app for public transport
optimization

BACHELOR THESIS

Scientific advisor:
Dr. Antal Margit,
Associate Professor

Student:
Kacsó Róbert

2023

UNIVERSITATEA „SAPIENTIA” din CLUJ-NAPOCA
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș
Programul de studii: Informatică

Viza facultății:

LUCRARE DE DIPLOMĂ

Coordonator științific:
Conf. dr. ANTAL Margit

Candidat: **Kacsó Róbert**
Anul absolvirii: **2023**

a) Tema lucrării de licență:

Path Planner: Aplicație pentru planificarea rutelor urbane în vederea optimizării transportului public

b) Problemele principale tratate:

- Studiu bibliografic privind sistemele REST API și aplicații Android
- Realizarea unei aplicații pentru planificarea rutelor urbane în vederea optimizării transportului public

c) Desene obligatorii:

- Schema bloc a aplicației
- Diagrame de proiectare pentru aplicația software realizată.

d) Softuri obligatorii:

- REST Api implementat în Java utilizând framework-ul Spring Boot
- testarea aplicației: teste unit și de integrare
- aplicație Android pentru planificarea rutelor urbane

e) Bibliografia recomandată:

- <https://spring.io/>
- <https://developer.android.com/docs>
- <https://spring.io/projects/spring-security>

f) Termene obligatorii de consultații:

g) Locul și durata practicii: Universitatea „Sapientia” din Cluj-Napoca,
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș, sala / laboratorul 417

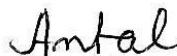
Primit tema la data de: 01.06.2022

Termen de predare: 02.07.2023

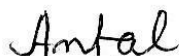
Semnătura Director Departament



Semnătura coordonatorului



Semnătura responsabilului
programului de studiu



Semnătura candidatului



Declarație

Subsemnatul/a KACSO' RO'BERT, absolvent(ă) al/a specializării INFORMATICA, promoția 2023 cunoscând prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a Universității Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licență/proiect de diplomă/disertație se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea, TÂRGU MUREȘ
Data: 14.06.2023

Absolvent

Semnătura.....Kacso'.....

Kivonat

A városi tömegközlekedés mindennapjaink elengedhetetlen része, de gyakran szembesülünk kihívásokkal és problémákkal az útvonaltervezés során. A városok növekedése és a lakosság mobilitásának növekedése olyan problémákat vet fel, mint a közlekedési torlódások, a hosszú utazási idő és a kényelmetlenség. Az útvonalak tervezése és optimalizálása kiemelten fontos a hatékony és fenntartható városi közlekedés biztosításához.

A Path Planner nevű alkalmazás ezen kihívásokra kínál választ. Célja, hogy segítsen a felhasználóknak hatékonyan tervezni és optimalizálni a városi útvonalakat, legyen szó buszútvonalakról vagy bicikliútakról. Az alkalmazás működése során a felhasználók egyszerűen megadhatják az indulási és célállomást, valamint választhatnak a rendelkezésre álló közlekedési eszközök közül.

Az útvonalakat térképi formában jeleníti meg, így könnyedén követhetőek és vizualizálhatóak. Az alkalmazás különlegessége abban rejlik, hogy a felhasználók aktív szerepet játszhatnak az útvonaltervezésben. Lehetőségük van ajánlásokat tenni új buszútvonalakra vagy bicikliutakra, amelyeket az alkalmazás feldolgoz.

Az alkalmazás egyszerűen használható felülettel rendelkezik, amely lehetővé teszi a felhasználók számára az intuitív és hatékony használatot. Az optimalizált útvonalak segítségével a felhasználók időt, energiát és pénzt takaríthatnak meg, miközben hozzájárulnak a fenntarthatóbb városi közlekedés kialakításához.

Kulcsszavak: városi tömegközlekedés, útvonaltervezés, klaszterezés.

Rezumat

Transportul public urban este o parte indispensabilă a vieții noastre de zi cu zi, însă adesea ne confruntăm cu provocări și probleme în procesul de planificare a traseelor. Creșterea orașelor și a mobilității populației ridică probleme precum blocajele de trafic, timpul lung de călătorie și disconfortul. Planificarea și optimizarea traseelor sunt extrem de importante pentru asigurarea unui transport urban eficient și durabil.

Aplicația numită Path Planner oferă o soluție pentru aceste provocări. Scopul său este de a ajuta utilizatorii să planifice și să optimizeze eficient traseele urbane, fie că este vorba despre linii de autobuz sau piste de biciclete. În funcționarea aplicației, utilizatorii pot introduce cu ușurință punctul de plecare și destinația, precum și să aleagă din mijloacele de transport disponibile.

Traseele sunt afișate pe hartă, facilitând urmărirea și vizualizarea acestora. Unicitatea aplicației constă în faptul că utilizatorii pot juca un rol activ în planificarea traseelor. Au posibilitatea de a face recomandări pentru noi linii de autobuz sau piste de biciclete, care sunt procesate de aplicație.

Aplicația are o interfață ușor de utilizat, care permite utilizatorilor să o folosească în mod intuitiv și eficient. Prin intermediul traseelor optime, utilizatorii pot economisi timp, energie și bani, contribuind în același timp la dezvoltarea unui transport urban mai durabil.

Cuvinte de cheie: transport public urban, planificarea rutelor, algoritmi de grupare.

Abstract

Urban public transportation is an essential part of our daily lives, but we often face challenges and problems during route planning. The growth of cities and the increasing mobility of the population raise issues such as traffic congestion, long travel times, and discomfort. Planning and optimizing routes are crucial for ensuring efficient and sustainable urban transportation.

The Path Planner application offers a solution to these challenges. Its purpose is to assist users in efficiently planning and optimizing urban routes, whether it's bus routes or bike paths. Users have the ability to simply input their starting and destination points and choose from available transportation options.

The application displays the routes on a map, making them easy to follow and visualize. What sets this app apart is that users can actively participate in route planning. They have the ability to make recommendations for new bus routes or bike paths, which are then processed by the application.

The application features a user-friendly interface that allows for intuitive and efficient usage. With the help of optimized routes, users can save time, energy, and money while contributing to the development of a more sustainable urban transportation system.

Keywords: urban public transportation, route planning, clustering.

Tartalomjegyzék

1. Bevezető	11
2. Célkitűzések	12
3. Elméleti megalapozás és szakirodalmi tanulmány	13
3.1. Az alkalmazás esettanulmánya	13
3.2. Hasonló szolgáltatások	14
3.3. Technológiák ismertetése	14
3.3.1. Java	14
3.3.2. Spring Boot	14
3.3.2.1. Spring Boot alkalmazás felépítése	15
3.3.2.2. Komponensek	16
3.3.3. TypeScript	16
3.3.4. Angular	17
3.3.4.1. Angular alkalmazások felépítése	17
3.3.4.2. Angular könyvtárak	18
3.3.5. Kotlin	18
3.3.6. Android	19
3.3.6.1. Android alkalmazás felépítése	19
3.3.6.2. Android könyvtárak	20
3.3.7. Gradle	21
4. A rendszer specifikációja	22
4.1. Felhasználói követelmények	22
4.1.1. Fontosabb használati esetek	24
4.2. Rendszerkövetelmények	26
4.2.1. Funkcionális követelmények	26
4.2.2. Nem funkcionális követelmények	27
5. Tervezés	29
5.1. A rendszer architektúrája	29
5.2. Szerveroldal	30
5.3. Webkliens	30
5.4. Mobilkliens	31
5.5. Felhasználói felület tervezése	32
6. Kivitelezés	34

6.1.	Szerveroldali megvalósítás	34
6.1.1.	Adat és repository réteg	34
6.1.2.	Service réteg	35
6.1.3.	Controller réteg	38
6.1.4.	Ajánlások feldolgozása	38
6.2.	Webkliens megvalósítása	40
6.2.1.	Routing	41
6.2.2.	Data Binding	41
6.2.3.	Szerverrel való kommunikáció	41
6.3.	Mobilkliens megvalósítása	41
6.3.1.	Állapotmenedzsment	41
6.3.2.	Navigáció	42
6.3.3.	Szerver oldallal való kommunikáció	44
6.3.4.	Korutinok	45
6.3.5.	Útvonaltervezés	45
6.4.	Az alkalmazás működése	46
6.4.1.	Kezdőoldal	46
6.4.2.	Új felhasználói ajánlás hozzáadása	46
6.4.3.	Felhasználói ajánlások megtekintése	49
6.4.4.	Létező és aggregált útvonalak listázása	49
6.4.5.	Profil	50
7.	Eszközök és munkamódszerek	52
7.1.	Segédeszközök	52
7.1.1.	Integrált fejlesztői környezetek	52
7.1.1.1.	Szerveroldal és webkliens fejlesztése	52
7.1.1.2.	Mobil alkalmazás fejlesztése	52
7.1.2.	Verziókövetés eszközök	52
7.1.3.	API tesztelés	53
7.1.4.	Design eszközök	53
7.2.	A szerveroldal tesztelése	54
	Összefoglaló	57
7.3.	Következtetések	57
7.4.	Továbbfejlesztési lehetőségek	57
	Köszönetnyilvánítás	58
	Ábrák jegyzéke	59
	Irodalomjegyzék	61

1. fejezet

Bevezető

Ebben a fejezetben megfogalmazom a dolgozatom témáját, és tárgyalom azokat a problémákat, amelyek ötletet adtak a téma választásához.

A városi tömegközlekedés mindennapjaink elengedhetetlen része, de gyakran szembesülünk kihívásokkal és problémákkal az útvonaltervezés során. A városok növekedése és a lakosság mobilitásának növekedése olyan problémákat vet fel, mint a közlekedési torlódások, a hosszú utazási idő és a kényelmetlenség. Az útvonalak tervezése és optimalizálása kiemelten fontos a hatékony és fenntartható városi közlekedés biztosításához. Az útvonalak megfelelő tervezése lehetővé teszi a tömegközlekedési eszközök hatékony kihasználását, csökkenti a közlekedési torlódásokat, minimalizálja az utazási időt és javítja a közlekedési tapasztalatot.

A Path Planner nevű alkalmazás ezen kihívásokra kínál választ. Célja, hogy segítsen a felhasználóknak hatékonyan tervezni és optimalizálni a városi útvonalakat, legyen szó buszútvonalakról vagy bicikliútról. Az alkalmazás működése során a felhasználók egyszerűen megadhatják az indulási és célállomást, valamint választhatnak a rendelkezésre álló közlekedési eszközök közül. Az útvonalakat térképi formában jeleníti meg, így könnyedén követhetők és vizualizálhatóak. Az alkalmazás különlegessége abban rejlik, hogy a felhasználók aktív szerepet játszhatnak az útvonaltervezésben. Lehetőségük van ajánlásokat tenni új buszútvonalakról vagy bicikliútról, amelyeket az alkalmazás feldolgoz. Ezáltal javítja az útvonalak hatékonyságát és időtakarékoságát. Az alkalmazás egyszerűen használható felülettel rendelkezik, amely lehetővé teszi a felhasználók számára az intuitív és hatékony használatot. Az optimalizált útvonalak segítségével a felhasználók időt, energiát és pénzt takaríthatnak meg, miközben hozzájárulnak a fenntarthatóbb városi közlekedés kialakításához.

A következőkben bemutatom a további fejezeteket röviden. A bevezető után a célkitűzések rögzítése következik, amelyeket az alkalmazás megvalósításakor figyelembe vettem. Ezután részletesen ismertetem az alkalmazás koncepcióját és a technológiákat, amiket a fejlesztés során használtam. A következő fejezet a rendszer specifikációról szól, ahol részletesen leírom az alkalmazás funkcionális és nem funkcionális követelményeit. Ezután a tervezési részletekre és az alkalmazás architektúrájára összpontosítok a hatodik fejezetben. A következő fejezet a kivitelezést, és az alkalmazás megvalósítását részletezi. Végül, a dolgozat összefoglaló részében áttekintem a főbb eredményeket és megfogalmazom a továbbfejlesztési lehetőségeket.

2. fejezet

Célkitűzések

Az alkalmazás fejlesztése a Codespring szakmai gyakorlata során kezdődött, az IT Plus Cluster Mentorprogram keretén belül. Az alkalmazás fejlesztésén közösen dolgoztam Imecs Bulcsúval, a Babeş-Bolyai Tudományegyetem informatika szakos hallgatójával és Barabás Csabával, a Babeş-Bolyai Tudományegyetem informatika és szoftverfejlesztés posztgraduális képzési programjának résztvevőjével. A szakmai gyakorlat során elkészült a szerveroldal és webkliens. A webalkalmazáson keresztül a felhasználóknak lehetősége van ajánlások hozzáadására, a már létező útvonalak megtekintésére, ezeket közötti szűrésre és a saját ajánlásaik listázására. Továbbá, elérhető a City Maintainer számára az aggregálási művelet elvégzése, az adminisztrátornak, pedig az új város hozzáadása, és a City Maintainer kinevezése.

A projekt fejlesztése során két fő célkitűzésem volt:

- a szerveroldal tesztelésének megvalósítása egység-, integrációs és végpont-tesztek alkalmazásával.
- mobilkliens fejlesztése a projekthez a következő funkcionálisokkal:
 - regisztrálás/bejelentkezés Google fiókkal,
 - új ajánlás hozzáadása,
 - tartózkodási hely használata ajánlás megadásánál,
 - előző ajánlások listázása,
 - létező/aggregált útvonalak megtekintése,
 - profil megtekintése/szerkesztése,
 - alapértelmezett város beállítása.

3. fejezet

Elméleti megalapozás és szakirodalmi tanulmány

3.1. Az alkalmazás esettanulmánya

Az alkalmazás célja, hogy támogatást nyújtson kisebb városok tömegközlekedési úthálózatának tervezésében és optimalizálásában. Az elsődleges funkcionalitás az, hogy a felhasználók ajánlásokat tudjanak tenni buszútvonalak és bicikliutak tekintetében. Az ajánlások beérkezése után ezeket feldolgozzuk, hogy értékes információkat nyerjünk a városi közlekedési igényekről és preferenciákról.

Az ajánlások feldolgozása egy klaszterező algoritmus segítségével történik. Az algoritmus összekapcsolja az egymáshoz közeli kezdő- és végpontokat, ezáltal kialakítva egy közös pontot, amely a különböző ajánlásokból származó útvonalak kiinduló- vagy végpontja lehet. Ez az aggregálási folyamat hozzájárul a hatékonyabb útvonaltervezéshez és megakadályozza az induló vagy végállomások túlzott közelségét egymáshoz képest. Az aggregálási folyamatot a City Maintainer szerepkörrel rendelkező felhasználó végzi el, aki felelős a városi közlekedési útvonalak kezeléséért. A City Maintainer személyre szabhatja az aggregálás paramétereit, például az összekapcsolásra vonatkozó távolsági küszöbértéket, hogy a lehető legjobb eredményt érje el a városi közlekedés optimalizálása terén.

Az alkalmazásban a felhasználók az ajánlásaikat követhetik nyomon, valamint megtekinthetik a városonként létrejött aggregált útvonalakat és a már meglévő útvonalakat. Ez lehetővé teszi a városlakók számára, hogy aktívan részt vegyenek a közlekedési útvonalak fejlesztésében és optimalizálásában, valamint segít a városi közlekedési infrastruktúra folyamatos fejlesztésében. Az alkalmazás adminisztrációs része az Adminisztrátor szerepkörrel rendelkező felhasználó számára elérhető. Az Adminisztrátor jogosultságokkal rendelkezik, hogy új városokat adjon hozzá az alkalmazáshoz, módosítsa a városok adatait és kijelölje a City Maintainer szerepkörű felhasználót egy városhoz. Ezáltal az Adminisztrátor gondoskodik az alkalmazás városi adatbázisának karbantartásáról és frissítéséről.

Az alkalmazásnak mobil és webes verziója is elérhető. A mobil alkalmazás kizárólag a felhasználói funkciókat tartalmazza, amelyek lehetővé teszik az ajánlások beküldését, az ajánlások követését és az útvonalak megtekintését. A webes alkalmazás szélesebb körű funkcionalitást nyújt, mivel a City Maintainer és Adminisztrátor szerepkörrel rendelkező felhasználók is használhatják. A webes alkalmazásban a City Maintainer beállíthatják

az aggregálási paramétereket és kezelhetik a városi útvonalakat, míg az Adminisztrátorok felelősek az alkalmazás városi adatbázisának karbantartásáért.

3.2. Hasonló szolgáltatások

Az alkalmazás koncepciója olyan egyedi megközelítést alkalmaz, ahol a felhasználók aktív résztvevőként vesznek részt a városi közlekedési úthálózat tervezésében és optimalizálásában. Az ötlet alapján nem találtam olyan alkalmazást, amely ilyen módon kombinálná a felhasználói ajánlásokat, az aggregálási algoritmusokat és a városi közlekedési útvonalak kezelését.

3.3. Technológiák ismertetése

Ebben a részben részletesen bemutatom a projekt implementációjához használt kulcsfontosságú technológiákat.

3.3.1. Java

A Java [1] platformfüggetlen nyelv, amely lehetővé teszi a programok egyszeri írását és futtatását különböző operációs rendszereken és hardverplatformokon. A Java programokat bájtkód formátumra fordítják, amelyet azután a Java virtuális gép (JVM) interpretál. Objektumorientált paradigmára épül, ami azt jelenti, hogy a programokat osztályok és objektumok hierarchiájával írják le. Ez segíti az alkalmazások moduláris szerkezetét, újrafelhasználhatóságát és karbantarthatóságát.

A Java nyelv beépített mechanizmusokkal rendelkezik a hibák felderítésére és a programok stabil futtatására. A Java nyelv és a JVM környezet számos biztonsági funkciót kínál, amelyek segítenek megelőzni a sérülékenységeket és a rosszindulatú kód futtatását. A Java nyelv és a JVM optimalizálási technikái lehetővé teszik a nagy teljesítményű és skálázható alkalmazások fejlesztését. A nyelv támogatja az egyszerűbb feladatoktól a nagyvállalati szintű rendszerekig terjedő alkalmazások fejlesztését.

A Java nyelv számos beépített osztályt és könyvtárat tartalmaz, amelyek különböző feladatok elvégzését segítik, például adatbázis-kezelés, hálózati kommunikáció, grafikus felhasználói felületek és sok más. Emellett a Java platform támogatja a sokoldalú fejlesztést, mint például webes alkalmazások, mobilalkalmazások, asztali alkalmazások és nagy adatok feldolgozása.

3.3.2. Spring Boot

Spring Boot [2] egy nyílt forráskódú keretrendszer a Java alkalmazások gyors és egyszerű fejlesztésére [3]. A Spring Boot célja az alkalmazások gyors indítása és konfigurálása, minimalizálva a fejlesztői erőfeszítéseket és az ismétlődő munkát. A Spring Boot a Spring keretrendszerre épül, amely a Java alapú alkalmazások fejlesztését támogatja. Azonban a Spring Boot tovább lépve a hagyományos Spring keretrendszeren, előre beállított konvenciókat és automatikus konfigurációt kínál, ami jelentősen egyszerűsíti az alkalmazásfejlesztést.

A Spring Boot lehetővé teszi a gyors alkalmazásfejlesztést azáltal, hogy az alapértelmezett beállításokat és könyvtárakat biztosítja az alkalmazások számára. Az alkalmazásoknak kevesebb konfigurációs kódot kell írniuk, mivel a Spring Boot intelligensen felismeri a használt technológiákat és automatikusan konfigurálja azokat. A Spring Boot különböző funkciókat és modulokat kínál, amelyek elősegítik a webalkalmazások, adatbázis-kapcsolatok, biztonság, naplózás és más funkciók fejlesztését. Emellett a Spring Boot integrálódik más népszerű keretrendszerekkel és technológiákkal, például a Spring Data, Spring Security, Hibernate, JPA stb. A Spring Boot a fejlesztőknek nagy rugalmasságot és kontrollt nyújt az alkalmazások futtatásához és telepítéséhez. Az alkalmazások függetlenek a futási környezettől, és különböző platformokon és konténerekben futtathatók, beleértve a beágyazott szervereket, a független konténereket és a felhőalapú platformokat is.

Ezenkívül a Spring Boot rendelkezik egy nagy, aktív közösséggel és széles körű dokumentációval. A közösség folyamatosan támogatja és fejleszti a keretrendszert, és számos erőforrást, útmutatót és példaprogramot biztosít az alkalmazásfejlesztők számára.

3.3.2.1. Spring Boot alkalmazás felépítése

Egy Spring Boot alkalmazás több rétegből és komponensből épül fel [4]. A következőkben ismertetem ezeket a fő komponenseket és rétegeket:

- **Controller:** Ez a réteg fogadja a HTTP kéréseket és kezeli a bejövő kérések útvonalazását. A kontrollerek osztályok, amelyek az *@Controller* vagy *@RestController* annotációval vannak ellátva, és a bejövő kérések kezelésére szolgáló végpontokat definiálják.
- **Service:** A service réteg az üzleti logika megvalósításáért felelős. Ez a réteg kezeli az adatok feldolgozását és az alkalmazás üzleti szabályainak végrehajtását. A service réteg osztályokat tartalmaz, amelyek az *@Service* annotációval vannak ellátva.
- **Repository:** A repository réteg az adatelérésért és az adatbázis-kezelésért felelős. Ez a réteg interfészeket és osztályokat tartalmaz, amelyek lehetővé teszik az adatbázis műveletek elvégzését, például adatok lekérdezését, beszúrását, frissítését vagy törlését. A repository réteg általában az *@Repository* annotációval ellátott interfészeket használja.
- **Model:** A model réteg az alkalmazás belső adatmodelljét reprezentálja. Ez a réteg tartalmazza az adatmodelleket, entitásokat vagy DTO-kat, amelyek a közvetítők adatait képviselik.
- **Configuration:** A konfigurációs rétegben a különböző beállítások és konfigurációk találhatók. Ez a réteg lehetővé teszi az alkalmazás számára, hogy beállítsa a környezeti változókat, adatbázis-kapcsolatokat, biztonsági beállításokat és más konfigurációs információkat. A konfigurációs rétegben osztályok találhatók, amelyeket gyakran az *@Configuration* annotációval látnak el.

Ezenkívül a Spring alkalmazás több más komponenst és funkciót is tartalmazhat, például komponenseket a biztonság, az adatbázis-kezelés, a naplózás, az ütemezés és más területeken. Az alkalmazás függ a használt moduloktól és a projekt konkrét igényeitől.

A Spring keretrendszer és az alkalmazások építésének alapja az invertált irányítás (Inversion of Control - IoC) és a függőségi befecskendezés (Dependency Injection - DI) elve. Ezek az elvek segítenek a rugalmas és könnyen karbantartható alkalmazások készítésében, valamint a kód újrafelhasználásában és a könnyű tesztelhetőségben.

3.3.2.2. Komponensek

A Spring Boot rengeteg beépített komponenst és modult tartalmaz, amelyek segítenek az alkalmazásfejlesztésben és számos funkciót biztosítanak. Néhány fontosabb beépített komponens és modul a következő:

- **Auto-configuration:** A Spring Boot automatikusan konfigurálja [5] az alkalmazást alapértelmezett beállításokkal és függőségekkel. Ez csökkenti a konfigurációs feladatokat és megkönnyíti az alkalmazás indulását.
- **Spring Boot Starter:** A Spring Boot Starterek gyűjteménye előre definiált függőségek [6], amelyek segítségével könnyen hozzáadhatók az alkalmazáshoz. Például a spring-boot-starter-web tartalmazza az összes szükséges függőséget a webalkalmazás fejlesztéséhez.
- **Spring Boot Actuator:** Az Actuator lehetővé teszi az alkalmazás monitorozását és üzembe helyezését [7]. Segítségével megfigyelhetjük az alkalmazás egészségi állapotát, a végpontok állapotát, a metrikákat, naplókat és egyéb információkat
- **Spring Data:** A Spring Boot beépített támogatást nyújt a Spring Data modulokhoz, amelyek segítségével egyszerűen és hatékonyan kezelhetjük az adatelérést [8]. Például a Spring Data JPA lehetővé teszi az objektum-relációs leképzést (ORM) a JPA (Java Persistence API) használatával.
- **Spring Security:** A Spring Boot beépített támogatást tartalmaz a biztonsági funkciókhoz a Spring Security modul segítségével [9]. Ez lehetővé teszi az autentikációt, autorizációt és egyéb biztonsági mechanizmusokat az alkalmazásban.
- **Spring Boot Test:** A Spring Boot beépített tesztelési támogatást nyújt a JUnit és más keretrendszerekkel [10]. Ez lehetővé teszi a könnyű és hatékony tesztek írását az alkalmazás különböző rétegeire.

3.3.3. TypeScript

A TypeScript [11] a JavaScript nyelv kiterjesztése, és lehetővé teszi a statikus típusellenőrzést, erőteljesebb típusrendszert, valamint modern nyelvi funkciókat és eszközöket, amelyek segítik a nagyobb méretű és összetettebb alkalmazások fejlesztését.

A TypeScript lehetővé teszi a változók, függvények és objektumok típusának definiálását, valamint a típushibák felderítését a fejlesztési időben. Támogatja a primitív típusokat, objektumokat, tömböket, függvényeket és egyéb összetett típusokat. Ez segít az erőteljesebb típusellenőrzésben és a fejlesztési folyamat során. A TypeScript támogatja az objektumorientált programozás alapelveit, mint az osztályokat, interfészeket, leszármaztatást, absztrakt osztályokat, a modern JavaScript nyelvi elemeket, mint az arrow function-öket, sablon literálokat, destrukurálást, async/await, generátorokat.

A TypeScript egyre népszerűbbé válik a webfejlesztésben, különösen az Angular keretrendszerrel való együttműködésben. Az Angular-t is TypeScript-ben írják, és a TypeScript nyújtotta előnyök, mint a statikus típusellenőrzés és a jobb fejlesztői élmény, segítenek az Angular alkalmazások megbízhatóságának és karbantarthatóságának javításában.

3.3.4. Angular

Az **Angular** [12] rendkívül sokoldalú és erőteljes, és számos funkciót és eszközt biztosít az alkalmazásfejlesztéshez [13]. Az Angular komponens-alapú architektúrára épül, ami lehetővé teszi a felhasználói felület építését önálló, újrafelhasználható komponensekből. Ez javítja a kód modularitását, tesztelhetőségét és karbantarthatóságát. TypeScript nyelven alapul, ami egy szigorúan típusos JavaScript kiterjesztés. Ez növeli a kód biztonságát és hibák korai észlelését, valamint jobb fejlesztői élményt nyújt az intelligens kód kiegészítések és hibakeresési lehetőségek révén.

Az Angular támogatja a komponensek közötti adatkötést, ami lehetővé teszi az adatok egyirányú vagy kétirányú áramlását a felhasználói felületen [14]. Ez egyszerű és hatékony módja az adatok megjelenítésének és kezelésének a felületen. Modulok segítségével lehetőség van az alkalmazás funkcióinak és komponenseinek csoportosítására. Ez javítja a kód szervezettségét, könnyűvé teszi a nagyobb projektek kezelését és lehetővé teszi a lazy loading technikákat a hatékony oldalbetöltés érdekében.

Az Angular számos beépített eszközt és modult kínál az alkalmazásfejlesztéshez. Ide tartoznak például a Routing modul a navigációhoz, az HTTP modul a szervertől való kommunikációhoz, az űrlapkezelés, a tesztelési keretrendszerek és még sok más.

3.3.4.1. Angular alkalmazások felépítése

Egy Angular alkalmazás általában a következő főbb komponensekből áll:

- **Modulok:** Az Angular alkalmazás modulokra épül [15]. A modulok segítségével csoportosíthatjuk az alkalmazás komponenseit, szolgáltatásait és egyéb funkcióit. Az AppModule a fő modul, amely tartalmazza az alkalmazás gyökerkomponensét és beállításait. Az alkalmazás többi modulja segítségével további funkciókat és komponenseket adhatunk hozzá az alkalmazáshoz.
- **Komponensek:** Az Angular komponensek az alkalmazás felhasználói felületét képviselik [16]. A komponensek feladata a megjelenítés és az interakció kezelése. Mindegyik komponenshez tartozik egy sablon, ami a HTML kódot és a megjelenítési logikát tartalmazza, valamint egy TypeScript osztály, ami a komponenshez kapcsolódó logikát és adatokat kezeli.
- **Sablonok:** A komponensekhez tartoznak sablonok, amelyek meghatározzák a komponensek kinézetét és struktúráját [17]. A sablonok HTML alapúak, de kiegészítve vannak Angular direktívákkal és adatkötési szintaxissal, amelyek lehetővé teszik a dinamikus adatmegjelenítést és az eseménykezelést.
- **Szolgáltatások:** Az Angular szolgáltatások olyan osztályok, amelyek közös funkciókat és adatkezelést biztosítanak az alkalmazás különböző részei számára [18]. A

szolgáltatások lehetnek adatszolgáltatók, hálózati kommunikációért felelős osztályok, biztonsági vagy hibakezelési szolgáltatások, stb. A szolgáltatások segítségével a komponensek elérhetik és kezelhetik a megfelelő adatokat és funkciókat.

- **Routing:** Az Angular Routing lehetővé teszi az alkalmazás különböző oldalai közötti navigációt [19]. A Routing modul segítségével definiálhatjuk az alkalmazás útvonalait, és megadhatjuk, hogy melyik komponens jelenjen meg az adott útvonalon. Ez lehetővé teszi az egyszerű és intuitív navigációt az alkalmazásban.
- **Vezérlők:** Az Angular vezérlők olyan direktívák, amelyek a DOM elemek viselkedését és megjelenését befolyásolják [20]. A vezérlők lehetővé teszik a dinamikus elemek hozzáadását és eltávolítását, eseménykezelést, adatkötést és más DOM manipulációkat.

3.3.4.2. Angular könyvtárak

Az Angular keretrendszer számos fontos könyvtárral rendelkezik, amelyek segítik az alkalmazásfejlesztést és különböző feladatokat oldanak meg. Néhány fontos könyvtár az Angularhoz:

- **Angular CLI:** Az Angular Command Line Interface (CLI) egy erőteljes parancssori eszköz, amely segít az Angular projekt létrehozásában, fejlesztésében és kezelésében. Az Angular CLI egyszerűsíti a fejlesztési folyamatot és automatizálja a feladatokat, például a projekt inicializálását, komponensek és szolgáltatások generálását, fejlesztői szerver indítását és az alkalmazás buildelését.
- **Angular Material:** Az Angular Material egy stílusos és reszponzív UI komponenskönyvtár, amely előre elkészített és testreszabható komponenseket kínál az Angular alkalmazásokhoz. Ez a könyvtár segít egyszerűen és konszisztensen megjeleníteni a felhasználói felületet, és lehetővé teszi az interakciók kezelését.
- **Angular Forms:** Az Angular Forms modul lehetővé teszi az űrlapok kezelését az Angular alkalmazásokban. Ez a könyvtár segít az adatok bevitelének és validálásának kezelésében, valamint az űrlapállapot nyomon követésében.
- **Jasmine:** A Jasmine egy tesztelési keretrendszer, amelyet az Angular fejlesztéséhez gyakran használnak. A Jasmine segítségével egységteszteket és integrációs teszteket írhatunk az Angular alkalmazás komponenseire, szolgáltatásaira és egyéb részeire. Ezáltal biztosíthatjuk a kód minőségét, hibamentességét és a funkcionalitás helyes működését.

3.3.5. Kotlin

Kotlin [21] egy modern, statikusan típusos programozási nyelv, amely a Java Virtual Machine (JVM) környezetre és más platformokra, mint például az Android, is kiterjed. A célja, hogy kényelmesebbé és hatékonyabbá tegye a Java nyelven történő fejlesztést.

Kotlin tökéletesen interoperál a Java nyelvvel, így a meglévő Java kódokat könnyen integrálhatjuk Kotlin projektekbe, és fordítva. Egyszerűbb és rövidebb szintaxissal rendelkezik, mint a Java az olvashatóság és írás könnyebbége érdekében. Kotlin beépített

támogatást nyújt a null-hibák elkerülésére. A típusrendszerben bevezetett nullable és non-nullable típusok segítenek a biztonságosabb kódolásban és a null-hiba okozta problémák minimalizálásában. Támogatja a funkcionális programozást, mint a lambda kifejezések, magasabbrendű függvények, immutable adatstruktúrák és stb. Ez lehetővé teszi a kód egyszerűsítését és az expresszívabb programozást.

Kotlin gazdag szabványkönyvtárral rendelkezik, amely számos hasznos funkciót és osztályt tartalmaz, például string műveletek, kollekciók kezelése, input/output műveletek stb. Kotlin hivatalosan támogatott nyelv az Android platformon. Az Android alkalmazásokat Kotlin nyelven is írhatjuk, és sok fejlesztő előnyben részesíti a Kotlin a hatékonyság és az olvashatóság miatt.

3.3.6. Android

Az Android [22] egy mobil operációs rendszer és egy fejlesztési platform is. Az Android platform lehetővé teszi fejlesztők számára, hogy mobilalkalmazásokat hozzanak létre és futtassanak Android-eszközökön, mint például okostelefonok, táblagépek, okosórák, televíziók és egyéb eszközök.

Az Android nyílt forráskódú platform, amely lehetővé teszi a fejlesztők számára az alkalmazások testreszabását és az operációs rendszerre történő mélyebb beavatkozást. Ez hozzájárul a fejlesztői közösség dinamikus növekedéséhez és az új innovatív alkalmazások létrehozásához. A platform támogatja a széles körű eszközöket, az egyszerű okostelefonoktól a nagy teljesítményű táblagépekig és okosotthoni eszközökig. Ez lehetővé teszi a fejlesztők számára, hogy különböző eszközökre készítsenek alkalmazásokat, és a felhasználók számára egy egységes és összekapcsolt élményt biztosítsanak.

Az Android platform számos fejlesztési lehetőséget kínál, beleértve a natív Android fejlesztést Kotlin vagy Java nyelven, valamint a keresztplatformos fejlesztést olyan keretrendszerek segítségével, mint például a React Native vagy a Flutter. Szorosan integrálódik a Google szolgáltatásokkal, mint például a Google Maps, Gmail, Drive, Hangouts stb. Ez lehetővé teszi az alkalmazások számára, hogy kihasználják ezeket a szolgáltatásokat és együttműködjenek a felhőalapú technológiával.

3.3.6.1. Android alkalmazás felépítése

Az Android komponensek [23] építik fel az Android alkalmazások struktúráját és funkcionalitását, és lehetővé teszik a felhasználói felületek létrehozását, adatkezelést, háttérműveleteket és egyéb funkciókat az Android platformon. A fő komponensek:

- **Activity:** Az Activity az alkalmazás egy képernyőjét vagy felhasználói interakciót reprezentálja. Az alkalmazás egy vagy több Activity-ből épül fel, és ezek közötti váltásokkal valósul meg a navigáció.
- **Fragment:** A Fragment egy önállóan használható része az Activity-nek, amely különálló UI elemeket és viselkedést képes biztosítani. Több Fragment kombinálása lehetővé teszi a rugalmasabb és újrafelhasználhatóbb felhasználói felületek kialakítását.

- **Service:** A Service egy háttérben futó komponens, amely hosszabb ideig futó feladatokat vagy folyamatosan végrehajtott műveleteket képes ellátni anélkül, hogy szükség lenne egy UI felületre. Például zenelejátszás vagy hálózati kérések kezelése.
- **Broadcast Receiver:** A BroadcastReceiver olyan komponens, amely az Android rendszerből érkező különböző eseményeket képes észlelni és kezelni, például hálózati állapotváltozás vagy telefonhívás érkezése.
- **Content Provider:** A Content Provider lehetővé teszi az alkalmazás számára, hogy adatokat osszon meg más alkalmazásokkal, vagy adatokat lehessen lekérdezni más alkalmazásokból. Ez egy központi adatforrás lehet, amelyhez más alkalmazások hozzáférhetnek.

3.3.6.2. Android könyvtárak

Az Android könyvtárak fontos szerepet játszanak az alkalmazások fejlesztésében, és segítenek abban, hogy könnyebbé és hatékonyabbá tegyünk az Android alkalmazások tervezését és implementációját. Ezeket általában a modern Android alkalmazások architektúrájának részeként használják.

- **ViewModel:** Az Android ViewModel egy komponens, amely az alkalmazás adatállapotának tárolására és kezelésére szolgál. A ViewModel segít az alkalmazás adatok és üzleti logika elkülönítésében a felhasználói felülettől. A ViewModel tárolja és biztosítja az adatokat az Activity-k és Fragment-ek között, így adataink megőrzik az állapotukat például kijelző forgatás esetén is. A ViewModel lehetővé teszi az alkalmazás adatok hatékony kezelését és az alkalmazásarchitektúra javítását.
- **Android Lifecycles:** Az Android Lifecycles egy könyvtár, amely lehetővé teszi az alkalmazás komponenseinek, például az Activity-k és Fragment-ek életciklusának kezelését. Az Android platform különböző eseményeinek, például az alkalmazás indításának, háttérbe helyezésének vagy leállításának kezelésére szolgál. A Lifecycles lehetővé teszi a folyamatos és koherens alkalmazási állapot fenntartását, valamint a források hatékony kezelését.
- **Data Binding:** Az Android DataBinding egy könyvtár, amely lehetővé teszi az adatkötést (data binding) az alkalmazás felhasználói felülete és a háttérben lévő adatok között. Az adatkötés segítségével a felhasználói felület automatikusan frissül az adatok változásával, így könnyebb és kényelmesebb módon kezelhetjük a felhasználói felületet. Az adatkötés jelentősen csökkentheti a kódmennyiséget és növelheti az alkalmazás fejlesztésének hatékonyságát.
- **Navigation:** Az Android Navigation egy könyvtár, amely segít az alkalmazás navigációjának kezelésében. Ez az architektúra segíti az alkalmazás különböző képernyőinek közötti átmeneteket, a visszalépést, mély linkelést és egyéb navigációs funkciókat. A Navigation segítségével egyszerűen és hatékonyan kezelhetjük az alkalmazás navigációs gráfját és a felhasználói élményt.

3.3.7. Gradle

A Gradle [24] egy nyílt forráskódú build automatizálási eszköz és építési rendszer. A Gradle lehetővé teszi a fejlesztők számára a projekt folyamatossági menedzsmentjét, a forráskód fordítását, a függőségek kezelését és az alkalmazások csomagolását.

A Gradle egy rugalmas és kifejező konfigurációs DSL-t (Domain Specific Language) használ, amely lehetővé teszi a feladatok, folyamatok és függőségek deklaratív leírását. Ez lehetővé teszi a fejlesztők számára, hogy könnyen testreszabják és beállítsák a projektük építési folyamatát az egyedi igényeknek megfelelően. Támogatja a pluginok használatát, amelyek kiterjesztik az alapértelmezett funkciókat és lehetővé teszik az alkalmazások specifikus feladatok végrehajtását. Számos előre elkészített plugin érhető el a Java projektekhez, de a Gradle támogatja más nyelvek és keretrendszerek, például Kotlin, Android, Spring stb. buildelését is.

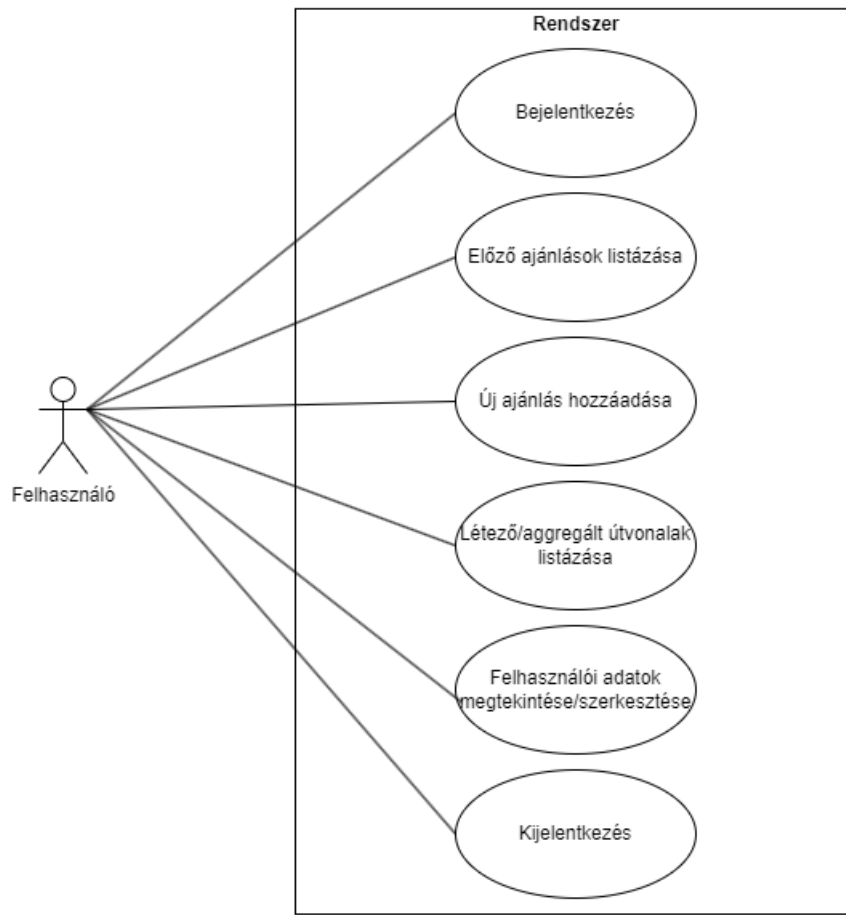
4. fejezet

A rendszer specifikációja

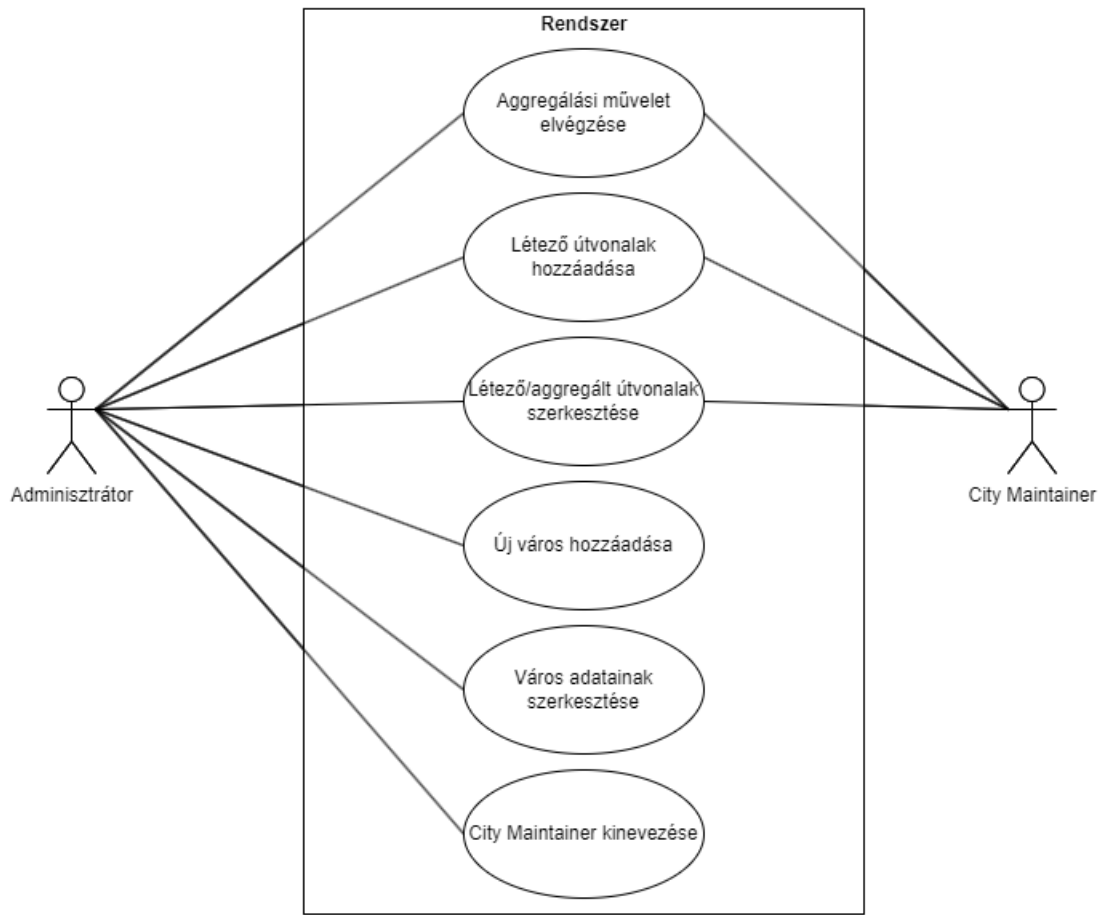
4.1. Felhasználói követelmények

Ebben a részben az alkalmazás fontosabb funkciói és felhasználási esetei kerülnek felsorolásra és leírásra, amelyekkel a felhasználó találkozhat az alkalmazás használata során. Három szerepkört lehet megkülönböztetni az alkalmazásban:

- az egyszerű felhasználó, aki a közlekedési ajánlásokat adhat.
- city maintainer, az aggregálási folyamatot irányítja és beállítja annak paramétereit a városi közlekedés optimalizálása érdekében.
- adminisztrátor, városokat adhat hozzá az alkalmazáshoz, kezelheti, módosíthatja a városi adatokat, és kinevezhet city maintaintereket a városokhoz, illetve szükség esetén a city maintainer feladatát is el tudja végezni.



4.1. ábra. Felhasználó használati eset diagram



4.2. ábra. Használati eset diagram a többi szerepkörre

4.1.1. Fontosabb használati esetek

- új ajánlás hozzáadása

- előfeltételek:

- * PR1: a felhasználó be van jelentkezve

- folyamat:

- * F1: a felhasználó bejelentkezik

- * F2: a felhasználó kiválasztja a menüből az új ajánlás hozzáadását

- * F3: a felhasználó kiválasztja a várost a legördülő listából

- * F4: a felhasználó bejelöli a térképen az induló- és végpontot

- * F5: a felhasználó kitölti az útvonalhoz szükséges adatokat

- * F6: a felhasználó megnyomja az *Add* gombot

- alternatív folyamat:

- * AF1: ha a felhasználó már be van jelentkezve, akkor a folyamat az F2-től kezdődik

- utófeltételek:

- * PO1: az új ajánlás bekerül a felhasználó ajánlásai közé
- * PO2: "Suggestion sent" toast message jelenik meg
- *hibák:*
 - * E1: "Form is not filled out properly" toast message jelenik meg. Vissza F4-hez
 - * E2: "Invalid token" toast message jelenik meg. Vissza F1-hez
- **előző ajánlások megtekintése**
 - *előfeltételek:*
 - * PR1: a felhasználó be van jelentkezve
 - * PR2: a felhasználónak már adott hozzá legalább egy ajánlást
 - *folymat:*
 - * F1: a felhasználó bejelentkezik
 - * F2: a felhasználó kiválasztja a menüből az ajánlások listáját
 - * F3: a felhasználó kiválasztja a várost a legördülő listából
 - * F4: a felhasználó megérinti az ajánlást
 - *alternatív folymat:*
 - * AF1: ha a felhasználó már be van jelentkezve, akkor a folymat az F2-től kezdődik
 - *utófeltételek:*
 - * PO1: az ajánlás útvonala megjelenik a térképen és alatta a hozzá kapcsolódó részletek
- **létező útvonalak megtekintése**
 - *előfeltételek:*
 - * PR1: a felhasználó be van jelentkezve
 - *folymat:*
 - * F1: a felhasználó bejelentkezik
 - * F2: a felhasználó kiválasztja a menüből az útvonalak listáját
 - * F3: a felhasználó kiválasztja a várost a legördülő listából
 - * F4: a felhasználó a *Filter Options* menüben kiválasztja a *Route Typenál* a *Existing* típust és a többi szükséges szűrőt
 - * F5: a felhasználó megérinti az útvonalat
 - *alternatív folymat:*
 - * AF1: ha a felhasználó már be van jelentkezve, akkor a folymat az F2-től kezdődik
 - *utófeltételek:*
 - * PO1: az útvonal megjelenik a térképen és alatta a hozzá kapcsolódó részletek

- **alapértelmezett város megváltoztatása**

- *előfeltételek:*

- * PR1: a felhasználó be van jelentkezve

- *folyamat:*

- * F1: a felhasználó bejelentkezik

- * F2: a felhasználó kiválasztja a menüből a profilt

- * F3: a felhasználó kiválasztja az új alapértelmezett várost a legördülő listából

- *alternatív folyamat:*

- * AF1: ha a felhasználó már be van jelentkezve, akkor a folyamat az F2-től kezdődik

- *utófeltételek:*

- * PO1: "Default city changed" toast message jelenik meg

4.2. Rendszerkövetelmények

4.2.1. Funkcionális követelmények

Az alkalmazás legfontosabb funkcionalitásait fogom bemutatni ebben az alfejezetben.

- **Bejelentkezés:** ha a felhasználó nem rendelkezik érvényes tokennel, akkor a bejelentkezési oldalon találja magát, az alkalmazás megnyitásakor. A *Sign In* gomb megnyomásával a Google fiók adatait kell megadni és sikeres bejelentkezés esetén már használhatja az alkalmazást. Ha a felhasználó először használja az alkalmazást az adott email címmel, akkor a Google fiók alapadatai lesznek használva a felhasználó személyes adataiként.
- **Új ajánlás hozzáadása:** a menüből elérhető ez az oldal. A felhasználó először kiválasztja a várost, a legördülő listából, majd a térképen bejelöli az útvonal kezdő- és végpontját. A *Suggestion Options* menüben pedig kiválasztja, hogy milyen útvonal legyen, illetve a többi beállítás is itt érhető el. Az *Add* gomb megnyomásával lehet elküldeni az ajánlatot.
- **Előző ajánlások megtekintése:** lista szerűen megtekinthetők a felhasználó előző ajánlásai. A városok között lehet szűrni, úgy hogy csak egy adott városban tett ajánlások jelenjenek meg. Egy adott ajánlást kiválasztva, megjelenik a pontos útvonal a térképen, illetve alatta az ehhez tartozó továbbá részletek.
- **Létező útvonalak megtekintése:** az útvonalak oldalon, a *Filter Options* menüt lenyitva, az útvonal típusnál ki kell választani az *Existinget*, majd az *Apply* gombbal alkalmazzuk ezt. Így csak a létező útvonalak fognak megjelenni. A várost pedig az oldal tetején lévő *Citynél* változtathatjuk. Ha kiválasztunk egy adott útvonalat, akkor megtekinthetők az ehhez kapcsolódó részletek, illetve a térképen látható lesz a teljes útvonal.

- **Aggregált útvonalak megtekintése:** az útvonalakat menüpontot kiválasztva, az oldal tetején a legördülő lista segítségével lehet megadni, hogy melyik városban lévő útvonalak legyenek betöltve. A *Filter Options*ben az *Aggregated* útvonaltípust kell megjelölni, ahhoz, hogy megtekinthető legyen egy adott útvonal a térképen, egyszerűen csak ki kell választani és egy új oldalon látható lesz. A térkép alatt további részletek lesznek olvashatóak.
- **Szűrés útvonalak között:** az útvonalak oldalon, a *Filter Options* menüt lenyitva találhatóak a szűrő beállítások. Itt lehet kiválasztani az útvonal típusát. Be lehet állítani, hogy csak bicikli- vagy buszútvonalak jelenjenek meg. Buszútvonalak esetén kiválasztható, hogy melyik napon, illetve milyen időpontban közlekedjen. Az *Apply* gomb segítségével alkalmazhatjuk a kijelölt szűrőket, illetve a *Clear* gombbal törölhetjük az összes aktívát.
- **Profil megtekintése:** a profil menüpont kiválasztásával a felhasználó megtekintheti a személyes adatait.
- **Profil szerkesztése:** a profil menüpontban a felhasználó megváltoztathatja ezeket az adatokat. Az *Edit Profile* gomb megnyomásával lehet elmenteni a változtatásokat.
- **Alapértelmezett város beállítása:** a profilon belül, a *City* legördülő listából lehet kiválasztani, hogy alapértelmezetten melyik város jelenjen meg a többi menüpontok térképén.
- **Aggregálási művelet elvégzése:** az útvonalak aggregálása menüpontban lehet beállítani az aggregálás paramétereit, illetve hogy melyik fajta útvonalon legyen elvégezve. *Generate Aggregated Routes* gomb megnyomásával meg fog történni a művelet.
- **Új város hozzáadása:** a város nevét és a koordinátáit kell megadni, *Add* gomb megnyomásával ez a város is kiválaszthatóvá válik.

4.2.2. Nem funkcionális követelmények

- **Termék követelmények**
 - *Felhasználhatóság:* a webkliens a leggyakrabban használt modern böngészőkkel (Chrome, Edge, Firefox stb.) használható. A mobil alkalmazás futtatásához legalább 7.0 (Nougat) verziójú Android rendszer szükséges, minimum 24-es API-val kell rendelkeznie.
 - *Hatékonyság:* az alkalmazás gyorsan válaszol a felhasználó kéréseire, szinte azonnal reagálva a műveletekre, amennyiben rendelkezik megfelelő internet-kapcsolattal.
 - *Megbízhatóság:* a fejlesztés során a legtöbb lehetséges hibalehetőség alaposan tesztelve volt, így minimalizálva a hibák és hibajelenségek előfordulását. Az alkalmazás normál használat során hibamentesen működik, és nem jelennek meg hibaüzenetek a felhasználó számára.

- *Biztonság*: az alkalmazás a Google OAuth segítségével valósítja meg a bejelentkezést és token alapú azonosítást használ a felhasználók számára.
- *Hordozhatóság*: a mobil alkalmazáshoz előzetes telepítés szükséges.

- **Külső követelmények**

- *Összeférhetőségi*: az alkalmazás teljes mértékben elkülönül a már meglévő alkalmazásoktól, és semmilyen más alkalmazásra nincs hatással. Nem befolyásolja más alkalmazások működését.
- *Etikai*: az alkalmazás biztosítja az adatvédelmet, és nem hozzáférhetőek külső források számára. Az általános adatvédelmi előírásokat maradéktalanul betartja.
- *Együttműködési*: a felhasználónak saját adatokat kell megadnia, és felelősséget vállal adatainak pontosságáért. Az adatok kizárólag az adminisztrátor által lesznek láthatóak.

5. fejezet

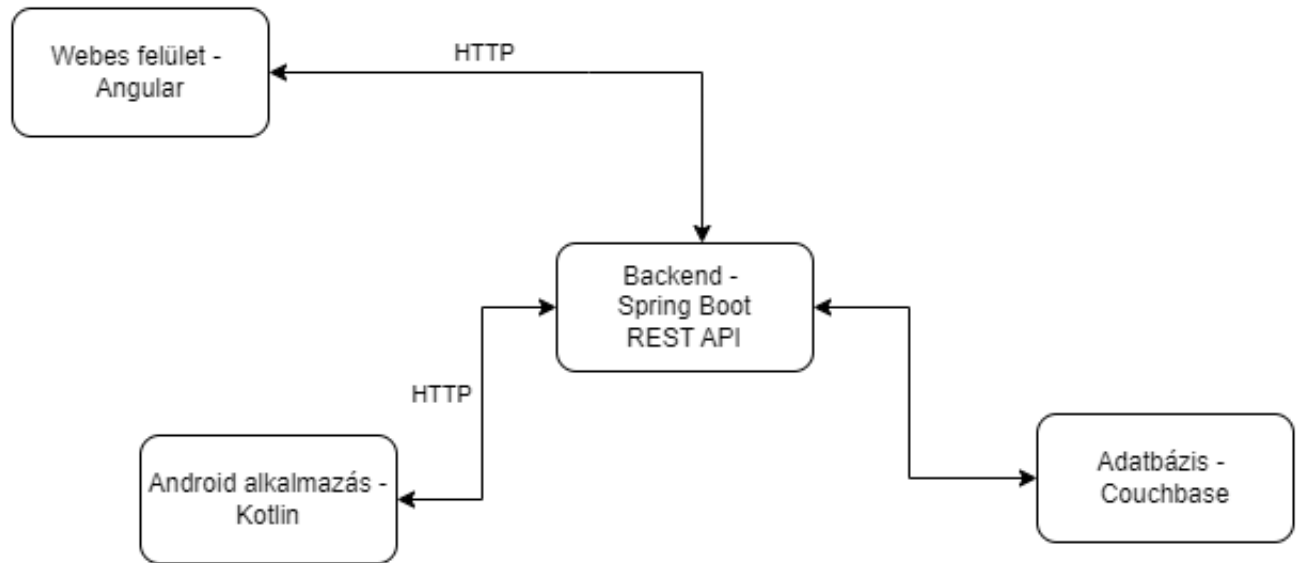
Tervezés

Ebben a fejeztben ismertetem a projekt rendszertervét és részletezem a szerver- és kliensoldali applikáció architektúráját.

5.1. A rendszer architektúrája

A rendszer három fő részből áll: adatbázis, szerveroldal (backend) és kliensoldal (frontend). Ezeken kívül be vannak építve harmadik féltől származó alkalmazások, mint a Google OAuth bejelentkezéshez használt hitelesítő és a Google Directions API, ami a helyszínek közötti útvonal meghatározására van használva.

- *adatbázis*: a legalsó réteg, amely lehetővé teszi a dokumentumok tárolását és kezelését, illetve megvalósítja a szerveroldallal való kommunikációt.
- *szerveroldal*: felelős az üzleti logika végrehajtásáért, a kérések fogadásáért és feldolgozásáért, valamint végrehajtja a szükséges adatbázis műveleteket, például adatlekérdezéseket.
- *kliensoldal*:
 - **webes felület**: MVC (Model-View-Controller) tervezési mintát követi, amely elkülöníti az adatokat (Model), a felhasználói felületet (View) és az üzleti logikát (Controller), hogy hatékonyan kezelje a kéréseket és adatokat.
 - **mobil alkalmazás**: MVVM (Model-View-ViewModel) alapú tervezési mintát követ, amely elkülöníti a felhasználói felületet (View) a üzleti logikától (ViewModel), és adatait (Model) a ViewModel közvetítésével kezeli.



5.1. ábra. A rendszer architektúrája

5.2. Szerveroldal

A szerveroldal egy **háromrétegű architektúrára** támaszkodik. Ez a háromrétegű architektúra lehetővé teszi a különálló felelősségi területek elkülönítését és a komponensek könnyű cseréjét vagy bővítését. Az elkülönült rétegek javítják a fejlesztési rugalmasságot, lehetővé teszik a könnyű karbantarthatóságot és a skálázhatóságot. Az alkalmazás a következő rétegből áll:

- **Prezentációs réteg:** ez a réteg fogadja a kliensoldalról érkező kéréseket és válaszokat küld a kliensnek és kommunikál a logika réteggel a kliensoldal felől érkező adatok továbbítása, illetve ezek feldolgozása érdekében.
- **Üzleti logikai réteg:** ez a réteg felel az alkalmazás üzleti logikájáért, itt történik az adatok validálása, feldolgozása a prezentációs rétegből érkező kérések alapján, közvetlenül kommunikál az adatréteggel
- **Adat réteg:** az adatok tárolásáért és kezeléséért felelős, közvetlenül kommunikál az üzleti logika réteggel az adatok lekérdezése és módosítása során, és biztosítja a szükséges adatelérést a szerveroldali komponensek számára.

5.3. Webkliens

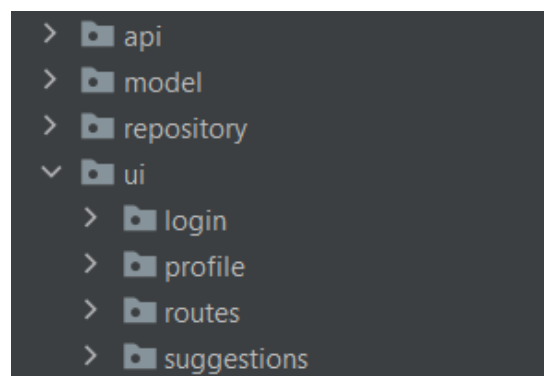
Az MVC (**Model-View-Controller**) architektúra lehetővé teszi a webkliens különböző rétegeinek elkülönítését és felelősségi területeinek tisztázását. Az MVC architektúra a következő három fő komponensből áll: Model, View és Controller. A Model réteg az alkalmazás adatait és az adatkezelést kezeli. A Model nem tartalmazza a felhasználói felülettel vagy a felhasználói interakcióval kapcsolatos logikát. A View réteg a felhasználói

felületet jelenti, amelyen a felhasználók az alkalmazással interakcióba lépnek. A View felelős a tartalom megjelenítéséért, és a felhasználói eseményeket (pl. kattintások, beviteli mezők stb.) észleli és továbbítja a Controller rétegnek. A Controller réteg kapcsolatot teremt a Model és a View között. Reagál a felhasználói eseményekre, majd feldolgozza ezeket a kéréseket. A Controller továbbítja az eredményeket a View felé, és kezeli az alkalmazás állapotát. Az MVC architektúra lehetővé teszi az alkalmazás komponenseinek különálló skálázhatóságát, karbantarthatóságát. Az elkülönült rétegek lehetővé teszik az egymástól független fejlesztést és tesztelést.

5.4. Mobilkliens

Az **MVVM (Model-View-ViewModel) architektúra** egy olyan tervezési minta, amely különválasztja az adatokat (Model), a felhasználói felületet (View) és a nézetmodellt (ViewModel) a mobil alkalmazásban. Az MVVM architektúra előnyei közé tartozik a felelősségek szétválasztása, a könnyű tesztelhetőség és a kód újrafelhasználhatósága. A Model réteg az alkalmazás adatait és adatainak kezelését tartalmazza. Ez lehet adatbázis, hálózati szolgáltatásokhoz való hozzáférés, adatforrások kezelése vagy bármilyen más adatmanipulációs logika. A Model réteg felelős az adatok lekérdezéséért, módosításáért és tárolásáért. Ez lehet például adatközpont, API-k vagy bármilyen más adatforrás, amelyre az alkalmazás támaszkodik. A View réteg az alkalmazás felhasználói felületét jelenti, amely a felhasználóval való interakciót és adatainak megjelenítését végzi. A View felelős az események kezeléséért és a felhasználói interakciók feldolgozásáért, valamint a megjelenített adatok frissítéséért és megjelenítéséért. A ViewModel réteg közvetítőként működik a Model és a View között, ami lehetővé teszi a hatékony adatkötést és a felhasználói interakciók könnyű kezelés és feldolgozását. Ez a réteg lehetővé teszi a deklaratív adatkötést, amely automatikusan frissíti a View-t, ha a ViewModel-ben található adatok megváltoznak.

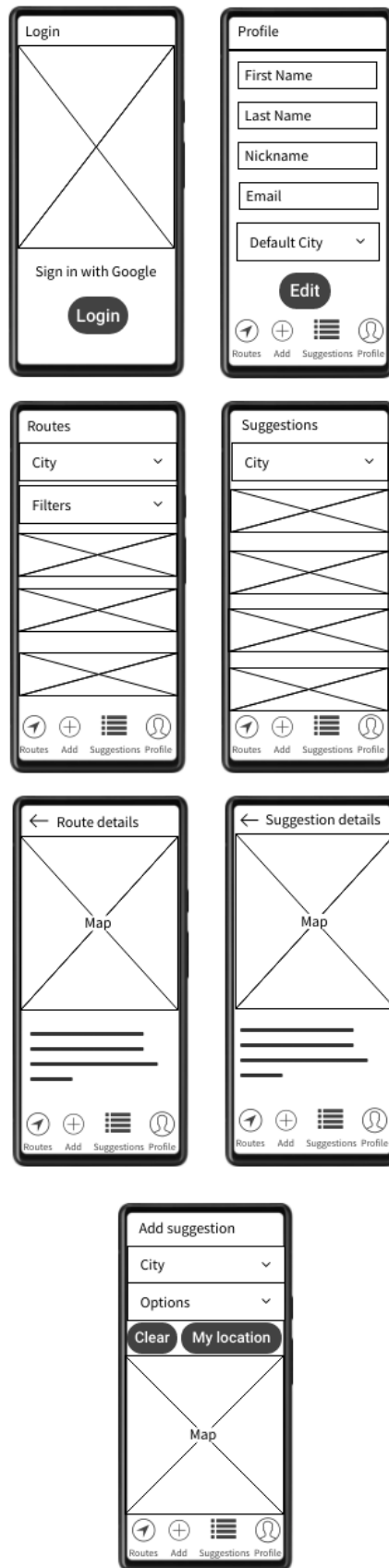
A mappaszerkezet kialakításakor ennek követésére törekedtem. A serveroldallal kommunikáló réteget, a repository réteget, a modelleket és felhasználói felület elemeit külön csomagokba rendeztem. A felhasználói felület elemeit, pedig az átláthatóság miatt külön csoportosítottam.



5.2. ábra. Android Studio mappaszerkezet

5.5. Felhasználói felület tervezése

A wireframe-ek (drótvázak) a felhasználói felület tervezésekor használatosak, még mielőtt a részletes design és stílus elemek bekerülnének, olyan alacsony szintű, fekete-fehér vázlatok vagy rajzok, amelyek bemutatják az alkalmazás alapvető elrendezését, szerkezetét és tartalmát. Lehetővé teszik a tervezők számára, hogy kialakítsák és kommunikálják a felhasználói felület alapvető szerkezetét és elrendezését. Ez segít megérteni, hogy milyen elemek fognak szerepelni az alkalmazásban, hogyan fognak elhelyezkedni és hogyan lesznek egymással kapcsolatban. A drótvázak segítenek megjeleníteni az alkalmazás funkcióit és a felhasználói interakciókat. Ez lehetővé teszi a tervezők számára, hogy megtervezzék a gombok, űrlapok, menük vagy más interaktív elemek helyét és működését.



5.3. ábra. A felhasználói felület drótváza

6. fejezet

Kivitelezés

Ebben a részben szemléltetem az alkalmazás elkészítésének kivitelezését és működését.

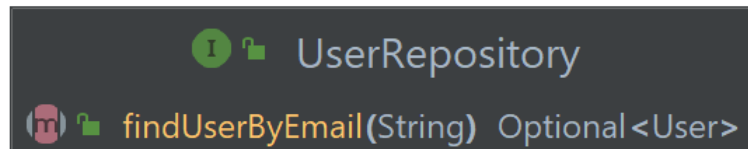
6.1. Szerveroldali megvalósítás

A szerveroldal Spring keretrendszer alatt fut. Az alábbiakban bemutatom a rétegeket és a komponenseket, amelyek a backend kialakításához vannak használva.

6.1.1. Adat és repository réteg

Az alkalmazás Couchbase adatbázist használ. Az adatréteg tartalmazza az entitásokat (modellek), amelyek leképezik az adatbázis dokumentumait és implementálja a CRUD (Create, Read, Update, Delete) műveleteket az adatok kezeléséhez. Az adatréteg használ repository-kat, amelyek lehetővé teszik az adatbázis-specifikus műveletek elvégzését, például adatok lekérdezését vagy módosítását. A kommunikáció az adatbázis és a szerveroldal között a Spring Data Couchbase keretrendszer segítségével valósul meg. Ez a keretrendszer lehetővé teszi az egyszerűbb és hatékonyabb adatbázis-műveletek elvégzését a Couchbase adatbázisban.

A repository réteg célja az adatelérések elvégzése és az adatok adatbázisba történő mentése vagy onnan történő lekérése. Ennek a rétegnek a feladata, hogy elrejtse az adatbázissal való közvetlen kommunikációt és a lekérdezések kezelését, így a service réteg nem kell közvetlenül foglalkozzon az adatbázissal kapcsolatos részletekkel. A repository interfész az adatelérésekhez szükséges metódusokat tartalmazza, amelyeknek a Spring Data automatikusan generálja az implementációt. A service réteg injektálhatja a repository-t és használhatja a definiált metódusokat az adatok elérésére vagy manipulálására. A Spring Data gondoskodik az adatelérési műveletek végrehajtásáról a háttérben.



6.1. ábra. Repository réteg osztálydiagramja

A repository réteg interfészei implementálják a *CrudRepository* interfészt, amely már tartalmazza az alapvető CRUD műveleteket megvalósító függvények definícióit.

6.1.2. Service réteg

A service réteg az alkalmazás logikai rétege, amely felelős az üzleti logika végrehajtásáért és a különböző folyamatok kezeléséért. A service réteg feladata az adatelérést végző repository-k használata és a repository-k közötti kommunikáció biztosítása. A service réteg meghívja a megfelelő repository metódusokat az adatok lekérdezéséhez, mentéséhez, frissítéséhez vagy törléséhez. Ezáltal a service réteg absztrakt módon kezeli az adatelérést és lehetővé teszi a könnyű cserélhetőséget különböző adatforrások között. A controller rétegtől érkező felhasználói adatok feldolgozásáért is felel, ellenőrzi, hogy az adatok megfelelnek-e a megadott szabályoknak és érvényesek-e az alkalmazás logikájában, kezeli az esetleges hibákat és kivételeket, amelyek az adatok feldolgozása során felléphetnek.

```

CityService

getCities() Iterable<City>
getMaintainedCityByCityMaintainer(String) City
createCity(City) City
getNewsListByCityId(String) Iterable<News>
deleteNewsById(String, String) void
getCityById(String, List<VehicleType>, List<Integer>, List<Integer>) City
deleteCityById(String) void
updateCity(City) City
saveNewsByCityId(String, News) News
updateNews(String, News) News

```

```

CityServiceImpl

getCityById(String, List<VehicleType>, List<Integer>, List<Integer>) City
getCities() Iterable<City>
getNewsListByCityId(String) Iterable<News>
filterRouteByTime(Route, List<Integer>, List<Time>) boolean
calculateSectionTime(int, List<Time>) int
updateNews(String, News) News
createCity(City) City
deleteCityById(String) void
getFilteredRoutes(List<VehicleType>, List<Integer>, List<Integer>, C
updateCity(City) City
filterRouteByDays(Route, List<Integer>) boolean
deleteNewsById(String, String) void
filterRouteByVehicleType(Route, VehicleType) boolean
getMaintainedCityByCityMaintainer(String) City
saveNewsByCityId(String, News) News

```

```

SectionService

filterAndAggregateSections(String, VehicleType, double, double) <Section>

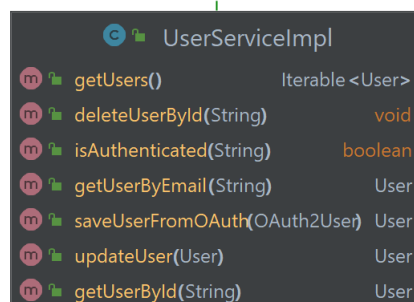
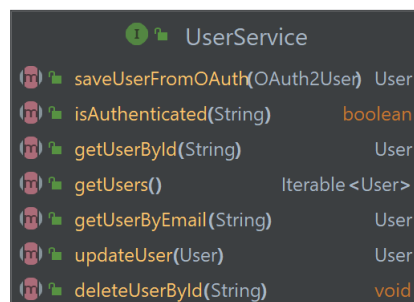
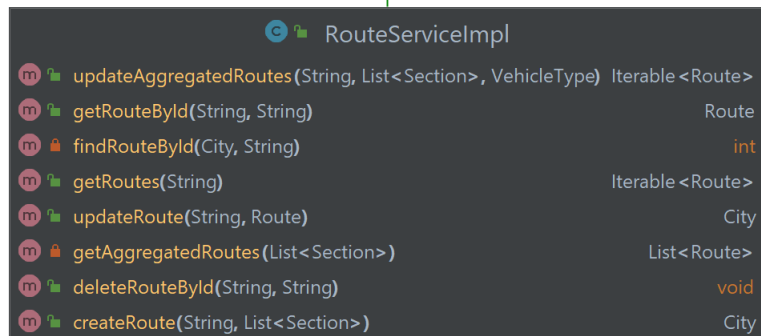
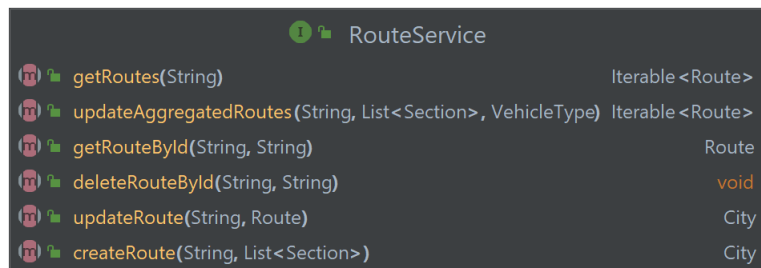
```

```

SectionServiceImpl

createSectionList(City) List<Section>
filterBicycleSections(List<Section>) List<Section>
filterBusSections(List<Section>, List<Time>, List<List<Integer>>) t<List<Section>>>
getFilteredLists(List<Section>, List<Time>, List<Integer>) List<List<Section>>
filterAndAggregateSections(String, VehicleType, double, double) List<Section>

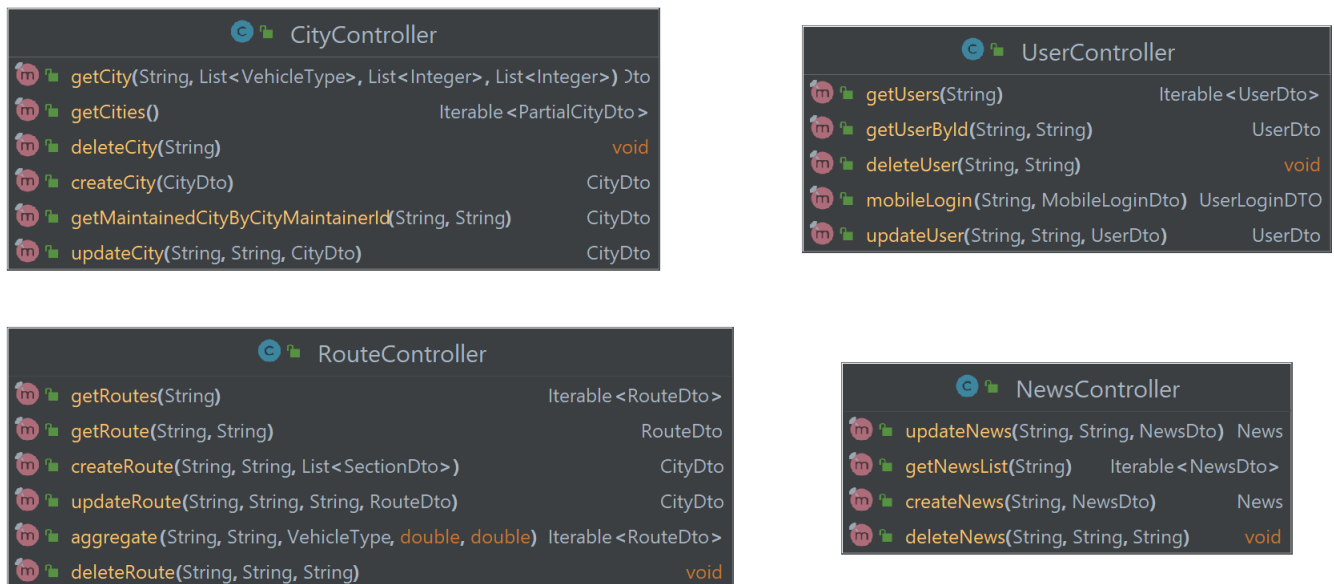
```



6.2. ábra. Service réteg osztálydiagramja

6.1.3. Controller réteg

A controller réteg az alkalmazásban közvetlen kapcsolatot biztosít a külső kérések (HTTP kérések) és a service réteg között. A controller réteg felelős a bejövő kérések fogadásáért, feldolgozásáért és a megfelelő válaszok generálásáért. A controller réteg figyeli a bejövő kéréseket, például a HTTP kéréseket, és azokat az adott végpontokhoz vagy útvonalakhoz irányítja. A controller réteg definiálja, hogy melyik metódus és melyik útvonalhoz tartozik, és meghatározza a végrehajtandó műveleteket, kinyeri a bejövő kérésből a szükséges paramétereket, például a vetőquery paramétereket és átadja ezeket a service rétegnek az adatelérés vagy a feldolgozás céljából. A controller réteg kezeli az esetleges hibákat és kivételeket, amelyek a kérések feldolgozása során felléphetnek. Ez lehet validációs hiba, erőforrás nem található vagy más típusú hiba. A controller réteg megfelelő válaszokat generál ezekre a hibákra, például hibaüzeneteket vagy HTTP státusz kódokat, és ezeket visszaküldi a kérő entitásnak.



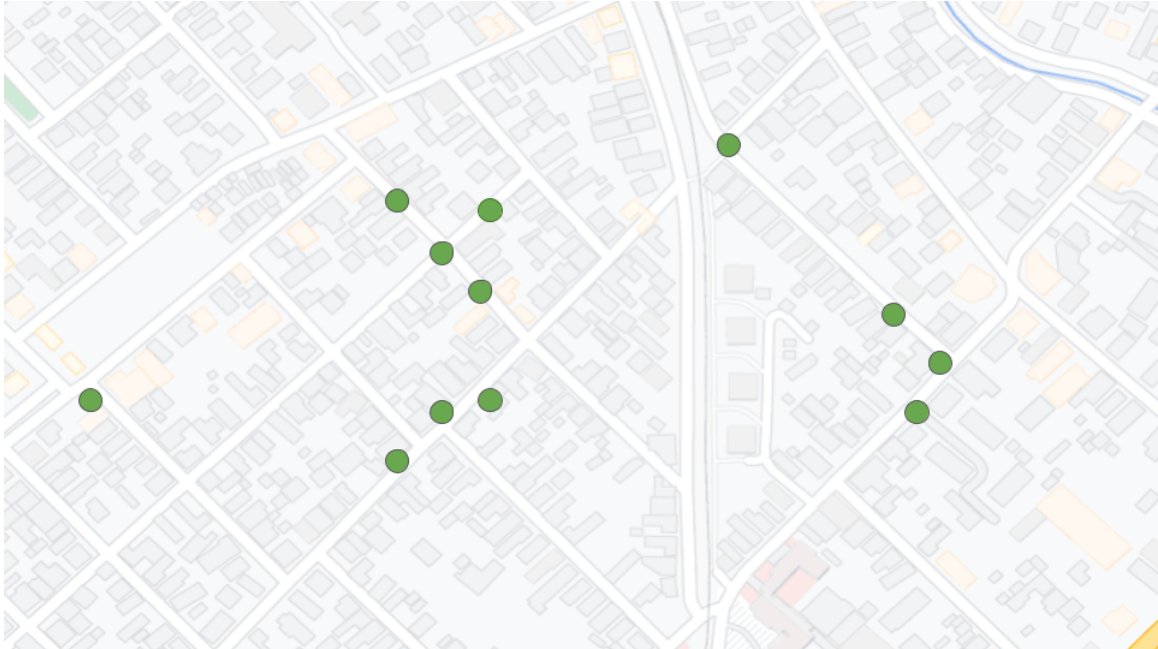
6.3. ábra. Controller réteg osztálydiagramja

6.1.4. Ajánlások feldolgozása

Az ajánlások feldolgozása egy klaszterező algoritmus segítségével történik. Az első lépésben be lesz kérve az adott város összes felhasználójának az adott városhoz tartozó ajánlása, majd ezeket szűrjük járműtípus szerint, illetve nap és időpont szerint, ha szükséges. Így megkapjuk az összes útvonalat, amelyeken el kell végezni az aggregálási műveletet. A második lépésben elválasztjuk az útvonalak kezdő és végpontjait, majd ezeket klaszterezzük. A következő lépéseket elvégezzük az útvonalak kezdőpontjaival, majd végpontjaikkal is (6.4 ábra).

Köröket használunk a klaszterezéshez. Az első pont lesz az első kör központja, és a további körön belüli pontok változatlanok maradnak. A körön kívülre eső újabb pontok esetén új körök jönnek létre (6.6 ábra). Utolsó lépésben az aggregált útvonalakat azokból

az útvonalakból hozzuk létre, amelyek ugyanabból a kezdőpontból indulnak és ugyanabba a végpontba érkeznek. Az új útvonalak kezdőpontja a kezdőpontjukat tartalmazó kör középpontja (6.6, 6.7 ábrák), végpontja pedig a végpontjukat tartalmazó kör középpontja lesz. A feldolgozás eredménye ezek az aggregált útvonalak lesznek.



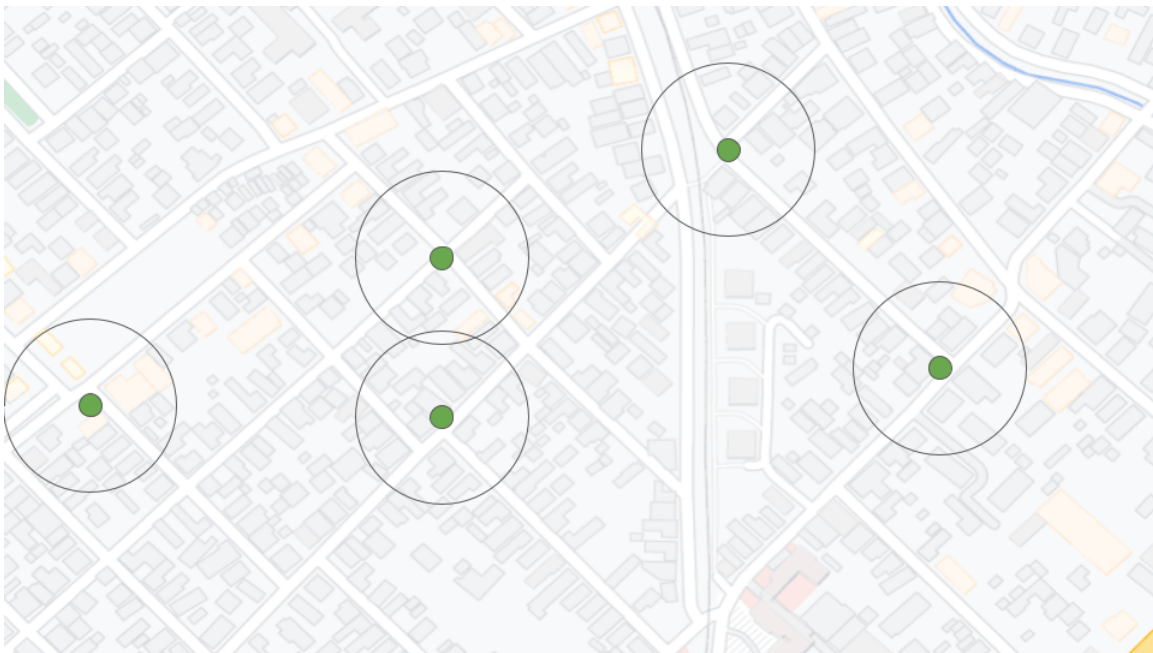
6.4. ábra. Kezdő/végpontok



6.5. ábra. Körök alkalmazása



6.6. ábra. A középpontok kiemelése



6.7. ábra. Maradék pontok eltüntetése

6.2. Webkliens megvalósítása

A webkliensként egy Angular alkalmazás fut. Az Angular keretrendszer segítségével könnyedén lehet fejleszteni reszponzív és korszerű webalkalmazásokat, amelyek dinamikus felhasználói felülettel és hatékony kommunikációval rendelkeznek a szerverrel.

6.2.1. Routing

Az Angular Routing egy fontos része az Angular keretrendszernek, amely lehetővé teszi a webalkalmazás különböző oldalai (komponensei) közötti navigációt és útvonalvezérlést. A routingnak számos előnye van, amelyek közé tartozik a felhasználói élmény javítása, a könnyű oldalváltás és a dinamikus tartalomkezelés. Az útvonalak meghatározzák, hogy mely komponensek jelenjenek meg az egyes URL-ekhez rendelt oldalakon, lehetővé teszi a dinamikus paraméterek használatát is. Az útvonalakat a RouterModule segítségével lehet definiálni az Angular modulokban. Az Angular Routing használja a lazy loadingot, vagyis hogy a komponensek csak akkor töltődjenek be, amikor szükség van rájuk. Ez javítja az alkalmazás teljesítményét, mivel kezdetben csak az első oldal komponensei töltődnek be, és a többi oldal csak akkor töltődik be, amikor a felhasználó arra navigál.

6.2.2. Data Binding

Az adatkötés (data binding) az Angular keretrendszer egyik kulcsfontosságú tulajdonsága, amely lehetővé teszi az adatok és a felhasználói felület (UI) közötti automatikus szinkronizációt. Az adatkötés segítségével a változások automatikusan továbbítódnak az adatmodell és a UI között. A *two-way binding* (kétirányú adatkötés) lehetővé teszi az adatok szinkronizációját a modell és a felhasználói felület között mindkét irányban. Amikor az adatok megváltoznak a modellben, azok automatikusan frissülnek a UI-n és fordítva, amikor a felhasználó módosítja az adatokat a UI-n, azok visszahatnak a modellre.

6.2.3. Szerverrel való kommunikáció

Az Angular alkalmazások a szerverrel való kommunikációhoz általában HTTP kéréseket használnak. Az Angular keretrendszer beépített HTTP modult kínál, amely lehetővé teszi a könnyű és hatékony kommunikációt a szerverrel. A HTTP kérésekkel JSON adatformátumok küldhetők és fogadhatók, és a válaszok kezelése is egyszerűen megoldható. A HttpClient segítségével egyszerűen megvalósítható a HTTP kérések küldése a szerver felé és a válaszok fogadása is. Ez lehetőséget biztosít a Rest API-kkal való kommunikációban és az adatok lekérdezésében vagy módosításában a szerveroldalon.

6.3. Mobilkliens megvalósítása

A mobilkliens egy natív android alkalmazás, ami MVVM architektúrát használ és HTTP kérésekkel kommunikál a szerveroldallal. Az alkalmazás futtatásához legalább 7.0 (Nougat) Androiddal (24-es API) rendelkező eszköz vagy emulátor szükséges.

6.3.1. Állapotmenedzsment

Az alkalmazás két activityből áll. Az egyik activity a bejelentkezéshez szükséges, a Google OAuth szolgáltatását igénybe véve hitelesíti a felhasználót és sikeres hitelesítés után automatikusan Shared Preferences segítségével eltárolja egy XML állományba a szükséges felhasználói adatokat. Ezután, a szerveroldalnak elküldi ezeket az adatokat

és a szerver válaszként visszaküldi egy tokenet. A másik activity a felhasználói felület többi részének megjelenítésért felel. A 6.1 kódrészletben a szerveroldaltól érkezett válasz elmentésére adok példát.

```
fun login(token: String, loginRequest: LoginRequest) {
    viewModelScope.launch{
        try {
            val response = userRepository.login(token = token, loginRequest =
                loginRequest)
            if (response.isSuccessful) {
                Log.d(TAG, "Login was successful")
                val edit = prefs.edit()
                edit.putString("token", response.body()!!.token)
                edit.putString("id", response.body()!!.id)
                edit.putString("user_role", response.body()!!.userRole.
                    toString())
                edit.apply()
                loginResult.value = RequestResult.SUCCESS
            } else {
                Log.d(TAG, "Login error: ${response.message()}")
                loginResult.value = RequestResult.UNKNOWN_ERROR
            }
        } catch (ex: Exception) {
            Log.e(TAG, ex.message, ex)
        }
    }
}
```

6.1. kódrészlet. LoginResponse elmentése

Minden felhasználói oldalnak megfelel egy fragment. A main activity tartalmazza a FragmentContainert, amely segítségével az oldalnak megfelelő fragment lesz betöltve. Minden fragmenthez tartozik egy ViewModel, aminek a segítségével megőrizhetőek a fragmentben tárolt adatok.

6.3.2. Navigáció

A navigáció megvalósítására szolgáló komponens NavHostFragment. A NavHostFragment a fragmentek konténere, amelyek között a felhasználó navigálhat az alkalmazásban, gondoskodik hogy a megfelelő fragmentek jelenjenek meg a megfelelő időben a felhasználó navigációja alapján. A NavHostFragment a navigációs gráfot tartalmazza, amely definiálja a lehetséges navigációs útvonalakat és a hozzájuk tartozó fragmenteket. Ez a navigációs gráf lehetővé teszi a NavHostFragment számára, hogy dinamikusan vezérelje a fragmentek közötti navigációt. A navigációs gráfot XML fájlban definiáljuk, és az alkalmazás inicializálásakor társítjuk a NavHostFragmenthez. A NavHostFragment támogatja a visszavigázást az alkalmazásban. Ez azt jelenti, hogy az alkalmazásban a felhasználó a vissza gombot megnyomva visszaléphet az előzőleg megjelenített fragmentre. A navigációs gráf a 6.2 kódrészletben látható.

```

<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/mobile_navigation"
    app:startDestination="@id/navigation_login">

    <fragment
        android:id="@+id/navigation_login"
        android:name="edu.sapientia.pathplanner.mobile.ui.login.LoginFragment"
        tools:layout="@layout/fragment_login" />
    <fragment
        android:id="@+id/navigation_route_list"
        android:name="edu.sapientia.pathplanner.mobile.ui.routes.
            RouteListFragment"
        android:label="@string/title_route_list"
        tools:layout="@layout/fragment_route_list" />
    <fragment
        android:id="@+id/navigation_route_details"
        android:name="edu.sapientia.pathplanner.mobile.ui.routes.
            RouteDetailsFragment"
        android:label="@string/title_route_details"
        tools:layout="@layout/fragment_route_details" />
    <fragment
        android:id="@+id/navigation_suggestion_add"
        android:name="edu.sapientia.pathplanner.mobile.ui.suggestions.
            SuggestionAddFragment"
        android:label="@string/title_add_suggestion"
        tools:layout="@layout/fragment_suggestion_add" />
    <fragment
        android:id="@+id/navigation_suggestion_list"
        android:name="edu.sapientia.pathplanner.mobile.ui.suggestions.
            SuggestionListFragment"
        android:label="@string/title_suggestion_list"
        tools:layout="@layout/fragment_suggestion_list" />
    <fragment
        android:id="@+id/navigation_suggestion_details"
        android:name="edu.sapientia.pathplanner.mobile.ui.suggestions.
            SuggestionDetailsFragment"
        android:label="@string/title_suggestion_details"
        tools:layout="@layout/fragment_suggestion_details" />
    <fragment
        android:id="@+id/navigation_profile"
        android:name="edu.sapientia.pathplanner.mobile.ui.profile.
            ProfileFragment"
        android:label="@string/title_profile"
        tools:layout="@layout/fragment_profile" />
</navigation>

```

6.2. kódrészlet. Navigációs gráf állománya

6.3.3. Szerver oldallal való kommunikáció

A szerveroldal és a mobil alkalmazás közötti kommunikáció HTTP kérésekkel, REST API segítségével van megvalósítva. A könnyű és hatékony kommunikációt a szerverrel a Retrofit, egy HTTP kliens könyvtár teszi lehetővé. A Retrofit segítségével a kliens alkalmazás egyszerűen definiálhatja az API végpontokat, és a Retrofit automatikusan generálja a szükséges hálózati kéréseket (GET, POST, stb.) a szerver felé. A Retrofit támogatja az adatok aszinkron letöltését és feltöltését, valamint a hibakezelést és a hálózati hibák kezelését. A szerver és kliens közötti adatcsere JSON formátumban van elvégezve. A Moshi egy JSON feldolgozó könyvtár, amely segít az adatok átalakításában és visszaalakításában a JSON formátum és a Kotlin objektumok között. A Moshi egyszerű használata és hatékony séma támogatása lehetővé teszi a JSON adatok könnyű feldolgozását és kezelését. A Retrofit példányok létrehozása a 6.3 kódrészletben van szemléltetve.

```
object ApiClient {  
  
    private val moshi = Moshi.Builder()  
        .add(KotlinJsonAdapterFactory())  
        .build()  
  
    private val retrofit: Retrofit = Retrofit.Builder()  
        .addConverterFactory(MoshiConverterFactory.create(moshi))  
        .baseUrl(BASE_URL)  
        .build()  
  
    private val googleConnection: Retrofit = Retrofit.Builder()  
        .addConverterFactory(MoshiConverterFactory.create(moshi))  
        .baseUrl(GOOGLE_API_URL)  
        .build()  
  
    val userApi: UserApi by lazy{  
        retrofit.create(UserApi::class.java)  
    }  
  
    val cityApi: CityApi by lazy{  
        retrofit.create(CityApi::class.java)  
    }  
  
    val googleApi: GoogleApi by lazy{  
        googleConnection.create(GoogleApi::class.java)  
    }  
}
```

6.3. kódrészlet. Retrofit objektumok létrehozása

A *retrofit* objektum valósítja meg a szerveroldallal, míg a *googleConnection* objektum pedig a Google Directions API-val a kapcsolatot.

6.3.4. Korutinok

A korutinok a Kotlin nyelvben lehetővé teszik az aszinkron programozást. A korutinok lehetővé teszik az egyszerű, strukturált és hatékony aszinkron kódírást, amely könnyebbé teszi a hálózati kérések, adatbázis műveletek vagy más hosszabb ideig tartó műveletek kezelését az alkalmazásban. A coroutines lehetővé teszik a konkurens műveletek kezelését anélkül, hogy szálakat kellene kezelni, a strukturált aszinkron kódírást. A coroutinesokat blokkokba vagy függvényekbe helyezhetők, amelyek könnyen olvashatók és érthetők. A coroutines lehetővé teszik a függvények felfüggesztését és folytatását anélkül, hogy blokkolnák a fő szálát. A coroutines használata a ViewModelen belül, a ViewLifecycle segítségével történik. A ViewModel inicializálásakor a ViewModelScope létrejön, ami egy CoroutineScope. A ViewModelScope automatikusan figyeli a ViewModel élettartamát és a hozzá tartozó UI komponenseket az élettartam változásaira, így coroutinesok élettartama megegyezik a ViewModel élettartamával. Ez segít elkerülni a memóriaproblémákat és a szivárgásokat. A coroutinesokban a suspend kulcsszóval ellátott függvényeket használjuk. A suspend függvények olyan függvények, amelyek felfüggeszthetők, és várakozhatnak más függvények végrehajtására anélkül, hogy blokkolnák a szálakat. A 6.4 kódrészletben látható egy példa suspend típusú függvényre, amelyet kötelező korutimból meghívni.

```
interface CityApi {
    @GET(Constants.CITIES_URL)
    suspend fun getCities(): Response<List<PartialCity>>

    @GET(Constants.CITY_URL)
    suspend fun getCity(@Path("id") id: String, @Query("vehicleTypes")
        vehicleTypes: List<String>?, @Query("days") days: List<Int>?,
        @Query("times") times List<Int>?): Response<City>
}
```

6.4. kódrészlet. Példa suspend függvényekre

6.3.5. Útvonaltervezés

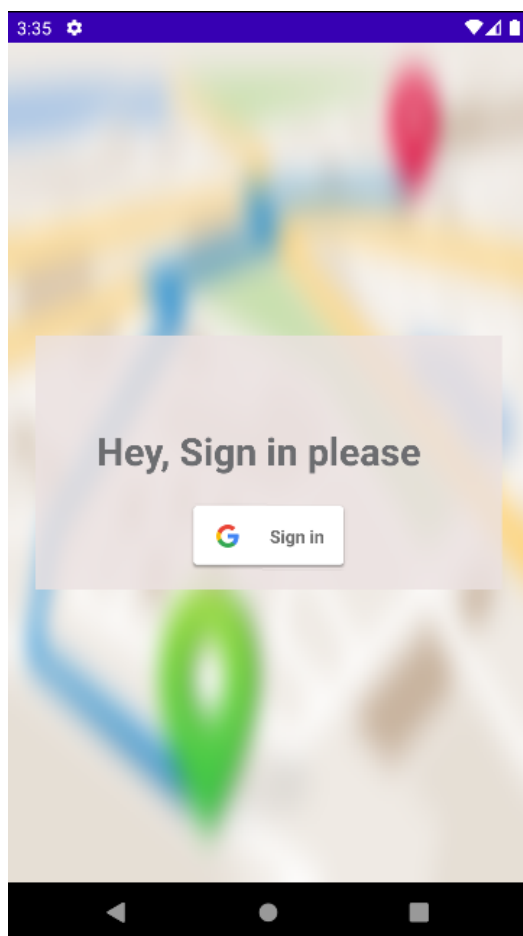
Az útvonalak lekérdezését és tervezését két helyszín között a Google Directions API szolgáltatás teszi lehetővé. Az API hozzáférést biztosít a Google térkép és navigációs adatbázisához és számos funkcionalitást kínál az útvonalak lekérdezéséhez, például a leggyorsabb útvonal. Az alkalmazásnak HTTP kéréseket kell küldenie a Google Directions API felé az útvonal lekérdezéséhez. A kérés tartalmazza a kiindulási és célhely információkat, valamint egy API kulcsot, amely azonosítja az alkalmazást a Google szerverek felé. A Google Directions API válaszként egy JSON formátumú adatobjektumot küld vissza az útvonallról. Ez az objektum tartalmazza az útvonal részleteit, például a lépések listáját, a távolságokat, időtartamokat, koordinátákat és más navigációs információkat. Ezután fel kell dolgozni a kapott választ és kinyerni azokat az adatokat, amelyekre szükség van az útvonal megjelenítéséhez.

6.4. Az alkalmazás működése

Ebben az alfejezetben alkalmazás működésének részletes bemutatása következik.

6.4.1. Kezdőoldal

Az alkalmazás megnyitásakor splash screen jelenik meg és ezalatt eldönti az alkalmazás, hogy be van-e jelentkezve a felhasználó vagy sem a token alapján. Ha nincs, akkor a bejelentkezési oldalra dobja (6.8 ábra), ahol Google fiókkal kell bejelentkezzen, ha ez sikeresen megtörtént vagy a token érvényes volt, akkor már használhatja is az alkalmazást.

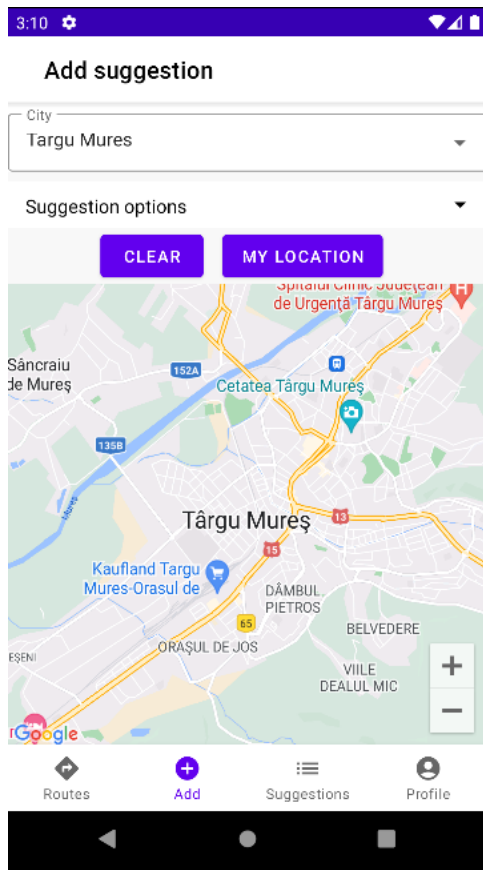


6.8. ábra. Kezdőoldal

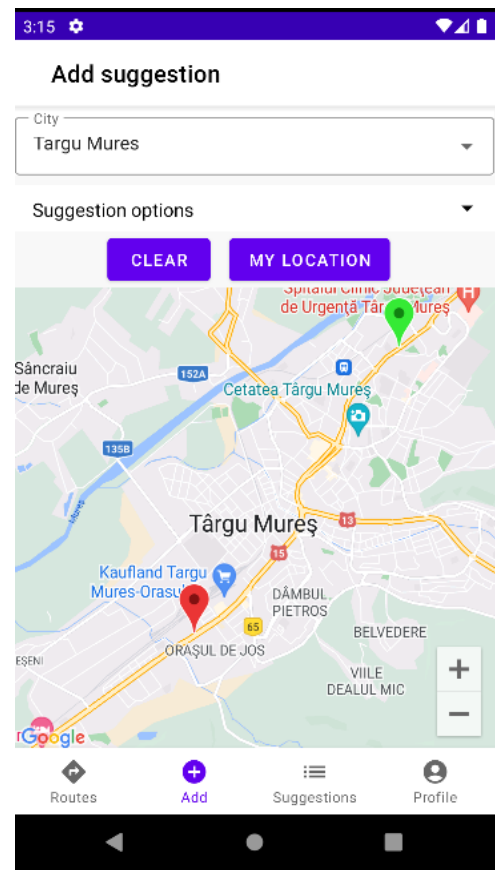
6.4.2. Új felhasználói ajánlás hozzáadása

Új ajánlás hozzáadása (6.9 ábra) esetében a felhasználó először kiválasztja a várost, majd a térképen megadja a kezdő- és a végpontot (6.10 ábra). A tartózkodási helyet is lehet használni az egyik végpont megadására. A megadott helyzetet vagy helyzeteket vissza is lehet vonni a *Clear* gomb megnyomásával. Ezután a *Suggestion optionsben* (6.11 ábra) meg kell adni, hogy busz vagy bicikli út legyen-e az ajánlás. Ha busz, akkor még

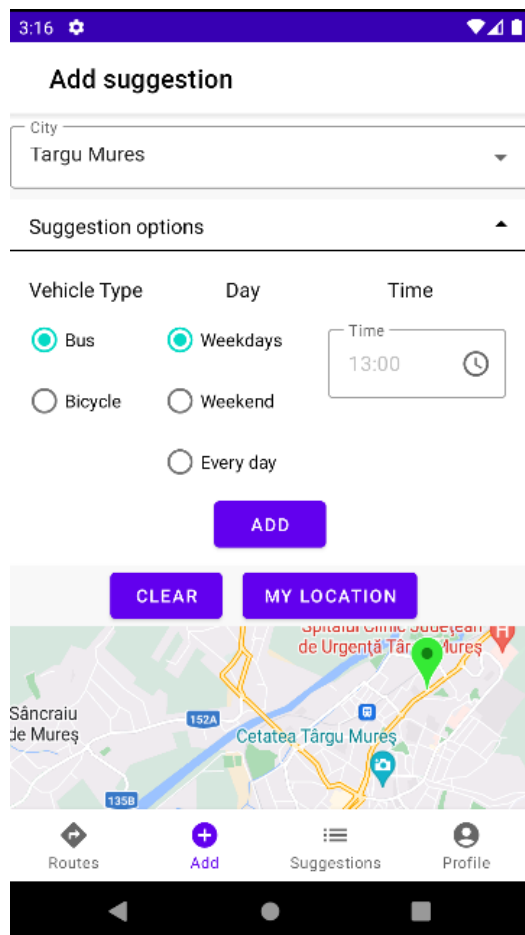
meg kell adni, hogy hétköznap, hétvégén vagy minden nap működjön a járat és az időponto(ka)t. Az ajánlás elküldését az *Add* gomb megnyomásával lehet megtenni, utána egy toast message jelenik meg, amiből meg lehet tudni, hogy sikeresen el lett-e mentve az ajánlás vagy sem. Ha nincs megadva minden szükséges adat, erről is érkezik toast üzenet. Az új ajánlás hozzáadása szekvencia diagramot szemlélteti a 6.12 ábra.



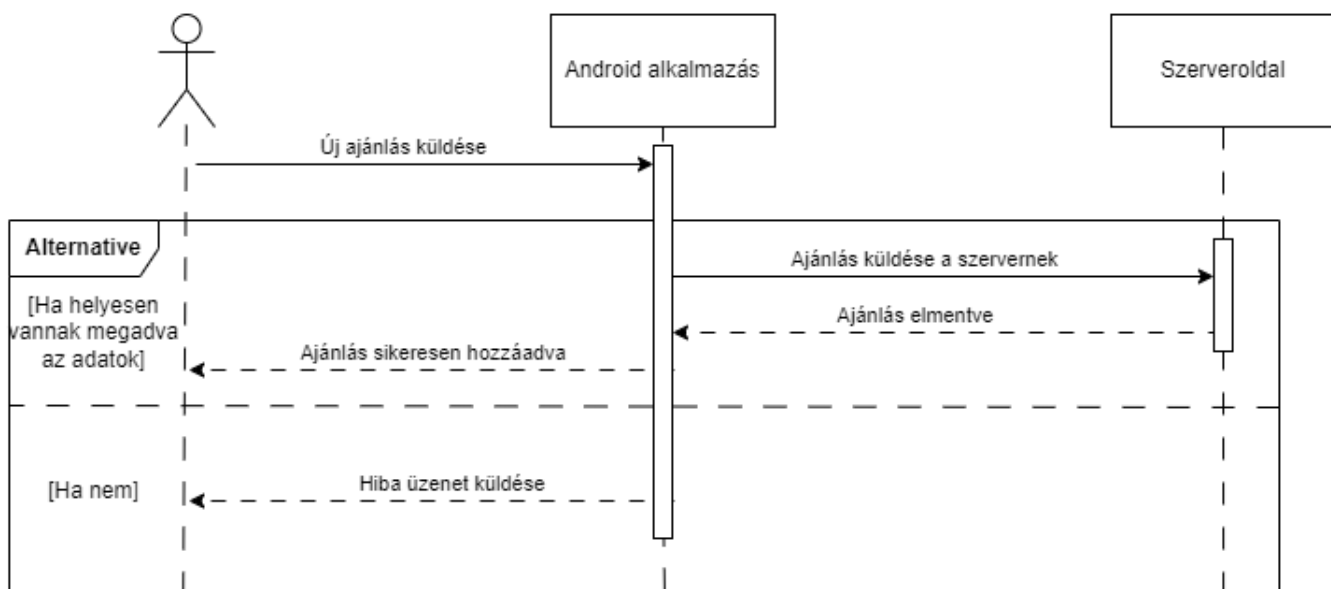
6.9. ábra. Új ajánlás



6.10. ábra. Induló- és végpont bejelölése



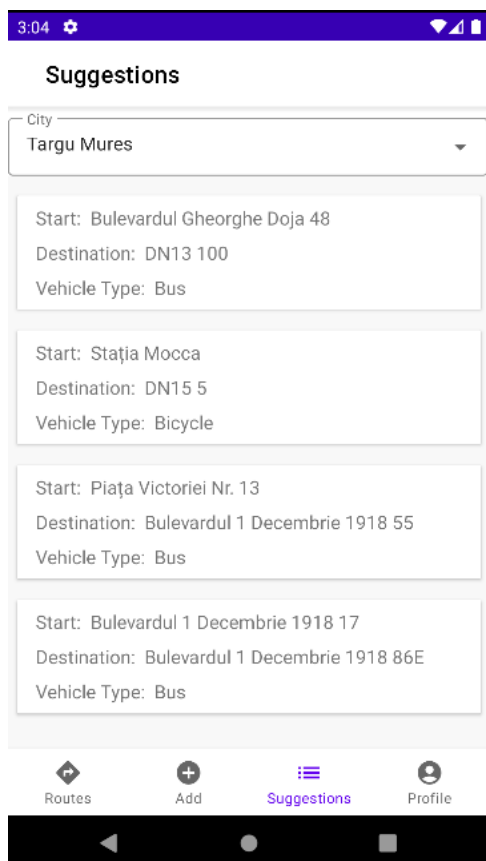
6.11. ábra. Ajánlási opciók megadása



6.12. ábra. Szekvencia diagram új ajánlás hozzáadására

6.4.3. Felhasználói ajánlások megtekintése

Felhasználói ajánlások megtekintése (6.13 ábra), ahol először ki kell választani, hogy melyik városhoz tartozó ajánlásokat kell megjeleníteni, kezdetben az összes megjelenik. Az induló-, végpont címe, illetve az útvonal típusa lesz látható minden esetben. Egy ajánlás kiválasztása során egy új oldalra fog navigálni az alkalmazás, ahol a térképen pontosan megjelenik az útvonal és más részletek ezzel kapcsolatosan (6.14 ábra).



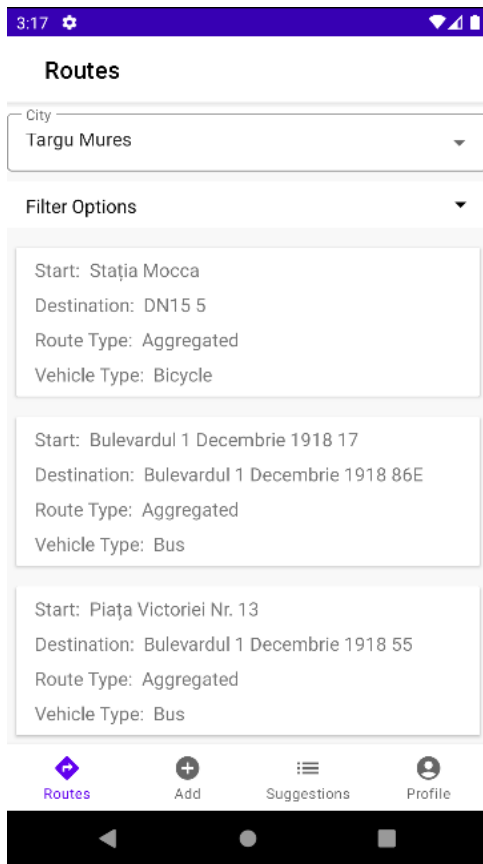
6.13. ábra. Ajánlások



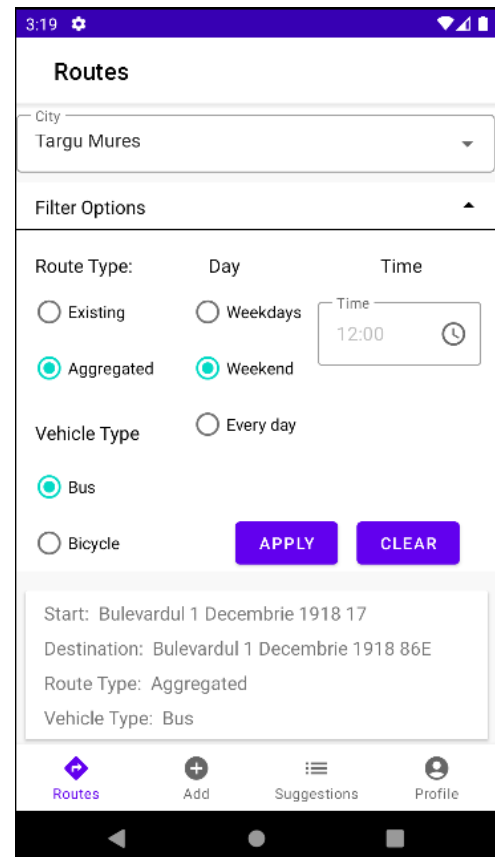
6.14. ábra. Részletek megjelenítése

6.4.4. Létező és aggregált útvonalak listázása

Létező és aggregált útvonalak listája (6.15 ábra), először ki kell választani, hogy melyik városhoz tartozó már létező útvonalakat vagy aggregált útvonalakat szeretné a felhasználó kilistázni. A *Filter* menü lenyitásával (6.16 ábra) kiválasztható, hogy a már létező vagy aggregált útvonalak legyenek listázva. Itt lehet kiválasztani az útvonal típusát és a típushoz tartozó egyéb opciókat. Az *Apply* gomb megnyomásával a megadott szűrők szerint jelennek meg az útvonalak, illetve a *Clear* gomb segítségével törölhetők az aktív szűrők. Az útvonalak esetében meg fog jelenni, az induló- és végpont címe, letező vagy aggregált útvonalak-e, illetve a típusuk. Egy útvonal kiválasztásával egy új oldalon a térképen meg fog jelenni a pontos útvonal és az útvonalhoz tartozó részletek.



6.15. ábra. Útvonalak



6.16. ábra. Szűrők a listázáshoz

6.4.5. Profil

A profil oldalon (6.17 ábra) a felhasználók megtekinthetik az adataikat és lehetőség van ezek szerkesztésére az *Edit Profile* gomb megnyomásával. A szerkesztés után toast message-ként megjelenni, hogy sikeres volt-e a változtatás vagy hogy jól vannak-e kitöltve az adatok. Továbbá, itt választható ki, hogy a többi oldalon melyik város legyen alapértelmezetten betöltve. Ehhez, egyszerűen csak ki kell választani a menüből a várost és érkezni fog a toast message, hogy meg lett változtatva az alapértelmezett város. A *Logout* gomb megnyomásával pedig a felhasználó kijelentkezik és a kezdőoldalra kerül.

3:02

Profile

First Name
Róbert

Last Name
Kacsó

Nickname
Robi

Email
kacsorobert01@gmail.com

City
Targu Mures

EDIT PROFILE

LOG OUT

Routes Add Suggestions Profile

6.17. ábra. Profil

7. fejezet

Eszközök és munkamódszerek

7.1. Segédeszközök

7.1.1. Integrált fejlesztői környezetek

7.1.1.1. Szerveroldal és webkliens fejlesztése

A szerveroldal és webkliens fejlesztéséhez IntelliJ IDEA-t használtam, ami egy integrált fejlesztői környezet (IDE), amelyet a JetBrains fejlesztett ki. Az IntelliJ IDEA lehetővé teszi a projektkezelés és az építési rendszerek, mint például a Gradle, könnyű integrálását, kiváló kód navigációt és keresést kínál, amely lehetővé teszi a gyors és hatékony munkavégzést. Rendelkezik kódkiegészítési funkciókkal, támogatja a verziókezelő rendszereket, például a Git-et.

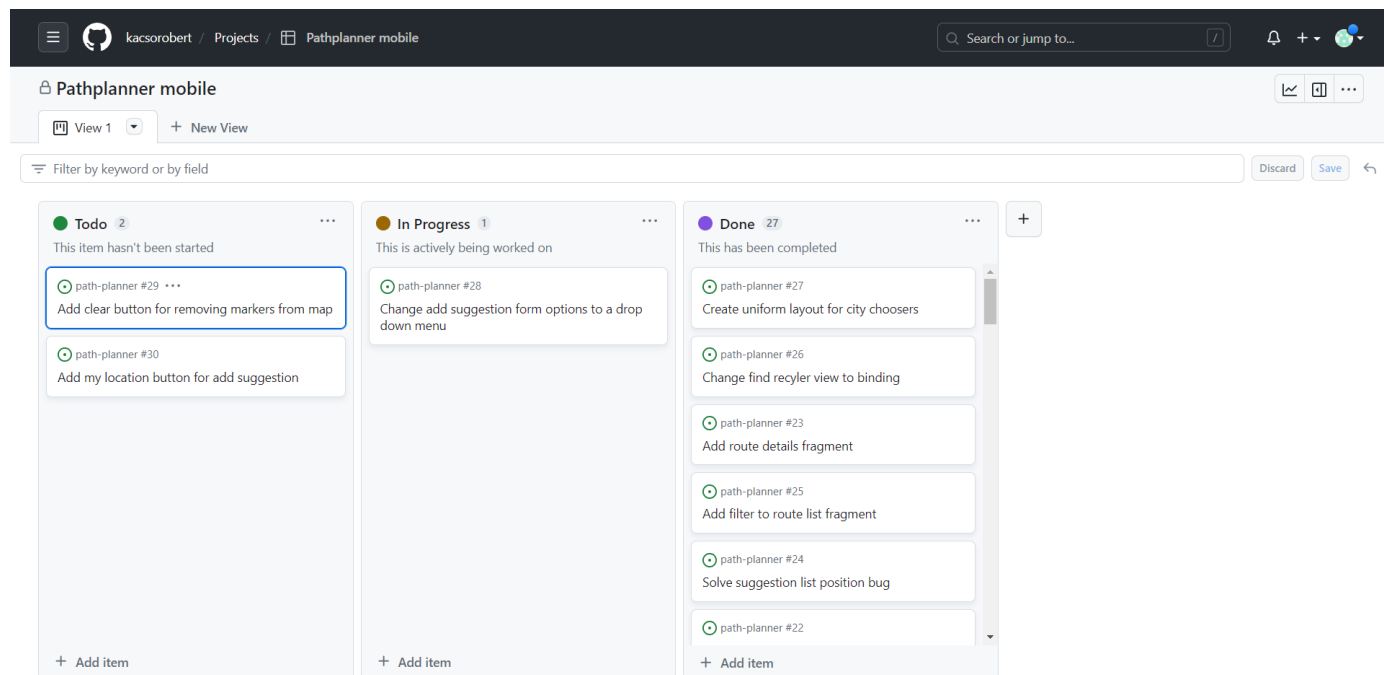
7.1.1.2. Mobil alkalmazás fejlesztése

A mobilkliens fejlesztéséhez az Android Studiot használtam. Az Android Studio a Google Android operációs rendszerének hivatalos integrált fejlesztőkörnyezete (IDE), amely a JetBrains IntelliJ IDEA szoftverére épül, és amelyet kifejezetten Android-fejlesztésre terveztek. Az Android Studio kódszerkesztője fejlett funkciókkal rendelkezik, mint például kódkiegészítés, automatikus formázás, hibakeresés és refaktorálás. Beépített XML Layout Editorral rendelkezik, amely segíti az UI (felhasználói felület) tervezést és szerkesztést. Az eszköz lehetővé teszi a vizuális elemek hozzáadását és elrendezését, valamint az attribútumok beállítását. Az Android Studio lehetővé teszi az Android emulátorok vagy fizikai eszközök használatát az alkalmazások teszteléséhez és hibakereséséhez. Az IDE beépített eszközkezelője segítségével könnyedén telepíthetünk, futtathatunk és kezelhetünk emulátorokat vagy eszközöket. A beépített Git integráció lehetővé teszi a könnyű és hatékony verziókezelést.

7.1.2. Verziókövetés eszközök

A Git rendszerrel történt verziókövetés lehetővé tette számomra, hogy nyomon kövessem az alkalmazás fejlesztésének előrehaladását. A Git lehetőséget nyújt új ágak (branch-ek), commitok létrehozására, különbségek áttekintésére a különböző verziók között. A GitHub Project Kanban tábla (7.1 ábra) egy nagyszerű eszköz a projekt feladatainak kö-

vetésére és szervezésére. A Kanban táblán különböző oszlopok vannak létre hozva, amelyek megfelelnek a feladatok különböző állapotainak, például "Elkezdésre vár", "Fejlesztés alatt", "Kész". Minden feladathoz kártyákat lehet rendelni, amelyeket az oszlopok között mozgathatóak a feladatok előrehaladása szerint. A kártyákhoz megjegyzéseket és címkéket is hozzá lehet adni. Ezáltal könnyen látható, mely feladatok vannak folyamatban, melyek vannak készen, és melyek vannak még hátra.



7.1. ábra. GitHub Kanban tábla

7.1.3. API tesztelés

A Postman segítségével létre lehet hozni és elküldeni a HTTP kéréseket a backend endpointjaihoz. Lehetőség van beállítani a kérés típusát (GET, POST, stb.), fejlécek, paraméterek és a kérés body részének hozzáadására. A kérésre érkező választ meg lehet tekinteni az eredmények panelen, amely tartalmazta a státuszkódot, a válasz fejléceit és a válasz törzs részét is.

7.1.4. Design eszközök

A drótvázak elkészítéséhez MockFlow-t használtam. A MockFlow egy online tervezőeszköz, amelyet használható interaktív drótvázak, prototípusok és diagramok létrehozására. A MockFlow egy intuitív és könnyen használható eszköz, amely az alkalmazások, weboldalak felhasználói felületének gyors és hatékony tervezésében segít. Gazdag eszközkészletet tartalmaz, amely segítségével elemeket és widgeteket lehet hozzáadni a tervezőfelülethez, például nyomógombokat, beviteli mezőket, menüket stb. Ezenkívül lehetőség van elhelyezni és csoportosítani az elemeket, illetve definiálni azok közötti kapcsolatokat.

7.2. A szerveroldal tesztelése

A tesztek célja, hogy ellenőrizzék a szoftver működését, biztosítsák a kód minőségét, és megbizonyosodjanak arról, hogy az alkalmazás megfelel a tervezett követelményeknek és funkcionalitásnak. A JUnit 5 egy népszerű Java alapú keretrendszer a tesztek írásához és végrehajtásához. A JUnit 5 lehetővé teszi a fejlesztők számára, hogy egységteszteket, integrációs tesztek és teljes körű végpont-teszteket hozzanak létre és futtassanak. A következő típusú teszteléseket végeztem: egységtesztek, integrációs tesztek és teljes körű végpont-tesztek.

A JUnit 5 tesztek írásához a *@Test* annotációt használjuk a tesztmetódusok jelölésére. Ezek a metódusok az adott funkcionalitás tesztelésére szolgálnak, és különböző bemenetekkel és elvárt kimenetekkel vannak meghatározva. A tesztmetódusokban használhatunk különböző assert függvényeket az elvárt értékek összehasonlítására, például az *assertEquals()*, *assertTrue()*, *assertNotNull()* stb. Az AssertJ könyvtár pedig egy továbbfejlesztett assert API-t nyújt a JUnit 5 felett, amely könnyebb és olvashatóbb tesztek írását teszi lehetővé.

Az egységtesztek célja az adott egység, például egy osztály vagy metódus működésének tesztelése elszigetelt környezetben, különféle bemenetekre és várható eredményekre. Az egységtesztek gyorsan futnak, és a fejlesztők számára segítenek az egyes komponensek helyes működésének ellenőrzésében. A 7.1 kódrészlet példa egységtesztekre.

```
@ExtendWith(MockitoExtension.class)
class UserServiceUnitTest {

    @Mock
    private UserRepository userRepository;

    @InjectMocks
    private UserServiceImpl userService;

    @Test
    void getUserByIdWhenFound() {
        //given
        User user = new User();
        user.setId("1");
        user.setFirstName("Firstname");
        user.setLastName("Lastname");
        user.setEmail("email@email.com");

        given(userRepository.findById(user.getId())).willReturn(Optional
            .of(user));

        // when
        User savedUser = userService.getUserById(user.getId());

        // then
        assertThat(savedUser).isEqualTo(user);
    }
}
```

```

@Test()
void getUserByIdWhenNotFound() {
    //given
    User user = new User();
    user.setId("1");
    user.setFirstName("Firstname");
    user.setLastName("Lastname");
    user.setEmail("email@email.com");

    given(userRepository.findById(user.getId())).willReturn(Optional
        .empty());

    // then
    assertThatThrownBy(() -> userService.getUserById(user.getId()))
        .assertInstanceOf(ServiceException.class)
        .hasMessage("User not found.");
}
}

```

7.1. kódrészlet. Példa egységtesztre

Az integrációs tesztek célja az alkalmazás komponenseinek együttműködésének és az interfészek helyes működésének ellenőrzése. Ezek a tesztek általában valós környezetben futnak, például az adatbázis vagy külső rendszerek integrációját tesztelik. Az integrációs tesztek segítenek az alkalmazás különböző részeinek összekapcsolásában és a rendszer egészének tesztelésében. A 7.2 kódrészlet példa integrációs tesztre.

```

@SpringBootTest
class CityServiceIntegrationTest {

    @Autowired
    private CityRepository cityRepository;

    @Autowired
    private CityService cityService;

    @Test
    void createCity() {
        //given
        City city = new City();
        city.setId("1");
        city.setName("City");
        Coordinate coordinate = new Coordinate();
        coordinate.setX(0.0);
        coordinate.setY(0.0);
        city.setCenter(coordinate);

        //when
        cityService.createCity(city);
    }
}

```

```

        //then
        City savedCity = cityRepository.findById(city.getId()).get();
        assertThat(savedCity).isEqualTo(city);
    }
}

```

7.2. kódrészlet. Példa integrációs tesztre

A végpont-tesztek (end-to-end tesztek) a teljes alkalmazás működését tesztelik, beleértve a kliens és a szerver közötti kommunikációt is. Ezek a tesztek valós környezetben futnak, és átfogóan ellenőrzik az alkalmazás működését a felhasználói szemszögből. A végpont-tesztek segítenek az alkalmazás teljesítményének, skálázhatóságának és megbízhatóságának ellenőrzésében. A 7.3 kódrészlet példa végpont tesztelésre.

```

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
class CityControllerEnd2EndTest {

    private static RestTemplate restTemplate;

    @Autowired
    private ServletWebServerApplicationContext webServerApplicationContext;

    private String baseUrl = "http://localhost:%d/cities";

    @BeforeAll
    static void beforeAll() {
        restTemplate = new RestTemplate();
    }

    @Test
    void getCitiesWhenEmpty() {
        int port = webServerApplicationContext.getWebServer().getPort();
        String url = String.format(baseUrl, port);

        Iterable<PartialCityDto> httpResult = restTemplate.getForObject(url,
            Iterable.class);

        assertThat(httpResult).isEqualTo(Collections.emptyList());
    }
}

```

7.3. kódrészlet. Példa végpont tesztelésre

Összefoglaló

7.3. Következtetések

Összefoglalva, sikeresen elértem a kitűzött célokat és egy olyan felületet sikerült létrehoznom, amely tökéletesen megfelel a felhasználók elvárásainak. Az alkalmazás teljes mértékben kielégíti a felhasználói igényeket, és letisztult felhasználói felületével könnyen kezelhetővé teszi a különböző funkcionalitásokat. A városi tömegközlekedést segítő ajánlások megvalósítása során kényelmes és intuitív használatot biztosít, hogy a felhasználók könnyen megszathassák véleményüket.

Az alkalmazás fejlesztése során sikerült integrálni a kitűzött funkcionalitásokat. A felhasználók lehetőséget kapnak arra, hogy Google fiókjukkal egyszerűen és kényelmesen használják az alkalmazást, így nem szükséges külön email címet és jelszót használni. Bejelentkezéskor a felhasználó a Google fiókjával azonosítja magát. A felhasználóknak lehetőségük van új ajánlásokat tenni az alkalmazásban. Az ajánlásokat a felhasználók a saját tartózkodási helyük alapján is megadhatják, amely segít személyre szabott és releváns javaslatokat nyújtani. A felhasználóknak lehetőségük van az előző ajánlásainak listázására. Emellett a felhasználók megtekinthetik a létező vagy aggregált útvonalakat, amelyek segítségével könnyedén tervezhetnek útvonalakat a városban. Szűrőket is alkalmazhatnak az ajánlásokhoz, hogy a saját preferenciáiknak megfelelően válogathassanak az útvonalak között. Megtekinthetik a saját profiljukat az alkalmazásban. Itt szerkeszthetik a személyes adataikat, emellett beállíthatják az alapértelmezett várost.

Az alkalmazás fejlesztése során sikeresen megvalósult a szerveroldal tesztelése is, egység-, integrációs és végpont-tesztek használatával. Ezek célja az alkalmazás stabilitásának és megbízhatóságának biztosítása volt.

7.4. Továbbfejlesztési lehetőségek

Az alkalmazás jelenleg megfelelően működik, de továbbfejlesztési lehetőségek állnak rendelkezésre. Az egyik fontos fejlesztési irány a platformtámogatás kiterjesztése lenne, mivel jelenleg az alkalmazás csak Android eszközökön érhető el. Az iOS rendszerekre történő implementáció lehetővé tenné, hogy több felhasználó is élvezhesse az alkalmazás előnyeit.

Ezenkívül fontos lenne olyan funkcionalitást bevezetni, amely lehetővé teszi a felhasználók számára, hogy visszajelzéseket küldjenek az aggregált útvonalakról. Ez lehetővé tenné, hogy a felhasználók tapasztalataikat, javaslataikat megosszák, illetve, hogy mennyire használnák az adott útvonalat.

Publikus GitHub repository: <https://github.com/kacsorobert/path-planner>

Köszönetnyilvánítás

Az alkalmazás fejlesztése a 2022-es Codespring szakmai gyakorlat során indult, ezért szeretnék köszönetet mondani akkori csapattársaimnak Imecs Bulcsúnak és Barabás Csabának, mentoraimnak Balogh Szabolcsnak, Hegedüs Hunornak, illetve témavezetőmnek, Dr. Antal Margitnak.

Ábrák jegyzéke

4.1. Felhasználó használati eset diagram	23
4.2. Használati eset diagram a többi szerepkörre	24
5.1. A rendszer architektúrája	30
5.2. Android Studio mappaszerkezet	31
5.3. A felhasználói felület drótváza	33
6.1. Repository réteg osztálydiagramja	35
6.2. Service réteg osztálydiagramja	37
6.3. Controller réteg osztálydiagramja	38
6.4. Kezdő/végpontok	39
6.5. Körök alkalmazása	39
6.6. A középpontok kiemelése	40
6.7. Maradék pontok eltüntetése	40
6.8. Kezdőoldal	46
6.9. Új ajánlás	47
6.10. Induló- és végpont bejelölése	47
6.11. Ajánlási opciók megadása	48
6.12. Szekvencia diagram új ajánlás hozzáadására	48
6.13. Ajánlások	49
6.14. Részletek megjelenítése	49
6.15. Útvonalak	50
6.16. Szűrők a listázáshoz	50
6.17. Profil	51
7.1. GitHub Kanban tábla	53

Irodalomjegyzék

- [1] „Java documentation - get started *Oracle*.” <https://docs.oracle.com/en/java/>. (elérés dátuma: 2023. jún. 12.).
- [2] C. Walls, *Spring Boot in Action*. Manning, 2015.
- [3] „Spring boot *Spring*.” <https://spring.io/projects/spring-boot>. (elérés dátuma: 2023. jún. 11.).
- [4] „Spring boot - architecture - geeksforgeeks *GeeksforGeeks*.” <https://www.geeksforgeeks.org/spring-boot-architecture/>. (elérés dátuma: 2023. jún. 11.).
- [5] „16. auto-configuration *Spring*.” <https://docs.spring.io/spring-boot/docs/2.0.x/reference/html/using-boot-auto-configuration.html>. (elérés dátuma: 2023. jún. 11.).
- [6] „Spring boot - starter - geeksforgeeks *GeeksforGeeks*.” <https://www.geeksforgeeks.org/spring-boot-starters/>. (elérés dátuma: 2023. jún. 11.).
- [7] „Spring boot actuator web api documentation *Spring*.” <https://docs.spring.io/spring-boot/docs/3.1.0/actuator-api/htmlsingle/>. (elérés dátuma: 2023. jún. 11.).
- [8] „Spring data *Spring*.” <https://spring.io/projects/spring-data>. (elérés dátuma: 2023. jún. 11.).
- [9] „Spring security *Spring*.” <https://spring.io/projects/spring-security>. (elérés dátuma: 2023. jún. 11.).
- [10] „41. testing *Spring*.” <https://docs.spring.io/spring-boot/docs/1.5.2.RELEASE/reference/html/boot-features-testing.html>. (elérés dátuma: 2023. jún. 11.).
- [11] „Typescript: Documentation - typescript for the new programmer *TypeScript*.” <https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html>. (elérés dátuma: 2023. jún. 12.).
- [12] Y. Fain and A. Moiseev, *Angular Development with TypeScript*. Manning, 2016.
- [13] „Angular - what is angular? *Angular*.” <https://angular.io/guide/what-is-angular>. (elérés dátuma: 2023. jún. 11.).

- [14] „Angular - binding syntax *Angular*.” <https://angular.io/guide/binding-syntax>. (elérés dátuma: 2023. jún. 11.).
- [15] „Angular - introduction to modules *Angular*.” <https://angular.io/guide/architecture-modules>. (elérés dátuma: 2023. jún. 11.).
- [16] „Angular - angular components *Angular*.” <https://angular.io/guide/component-overview>. (elérés dátuma: 2023. jún. 11.).
- [17] „Angular - template syntax *Angular*.” <https://angular.io/guide/template-syntax>. (elérés dátuma: 2023. jún. 12.).
- [18] „Angular - introduction to services and dependency injection *Angular*.” <https://angular.io/guide/architecture-services>. (elérés dátuma: 2023. jún. 11.).
- [19] „Angular - angular routing *Angular*.” <https://angular.io/guide/router>. (elérés dátuma: 2023. jún. 11.).
- [20] „Angular - structural directives *Angular*.” <https://angular.io/guide/structural-directives>. (elérés dátuma: 2023. jún. 12.).
- [21] „Get started with kotlin | kotlin documentation *Kotlin*.” <https://kotlinlang.org/docs/getting-started.html>. (elérés dátuma: 2023. jún. 12.).
- [22] „Platform architecture | android developers. *Android Developers*.” <https://developer.android.com/guide/platform>. (elérés dátuma: 2023. jún. 12.).
- [23] „Application fundamentals | android developers *Android Developers*.” developer.android.com/guide/components/fundamentals.html#Components. (elérés dátuma: 2023. jún. 12.).
- [24] „What is gradle? *Gradle*.” docs.gradle.org/current/userguide/what_is_gradle.html. (elérés dátuma: 2023. jún. 12.).