

Pannon Egyetem

Műszaki Informatikai Kar

Rendszer- és Számítástudományi Tanszék

Gazdaságinformatikus BSc

SZAKDOLGOZAT

Veszprémi tömegközlekedést támogató okostelefon

alkalmazás

Böröndi Evelin

Témavezető: Dr. Hegyháti Máté

2017

KÖSZÖNETNYILVÁNÍTÁS

...

TARTALMI ÖSSZEFOGLALÓ

A nagyvárosok bonyolult és szerteágazó tömegközlekedése összetett hálózatot alkotva húzódik végig a város különböző pontjait érintve. A tömegközlekedéshez tartozó útvonalak, megállók és indulási idők sokaságán az tud igazán kiigazodni, aki hosszabb ideje használja már. A város által nyújtott közösségi közlekedés használatához szükség lehet egy olyan eszközre, amely segítségével az utazóközönség gyorsan és kényelmesen szerez információt, ezzel eljutva céljukhoz. Ez a lehetőség főleg a turistáknak, a várost még nem ismerő személyeknek nyújthat nagy segítséget.

Veszprémben jelenleg a tömegközlekedést autóbuszjáratai szolgáltatják. Bár a megyeszékhely nem tartozik a legnagyobb városok közé Magyarországon, menetrendje ennek ellenére mégis szerteágazó. Emellett a városban sok olyan ember megfordul, akik nem ismerik ki magukat Veszprémben, például turisták illetve egyetemisták. Számukra nagy hátrányt jelent, hogy nincsenek tisztában a buszmegállók elhelyezkedéséről, de sok esetben a saját pozíciójukat sem ismerik. A városban egyelőre nem található olyan szolgáltatás, ami eleget tenne annak, hogy segítse az utazóközönséget a tájékozódásban. A jelenlegi megoldások túlságosan statikusak azok számára, akik újonnan látogatnak Veszprémbe. Igény lenne egy olyan megvalósításra, ahol térkép alapján tudnának tájékozódni a megállókról.

Munkám során erre a hiányra próbál megoldást nyújtani az Androidos platformra készült alkalmazás. Segítségével a felhasználók képesek megtervezni az utazásukat a tömegközlekedés bevonásával.

Kulcsszavak: tömegközlekedés, Android, Veszprém, útvonaltervezés

ABSTRACT

Angol tartalmi összefoglaló...

Keywords: public transport, Android, Veszprém, route planning

Tartalomjegyzék

1. Bevezetés	6
2. Veszprém tömegközlekedése	8
2.1. Az útvonaltervezés és a Google Maps	8
2.2. Veszprém tömegközlekedése	10
2.3. Jelenlegi megoldások	11
3. Követelmények, technológiák	14
3.1. Követelmények	14
3.2. Funkcionális követelmények	15
3.3. Felhasznált technológiák	16
4. Felhasználói kézikönyv	19
4.1. Android alkalmazás	19
4.1.1. Főmenü	19
4.1.2. Útvonaltervezés	20
4.1.3. Menetrendek	23
4.1.4. Megállók	24
4.1.5. Kedvencek	24
4.2. Admin oldal	25
5. Fejlesztői dokumentáció	26
5.1. A szerver	26
5.2. Szerver feladatai	29
5.3. Az Android kliens	31
6. Továbbfejlesztési lehetőségek	39
7. Összefoglalás	41

Ábrák jegyzéke

2.1. Google Maps útvonaltervezés	9
2.2. Élő útvonalinformáció	9
2.3. Veszprém tömegközlekedési hálózata	10
2.4. Az 1-es buszhoz tartozó menetrend táblázat	11
2.5. Az 1-es és a 4-es járat útvonala	12
2.6. Az alkalmazás menüje	13
3.1. A Google Maps és az OSM összehasonlítása	16
4.1. Az alkalmazás ikonja	19
4.2. Főmenü	20
4.3. Helység keresése az applikációban	21
4.4. Útvonaltervezés saját pozícióval	21
4.5. A találati képernyő és az térképes útvonalterv	22
4.6. Menetrendek	23
4.7. Megállók és Kedvencek	24
5.1. Az autentikáció folyamata	27
5.2. A járat entitás kapcsolatai	30
5.3. A járathoz kapcsolatai	30
5.4. Különleges és indulási időpontok közötti kapcsolat	31
5.5. Google Maps útvonaltervezés	36

1. fejezet

Bevezetés

A nagyvárosokban található tömegközlekedés bonyolult és szerteágazó módon húzódik végig a város különböző pontjait érintve. A tömegközlekedéshez tartozó útvonalak, megállók és indulási idők sokaságán az tud igazán kiigazodni, aki hosszabb ideje használja már. A városba „idegenként” érkezők, turisták számára szükséges lehet egy olyan eszköz, melynek segítségével eljutnak a céljukhoz, gyorsan és kényelmesen tudják használni a város nyújtotta közösségi közlekedést.

Veszprémben jelenleg a közlekedés e fajtáját az autóbuszok szolgálják ki. Bár a megyeszékhely nem tartozik a legnagyobb városok közé Magyarországon, mégis szerteágazó menetrendet tudhat magáénak. Emellett a városban gyakran fordulnak meg egész évben turisták, egyetemisták, akik számára a legnagyobb hátrány, hogy nem ismerik a buszmegállókat, esetlegesen a saját helyzetüket sem. Egyelőre még nem található olyan szolgáltatás városunkban, ami eleget tenne annak, hogy segítse az utazóközönséget a tájékozódásban. Jelenleg több, a veszprémi tömegközlekedést segítő megoldással is találkozhatunk. Ezen megvalósítások ugyanakkor túlságosan is statikusak egy újonnan a városba látogató számára. Egyik ilyen fennálló probléma az, hogy megtudja az utas egy adott megállóban milyen buszok állnak meg, végig kell néznie az egész menetrendet, továbbá nem szolgálnak vizuális visszajelzéssel, azaz ugyan ismeri a megálló nevét, azonban nem tudja pontosan meghatározni a város mely területén található. Ilyen helyzetekben igény lenne egy olyan megoldásra, hogy térkép alapján is tudjanak tájékozódni, viszont a jelenlegi megoldások közül egyik sem felel meg az előzőekben felállított igényeknek a kielégítésére. Erre a fennálló problémára nyújt megoldást a szakdolgozatomban megvalósított Android alkalmazás, amely hordozható, megbízható és gyors formában tájékoztatja az utazni kívánókat.

A 2. fejezetben bemutatom az útvonaltervezést, mint szolgáltatást, illetve a városban működő jelenlegi megoldásokat. A 3. fejezetben ismertetem a program szükséges funkcióit, és az azokkal szemben támasztott követelményeket. A 4. fejezetben bemutatom az elkészült alkalmazást

felhasználói szinten. Az 5. fejezetben kitérek a fejlesztésre részletesebben, az alkalmazás egyes részegségeire és azok implementációjára. Végül a 6. fejezetben felvázolok pár elképzelést az alkalmazás továbbfejlesztéséhez.

2. fejezet

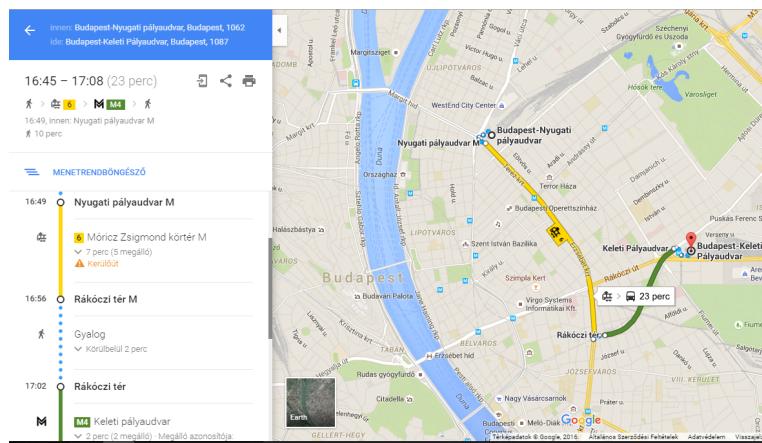
Veszprém tömegközlekedése

A fejezetben ismertetésre kerül az útvonaltervezés, mint szolgáltatás. Sok weboldal és telefonos alkalmazás segítségével tudjuk az utunkat előre megtervezni, továbbá különböző egyedi funkciókat is igyekeznek fejleszteni, ezzel csalogatva magukhoz a felhasználókat. A fejezet első részében az útvonaltervezést fogom bemutatni pár népszerű alkalmazáson keresztül, amely sikeresen elégíti ki az utazni vágyók igényeit. Ezután áttekintést adok Veszprém tömegközlekedéséről, hogy átfogó képet adjak a város közlekedési helyzetéről. Végül pedig bemutatom a jelenleg is a piacon lévő megoldásokat, amik a városban való tájékozódást segítik.

2.1. Az útvonaltervezés és a Google Maps

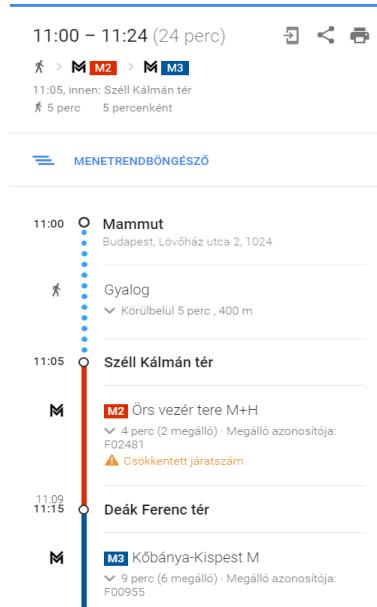
Útvonaltervezők nevezik az olyan szoftvereket, amelyek két földrajzi pont között keresnek optimális útvonalat egy keresőmotor segítségével. Ezen motorok gyakran intermodális működések. Már az 1970-es évektől használják a támogatás ezen fajtáját. Akkoriban ez annyit jelentett, hogy egy terminálos felhasználói interfészen keresztül csatlakozott a hívóközponthoz, és onnan érdeklödtek meg a tömegközlekedéssel kapcsolatos információkat. Miután elterjedt az a szokás az emberek között, hogy maguknak terveztek meg a nyaralásokat, és nem vették igénybe az utazási irodák ügynökeit, elkezdtek fejlődni az interneten elérhető útvonaltervezők.

A tervezők ebbe a fajtájába tartozik az akkoriban Google Transitként ismert útvonaltervező, ami napjainkban a Google Maps [1] térképes szolgáltatás része. Az alkalmazás interneten és mobil eszközökre is elérhető, rengeteg plusz funkcióval. A térképes adatbázisát különböző partnerek segítségével szerzi be, de sok helyen (például a fejlődő országokban) a közöség frissíti a térképes adatokat. Az útvonaltervező funkció kezdetben gyalogos és autós közlekedés tervezésére volt képes, azonban 2007-ben integrálták a tömegközlekedést is az útvonaltervezésbe. Magyarországon 2011 óta kizárálag Budapesten érhető el a funkció.



2.1. ábra. Google Maps útvonaltervezés

Ahogy a 2.1. ábrán is látszik, tervezéskor beállíthatjuk a közlekedési formát, hogy éppen gyalogosan vagy tömegközlekedéssel szeretnénk igénybe venni. Gyalogos és autós közlekedéskor az elérhető járdákat és autóutakat veszi figyelembe az alkalmazás, majd az eredményt kirajzolja a térképre, esetlegesen több elérhető opció esetén a többöt szaggatott vonallal jelöli. Tömegközlekedés esetén is több lehetőséget kínál fel, ezekből több szempont alapján tudjuk kiválasztani a számunkra optimálisat. Ilyen szempont lehet a legkevesebb átszállállás, illetve legkevesebb gyaloglás. A Google Maps továbbá indulási időket is rendel a járatokhoz, így



2.2. ábra. Élő útvonalinformáció

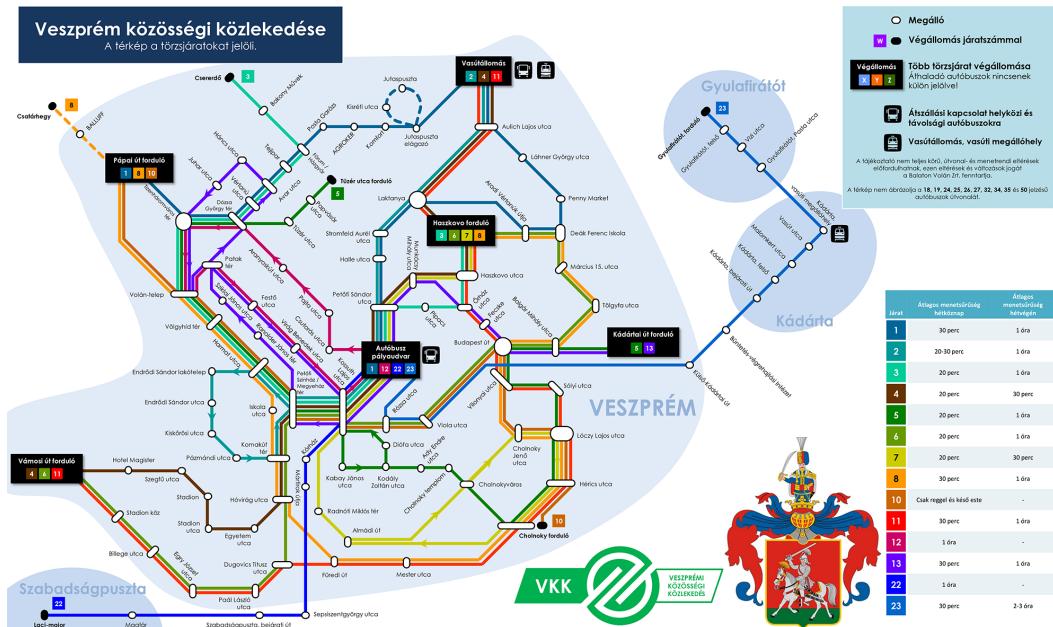
képesek vagyunk tervezni mostani időpillanathoz, illetve ha később szeretnénk csak utazni, azt is beállíthatjuk. Mindemellett rendelkezik élő menetrendinformációval, amit a Budapesti

Közlekedési Központ hivatalos oldaláról szerez be. Itt láthatjuk ha felújítás, útlezárás miatt nem közlekednek járatok, vagy más útvonalon járnak, emiatt más megállókat érintenek, ezt láthatjuk a 2.2. ábrán is.

2.2. Veszprém tömegközlekedése

Veszprém tömegközlekedését jelenleg a város méretéből és rendelkezésre álló infrastruktúrájából adódóan autóbuszok adják. A buszokat a Balaton Volán Zrt. biztosítja az 1960-as évek óta. Veszprém tömegközlekedésének gondolata azonban már 1884-ben megfogalmazódott Czollenstein Ferenc által, aki omnibuszokat indított Veszprém és Balatonalmádi között. A jelenlegi közlekedési forma 28 vonalat foglal magában, és a város belterületén kívül közlekedik a közigazgatásilag a városhoz tartozó településekhez is, úgymint Szabadságpuszta, Jutaspuszta, Kádárta és Gyulafirátót, valamint Csatár. A tömegközlekedés több átalakításra is átesett azóta, amíg el nem elérte a mai napi formáját, amit a 2.3. ábra mutat.

A menetrend integrálása lehetséges a Google Maps rendszerébe, így az útvonaltervezés funkció Veszprém városában is használható lehetne. A busztársaság részéről, egy meghatározott formátumú adatbázis továbbítása szükséges a Google felé, mivel ez csak Budapesten valósult meg, így csak a fővárosban érhető el Magyarországon belül ez a szolgáltatás.



2.3. ábra. Veszprém tömegközlekedési hálózata

2.3. Jelenlegi megoldások

Veszprémben a tömegközlekedés támogatására jelenleg is létezik több megoldás. A busztársaság is igyekszik minél kielégítőbb segítséget nyújtani az utazóközönségének, hiszen fontos számára, hogy minél többen vegyék igénybe a tömegközlekedést. Továbbá léteznek olyan harmadik fél által készült eszközök is, amelyek szintén hozzájárulnak az információszerzéshez. Ezen alkalmazásokat fejlesztőik nem profitszerzési céllal készítették el, hanem csupán önkéntes alapon, az emberek megsegítésének céljával. Funkcionalitásuk ezáltal elmarad egy céges környezetben, nagyobb fejlesztőgárda által készített szolgáltatástól, melytől nyereséget várnak a tulajdonosok. Ezek közül mutatok be pár ismertebb példát, amik jelenleg elérhetőek a piacon.

ÉNYKK

Az ÉNYKK vagyis az Észak-nyugat-magyarországi Közlekedési Központ felel a tömegközlekedés üzemeltetéséért. Weboldalukon található dokumentum magába foglalja az összes buszjáratot, és az azokhoz tartozó megállókat és indulási időket. A 2.4. ábrán látható módon szerepel egy buszjárat a dokumentumban. Ez a fajta statikus megoldás általános segítséget nyújt az utazni vágyóknak, ha már rendelkeznek információval a városról, például a megállók elhelyezkedését illetően.

1		Autóbusz-állomás – Laktanya – Vasútállomás – Jutaspuszta – Dózsa Gy. tér – Pápai úti ford.	
Menetidő 1	Menetidő 2	MEGÁLLÓHELYEK	
0	0	Autóbusz-állomás	
2	2	Petőfi S. u.	
3	3	Jutasi út 61.	
4	4	Jutasi úti Itp.	
5	5	Laktanya	
7	7	Aradi Vérteműk u.	
8	8	Deák Ferenc Iskola	
9	9	Penny Market	
10	10	Láhner Gy. u.	
11	11	Vasútállomás	
13	13	Kisréti u.	
15	15	Jutaspuszta	
17	17	Kisréti u.	
19	12	Jutaspuszta elág.	
20	13	Komfort	
21	14	AGROKER	
22	15	Posta Garázs	
23	16	Fórum	
24	17	Telipar	
25	18	Avar u.	
26	19	Vértemű u.	
27	20	Dózsa Gy. tér	
28	21	Tizenháromváros tér	
29	22	Pápai úti forduló	

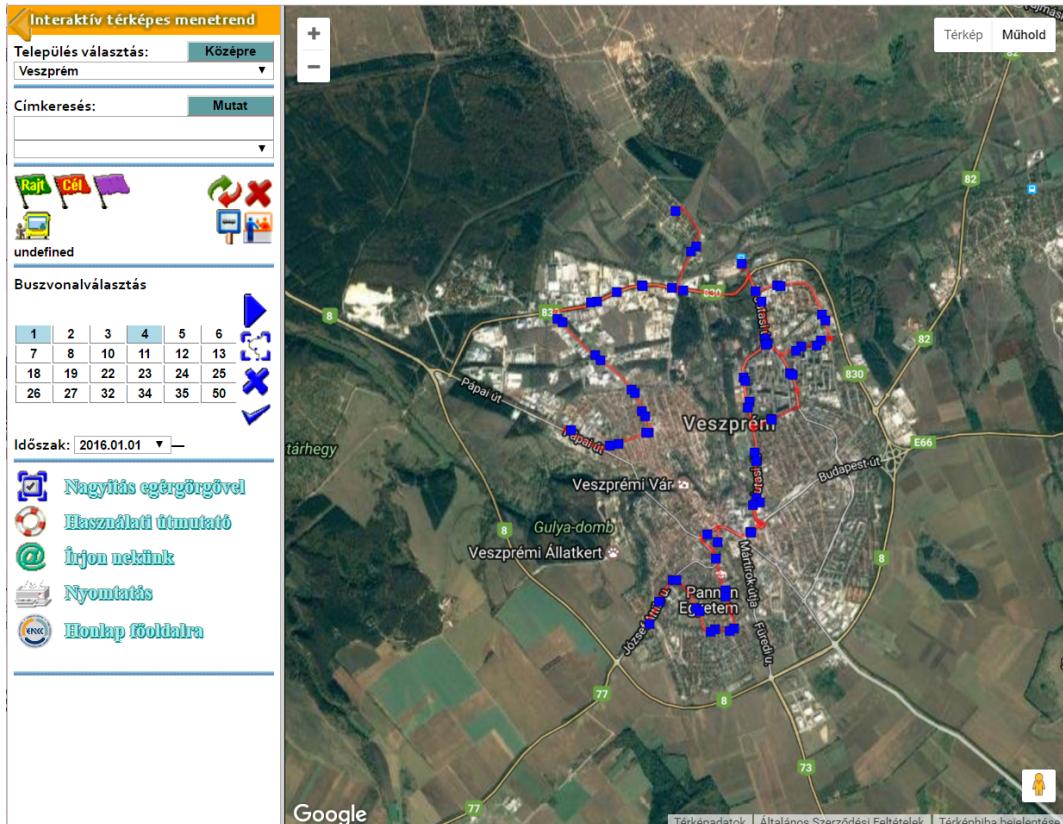
Óra	Munkanapokon	Szabad- és munkaszüneti napokon
	perc	perc
05 :	10, 40	35
06 :	10, 40	05
07 :	10, 35	05
08 :	10, 40	05
09 :	10, 40	05
10 :	10, 40	05
11 :	10, 40	05
12 :	10, 40	05
13 :	10, 40	05
14 :	10, 40	05
15 :	10, 40	05
16 :	10, 40	05
17 :	10	05
18 :	10	05
19 :	10	05
20 :	30	30

JELMAGYARÁZAT:
Az alábbzásal jelölt járatok
Jutaspuszta betéréssel közlekednek az 1-es menetidőszlop szerint

2.4. ábra. Az 1-es buszhoz tartozó menetrend táblázat

A társaság igyekszik több segítséget nyújtani az utasoknak, emiatt új funkciókat készítettek a

weboldalra az elmúlt időben. Létrehoztak egy térképes funkciót, ahol a járatokat kiválasztva az alkalmazás felrajzolja ezen buszoknak az útvonalát a térképre. Továbbá elkezdtek fejleszteni egy útvonaltervező funkciót is, viszont ezek jelenleg kezdetleges formában működnek csak. A térképes szolgáltatásnál kiválasztották, hogy milyen buszjáratakre vagyunk kiváncsiak, és a program kirajzolja azokat a térképre, ahogy a 2.5. ábra mutatja.



2.5. ábra. Az 1-es és a 4-es járat útvonala

BamBusz

Online felületen elérhető segítség, célközönsége főleg az egyetemisták. Kedvezőbb megoldást nyújt, mint a busztársaság oldala abból a szempontból, hogy nem kell átböngészniük az egész dokumentumot az indulási időkért, hanem beállíthatjuk az indulási és az érkezési megállókat. Ezt követően az oldal kilistázza nekünk azokat a buszokat és a hozzájuk tartozó indulási időket, amikkel eljuthatunk a célunkhoz a 2.6. ábra a) részén látható módon. Hátránya hasonlóan a hivatalos oldalhoz, hogy ismernünk kell a megállókat, ahhoz hogy használni tudjuk.



a)



b)

2.6. ábra. Az alkalmazás menüje

Veszprémi buszmenetrend

Okostelefonra elérhető alkalmazás, ami letisztultan, egyszerűen és gyorsan jeleníti meg a buszjáratokat külön menüpontba szedve, ahogy a 2.6. ábra b) képén látható. Előnye, hogy akár útközben tudunk információt szerezni az autóbuszok közlekedési rendjéről. Továbbá elérhető egy éves menetrendi naptár a főoldalon, ami segítségével megállapíthatjuk hogy milyen rend szerint közlekednek a buszok adott napokon. Ugyanakkor az alkalmazás nem naprakész, a naptár a tavalyi évet reprezentálja, illetve ebből kifolyólag a buszjáratok menetrendje is az elmúlt évre érvényes indulásokat mutatja.

3. fejezet

Követelmények, technológiák

A fejezetben bemutatásra kerülnek a funkciókövetelmények, amelyeket a fejlesztés előtt és közben figyelembe kellett venni, hogy az utazóközönség számára egy mai igényeket kielégítő, modern alkalmazás készüljön el. Cél volt továbbá, hogy a rendszer másik szereplői, az adminisztrátorok számára is könnyen kezelhető, felhasználóbarát felület valósuljon meg.

3.1. Követelmények

Az alábbi követelmények tartalmazzák azokat a tulajdonságokat, amiket az elkészült alkalmazásnak tartalmaznia kell.

- Sebesség

Válaszidő lecsökkentése minél alacsonyabb szintre. A felhasználóknak fontos, hogy az alkalmazás segítségével gyorsan és megbízhatóan tudjanak információhoz jutni.

- Alacsony erőforrás felhasználás

Az egyik legfontosabb követelmény, hogy az alkalmazás kevés erőforrás igénybevételével is megfelelően működjön. Ehhez szükség volt arra, hogy a telefon egy lokális adatbázist üzemeltesse. Így az alkalmazás használatakor nem kell internetkapcsolatot biztosítani az adatok elérhetőek lesznek a telefon adatbázisából.

- Megbízhatóság

Létre kellett hozni egy olyan webes felületet, ahol az adatbázis kezelhető, változások esetén pedig módosíthatóak az adatok. Ebből kifolyólag az alkalmazás minden naprakészen szolgálja az információt a felhasználóknak.

- Könnyű kezelhetőség

A felhasználók számára fontos, ha nincsenek bonyolult akciók, hanem lehetőleg az összes funkció használata egyértelmű, ezzel is gyorsítva az alkalmazás használatát. Ha bizonyos jelölések, rövidítések igénylik, akkor súgót kell hozzáadni az alkalmazáshoz.

3.2. Funkcionális követelmények

Ebben a alfejezetben a felhasználóval és az adminisztrátorral kapcsolatos használati esetek kerülnek kifejtésre.

Android alkalmazás

– Útvonaltervezés

Az útvonaltervezés az alkalmazás fő funkciója, ezért nagy hangsúlyt kellett a megvalósításra fektetni. Könnyű kezelőfelületet kívánt, egyértelmű jelölésekkel, amik bárki számára egyszerűvé teszik a használatát. Ez a menüpont azokat az utasokat célozza meg elsődlegesen, akik számára Veszprém ismeretlen terület. Fontos volt, hogy különböző utazási módozatok is elérhetők legyenek, például azoknak, akik buszjegyet váltanak, minél kevesebb átszállással kínálja az alkalmazás az útvonalat. Továbbá, mivel ez egy térképet integráló funkció, ezért megfelelő módon kellett megjeleníteni a térképet.

– Menetrend

A menetrend funkció akkor kap szerepet egy felhasználónál elsősorban, amikor nincs elérhető internetkapcsolat a mobilkészülékén. Ezen kívül azoknál, akik rendelkeznek a tömegközlekedésről legalább alapszintű ismerettel, így jártasak a menetrendben és az indulási időkről szeretnének informálódni. Megjelenésénél figyelni kellett arra, hogy felülete letisztult, könnyen kezelhető és informatív legyen.

– Megállók

Az alkalmazásban szükség volt egy olyan szolgálatásra is, ahol a felhasználók tájékozódni tudnak a városban a megállókról. Mivel előfordulhat olyan eset, hogy valaki tisztában van a megálló elhelyezkedésével, viszont meg szeretné tudni, hogy milyen járatok érintik anélkül hogy az egész menetrendet át kéne olvasnia, ezért fontos volt implementálni egy ilyen funkciót az alkalmazásba.

Weboldal

- Buszjáratok kezelése**
- Menetidők kezelése**
- Megállók kezelése**
- Ideiglenes járatmodosulások kezelése**

3.3. Felhasznált technológiák

Térkép

Egy útvonaltervező alkalmazás nagyon fontos tulajdonsága a megjelenés. Egy ilyen alkalmazásnál alapvető elvárás, hogy a térképen megjelenő információ könnyen kiolvasható legyen, hogy a felhasználók könnyedén el tudják választani a lényeges információt azoktól, amelyek nem szükségesek a tájékozódáshoz. Emiatt szükségem volt egy olyan térképes szolgáltatásra, ami az előbbi céloknak megfelel. Mielőtt megkezdtem a fejlesztést, több ilyen szolgáltatást is megvizsgáltam, abból a célból, hogy a legmegfelelőbbet tudjam kiválasztani a tapasztalatok alapján a szakdolgozatomhoz. Az egyik legfontosabb szempont a kiválasztásnál az volt, hogy ingyenesen elérhető legyen a szolgáltatás. A kutatás során két fő jelöltre sikerült leszűkítenem a listát, a Google Maps-re illetve az OpenStreetMap-re. Mivel az általam készített programot Android platformra terveztem elkészíteni, ezért olyan térképes API-ra volt szükségem, amit be lehet építeni Android applikációba, és ennek a célnak mind a kettő szolgáltatás megfelelt. Mindezek mellett fontos volt számomra, hogy olyan térkép alkalmazást válasszak, ami felhasználói körökben jól ismert. Emiatt megvizsgáltam, hogy az alkalmazásoknak mekkora a felhasználói köre. Az OpenStreetMap Android alkalmazása körülbelül 5 millió felhasználóval rendelkezik, míg a Google Maps letöltése meghaladja az 1 milliárd felhasználót. Ezekből az adatokból következtetni tudtam arra, hogy a felhasználók szélesebb köre miatt a Google Maps felülete sokkal szélesebb körben ismert az emberek között.

	Google Maps	OpenStreetMap
Ingyenesen elérhető	igen	igen
Lekérdezési limit	nincs limit	1 lekérdezés/másodperc
Felhasználók száma	~1 milliárd	~5 millió
Beépíthető modul	saját modul	külső modulok
Megbízhatóság	ellenőrzött, karbantartott	közösségi adatfelvitel

3.1. ábra. A Google Maps és az OSM összehasonlítása

A 3.1. ábrán látható szempontok figyelembe vételevel végül a Google Maps-re esett a választásom, mint integrálható térképes szolgáltatás.

Android, Java

Az Android platformon futó alkalmazások elsődleges programozási nyelve a Java. A Java platformfüggetlen és széles körben elterjedt, a Sun Microsystems által fejlesztett nyelvet az

1990-es évek óta folyamatosan fejlesztik. Az Android egy Linux alapú operációs rendszer, amely több pozitívummal is rendelkezik, ilyen például a hordozhatóság és a biztonság. Futtatási sebessége a Java-hoz képes javult, hiszen a kibővített Java programozási nyelvet Android esetén egy olyan virtuális gépen futtatják, amely szerves részét képezi az Android operációs rendszernek. [telenyomni hivatkozásokkal a részt](#)

Symfony

A Symfony egy PHP alapú keretrendszer. A vele készült webes alkalmazások könnyen karbantarthatóak és jól skálázhatóak, mivel a keretrendszer az MVC tervezési mintára épül. A Symfony felépítése moduláris, ennek köszönhetően könnyen bővíthető. Továbbá a Symfony egy-egy komponensét más keretrendszerrel készült alkalmazásban külön is használhatjuk, amit szintén a moduláris szerkezet tesz lehetővé. minden beérkező kérést a keretrendszer dolgoz fel, melyek az útvonalválasztás után meghívják a megfelelő kontrollerek metódusát. Továbbá a kontrollerek hozzáférnek a services.yml fájlban definiált szolgáltatásokhoz is. Az itt definiált szolgáltatások példányait, a Symfony dependency injection segítségével juttatja el a kontrollerekhez. A Symfony segítségével a dinamikus tartalom is könnyen átalakítható a benne található template nyelvek egyikének köszönhetően. Az általam választott nyelv a twig, amellyel a megjelenítés elválasztható az üzleti logikától.

MySQL

A MySQL egy SQL (Structured Query Language, vagyis struktúrált lekérdező nyelv) alapú relációs adatbázis-kezelő szerver. Teljesen nyílt forráskodú, feltételezhetően emiatt terjedt el széles körben. Egyszerűen használható és költséghatékony megoldást nyújt dinamikus webhelyek számára. Az adatbázis kezelésére a phpMyAdmin adminisztrációs eszközt használtam, amely segítségével Interneten keresztül menedzselhettem az adatokat.

Doctrine

A Doctrine keretrendszer az adatbázisban található adatok elérésére alkalmas. A segítségével PHP osztályok képezhetőek le relációs adatbázis táblákká. A keretrendszer lehetőséget ad arra, hogy az alkalmazás adatbázisa különösebb ráfordítás nélkül lecerélhető legyen. A DQL, vagyis a Doctrine lekérdező nyelv használatával lehet az adatokat elérni. A Doctrine véd továbbá az SQL injektálásos támadásokkal szemben is. A Doctrine úgynevezett Repository-kat használ az adatok lekérdezésére, amelyek lehetővé teszik, hogy alkalmazás logikája és a lekérdezés megvalósítása elvonatkoztatható legyen egymástól. Továbbá azért esett a választásom a

Doctrine-ra, mert a keretrendszer integrálva van a Symfony-ba, illetve szolgáltatásként elérhető a Doctrine EntityManager objektuma. Új entitások generálásában és az adatbázis tábláinak frissítésében is segítséget nyújt a Symfony parancssoros eszköze.

4. fejezet

Felhasználói kézikönyv

A fejezetben bemutatásra kerül a szakdolgozat keretében fejlesztett Android alkalmazás felhasználói szemszögből. Kifejtésre kerül, milyen menüpontok találhatóak az applikációban, és ezek milyen funkcionálitással rendelkeznek. Továbbá ismertetem az adatok menedzselésére szolgáló weboldal felületét és működését, amely az admin felhasználók munkáját teszi könnyebbé.

4.1. Android alkalmazás

Ahogy a 4.1. ábrán is látszik, az applikáció ikonja egy Veszprém járműparkjában is szereplő autóbusz, az Ikarus 280. Az alkalmazás indítása után az alkalmazás lekéri az adatbázisból



4.1. ábra. Az alkalmazás ikonja

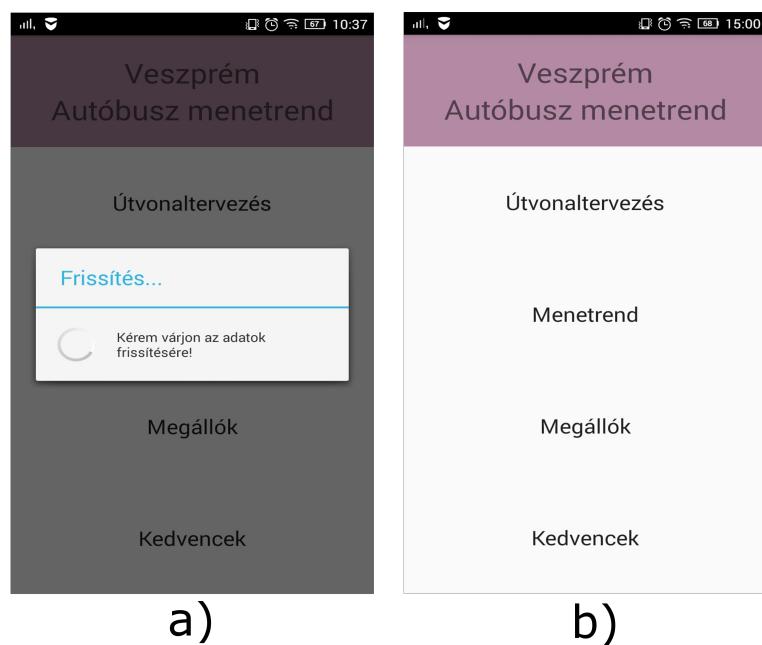
azokat az adatokat, amik módosítási dátuma későbbi, mint az utolsó letöltés ideje. Ezt a felhasználó egy felugró ablak képében látja, amelyen az alkalmazás közli, hogy frissítés van folyamatban, és kéri a felhasználók türelmét. Ezt a 4.2. ábra a) képén tekinthető meg. Amint az adatok aktualizálása befejeződött, a felugró ablak eltűnik, átadva helyét a főmenü menüpontjainak, melyet a 4.2. ábra b) része mutat.

4.1.1. Főmenü

A főmenü az alkalmazás nevét és négy almenüt foglal magába:

- Útvonaltervezés
- Menetrendek
- Megállók
- Kedvencek

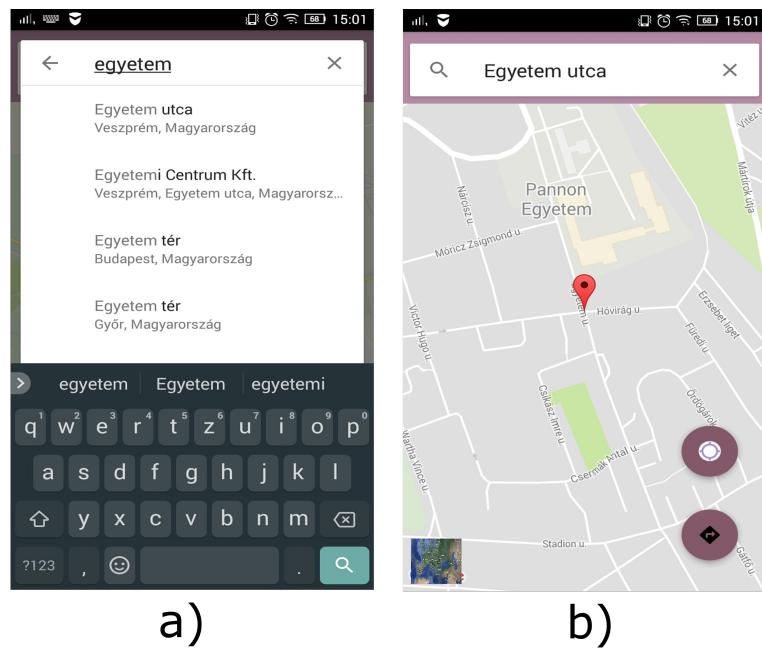
Az applikáció témája a lila szín és annak árnyalatai, amely lehetővé teszi a felhasználók figyelmének felkeltését, mégis letisztult külsőt kölcsönöz.



4.2. ábra. Főmenü

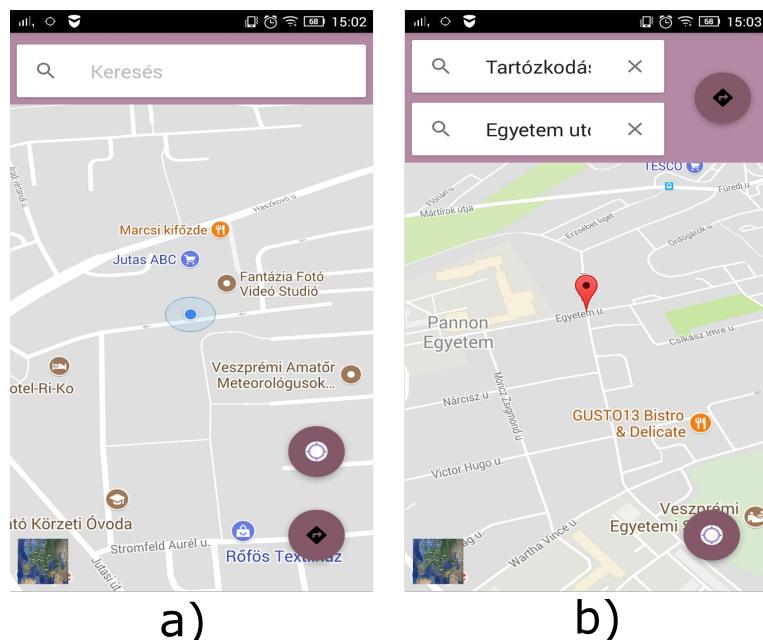
4.1.2. Útvonaltervezés

Az Útvonaltervezés menüpontot kiválasztva egy Google Maps térkép jelenik meg Veszprém városára fókuszálva. A képernyón ezen kívül egy beviteli mező és három gomb található. A beviteli mezőt a felhasználók a térképen való keresésre használhatják. A keresés fő fókusza Veszprémre irányul, a mező automatikusan próbálja kiegészíteni a begépelt helyszínt, Veszprém környékére összpontosítva, ahogy az a 4.3. ábra a) képén is látható. Az alkalmazás képes más földrajzi helyre irányuló keresésekkel is kiegészíteni, viszont ez a funkció - az alkalmazás minél effektívebb működése érdekében - le van korlátozva. Ahogy a a 4.3. ábra b) képén is látszik, a találatot az alkalmazás egy úgynevezett 'marker' lerakásával jelöli a térképen. A bal oldalon található gombbal a felhasználó képes a térkép nézetének megváltoztatására. Az alapérmezett mód a domborzati megjelenés, a gomb megnyomávásával átválthatunk



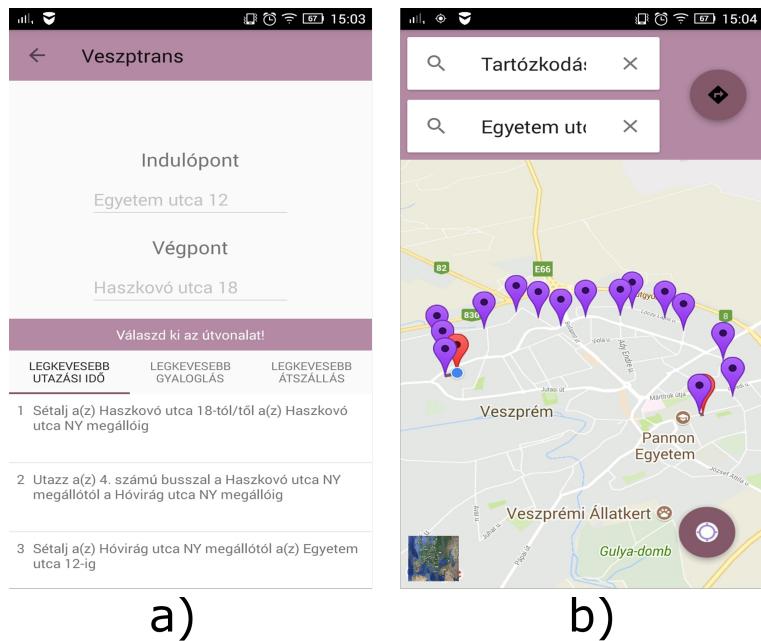
4.3. ábra. Helység keresése az applikációban

műholdas nézetre. A jobb oldalon két gomb helyezkedik el: a Saját pozíció, illetve az Útvonaltervezés. A Saját pozíció gomb használhatához szükség van GPS funkcióra a telefonban. Ha ez nincs engedélyezve, az alkalmazás egy felugró ablak segítségével átirányítja a felhasználót a GPS funkció bekapcsolására alkalmas képernyőre. Visszatérve az alkalmazásba, a térképen



4.4. ábra. Útvonaltervezés saját pozícióval

megjelenik a felhasználó feltételezhető helyzete egy kék ponttal jelölve. Továbbá a térképen megjelenik egy világoskék kör az előbb említett kék pont körül. Ezt a 4.4. ábra a) képében láthatjuk. A világosabb színezetű kör területén valamelyik pont a felhasználó biztos pozícióját jelöli, a kör nagysága az internet elérési módjától (mobilnet vagy Wifi) függ. Ez a funkció azoknak a felhasználóknak nyújt segítséget, akik számára Veszprém városa ismeretlen terület. A jobb alsó gombra kattintva a felhasználó átkerül az Útvonaltervezés oldalra, ahol a felső beviteli mező mellé egy újabb mező kerül, ahogy az a 4.4. ábra b) részén is látszik. Ha az előző oldalon a mezőbe került már földrajzi hely, az alkalmazás automatikusan az alsó mezőt, az Utazás célját tölti fel vele. Az új képernyőn megjelenő ablakban is lehetőség van a saját pozíció lekérésére, ebben az esetben az alkalmazás a paramétereit a Kiindulási pont mezőbe tölti be. Abban az esetben, ha mind a kettő mezőt kitöltött a felhasználó, akkor az Útvonaltervezés gomb ismételt megnyomásával átkerül a találati listára. Az alkalmazás a 4.5. ábra a) részén látható módon jeleníti meg a találatokat. A képernyő felső részén a



4.5. ábra. A találati képernyő és az térképes útvonalterv

beállított indulási és érkezési helyszín van feltüntetve, alul pedig három féle szempont alapján a megoldás:

- Legkevesebb utazási idő

Ennél a lehetőségnél az alkalmazás kiszámolja a lehetséges útvonalakban a gyaloglással és utazással töltött idejét, és ezek közül a legrövidebbet jeleníti meg.

- Legkevesebb gyaloglás

A második opció azt a találatot listázza ki, amelynél a gyaloglással töltött idő a

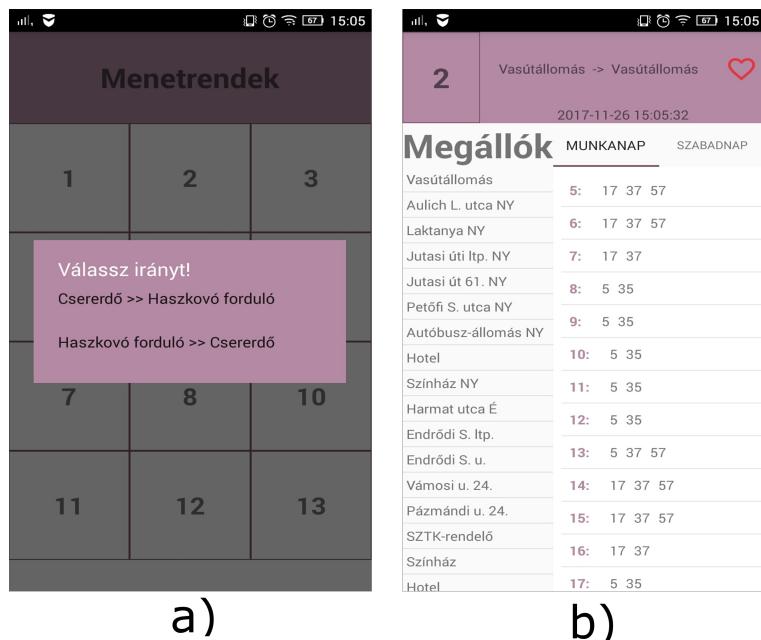
legrövidebb.

- Legkevesebb átszállás

Az utolsó alternatíva a jeggyel utazóknak kínál utazási megoldást, hiszen a legkevesebb átszállással járó utazást jeleníti meg.

A felhasználó az egyik megoldásra kattintva választhatja ki a számára optimális útvonalat. Az alkalmazás a következő képernyőn az útvonalat térképen jeleníti meg, ahogy az a 4.5. ábra b) képén is látszik. A kezdő és a végpont jelölje piros színű, a busz útvonalát az eltérő színnel (jelen esetben lilaival) jelölt megállók mutatják. Ha az útvonalban több busz is szerepel, akkor azok útvonalát egy harmadik színnel jelöli az alkalmazás. A gyalogos útvonalat az applikáció színes vonallal jelöli a térképen a piros jelölők és az első/utolsó buszmegállók között. Ha a választott megoldás mégsem felel meg a felhasználó igényeinek, a képernyő felső részén elhelyezkedő Útvonaltervezés gomb ismételt megnyomásával visszajut az utazási lehetőséget listázó oldalra.

4.1.3. Menetrendek



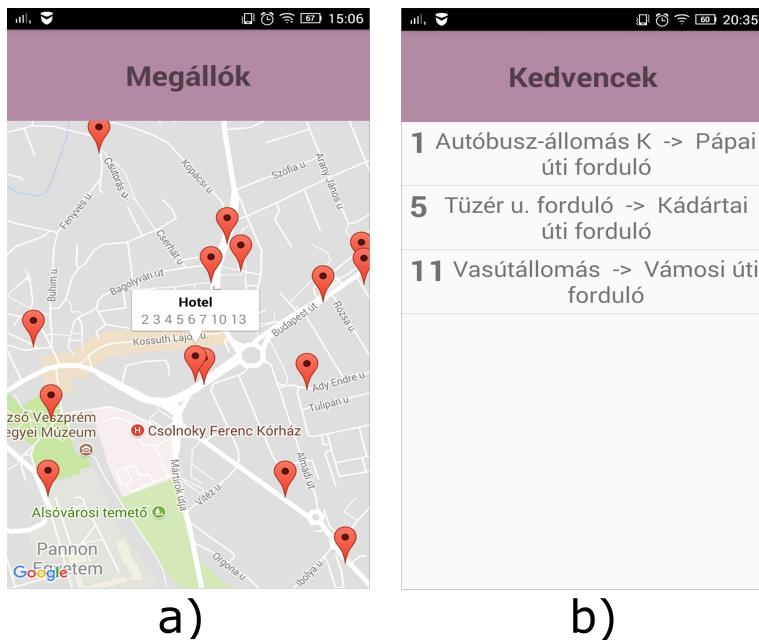
4.6. ábra. Menetrendek

A Menetrendek a főmenü második menüpontja. Ez az opció azokat a felhasználókat segíti, akik már ismerik Veszprém városát, illetve tömegközlekedését legalább alap szinten. A menüpontra kattintva az alkalmazás kilistázza a buszjáratok számait, mely a 4.6. ábra a) képén tekinthető meg. Amint kiválasztott a megtekinteni kívánt járatot, az applikáció bekéri

a járat irányát is, majd kilistázza az indulási időket a 4.6. ábra b) részén látható módon. Az megnyíló ablak fejléce tartalmazza az előbbi felhasználói akciók során bekért adatokat, illetve a Kedvencekhez adás funkciót egy szív ikon formájában. A képernyő bal oldalán a járat megállói láthatóak, a választott iránytól függő sorrendben. Az időpontok jobb oldalon, munkanapokra és szabadnapokra felosztva láthatóak.

4.1.4. Megállók

A Megállók menüpont alatt egy Google Maps térkép jelenik meg rajta Veszprém város buszmegállójával. A megállók pontos helyét piros színű jelölők jelzik a térképen, amely a közelítés mértékével arányosan csoportosítja a jelölőket. A felhasználó megkeresi a térképen azt a megállót amelyikre kíváncsi, az alkalmazás pedig kilistázza az ott megálló járatok számát, ahogy a 4.7. ábra a) képéén is látszik.



4.7. ábra. Megállók és Kedvencek

4.1.5. Kedvencek

A menüpontban a Menetrendek alatt Kedvencnek jelölt járatokat lehet elérni. Az alkalmazás kilistázza azokat az adott menetirányhoz tartozó járatokat, amelyeket a felhasználó kedvencnek jelölt a 4.7. ábra b) ábráján látható módon. A funkció egy szív ikon formájában jelenik minden menetrend adatlapján. Alapérmezzen a szív üres, rájuk kattintva tudja a felhasználó a Kedvencek menüpont alá rakni, ekkor a szív pirosra színeződik. A listából eltávolítani hasonló módszerrel lehet, a telt szívre rákattintva kikerül a Kedvencekből.

4.2. Admin oldal

Az alkalmazás létrehozásakor szükség volt egy olyan kezelőfelület létrehozására is, amelyen keresztül könnyen kezelhetőek az adatbázisba feltöltött adatok. Mivel az adatok közérdekűek és sok embert érintenek, fontos szempont volt, hogy csak azok férhessenek hozzá akiknek van jogosultságuk. Az oldal eléréséhez a felhasználónak igazolnia kell magát egy felhasználónév-jelszó párossal, ahogy az a **ABRA** ábrán is látszik. Az **ABRA** ábrán látható admin oldal jelenik meg a sikeres bejelentkezés után. Bal oldalon található a menü, amiből a felhasználó kiválaszthatja azt az elemet, amit menedzselni szeretne. Alapértelmezetten a Járat menüpont jelenik meg, ez az adatbázis architektúrájának alapja. A másik alapvető entitás a Megállók, amelyek a buszmegállók pontos földrajzi helyzetét tárolják. Egy megálló felviteléhez hosszúsági és szélességi koordinátákra van szükség, amelynek felvitelét egy beágyazott Google Maps térkép segíti a weboldalon. A felhasználó kijelöli a térképen a megálló pontos helyét, aminek a koordinátái megjelennek a térkép mellett és az adminisztrátornak már csak nevet kell hozzárendelnie az új rekordhoz. Hasonló kezelőfelület található a megállók útvonalakhoz való rendelését kezelő menüpontban. Egy új útvonal létrehozása esetén az adminisztrátor a Google Maps-en elhelyezkedő jelölők közül kiválasztja a megfelelőt. Ezek a jelölők az adatbázisban található megállókat jelölik a térképen elhelyezve a könnyebb kezelés érdekében. Ha egy útvonal módosul (például egy felújítás miatt), akkor az egy új útvonalként felvezetésre kerül, az alapvető útvonal állapota pedig inaktívvá válik. Amint az adminisztrátor sikeresen az adatbázishoz adott egy buszjáratot, a megfelelő útvonalhoz és járathoz indulási időket rendel. Ezeket az időket előre definiált kategóriákba menthetjük el, ilyen kategória például a szabadnap, a munkanap vagy a tanszüneti munkanap. Mindemellett előfordulhatnak olyan időpontok a naptárban, amikor nem az alapértelmezett közlekedési rend a mérvadó, például hétköznapra eső ünnepnapokon. Mivel ezek évről évre változnak, így az adminisztrátor feladata, hogy megfelelően felvezesse ezeket a napokat az adatbázisba. Erre szolgál az Elterő közlekedési rendek menüpont, ahol a megadott időintervallumra vonatkozóan felülírhatjuk az alapértelmezett rendet.

Az adatok átláthatóságának érdekében táblázatban jelennek meg az adatbázisból lekért rekordok. Az adminisztrátor a keresés funkció segítségével kereshet az adatok attribútumai között, illetve sorba rendezheti azokat a gyorsabb kezelés érdekében.

5. fejezet

Fejlesztői dokumentáció

Az elkészült rendszer szerver-kliens architekturára épül. Az architektúra modellje a ???. ábrán látható. A szerver felelős az adatok kezeléséért, mely a Symfony keretrendszerrel lett megvalósítva. Az adatokat a Doctrine keretrendszer segítségével éri el az adatbázisból. A rendszerben található a szerver, ami egy adminisztrációs feladatokat betöltő weboldal, és a kliens, egy Android alkalmazás, amely a felhasználók részére készült. Az adminisztrációs weboldalon az adatok a Twig sablonkezelő segítségével jelennek meg. Az Android-os kliens a kért adatokat JSON formátumban kapja meg. A továbbiakban részletesen bemutatásra kerül a szerver és a kliens felépítése, működése.

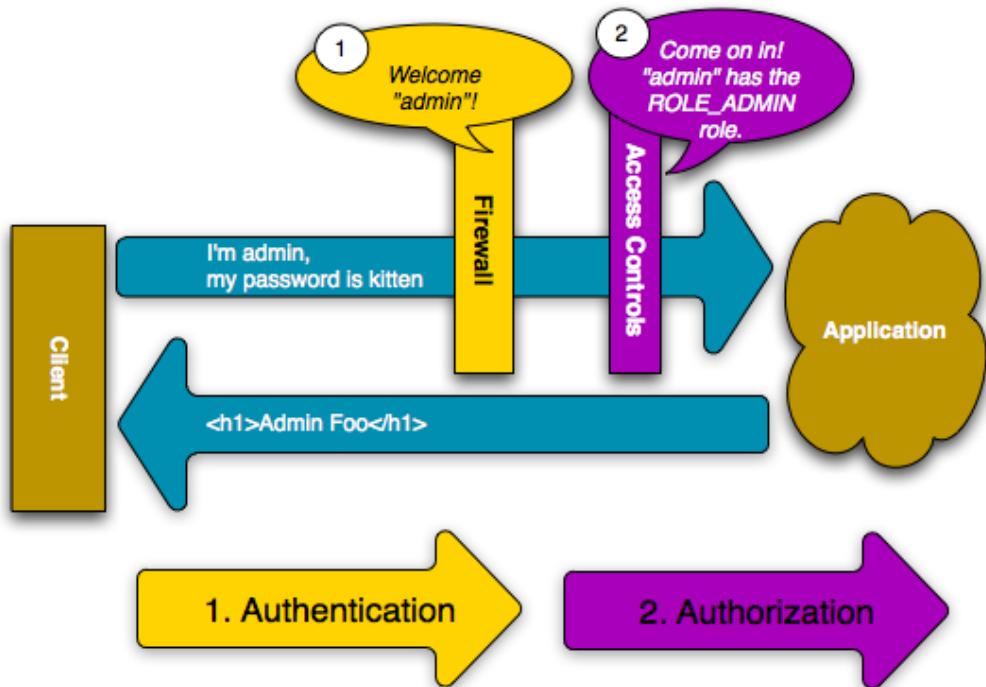
5.1. A szerver

A szerver a Symfony keretrendszer 2.8-as verziójára épült. A beérkező kérés áthalad a Symfony tűzfalán, ami elindítja az autentikációt. Ha sikeres volt az autentikáció, a kérés a megfelelő kontroller osztályhoz kerül feldolgozás után.

A kontroller osztályban a beírt URL-hez tartozó metódus hívódik meg. Alapesetben ezek az osztályok a *AppBundle\Controller* névterben találhatóak. Az URL-címek metódusokra való leképzése kétféleképp valósítható meg a Symfony keretrendszer használatával.

- routing.yml fájlban való definiálás
- kontroller osztályokban annotációként

Az első opció akkor előnyös, ha útvonal alapján akarjuk meghatározni, hogy mely kódrészlet fog végrehajtódni a kontrollerek megfelelő kialakításával és az annotációk jó elhelyezésével. Másrészről a második megoldás itt előnyösebb lehet, mert ilyenkor a kódot látva a fájl elhagyása nélkül meg tudjuk állapítani, hogy milyen esetekben fog a kódrészlet meghívásra kerülni. A metódusok annotációi között megszabhatunk különböző paramétereket is, melyet az alábbi kódrészlet demonstrál:



5.1. ábra. Az autentikáció folyamata

```

1  /**
2  * @Route("/{id}/edit", name="line_edit")
3  * @Method({"GET", "POST"})
4  */
5  public function editAction(Request $request, Line $line)
6  {
7      // ...
8  }

```

Autentikáció

Az autentikáció a Symfony tűzfal moduljával történik. Az ehhez szükséges beállítások a `app\config` mappa `security.yml` konfigurációs fájljában találhatóak. A szerverhez jelenleg két fajta felhasználó van rendelve, melyek az előbbiekben említett fájlban vannak definiálva. A szerver feladatköréből kiindulva a jelenlegi igényeket ez a két felhasználó kielégíti, ezért nincs szükség újabb felhasználók hozzáadására. Ez azt jelenti, hogy a felhasználókhöz tartozó jelszavakat is ebben a fájlban tároljuk. Biztonsági szempontkból kritikus tényező, hogy ezek a jelszavak valamilyen módon titkosítva legyenek. A titkosításhoz a bcrypt algoritmust használtam fel. A Symfony segítségével parancssorból lehetséges bármilyen jelszót titkosítani a bcrypt algoritmussal.

```
1  php bin/console security:encode-password
```

A parancs futása közben bekéri, mi az a jelszó, amit titkosítani kell. Miután megadtuk a jelszót, a bemeneten lefuttatja az algoritmust és végül kiírja a titkosított karakterszorozatot. Ezt a szót megadva a fájlban jelszóként, továbbra is a titkosítatlan jelszóval be tud lépni a felhasználó az oldalra, viszont nem áll fent tovább a veszély, hogy illetéktelenek kezébe jutna a jelszó. [hitvatzkozás a hashre](#)

Autorizáció

Az autorizáció folyamata szintén a Symfony tűzfal moduljának segítségével valósult meg. Ez a sikeres bejelentkezést követően hajtódik végre. A folyamat célja, hogy a bejelentkezett felhasználó csak a számára kijelölt szolgáltatásokat, oldalakat érje el. Az előző pontban említett két felhasználóhoz tehát az alábbi két szerepkör tartozik:

- Látogató
- Adminisztrátor

A látogató szerepkörhöz tartozó felhasználónak csak az adatbázis-entitások lekérdezéséhez van jogosultsága. Az adminisztrátori engedéllyel rendelkező viszont a weboldal minden szolgáltatásához hozzáfér.

A szerepkörökhez hozzá lettek rendelve meghatározott formájú URL címek.

```
1  access_control:
2      - { path: /json$, roles: ROLE_USER }
3      - { path: ^/, roles: ROLE_ADMIN }
```

Az alábbi kódrészlet azt mutatja be, hogy a látogatói jogkör csak a *json* végű URL-címekhez ad elérést, míg egy adminisztrátor minden címet elér. Ugyanezt a funkcionálitást meg lehet valósítani szintén annotációkkal is, de a weboldal kialakítása miatt célszerűbb volt ezt a megoldást választani, hisz így egyetlen fájlban elegendő meghatározni a hozzáférési szabályokat, annotációk használatával viszont minden kontroller osztályban egyenként kellett volna meghatározni ugyanazokat a szabályokat.

Adatbázis kapcsolat

Az adatbázist a Doctrine keretrendszer használva éri el a szerver. Az alkalmazáshoz tartozó entitások helye a *AppBundle\Entity* névtér. Ezek mindegyike egyszerű PHP osztályok,

a felhasznált keretrendszer alakítja ezeket az adatbázis tábláivá. Az osztályban szereplő adatmezőkhöz meg kell adni az annotációkat, hogy az átalakítás milyen módon történjen meg. Az entitások egymás közti kapcsolatait a következő annotációk jelzik:

- OneToOne
- OneToMany
- ManyToOne

Az adatbázisban található adatok lekérdezés és objektumba való alakítása a *Repository* osztályok segítségével történik. Ezek az osztályok a Doctrine *EntityRepository* osztályából származnak, mely már rendelkezik a legalapvetőbb lekérdezésekkel, mint az azonosító vagy valamilyen tulajdonság alapján történő adatlekérés. Az összetettebb lekérdezésekhez a DQL nyelvet használtam fel, mely az entitások kapcsolatait az entitáshoz tartozó osztályban megadott annotációk alapján kezeli. Amennyiben az adatbázisból lekérdezett adatok módszertak, ahhoz, hogy ez a változás megmaradjon el kell menteni az adatbázisba. A Doctrine keretrendszerben található automatikus megoldás, mégpedig az EntityManager osztály, amely érzékeli a változást és elmenti a *flush()* metódus használatával. Ha új adatot akarunk az adatbázishoz adni, azt is az EntityManager osztályal tehetjük meg. Az elkészült új objektumot paraméterként átadva az osztály *persist()* metódusának, a Doctrine keretrendszer úgy kezeli a továbbiakban az objektumot, mintha az az adatbázisból lekérdezett lenne, és így a *flush()* metódussal az adatbázisba tudja írni.

5.2. Szerver feladatai

Az alábbiakban bemutatásra kerülnek a szerver főbb feladatai.

Járatok

A városban közlekedő buszok más-más végpontok között közlekednek. Előfordulhat, hogy ugyanolyan számmal ellátott busz ugyanazon végpontok között közlekedik, de a köztes megállók eltérhetnek, vagy a köztes megállók ugyanazok, de a végpontok mások. Ezeknek az információk összefogására létrehozott *LineInfo* entitás felelős. Ez az entitás tárolja a két végpontot, a közöttük közlekedő buszjáratot, és a járat típusát. A végpont és a buszjáratok szintén entitások, az ezek között fennálló kapcsolat az 5.2. ábrán látható.

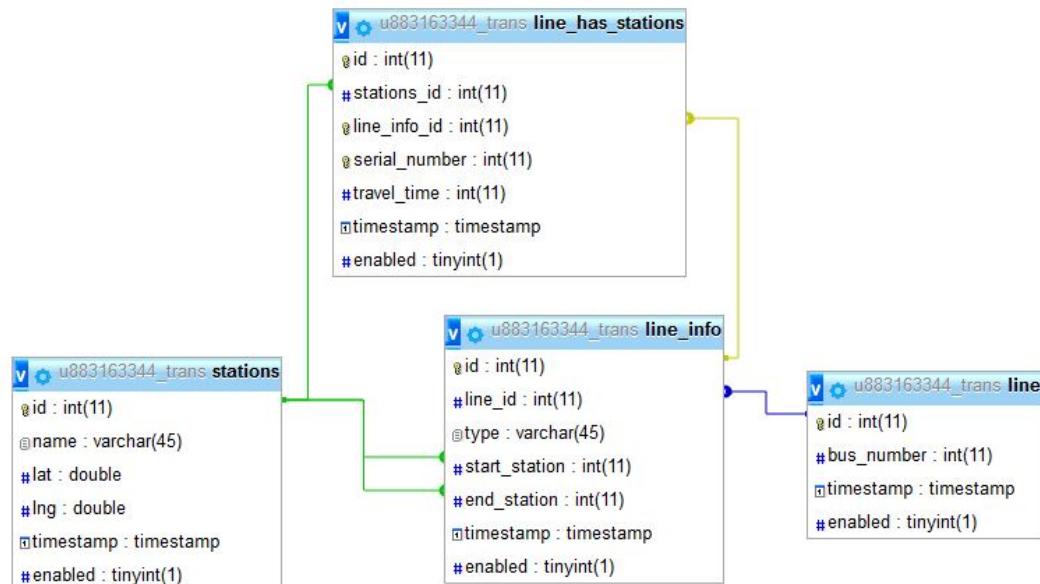
Útvonalak

A buszjáratok útvonaluk bejárása során több, köztes megállót is érintenek. Meg kell határoz-nunk, hogy egy adott busz milyen irányban haladva mely köztes megállókban áll meg. Ezt a



5.2. ábra. A járat entitás kapcsolatai

feladatot a *LineHasStations* entitás végzi. Az entitás segítségével meg tudjuk határozni, hogy az előző pontban vázolt járat egy példányának a végpontjain kívül minden más állomásai vannak. Továbbá tároljuk a köztes megálló útvonalon belüli sorszámát, melyből megállapítható az útvonalon található állomások helyes sorrendje. minden egyes objektum esetén tárolásra kerül még, hogy az előző megállóból minden gyorsan tudunk eljutni. Természetesen a busz indulópontjának ezen attribútuma nulla. Az 5.3. ábrán megtekinthető minden kapcsolatot tartalmaz az útvonal entitás.

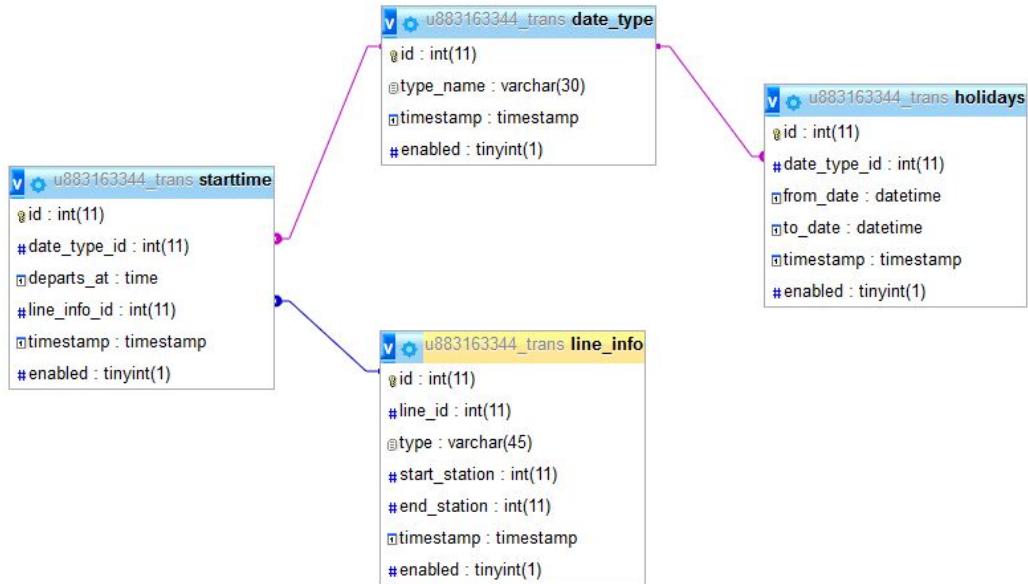


5.3. ábra. A járathoz kapcsolatai

Indulási idők, különleges időpontok

Útvonaltervezés szempontjából fontos információ továbbá, hogy az adott járat az útvonalán található megállóiba minden időpontban érkezik meg, továbbá különleges alkalmak esetén ezek hogyan változnak meg. Az indulási időpontot egy Starttime nevű entitás tartalmazza,

melyben tárolódik továbbá az is, hogy ez az indulás mely járatra vonatkozik, valamint milyen típusú dátum esetén értetendő. A különböző típusú dátumokat külön entitás tartalmazza. Az alkalmazáshoz tovább tartozik egy Holidays entitás is, melyben az olyan, előre nem tervezett időpontokat tartalmazza amelyek befolyásolják bizonyos járatok indulási rendjét és esetleges útvonalát is. Az előbbiekben bemutatott entitások közötti kapcsolatot a 5.4. ábra mutatja be.



5.4. ábra. Különleges és indulási időpontok közötti kapcsolat

5.3. Az Android kliens

Az alkalmazás fejlesztéséhez az Android Studio fejlesztői környezetet használtam. Az Android Studio a Gradle rendszerre épül, amelynek számos funkciója megkönnyíti az Androidos alkalmazások fordítását. A Gradle Androidos bővítményében található több Android specifikus parancs, amelyek közül az *assemble* parancssal készíthetjük el a futtatható és telepíthető állományt. A futtatható fájl elkészítéséhez szükséges információkat a build.gradle fájlban adhatjuk meg. Ebben találhatóak a szükséges függőségek, amiket a Gradle bővítménye letölt, és belecsomagol az .apk-be. Továbbá itt található az applikáció által támogatott Android verziók száma. Az általam készített alkalmazás futtatásához legalább a *minSdkVersion 14* verzió szükséges, ami az Android operációs rendszer 4.0 verziójának felel meg. Az *AndroidManifest.xml* fájl az alkalmazás alapvető jellemzőit tartalmazza. Ilyen jellemzők többek között a szükséges engedélyek és a Google Maps-hez tartozó metainformáció. Az alkalmazás telepítéséhez és futtatásához szükség van engedélyekre a felhasználó részéről. Ezek az

engedélyek a következők:

```
1 <uses-permission android:name="android.permission.INTERNET" />
2 <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
3 <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Az első hozzáférés az Internettel való kommunikációhoz szükséges. Szükségünk van továbbá olyan engedélyre amivel a Saját pozíció lekérhető és engedélyezhető a GPS a telefonon. Mivel az alkalmazásnak szüksége van Internet hozzáférésre az adatok letöltéséhez, ezért az alkalmazás engedélyt kér a telefon hálózati állapotának lekéréséhez is.

Helyi adatbázis

Fontos szempont volt helyi adatbázist létrehozni, mivel az alkalmazás és a szerver között folyamatosan felépülő kapcsolat sok erőforrást igényel. Továbbá az olyan eseteknél, amikor a készülék nincs hálózatra kapcsolva, akkor nem érné el a szervert, így nem töltődnének le a működéséhez szükséges adatok. Ezek következtében szükség volt arra, hogy az applikáció offline üzemmódban is képes legyen funkcionálisan működni.

Megvizsgáltam több Android platformhoz elérhető ORM (object-relational mapping vagyis objektum-relációs leképezés) keretrendszer, ezen belül is az ORMLite keretrendszer. Az ORMLite akkor előnyös, ha nem létezik adatbázis séma, amihez alkalmazkodnia kell. Mivel az alkalmazás esetében már a szerver oldalon kialakításra került egy séma, emiatt a keretrendszer használata nehézkes volt. Ebben a kontextusban szükséges volt, hogy manuálisan személyre szabhattam a helyi adatbázis sémáját, így végül a natív SQL utasítok használata mellett döntöttem.

Az androidos kliens implementálását követően kiadásra került több architekturális felépítést segítő könyvtár. Ezek közül a *Room* nevű könyvtár az adatok tárolásáért felel. A *Room* egy adatbázis absztraktív könyvtár, amely elrejti az alatta található adattárolós részleteit. Fordítási időben validálja az SQL lekérdezéseket, így ha ezek között található olyan, amely nem felel meg az adatbázis sémának, fordítási hibáként jeleníti meg futás idejű hiba helyett.

A helyi adatbázis-kezelő rendszer az SQLite, amely az Android operációs rendszerbe gyárilag beépített adatabázis-kezelő motor. Mivel az SQLite relációs adatbázis, ezért a kommunikáció szimpla SQL utasításokkal történik. Az applikáció sikeres működéséhez szükséges egy *SQLiteOpenHelper* osztályból örökölöttetett osztály, amelyben az adatbázis nevét, verzióját és az adatbázis létrejöttekor végrehajtandó utasításokat tároljuk. Az ősosztályból kettő metódust kötelező implementálni, az *onCreate* és az *onUpgrade* metódust. Az *onCreate* metódus segítségével van lehetőség táblákat létrehozni az éppen elkészült adatbázisba, amit az alábbi kódrészlet szemléltet.

```

1  @Override
2  public void onCreate(SQLiteDatabase sqLiteDatabase) {
3      try {
4          sqLiteDatabase.execSQL(DataTypeEntry.CREATETABLE);
5          /* Create other tables similar to the above one. */
6      } catch (SQLiteException e) { /*...*/ }
7  }

```

Ugyanakkor az onUpgrade metódus abban az esetben fut le, ha megnöveltük az adatbázis verzióját. Ez a verziószám növekedés azt jelzi, hogy olyan változás történt az adatbázisban, amely visszafele nem kompatibilis, és egy transzformáció szükséges a régi és az új séma között. Az *SQLiteOpenHelper* osztály az SQL utasításokat tranzakcionálisan hatja végre. Egy tranzakció tartalmazhat több SQL utasítást is, amelyek végrehajtása egyszerre történik. Ha egy SQL utasítás sikertelen, akkor az összes utasítás visszavonásra kerül.

Az adatbázis minden táblájához egy statikus osztály lett létrehozva, melyek egy DatabaseContract nevű osztályban tárolódnak. minden belső osztály implementálja a BaseColumns osztályt, melynek köszönhetően megkapják az egyedi azonosításra használatos `_id` mezőt. Erre egy példa az alábbi kódrészletben látható. A további attribútumokat is implementálásra kerültek az adott osztályokban, majd a mezők alapján elkészült a tábla létrehozásához szükséges SQL parancs.

```

1  static class DataTypeEntry implements BaseColumns{
2      static final String TABLE_NAME = "date_type";
3      static final String COLUMN_NAME_TYPE_NAME = "type_name";
4      static final String COLUMN_NAME_ENABLED = "enabled";
5      static final String CREATETABLE = "CREATE TABLE IF NOT EXISTS " +
        TABLE_NAME + "("+_ID+"` INTEGER PRIMARY KEY NOT NULL, `"+
        COLUMN_NAME_TYPE_NAME+"` TEXT NOT NULL, `"+COLUMN_NAME_ENABLED+"` INTEGER
        DEFAULT 1);";
6  }

```

Az applikáció lokális adatbázisában lévő adatok POJO-ban tárolódnak, minden táblához külön POJO tartozik. A POJO egy olyan Java objektum, amely nem rendelkezik speciális tulajdonsággal.

Ahhoz, hogy az alacsonyabb szintű SQL utasítások elkölnöljenek az üzleti logikától, létrehoztam egy *DataManager* nevű absztrakt osztályt, melynek típus paramétere egy POJO objektum. Mivel az osztály absztrakt - így nem példányosítható - ezért ebben az osztályban kaptak helyet az olyan metódusok, melyek minden típusú POJO-ra általános érvényeségek. Az alábbi kódrészletben a *DataManager* osztály implementációja látható, ahol a `createOrUpdate` és a `queryForAll` metódusok az adatbázis tábláinak kezeléséért felelősek.

```

1  public abstract class DataManager<T> {

```

```

2     DatabaseHelper helper;
3
4     /* ... */
5     public abstract void createOrUpdate(T data);
6     public abstract List<T> queryForAll();
7 }
```

Ebből az osztályból származnak a specifikus osztályok, melyeken keresztül adott típusú POJO-k kerülnek lementésre vagy visszakérdezésre a helyi adatbázisból. Az ősosztályban található metódusokat mindenképp implementálnia kell minden származtatott osztálynak. Az olyan funkcionálitások esetén - amelyek nem minden POJO esetén voltak értelmezhetőek - a specifikus osztályban kerültek definiálásra. Az alkalmazás mindenhol ezeket a manager osztályokat használja, ahol az adatbázissal való interakció szükséges.

Szerver-kliens kapcsolat

A helyi adatbázisban található adatok csak egy adott idő-intervallumban érvényesek, a menetrend változtatásával érvényüket vesztik. Ezért bizonyos időközönként frissíteni kell a benne szereplő információkat. Az adatok aktualizálása az alkalmazás indításakor történik, erre szolgál a korábban bemutatott szerver, ahonnan az alkalmazás az aktuális információkat kapja. A szerver REST üzenetekkel válaszol a beérkező kérésekre. A kommunikációt a Retrofit nevű REST klienssel oldottam meg, melynek segítségével könnyű a szervernek kéréseket küldeni. A *NetworkManager* osztály képes lekérdezni, hogy az adott eszköz rendelkezik-e jelenleg bármilyen internet eléréssel. Ha a készülék nem rendelkezik internet eléréssel, akkor csak a helyi adatbázist használja. Ellenkező esetben pedig elkezdődik az adatok frissítése az applikáció elindításával. A minél kisebb adathasználat érdekében a kliens tárolja, hogy mikor történt az utolsó frissítés. Az applikáció indításkor elküldi a legutolsó frissítési dátumot a szervernek, a szerver pedig csak azokat az adatokat küldi vissza, amelyek újabbak, mint a kapott időpont. Miután az adatok aktualizálása sikeresen megtörtént, a frissítés időponja felülírásra kerül a helyi adatbázisban. Az IDownloader interfész a A *BaseDownloader* absztrakt osztály implementálja, melynek konstruktoraiban kerülnek inicializálásra a Retrofit számára szükséges osztályok. Lehetőségünk van minden kérést személyre szabni. Ezt felhasználva juttatjuk el a szerver számára a felhasználó azonosítására szolgáló adatokat és a legutolsó frissítési időpontot. Az IDownloader interfész egyetlen metódus-definíciót tartalmaz:

```

1 public interface IDownloader {
2     void download();
3 }
```

A *BaseDownloader* osztály tartalmaz még egy saveToDatabase metódust, amely az adatok helyi adatbázisba történő lementésére alkalmas. A *BaseDownloader* osztályból származtatott gyerekosztályok implementálják az előzőekben említett két metódust.

A Retrofit konfigurálása során egy interfész került megvalósításra, amelyben metódus-definíció formájában felsorolásra kerülnek azok a hívások, amelyeket a Retrofit képes végrehajtani.

```
1 public interface MyApiEndpointInterface {
2     /* ... */
3     @GET("stations/json")
4     Call<List<Stations>> getStations();
5 }
```

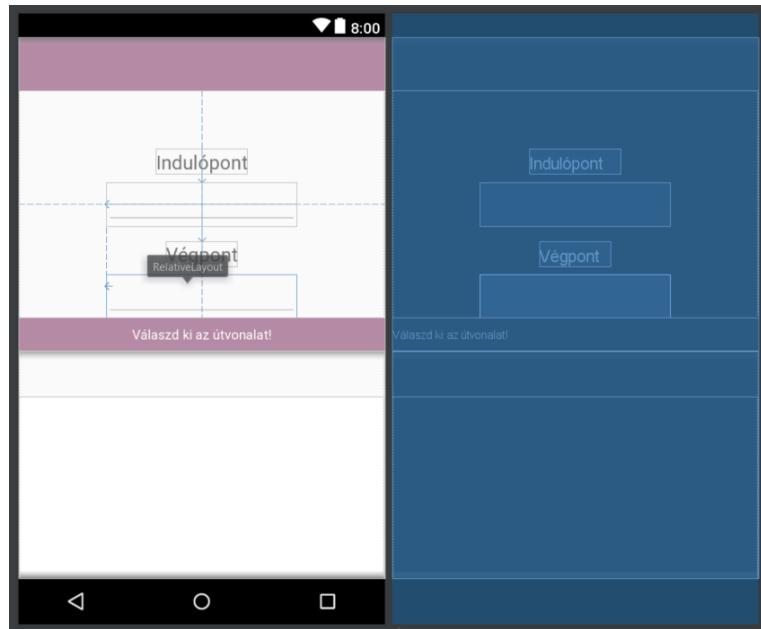
A fenti példán látható, hogy a megállók eléréséhez annotáció formájában megadásra került az elérési útvonal. Ennek visszatérési értéke egy olyan lista lesz, amely megállókat tartalmaz. A listában található megállókat a Retrofit automatikusan serializálta ki a szervertől kapott JSON válaszból. A sserializáláshoz az alábbi entitásban található annotációs konfigurációt veszi alapul.

```
1 public class Stations {
2     /* ... */
3     @SerializedName("lat")
4     @Expose
5     private double lat;
6 }
```

A felhasználó felület kialakítása

A kód karbantarthatóságának érdekében a felhasználói felület kialakítása az MVP (Modell-View-Presenter) szoftvertervezési mintára épül. Implementálása a *Mosby* könyvtár segítségével történt. Előnye, hogy az üzleti logika elhatárolódik a felhasználó felülettől, melynek köszönhetően a kezelőfelület módosítható a logika megváltoztatása nélkül. Az alkalmazás minden képernyő-nézetéhez tartozik egy úgynevezett activity. Az activity a felhasználók és az applikáció közötti legfőbb alapelem. A Mosby könyvtárban találhatóak előre definiált osztályok és felületek, melyek kiindulási pontként szolgálnak az MVP megvalósításához. A tervezési minta alkalmazásához létre kell hozni minden activity esetén egy interfészt, melyben definiálásra kerül, hogy milyen műveleteket tud végrehajtani az activity. Az üzleti logika tényleges megvalósítása a presenter osztályban történik, így ez az aspektus nem lesz kihatással a felhasználói felületre. A presenter osztályok típusparaméterként megkapják az előzőekben említett felületeket. A létrehozott felületet implementálnia kell az activity-nek, és típusparaméterként meg kell adni mind a felületet, mind a presenter osztályt. A felület, presenter és activity osztályok minden a könyvtárban található alaposztályok leszármazottjai. minden activity-hez tartozik egy .xml kiterjesztésű fájl, mely a felhasználói felület építő elemeit tartalmazza. A tervezésre lehetőség van az XML fájlban történő szöveges kifejtésre, illetve grafikus felületen keresztül,

amelyre drag-and-drop módszerrel helyezhetjük rá az elemeket. Az előbb bemutatott tervező felület az 5.5. ábrán látható.



5.5. ábra. Google Maps útvonaltervezés

Google Maps integráció

A Google Maps alkalmazásba való integrációját több előkészületi lépés előzte meg. Első lépésként a Google Developer weboldalon egy projekt létrehozása szükséges. A projektben engedélyezésre kerültek azon szolgáltatások, amelyek az alkalmazás funkcionálisához elengedhetetlenek. Az engedélyezett szolgáltatások használatához az alkalmazásban egy hitelesítő kulcsot kötelező megadni, amely a fent említett Google Developer oldalon generálódik. Mindezek után minden lehetőség adott a térkép megjelenítésére az alkalmazásban. Ennek legegyszerűbb módja, ha a *SupportMapFragment* osztályt használjuk. Ez az osztály körül fog egy térképet és automatikusan kezeli annak életciklusát. Egyszerűen megjeleníthető a kívánt activity-ben, az XML fájlhoz az alábbiakban bemutatott módon kell hozzáadni:

```
1 <fragment
2     android:id="@+id/map"
3     android:name="com.google.android.gms.maps.SupportMapFragment"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"
6 />
```

Ahhoz, hogy a térkép dinamikusan módosítható legyen, egy *GoogleMap* példányra van szükség. A *SupportMapFragment* osztályon a *getMapAsync(OnMapReadyCallback)* metódust meghívva

az osztály inicializálja a térképet. Miután ez megtörtént, a metódus paramétereként átadott *OnMapReadyCallback* felületen meghívódik az *onMapReady* metódust, ami egy *GoogleMap* példányt biztosít. A *OnMapReadyCallback* felület lehet külön osztály, de akár egy activity is implementálhatja azt, így amikor a példány elérhető, azonnal interakcióba léphetünk vele az activity-ben. A *GoogleMap* példány segítségével a térképhez adhatjuk a saját pozíciót lekérő gombot vagy megadhatjuk a térkép nézetének típusát.

Útvonaltervező

A felhasználó az Útvonaltervezés menüpontot kiválasztva Veszprém városán belül két tetszőleges hely között tervezhet közlekedési járatokat is alapul véve. Ehhez meg kell adnia a kiindulási és érkezési pontokat. Ezt megteheti úgy, hogy a saját pozíóját megadva automatikusan kitölödik az indulási pont és már csak az érkezési helyet kell megadni, vagy mindenkor manuálisan kell felvinnie. A felvitelt *SupportPlaceAutocompleteFragment*-en keresztül teheti meg a felhasználó. Ahogy a felhasználó elkezdi beírni a helyszínt az *SupportPlaceAutocompleteFragment* automatikus kiegészítés alkalmával. Az előnyben részesítendő területet a *setBoundsBias* metódus segítségével jelölhetjük ki.

Ahhoz, hogy a két kijelölt hely között megkapjuk a lehetséges útvonalakat, le kell futtatnunk valamilyen útvonaltervező algoritmust. A későbbi továbbfejlesztést elősegítendő létrehoztam egy *IPlanner* nevű felületet, melynek egyetlen metódusa a *calculate*. A felület mellé elkészült még egy *BasePlanner* nevű absztrakt osztály is, amely megvalósítja a felületet. A tényleges útvonaltervezést implementáló osztályoknak ebből kell leszármazniuk. Az ősosztály rendelkezik minden olyan fontos információval, amire az útvonaltervezés közben szüksége lehet az algoritmusnak, például az indulási és érkezési pontok. Rendelkezik továbbá egy *OnPlannerListener* nevű callback felülettel, melynek *onPlannerFinished* metódusát akkor kell meghívni, amikor az algoritmus végzett a tervezéssel.

Elkészült egy *AdvancePlanner* nevű útvonaltervező osztály, mely képes a két pont között útvonalat tervezni. Tervezés közben számításba veszi a lehetséges útvonalak között, hogy mennyit kell a felhasználónak sétálni, mennyit kell utazni időben és távolságban, valamint hogy hányszor kell átszállnia. Az algoritmus menete a következő:

1. A két megadott helyhez megkeressük a hozzájuk közel található buszmegállókat.
2. Ezután leellenőrizzük, hogy a két listában található-e olyan eset, amikor közvetlenül eljuthatunk a két ponthoz közeli megállókba egy járattal. Amennyiben igen, ezeket a lehetőségeket hozzáadjuk a lehetséges útvonalakhoz.
3. Ha nem található közvetlen járat, az átszállásos járatok vizsgálata történik meg.

Az algoritmus eredményül egy olyan listát ad át a *onPlannerFinished* metódusnak, melynek minden eleme több utasítást tartalmaz. Négy fajta utasítást különböztet meg az alkalmazás: sétálás indulási ponttól megállóig, sétálás megállótól érkezési pontig, buszutazás és amennyiben szükséges átszállás esetén séta két megálló között. Az algoritmus befejezése után egy új képernyő jelenik meg. Itt megtekinthetők az eredményül kapott javaslatok csoportokra bontva. Jelenleg három csoport szerint vannak rendezve az útvonalak:

- Legkevesebb utazási idő
- Legkevesebb gyaloglás
- Legkevesebb átszállás

Amint a felhasználó kiválasztja a számára optimális megoldást, visszatér az előző képernyőre, ahol már a kiválasztott útvonal vizuálisan is megjelenik, a térképen jelölve a gyaloglási útvonalat és a járat(ok) által érintett megállókat.

6. fejezet

Továbbfejlesztési lehetőségek

Az elkészült alkalmazás a felé támasztott követelményeknek eleget tesz, ennek ellenére funkcionálisára tovább bővíthető. Fontos szempont, hogy a fejlesztés végeztével tesztelési céllal is ellenőrzésre kerüljön az applikáció. Teszteléskor olyan hibák is jelentkezhetnek, amelyek a felhasználók által használt akciókból erednek. Ennél a szakasznál lényeges lehet egy külső személy bevonása az ellenőrzésbe, ezáltal lecsökkentve az alkalmazás nem megfelelő használatából eredő hibák számát. Mindemellett fontos, hogy az applikáció minél magasabb szinten elégítse ki a felhasználói igényeket, így továbbfejlesztésre és új funkciók bevezetésére is szükség lehet.

Android Room framework

Az alkalmazás egyszerűsége és fenntarthatóságának érdekében érdemes lenne az előző fejezetben ismertetett Room adatbázis absztraktiós könyvtárat használni. Segítségével egyszerűbbé és átláthatóbbá válna a kód bázis.

További városok hozzáadása

Veszprém megyében több kisváros is rendelkezik helyi tömegközlekedéssel, ami egy-kettő járattól akár húsz járatot is jelenthet. Ebből kifolyólag a továbbiakban fontos lehet a környező városok tömegközlekedésének integrálása. Ehhez az adatbázis átalakítása, továbbfejlesztése szükséges, hogy az általános érvényű legyen minden városra. Ez a funkció segítséget nyújtana a kisebb városokban élőknek, illetve azoknak a turistáknak akik a megye több városát felkeresve nyaralnak.

Többnyelvűség

Külföldről érkező turisták esetében megfontolandó a többnyelvűség támogatása. Ebben az esetben lényeges lehet egy nyelvválasztó funkció implementálása az alkalmazásba. Első sorban az angol és német nyelvek megvalósítása, de a későbbiekben akár még több nyelv bevonásával.

Útvonaltervezés továbbfejlesztése

Az alkalmazás optimálisabb működése érdekében fontos tényező lehet az útvonaltervezés továbbfejlesztése. A felhasználói élmény növekedése érdekében a tervező algoritmus pontossabbá, gyorsabbá tétele. Ehhez a jelenlegi megoldást le lehetne cserélni okosabb, heurisztikus algoritmusra. Az útvonaltervező mellett a felhasználói felületet is tovább lehet fejleszteni, ilyen feladat lehet többek közt a járatok megjelenítésének pontosítása.

Mobil értesítések

További ötletként felmerült, hogy a felhasználók számára értesítéseket küldjön ki az alkalmazás. A felhasználó beállíthatná a tartózkodási helyét, illetve egy járat indulási időpontját, az alkalmazás pedig jelezné egy értesítés formájában, ha a felhasználónak indulnia kell a megadott helyről, hogy elérje a korábban beállított járatot. Olyan esetben is hasznos lenne ez a funkció, ha útvonalváltozás történik egy buszjáratnál, így a felhasználó időben értesítve lenne a módusításról.

Widget

Érdemes lenne megvalósítani egy olyan modult, amely kitehető a telefon főképernyőjére. Az alkalmazásban, a Kedvencek menüpont alatt beállított járatok táblázatai között válthatna a felhasználó, így az alkalmazás elindítása nélkül, hamar megtekinthető lenne az egyes járatok menetrendje. Mindemellett fontos, hogy ez a modul is folyamatosan friss információkat jelenítsen meg, ha elérhető frissebb adat, az legyen jelezve a felhasználó felé is.

7. fejezet

Összefoglalás

Szakdolgozatom célja egy olyan lehetséges megoldás elkészítése volt arra a problémára, hogy Veszprém szerteágazó tömegközlekedéséről mindenki gyorsan és kényelmesen tudjon informálódni. Az alapötlet egy olyan program elkészítése volt, amely elérhető mindenki számára, és megbízható forrást nyújt az utazóközönségnek.

Mivel a város nem rendelkezik hasonló megoldással, ezért erre a hiányra alapozva készítettem el a *Vesztrans* alkalmazást. Fontos volt, hogy bárhol elérhető lehessen, emiatt Android operációs rendszerre készült el, amely a legelterjedtebb mobil operációs rendszer napjainkban. Több funkció is implementálva lett, figyelemre méltó arra, hogy a felhasználók több típusának igényeit is kielégítse. Az útvonaltervezés opció mellett az alkalmazás tartalmazza a járatok menetrendjeit, de megtekinthető a megállók pontos helyzete a városban. A jövőben könnyen illeszthetőek lesznek további funkcionálisok az alkalmazáshoz, mivel moduláris felépítésű. Ilyen fejlesztés lehet például további városok hozzáadása, alkalmazás értesítések küldése, vagy egy asztali minimodul elkészítése.

Irodalomjegyzék

[1] Google maps. URL <https://www.google.hu/maps>.

MELLÉKLET

A mellékelt CD könyvtárszerkezete