ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK	Hallgató neve:	**Baranyi Bence**
2022	Hallgató törzskönyvi száma:	**T/006123/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve:	**Baranyi Bence**
Törzskönyvi száma:	T/006123/FI12904/N
Neptun kódja:	XKI2XS

A dolgozat címe:

Valós idejű tömegközlekedési útvonaltervező alkalmazás
Real-time route planning for public transportation

Intézményi konzulens:	Simon-Nagy Gabriella
Külső konzulens:	
Beadási határidő:	2022. május 15.
A záróvizsga tárgyai:	Számítógép architektúrák Szoftvertervezés és –fejlesztés specializáció

A feladat

Tervezzen és valósítson meg egy olyan mobilalkalmazást, amely közlekedési adatok real-time elemzésével képes optimális útvonalak tervezésére a BKV hálózatán. Vizsgálja meg a BKK Futár API működését és a rajta keresztül letölthető forgalmi adatok szerkezetét. Valósítson meg egy olyan mintaalkalmazást, amely egy előzetesen beállított útvonalon valós időben monitorozza és térképen ábrázolja a késéseket, és a felhasználó haladását követve akár menet közben is képes alternatív, gyorsabb útvonalat felajánlani.

A dolgozatnak tartalmaznia kell:

- a feladat pontos ismertetését,
- a Futár API és egyéb közlekedéssel kapcsolatos adatforrások bemutatását,
- a mobil operációs rendszerek áttekintését, és a választott rendszer kapcsolódó szolgáltatásainak ismertetését,
- a feladat megoldásához alkalmazható technológiák tömör bemutatását,
- a választott megoldások indoklását,
- a tervezés és megvalósítás részletes leírását,
- a tesztelés tervét és eredményeit,
- a továbbfejlesztési lehetőségek számba vételét.

ÓBUDAI EGYETEM Neumann János Informatikai Kar 1. Budapest

Pl.

.....
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. május 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022. május 15.

.....

hallgató aláírása

TARTALOMJEGYZÉK

KÉPJEGYZÉK

TÁBLAJEGYZÉK

1. Bevezetés

Ma már a tömegközlekedést használók mindennapjainak részévé váltak az utazástervező alkalmazások. Segíthetnek abban, hogy időben induljunk el, ismeretlen helyről, vagy csak ismeretlen idősávban, de a lehető leggyorsabban, és legkényelmesebben jussunk haza. Bár több alkalmazás is nyújt már számunkra utazástervezési lehetőségeket, még mindig van fejlesztési lehetőség a területen. Két legfontosabb szempont van egy ilyen alkalmazás választásakor: a tervezett útvonalak pontossága, naprakészsége, illetve az útvonalak tervezésének időigénye. Egy tökéletes útvonaltervező alkalmazás mindig pontos, optimális útvonalakat ad a felhasználónak, lehető legkevesebb idő alatt. Ezeket a szempontokat figyelemmel kísérve, az alkalmazásom jelen időben felügyelve a késéseket és forgalmi akadályokat, akár utazásunk közben is megkísérelné az útvonalunk javítását. Ez a technológia már létezik autós navigáció alkalmazásokban, de tömegközlekedési útvonaltervezőkben nem elterjedt. Célom tehát egy gyors, erőforrás kímélő módszer implementálása, mely futtatása optimális útvonalakat biztosít, egy könnyen kezelhető felülettel társítva.

2. ALGORITMUSOK

Egy útvonaltervező alkalmazás egyik alappillére maga az útvonal megtervezése. Az útvonaltervezésnél használt megvalósítások jelentős része a tervezési teret gráfként értelmezi, az útvonal megtervezése pedig a gráfelmélet legrövidebb út problémájával [1] köthető össze. Ennek a problémának megoldására számos algoritmus létezik, és a mai napig aktuális kérdés, melyre időről időre újabb, hatékonyabb megoldásokat publikálnak.

2.1. Alapvető algoritmusok

Ebben a fejezetben olyan algoritmusokat fogok tömören bemutatni, amelyek a legrövidebb út problémára régebb óta ismert megoldások. Ezek általában ma már nem jelenteken önmagukban optimális megoldást, viszont sok modernebb algoritmus alapját képzik.

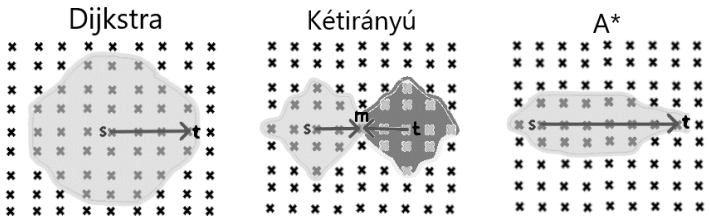
Dijkstra algoritmusa [1] [2] [3] egy olyan megvalósítás, amely a kijelölt kezdő csúcsból a gráf összes többi csúcsába megállapítja a legrövidebb utat. Az egyes csúcsokba vezető legrövidebb hosszt tárolja egy halmazban, de apró kiegészítéssel az útvonal is feljegyezhető. Egy lehetséges módosítás a hatékonyság nevében, ha az algoritmust csak addig hagyjuk keresni, míg az útvonal célcsúcsáig el nem jut. [1] [4]

Egy másik módosítása az algoritmusnak, ha a kezdőpontból és a célpontból is indítunk egyszerre egy-egy keresést. Ez a verzió akkor állítható le, ha a két keresés vizsgált elemeinek halmazaiban közös elemet találunk. [4] [5]

Más megközelítés a Bellman-Ford algoritmus [1] [3] [4], amely körökben dolgozik, minden körrel közelítve a legrövidebb utak értékét. Futása közben egy táblázatot tölt ki, melynek értékei lesznek a legrövidebb utak hosszai, minden csúcsba a kezdőcsúcsból.

Az A* keresés is a Dijkstra algoritmus egy kiegészítése, ahol az alap funkcionalitást egy potenciálfüggvénnyel egészítjük ki. Ez alapján a feldolgozandó csúcsokat sorrendbe rakjuk, így csökkentve a feldolgozott csúcsok számát.

A Dijkstra féle algoritmus, és két ráépülő módosítás által vizsgált csúcsok számát az 2.1-es ábra mutatja be



2.1 Vizsgált csúcsok ábrázolása

2.2. Komplexebb algoritmusok

Számos megoldás létezik, amelyek a gyorsabb lekérdezési idő érdekében előfeldolgozást alkalmaznak. Ilyen például az ívzászlókat használó megoldás [6], az összehúzódási hierarchián alapuló algoritmus [7], vagy a „Reach" [4]. Ezek azonban vagy egyáltalán nem, vagy nehezen egészíthetőek ki jelen idejű adatok használatához, így számomra nem alkalmasak.

Az előző fejezetben felsorolt algoritmusok hátránya, hogy csak egy szempontot vesznek figyelembe a keresés során, például a leghamarabbi érkezést. Az alábbi két algoritmus viszont úgynevezett Pareto optimális [8] megoldásokat nyújt, tehát több szempontot vesz figyelembe, és több azonosan jó eredményt is nyújt, amennyiben vannak ilyenek.

Egy ilyen megvalósítás az MLS (Multicriteria Label-Setting) [8]. Ez egy módosított Dijkstra algoritmus, ami egyszerű élsúlyok helyett címke szetteket helyez minden élre, majd ezek alapján hasonlítja össze a különböző útvonalakat. Amennyiben egy-egy útvonal nem egyértelműen Pareto domináns egy másik felett, mindkettőt eltárolja.

A másik Pareto optimumon alapuló megvalósítás, amit ki szeretnék emelni, a RAPTOR (Round bAsed Public Transit Optimized Router) [8]. Ez az algoritmus már nem Dijkstra alapú, és nem is gráfot használ alapul. Itt a keresési teret a közlekedési eszközök útvonalai jelentik, amelyeken körökben halad hullámszerűen. Egy kör egy átszállást jelent, tehát ha öt kör alatt jut el a célmegállóba, öt átszállásra lesz szükségünk. Az algoritmusnak létezik egy McRAPTOR nevű kiegészítése, amely lehetővé teszi, hogy egyéni számú kritérium alapján keresse a Pareto optimumokat.

Ezek a komplexebb algoritmusok azonban általában csak nagyobb méretű adatokkal dolgozva jelentenek gyorsabb működést. [4]

3. ÚTVONALTERVEZÉSI LEHETŐSÉGEK

Alkalmazásom központi részét képzi az útvonal tervezési megvalósítás, hiszen nem csak indulás előtt történik tervezés, hanem utazás közben is többször végezhet újratervezést az alkalmazás. Így ennek helyes megválasztása kiemelten fontos lépés az optimális felhasználói élmény érdekében.

3.1. Lokálisan, az eszközön történő útvonaltervezés

Egyik lehetséges megvalósítás, ha az útvonaltervezést az eszközön végzem el. Ennek a megvalósításnak az előnyei, hogy hálózati kapcsolat nélkül is tudunk tervezni, és a tervezés sebességét nem befolyásolja a hálózati kommunikációs késleltetés. Jelentős hátrányt jelent azonban az algoritmusok hardver igénye. Lényeges inkonzisztencia lépne fel egy gyengébb és egy erősebb eszköz között, illetve egyes megvalósítások számítási vagy memória igénye miatt egyáltalán nem is futtathatóak mobiltelefonon. Másik jelentős problémát a tervezéshez használt adatok frissen tartása jelentené. Ahhoz, hogy a tervezés mindig pontos eredményt nyújtson, az összes járat és jármű adatait nagy rendszerességgel kellene frissíteni, ami hatalmas hálózati forgalmat jelenteni a felhasználó számára.

3.2. Saját szerveren végzett útvonaltervezés

Másik lehetőség, ha a tervezést egy erős szerver gépen végzem, így nem számít a felhasználó eszközének hardvere, illetve jelentős adatforgalommal sem jár számára, hiszen a gráf frissítése a szerver feladata. A felhasználó számára az útvonal tervezés csak egy kérés lenne a szerver felé. Erre a feladatra tökéletes megoldást nyújtana az OpenTripPlanner [9], ami képes GTFS [10] tömegközlekedési adatokat és OpenStreetMap [11] térképi adatokat használva útvonal tervezést végezni. Ez egy nyílt forráskódú tervező, melynek két fő verziója van, az OTP1 és az OTP2. Ezek között a főbb kiemelendő különbség a tervező algoritmus: míg az OTP1 egy módosított A* keresést használ, az OTP2 már egy McRAPTOR implementációt. Az OTP dokumentációja azonban figyelmeztet, hogy a modernebb McRAPTOR algoritmus előnyei csak nagyobb méretű területeken érvényesek [12]. A fejlesztők az OTP2-t például egy egész országot lefedő keresési térhez javasolják.

Az általam végzett tesztelés során is azonos konklúzióra jutottam. Mindkét verziót azonos környezetben telepítettem és futtattam: egy Intel i7-8850H-s processzoron, 2.6Ghz-es órajellel, illetve a JVM számára 28 GB 2667 Mhz-es memóriát allokálva. A programokat futtattam Budapest területén, illetve a teljes norvég tömegközlekedési hálózaton. Mindkét esetben 10.000 generált útvonaltervet futtattam, a hívás elküldése és a válasz megérkezése közötti időt mérve. Budapesten az OTP1 átlagos ideje 144,72 ezredmásodperc, míg a medián érték 126 ezredmásodperc volt. OTP2-es verzióval ez az érték jelentősen csökkent 33,73 átlagos, és 31 ezredmásodperces medián értékekre. Bár ez arányosan jelentős gyorsulás, felhasználói élmény szempontjából nem jelent érezhető

változást. A norvég adatok esetében az OTP1 1202 ezredmásodperces átlagos és 1062 ezredmásodperces medián idővel adott választ, míg OTP2 esetén ezek az értékek 1036 és 969 ezredmásodperc. A különbség a két verzió között akkor látszik igazán jól, ha a keresések közül komplexitás alapján rendezve nézzük az eredmények felső 10 százalékát. Ezeket a komplex útvonalakat az OTP1 átlagosan 2551,9 ezredmásodperc alatt tervezte meg, az OTP2 pedig csak 1498,3 ezredmásodperc alatt. A medián értékek esetén is hasonló a különbség, 2484 ezredmásodperc az OTP1 esetében, és 1454 az OTP2-nél.

Ez bizonyítja, hogy a fejlettebb McRAPTOR nem minden esetben biztosít gyorsabb működést, de az is jól látszik, hogy egy Budapest méretű gráfon az egyszerűbb A* és a komplexebb RAPTOR is elfogadható gyorsasággal működik, hiszen felhasználói szempontból ezek az ezredmásodpercek nem jelentenek akkora különbséget, ami befolyásolná a használati élményt.

3.3. Futár API használata

A Futár API [13] a BKK BudapestGO alkalmazását kiszolgáló API. Ez egy szabadon elérhető, „RESTful” megvalósítás, ami számos szolgáltatást nyújt. Ezen keresztül érhető el az összes valós idejű adat a Budapesti tömegközlekedéssel kapcsolatban, például megállók menetrendi adatai, járművek pontos helyzete, de forgalmi változások is. Akár a térképen való keresést is képes kezelni, ahol budapesti utcákat, vagy megállókat ad válaszul. A kérésekre az API egy JSON formátumú választ ad. Az útvonal tervezésre az OpenTripPlanner 1-et használja, a teljes szerver kommunikációval együtt kevesebb, mint két másodperces válaszidővel tervez útvonalat, aminek válaszában több alternatívát is kínál.

4. TECHNOLÓGIAI LEHETŐSÉGEK

4.1. Java

A két hivatalos Android nyelv egyike, így a támogatottság széleskörű, a kompatibilitás biztosított. Objektum orientált, sok hasonlósággal a C#-hoz, így könnyen tanulható azoknak, akik ezt már ismerik.

iOS-re viszont nem lehet Java-ban fejleszteni, így ha programomat mindkét fő okostelefon platformon megszeretném jelentetni, akkor külön kell fejlesztenem egy iOS kompatibilis alkalmazást.

4.2. Kotlin

Az Android platform másik hivatalos nyelve, így a támogatottság szintén biztosított. Java-val együtt is működik, használata nem okoz semmilyen jelentős hátrányt, előnye viszont, hogy kevesebb a boilerplate kód, ami könnyebb olvashatóságot jelent a projektjeimnek. Bizonyos hibákat is kiküszöböl, ami tovább könnyíti a fejlesztést. [14]

Viszont Kotlin kódot sem lehet iOS-en futtatni, így a Java-hoz hasonló probléma itt is fennáll.

4.3. Objective-C

Egy objektum orientált, általános felhasználású nyelv, amely minden Apple terméken fut, lényegében C a Smalltalk nyelvvel kiegészítve. Ma már kevésbé használt, a közvetlenül iOS-re, vagy macOS-re készült alkalmazások nagy része Swiftben készül. [15]

Android alkalmazás nem írható Objective-C-ben, így ez a megvalósítás is nagyban növeli a fejlesztési igényeket.

4.4. Swift

A Swift egy Apple által fejlesztett programozási nyelv, amely a már idősödő Objective-C-t hivatott leváltani a platformjain. Ez egy multiparadigmás nyelv, amelynek célja a gyors és biztonságos programok készítése kényelmesen, az Apple ökoszisztémán belül. Objective-C és C/C++ kóddal is részlegesen kompatibilis, meghívhatunk kódot ezekből a nyelvekből is. [16]

Bár használata iOS alkalmazás fejlesztésénél nagyon kényelmes, Android alkalmazás nem írható vele, nem platform semleges.

4.5. C/C++

Androidon az Android NDK támogatja a C-ben való fejlesztést, ebben az esetben viszont nem a Java Virtual Machine-n fut a kódunk, hanem közvetlenül az eszközön. Ez nagyobb mértékű hozzáférést jelent a hardverhez, például a memóriához, illetve használhatóak C/C++ könyvtárak is fejlesztés során. Hátránya viszont, hogy jóval nehezebb fejlesztési élményt nyújt, több hibalehetőséggel. Játékfejlesztésnél nagyon elterjedt, hiszen ekkora hozzáférés mellett lehet minél jobban optimalizálni a kód futását, de az én esetemben nem biztos, hogy szükséges. [17]

iOS-re is lehet C-ben fejleszteni, de szinte minden felhasználói felület elem Objective-C-ben írt API-t használ, így elkerülhetetlen, hogy valamilyen szinten a program részévé váljon. Platformfüggetlenség részben lehetséges.

4.6. C#

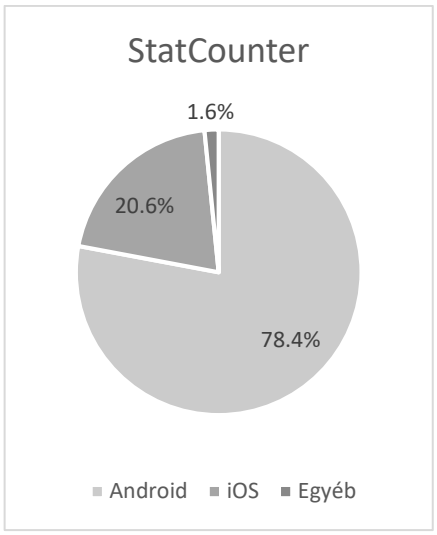
A C# a Java-hoz sokban hasonlító, de modernebb nyelv. Xamarin-nal kombinálva egyszerre lehet fejleszteni vele mindkét platformra. A Xamarin egy Visual Studio eszköz, amellyel könnyen átvihető egy alkalmazás azonos kódbázisból a két platform között. Mivel elterjedt platformközi megoldás, a támogatottsága széleskörű, és platform függetlensége nem jár különösebb teljesítményveszteséggel. Hátránya is egyben a platform függetlensége, hiszen így egyik oldalon sem hivatalos megoldás. Így néhány funkció használatba vétele extra munkát igényelhet a fejlesztők részéről, illetve általában késéssel kapnak támogatottságot a platformok legújabb funkciói. [18]

C# alapú kódot lehet használni Blazor-on keresztül is, a Mobile Blazor Bindings segítségével. Bár még fejlesztés alatt áll, segítségével a webfejlesztésnél megszokott Razor szintakszissal lehet platform független alkalmazásokat készíteni. Az alapvető felhasználói felületi elemekhez a Xamarin-t használja fel, de ez HTML kóddal is kombinálható, akár egy oldalon belül, ugyan azt a kódbázist használva. Bár alapja a webfejlesztéshez hasonlít, felhasználói oldalról mégis hagyományos alkalmazásnak látszik. [19]

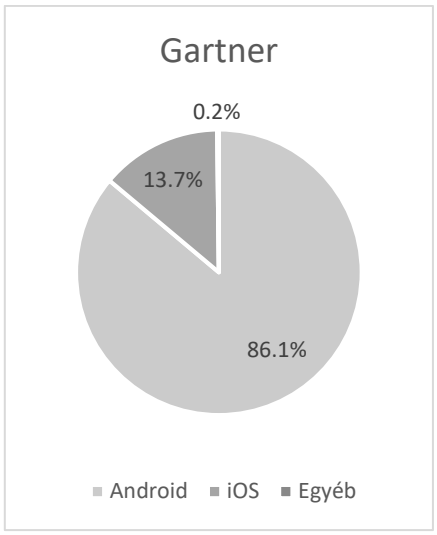
5. PLATFORM

5.1. Piaci helyzet

Fontos döntés tehát, hogy az alkalmazást milyen platformon készítem el. Ehhez tekintsünk meg pár, a mobil operációs rendszerek eloszlásához kötődő adatot. Először is ezek eloszlását két felmérési mód szerint:



5.1 StarCounter felmérés



5.2 Gartner felmérés

A StatCounter cég az internetes felkeresések száma alapján határozza meg statisztikáit. Ezek alapján az Android operációs rendszer Magyarországon 2020 novemberében a piac nagyjából 78%-át teszi ki. [20]. A statisztika eredményei az 5.1-es ábrán láthatóak

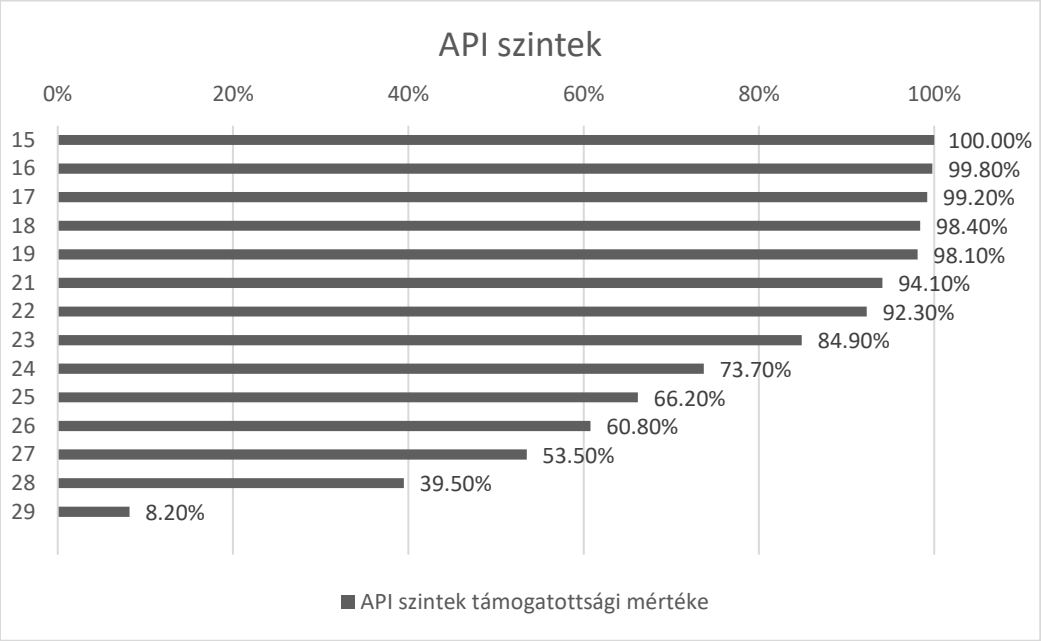
A Gartner tanácsadó cég felmérései az eladott eszközök számát veszik figyelembe. Ezen adatok alapján 2017-ben a telefon értékesítések 86% Android alapú eszköz volt. [21] A felmérés eredményei az 5.2-es ábrán láthatóak

Látható, hogy az eszközök jelentős részét Android alapú okos eszközök teszik ki, ezért ha egy alkalmazást csak egy platformon szeretnénk kiadni, sokkal nagyobb piaci réteget érünk el. A következőkben ezért csak az Android fejlesztés egyéb jellemzőit részletezem.

5.2. API szintek

Az Android platformon egy keretrendszer API segítségével érhetjük el az operációs rendszert. Ez tartalmazza többek között az alapvető osztályokat és csomagokat, az alkalmazásunk számára elérhető engedélykéréseket, és az erőforrások kezelésének módját. Általában minden Android verzió új API szinttel jár, ami elérhetővé teszi számunkra az új verzió naprakész funkcióit. Éppen ezért fontos lehet meghatározni a minimum API szintet, amin egy alkalmazás futni képes. Ha egy alkalmazás például

használ ujjlenyomat azonosítást, akkor legalább 23-as API szintet kell meghatároznia a fejlesztőknek, ugyanis az ujjlenyomatos azonosítás ettől a verziótól támogatott. [22] A minimum verzió meghatározása és az ezzel elérhető, jelenleg (2020. December) támogatott Androidos eszközök száma az 5.3-as ábrán látható:



5.3 Api szintek támogatottsági mértéke

5.3. Fejlesztési környezetek

Két főbb IDE-t vettem figyelembe lehetőségként, az IntelliJ IDEA-t és az Android Studio-t.

Az IntelliJ IDEA a JetBrains által fejlesztett Java fejlesztői környezet. Két verziója létezik, egy nyílt forráskódú ingyenes és egy fizetős verzió. Az ingyenes verzió is megfelelő Androidos fejlesztéshez, támogatja a Kotlin-t és a Java-t is, de például adatbázis kezelése limitált. A cég állítása szerint a Java alapú szoftverek több mint 70% ezzel a fejlesztői környezettel készül. [23]

Az Android Studio egy fókuszáltabb fejlesztői környezet, ezért nem minden fejlesztéshez használható, de Android alkalmazásokra igen. Támogatja a Java-t és a Kotlin nyelvet is. Fókuszáltságának előnye, hogy több Android fejlesztéssel kapcsolatos plug-in érhető el benne, mint az IntelliJ IDEA-ban. Teljesen ingyenes, ezért semmilyen funkció nincs előfizetés mögé zárva. A minimum API szint kiválasztásában is segít, adatokat nyújtva arról, hogy egyes szinteknél milyen elérhetősége lesz az alkalmazásnak. [24]

Vizuális elrendezés szerkesztést mindkettő környezet biztosít, hogy könnyen lehessen az alkalmazás kezelőfelületét fejleszteni. Android emulációra is van lehetőségünk mindkét környezetnél. [24] [25]

5.4. Térkép API

Android fejlesztésnél adja magát a lehetőség, hogy a Google Maps API-t használja az alkalmazás. Ezzel biztos teljes körű a támogatás, és a legmodernebb funkciók is elérhetőek az eszközökről. Lehet rajzolni saját elemeket a térképre, ami a járatok útvonalainak ábrázolásakor szükséges. Részben ingyenes, de komplexebb funkciói, mint például a keresésnél az automata kiegészítés funkciója már költséggel jár, melyet a használat alapján számol fel a cég. [26]

Másik lehetőség az OpenStreetMap, ami egy nyíltforrású világtérkép. Ezeket az adatokat letölthetjük, és saját szerverünkön futtatva használhatjuk, rajzolhatunk rá, kiegészíthetjük, módosíthatjuk. Léteznek hozzá API megoldások is, ám ezek non-profit jellegűek, alacsony lekérdezés igényű alkalmazásokhoz lehet csak használni. Míg a megoldás alapjában ingyenes, de ha nem szeretnénk minden adatot az alkalmazásban tárolni, saját szerverünk futtatása költséges lehet. [27]

6. EGYÉB MEGVALÓSÍTÁSOK

6.1. BudapestGO (volt BKK Futár)

A BKK saját fejlesztésű alkalmazása, amely a publikusan is elérhető FUTÁR API-t használva tervez útvonalakat. [13] Az alkalmazás 2022 februárjában egy nagy frissítésen esett át, melynek részeként új felhasználói felületet kapott, és elérhetővé vált a menetjegyek és bérletek vásárlása az appon keresztül. Az alkalmazás nevét is megváltoztatták, az eddigi BKK Futár helyett most már BudapestGO néven érhető el.

A BudapestGO kezelőfelülete könnyen áttekinthető minden platformon, keresési ideje is megfelelő. Tervezéskor több útvonalat is felajánl, ami közül választhat a felhasználó. Kiválasztás után még vizuálisan is kirajzolja az útvonalat, ezzel segítve a közlekedésünket. Mobil felületen el is tudunk menteni címeket és megállókat, sőt akár járatokat is, így könnyen elérjük adataikat, cím esetén pedig egyből tudunk oda útvonalat tervezni jelenlegi helyzetünkből. A keresést optimalizálhatjuk leggyorsabb utazási időre, legkevesebb átszállásra, vagy legkevesebb sétára. A járatok jelen idejű helyzetét is megnézhetjük az applikáción belül, vagy a netes felületen is.

Bár az API-ból származó teljesen valós idejű adatok alapján készíti útvonalterveit, utólag ezeket nem figyeli, nem vizsgálja felül. Ezért hosszabb utazáskor lehet, hogy mire egy átszálláshoz elérünk, annak helyzete változott, például dugóban ragadt, vagy esetleg pár perce már elhaladt a megállónkban. Ez ahhoz vezet, hogy jártas utazók gyakran könnyen felül tudják bírálni a FUTÁR javaslatait, ha például tudják, hogy átszállásukhoz érve már megkezdődik a reggeli csúcsforgalom, és egy villamos sokkal gyorsabb érkezést tud nyújtani, mint egy busz, hiába gyorsabb az, az indulás időpontjában előrelátható menetrend szerint.

6.2. Menetrend.app

Az alkalmazás különlegessége, hogy nem csak Budapesten, de vidéki városok némelyikében is működik. Általánosságban a GTFS specifikáció alapú, publikus menetrend adatokat használva ad válaszokat a kérésekre. Budapesten, ahol GTFS adatok mellett elérhető a FUTÁR API külső fejlesztők számára is, ott internetkapcsolat meglétekor az API-n keresztül tud valós idejű adatok alapján is útvonalat tervezni. Térképként az OpenStreetMap-et használja, ami egy szabadon szerkeszthető térkép. [28]

A Menetrend.app sok szempontból hasonlít a FUTÁR-ra, előnyei és hátrányai is nagyrészt azonosak. Előnye viszont a FUTÁR-hoz képest, hogy az elmentett „kedvencek” fülön a felhasználó részletes leírást kap a megállók forgalmát illetően, amennyiben elérhető, valós időben frissülő adatokkal. Előnyt jelent még, hogy több magyar városban is használható, folyamatosan frissülő adatokkal viszont csak Budapesten képes dolgozni.

Hátrányai közül a legszembetűnőbb, hogy csak Andriodon elérhető. Sem iOS-en, sem webes felületről nem használható az alkalmazás. Felhasználói felülete, bár minden funkcióval rendelkezik, amivel a FUTÁR, nehezebben értelmezhető, tömörebb, és vizuálisan kevésbé tetszetős. Útvonaltervezésnél a megszabható kritériumok száma is szűkebb, mint a FUTÁR-nál.

6.3. Google Maps

A Google Térképbe való integrálását a budapesti közlekedésnek a GTFS-realtime specifikáció által valósítja meg a cég. Az adatokat egy ún. Procotol Buffer adatformában továbbítja a BKK a Google felé. Ez a hivatalos leírás szerint olyan, mint az XML, csak „kisebb, gyorsabb és egyszerűbb". Mivel a GTFS-realtime adatai a FUTÁR API-nak nem elérhetőek a publikum számára, ezért ennek a használata jelenleg nem lehetséges. [10]

A Maps hatalmas előnye, hogy minden útvonaltervezést egy helyen összesít. Egy kattintással változtathatunk sétáló útvonal, autós utazás, vagy tömegközlekedési tervezés között. A Google által kezdeményezett GTFS alapú közlekedési adatbázisok segítségével még tömegközlekedési cégek, illetve fajták között is könnyen tervezhetünk. A vizuális megjelenítés is jobban kivitelezett itt, mint a másik két alkalmazásnál, ugyanis a Maps minden lehetséges útvonalat kirajzol a térképre, így nem csak számadatok alapján kell útvonalunkat megválasztanunk.

A valós időben frissülő adatokat használva itt is dinamikus adatok alapján kapunk útvonaltervet, viszont szintén csak a tervezés pillanatában optimálisnak tűnő útvonalakat veszi figyelembe. Átszállásokat a Budapest GO-val ellentétben csak gyaloglásnak tünteti fel, nem veszi figyelembe például a metróknál előforduló aluljárókat, és a megfelelő kijárat feltüntetését. Előny mellett hátrányt is nyújt, hogy több szolgáltató közlekedési eszközeit is beépíti a tervezéskor. Míg biztosak lehetünk benne, hogy minden optimális lehetőséget figyelembe vesz, a különböző fizetési zónákról, esetleges jegyvásárlási kötelezettségekről csak apróbetűs figyelmeztetések vannak. A nem valós idejű adatokat megosztó vonatok útvonalba iktatása pedig súlyosan tévedő számításokat eredményezhet, ha a vonat például késik.

7. IRODALOMKUTATÁS KONKLÚZIÓI

Programomnak egy olyan útvonaltervező alkalmazásnak kell lennie, mely képest lépést tartani sebességében és megbízhatóságában a már piacon lévő alkalmazásokkal. Lényeges, hogy megfelelő számú kritériummal lehessen benne keresni, hiszen ez már alapvető egyéb alkalmazások esetén. A mobileszköz hardverén történő megvalósítás hátrányai jelentősek, ezért mindenképpen egy szerveren futó megvalósítás az előnyös megoldás. Az elvégzett tesztjeim és az elérhető kutatási anyagok alapján nem származna lényeges előnyöm egy egyedi megvalósítás implementálásból. Így az implementációmat a Futár API használata köré építem ki.

Fontos a kényelmes használat is a sebesség mellett, hogy utazás közben a felhasználó minél kevesebb időbefektetéssel kapja meg kérésére a választ, ezért fontos a jól kidolgozott kezelőfelület. Ezért, azt tekintem legelőnyösebbnek, ha egy platformra fókuszálok, és ezen megpróbálok minél jobb alkalmazást készíteni. A piaci adatokat figyelembe véve, ezért tartom egy Android alkalmazás fejlesztését előnyösebbnek.

A megvizsgált lehetőségek közül az Android Studio-t és a Kotlin nyelvet tekintem a legjobb lehetőségnek. Jelen helyzetben mindkettő a támogatottság csúcsa Android platformon, ez a javasolt fejlesztői környezet, illetve nyelv. A Microsoft Blazor platformközi környezete is ígéretesnek tűnik, sajnos ez még csak részben kifejlett, ezért egyelőre nem tudni, milyen minőségű egy célzott, adott platformra készített alkalmazáshoz viszonyítva. Térkép API szempontjából a Google Maps használata tűnik célszerűnek számomra, egyszerűbb használata és nagyobb támogatottsága miatt. Nem tartom véletlennek, hogy a vizsgált, hasonló megvalósítások Android felületen mind a Google Maps-re építik térkép megjelenítésüket.

A jelenleg elterjedt útvonaltervező alkalmazásokat megfigyelve úgy gondolom, hogy ezt a feladatot többen is már elfogadható minőségben megoldották. Alapvető funkcióikban nehéz negatívumokat találni, mindegyik elvégzi a felhasználók A-ból B-be való eljuttatását, de a részletekben mégis vannak különbségek. Szerintem ezen alkalmazások pozitívumait egyesítve, és egy, az utazást aktívan végig kísérő, és ezt akár menet közben javító rendszerrel kiegészítve, egy olyan alkalmazást lehetne készíteni, ami tovább javítja az utazási élményt. Az autózás világában is kis változást jelentett egy GPS-hez képest a forgalmat folyamatosan figyelő mobilalkalmazás, mégis elég jelentős ahhoz, hogy ma már szinte teljesen dominálják ezek a programok.

8. SPECIFIKÁCIÓ

Egy olyan alkalmazást szeretnék készíteni, amely képes útvonalakat tervezni, és információt adni a budapesti tömegközlekedési eszközök helyzetéről. Emellett nyomonköveti az utazást és értesíti, alternatívát nyújt a felhasználónak, bármilyen fennakadás okozta késés esetén.

- Szükséges engedélyek
  - Az útvonaltervezés könnyítése érdekében a saját pozíció megadása lehetőség, akkor tervezésnél automatikusan a felhasználó pozíciójából tervez útvonalat az alkalmazás. A geolokáció engedélyezése szükséges a jármű elhagyás érzékelését végző funkcióhoz is.
  - Az érkezés előtti automatikus SMS üzenet küldés használatához engedélyezni kell az alkalmazás számára az üzenet küldést A funkció használata viszont nem kötelező.
  - Bejelentkezni/regisztrálni nem szükséges, a „Kedvencek” adatok helyben tárolódnak
- Kezdőképernyő
  - A kijelző nagy részét a térkép foglalja el, amin láthatóak a közeli megállók és járatok
    - Ha a felhasználó nem engedélyezi a helymeghatározást, akkor szimplán Budapest térképének közepe jelenik meg.
  - A kijelző tetején elérhető egy menü gomb
  - A térképet az ujjainkkal tudjuk mozgatni, nagyítani vagy kicsinyíteni
- Kezdőképernyőről nyíló menü
  - Térkép gomb
    - Visszatér a kezdőképernyőként szolgáló térkép oldalra
  - Útvonaltervezés gomb
    - Megjelenik az útvonaltervezés oldal
  - Keresés
    - Megjelenik a kereső oldal
  - Beállítások
    - Megjelenik a beállítások oldal
    - Személyes adatok törlése
    - Egyéb beállítások
- Keresés
  - Egy szöveges mező segítségével kereshetünk
  - Üres keresési mező esetén, a lista tetején a jelenlegi pozíciónk érhető el, alatta pedig a kedvencként elmentett elemek
- Útvonaltervezés
  - Kitölthető induló, köztes és érkezési pont mezők

  - Keresési beállítások módosítása
    - Tervezéshez használt járattípusok
    - Optimalizációs szempont kiválasztása
    - Sétálási sebesség megadása
  - Indulási/érkezési idő megadása
    - Időpont kiválasztása
    - A keresés fajtájának beállítása
      - Mikorra érjünk oda → legkésőbbi indulás
      - Mikor szeretnénk elindulni → legkorábbi megérkezés
  - Keresés gombra kattintva az alkalmazás kiad több lehetséges útvonalat, ami közül a felhasználó választhat
- Információt nyújtó nézetek
  - Egy-egy nézet, ahol egy kiválasztott megállóról, járatról vagy járműről kap a felhasználó tájékoztatást
- Utazást leíró nézet
  - Útvonaltervezés után jelenik meg, térképen és listán követhetjük nyomon haladásunkat
  - Részletes leírást ad a használatra kerülő járatokról, átszállási időpontokról és megállókról
  - Ha változás történik a tervezett útvonalunkon, értesítést kapunk róla, és újratervezhetünk, ezzel esetleg csökkentve az időveszteséget
  - Bekapcsolható egy SMS alapú értesítésküldés, amely a névjegyzékből kiválasztott telefonszámra üzenetet küld érkezésünk előtt pár perccel

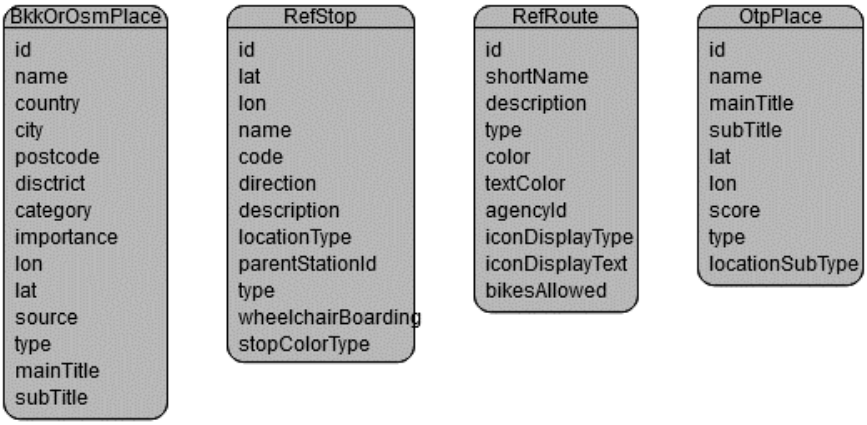
9. RENDSZERTERV

9.1. Adatbázis

Alkalmazásom tárhely igénye minimális, a felhasználó elmentett kedvencek tárolása és az alkalmazás beállításai az eltárolandó adatok összessége. Az utazástervezéshez szükséges adatokat is eltárolhatnánk lokálisan, ám offline keresés esetén az utazáshoz használt járatok figyelése elveszne, ami programunk lényeges előnye. Konkurens programok, mind autózás, mind tömegközlekedéses utazástervezés esetén is csak online működnek. Ez ma már nem szürreális elvárás, tekintve, hogy több szolgáltató már csak mobilinternet kerettel együttes csomagot ajánl vevőinek.

Ezért a preferencia adatokat a beépített SharedPreferences fájlban tárolom, a kedvencnek jelölt keresési elemek adatainak elmentésére pedig az SQLite alapú Room adatbázist. Ez a javasolt adatbázis az Android fejlesztői dokumentáció szerint, és minden igényét kielégíti az alkalmazásomnak.

Az adatbázis táblái között nincsenek kapcsolatok, hiszen különálló entitások az elmentésre kerülő keresési eredmény elemek. A táblázatok felépítését a 9.1-es ábra mutatja



9.1 Adatbázis táblázatok

Az adatbázisom négy táblából áll, oszlopaik az azonos nevű adatosztályok tulajdonságaival egyeznek. Ezek az osztályok az alkalmazásban való kereséskor megjelenő öt elem típus közül négy típust reprezentálnak: a megállókat, az útvonalakat, a BKK adatbázisában szereplő térképi pontokat, és az OpenTripPlanner adatbázisában szereplő térképi pontokat. Az ötödik kihagyott elem a forgalom változásokkal kapcsolatos riasztások, melyek általában csak rövid ideig aktuálisak, ezért elmentésüket feleslegesnek tartom. A négy kiválasztott típust elmentheti a felhasználó kedvencnek, így keresés nélkül egyből elérhető lesz számára a listában, például útvonaltervezéshez.

A két *Place* végű tábla rekordjai megegyeznek a Futár API által visszaadott keresési értékekkel, míg a megállók (*RefStop*) és útvonalak (*RefRoute*) esetében az adott elem a híváshoz referenciaként visszakapott párját mentem el. Erre azért van szükség, mert a keresési válaszban az API csak azonosítókat ad vissza, így az azonosítóhoz tartozó referencia elem elmentésével részletesebb információt kaphatunk később a kedvencnek jelöltről.

9.2. Platform

Alkalmazásom platformjának az Androidot választottam, lényeges piaci fölénye miatt. Szeretném alkalmazásomat minél jobban működővé és kényelmesen használhatóvá tenni, ezért egy platformra fogok fókuszálni. Így esélyem van a lehető leghatékonyabb kód elkészítésére.

Minimum API szintnek a 23-es szintet fogom megcélozni, ugyanis ettől a szinttől érhetőek el a záró képernyőre küldött értesítések, ami szükséges a felhasználó utazás közbeni tájékoztatásához az esetleges változásokról.

9.3. Fejlesztés

Programozási nyelvként a Kotlin-t választom támogatottsága és népszerűsége miatt. Igazán gyakorlottnak csak C#-ban érzem magam, de a Blazoros megoldást még nem találom célszerűnek. Ezért szerintem a legtöbb tanítási anyaggal, és legnagyobb támogatottsággal rendelkező nyelvet választottam.

Fejlesztői környezetnek az Android Studio-t választottam. A hivatalos dokumentáció is ezt javasolja, és kutatásaim is arra vezetett, hogy szakértőbb fejlesztők is ezt javasolják, arra az esetre, ha csak Android alkalmazást szeretnénk írni Kotlinban. Első benyomásaim szerint az emuláció és a felhasználói felület szerkesztés is sokkal egyszerűbbnek tűnik ebben a környezetben.

Jelen idejű adatok eléréséhez a Futár API-t fogom használni, amely kérésekre JSON formátumban minden releváns adatot elküld. Ezek alapján tud majd a program mindig pontos, dinamikus útvonalterveket készíteni. Publikusan elérhető és használható, így a legjobb megoldást nyújtja számomra.

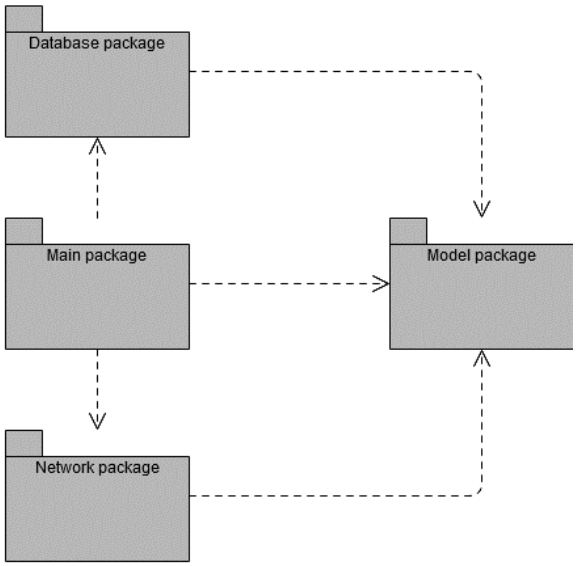
Térkép megjelenítéshez a Google Maps API-t fogom használni, mert könnyű megoldást jelent egy jó minőségű térkép integrálásához. Mivel minden eddig választott megoldásom is a Google által fejlesztett, vagy a cég által ajánlott megoldás, ezért a kompatibilitás a lehető legnagyobb fokban biztosított.

A Futár API válaszainak feldolgozásához a Retrofit [29] http klienst használom, aminek segítségével kezelem a hívásokat és az azokra érkező válaszokat is. A válaszokat

konvertálni is tudja, az én esetemben a Gson modullal kiegészítve a válaszokból data class példányokat hoz létre.

9.4. Felépítés

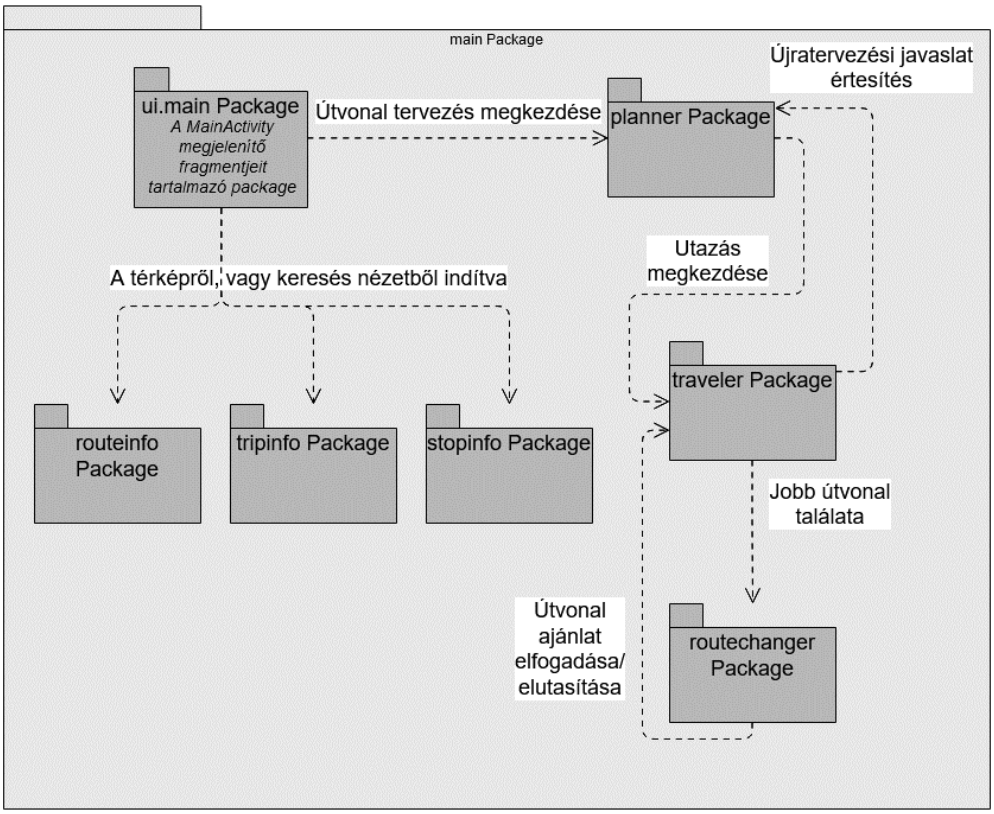
Az alkalmazás osztályai négy főbb csomagba sorolhatóak, ezek kapcsolatit a 9.2-es ábra mutatja be.



9.2 Csomagok kapcsolata

- **Database package:** A csomag feladata a perzisztens adatok tárolása, és a logikai osztályok kérésre azok előállítása
- **Main package:** Ebben a csomagban találhatóak az üzleti logika osztályai. Ezek felelősek a program főbb funkciójaiért és a megjelenítésért.
- **Model package:** Ez a csomag tartalmazza a program modell osztályait. Ezek adatosztály fájlok, amik a hálózati hívások eredményeit képviselik.
- **Network package:** A program hálózati kommunikációját végzi a Futár API-val. Ebben a Retrofit kliens segít, amely a JSON válaszokat az előbb említett adatosztályokká alakítja.

Az alkalmazás fő csomagjának (*main package*) áttekintő felépítését a 9.3-as ábra mutatja. Ezek az osztályok felelősek minden logikai feladatért, itt találhatóak a különböző *Activity*-k és *Fragment*-ek.



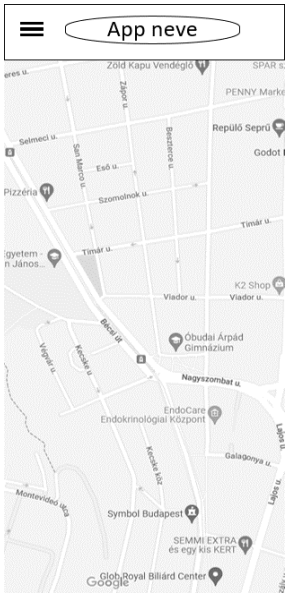
9.3 Fő csomag felépítése

Alkalmazásom „traveler” csomagjában található főbb funkcióinak logikai felépítése az alábbi szabályokon alapszik. Ezek az utazás valós idejű megfigyeléséért és javításáért felelősek.

- **Tervezetthez képest túl lassan haladó jármű észlelése:** Bizonyos időintervallumonként az alkalmazás ellenőrzi, hogy a jármű, amin éppen utazunk, jelentősen többet késik-e, mint az előző vizsgálatkor. Önmagában a menetrendhez viszonyított késést nem elegendő vizsgálni, hiszen ez lehet, hogy már az eredeti útvonalterv készítésekor is számításba lett véve. Amennyiben a jármű jelentősen növeli késésének mértékét, az alkalmazás megpróbál újratervezni, ezt a tervezést viszont kötött pályás járművekre limitálja, ezekre ugyanis kisebb eséllyel hat ki egy forgalmi dugó.
- **Jármű elhagyás érzékelése:** Bizonyos időintervallumonként az alkalmazás összehasonlítja a tervezett utazás éppen aktuális járatának pozícióját és a felhasználó pozícióját. Amennyiben a felhasználó a jármű mögött van egy bizonyos távolságon felül, megállapítható, hogy nincs a járművön. Ennek oka lehet a jármű lekésése, de a tervezett megállónál korábbi leszállás is. Ilyenkor az alkalmazás felajánlja, hogy a felhasználó jelenlegi pozíciójából újra tervez.

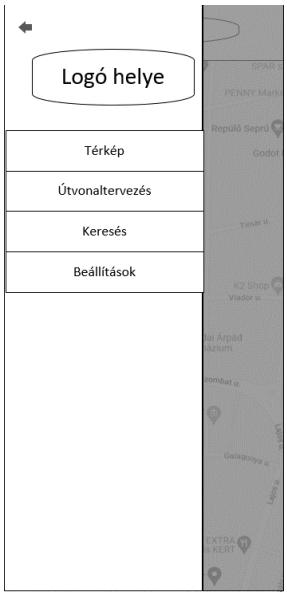
- **Jobb útvonal keresése:** Bizonyos ritkább időintervallumonként az alkalmazás megpróbál egy gyorsabb megérkezést adó útvonalat keresni. Ehhez kiindulópontnak a jelenlegi járat egy hamarosan következő megállóját használja. Amennyiben talál gyorsabb útvonalat, azt jelzi a felhasználó felé, aki eldöntheti, szeretne-e váltani, vagy megtartja a jelenlegi tervet.
- **A jelenlegi terv időadatainak frissítése:** Bizonyos időintervallumonként az utazási tervben résztvevő járművek adatait az alkalmazás lekéri a Futár API-tól, és amennyiben változás történt az eredeti tervhez képest, ezt a felhasználói felületen is megjeleníti. Késés esetén előfordulhat, hogy a felhasználó lekési a tervezett átszállást. Ilyenkor, amennyiben elérhető ilyen adat, az alkalmazás átvált a következő járatra az adott vonalon.

9.5. Felhasználói felület



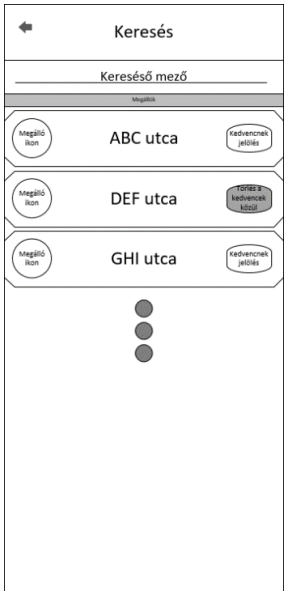
9.4 Kezdőképernyő

Az alkalmazásom kezdőképernyője egy térképes felület, melyen a felhasználó láthatja a környezetében lévő megállókat és járműveket. A kezdőképernyő terve a 9.4-es ábrán látható. A címsorból kinyitható egy menü, ahol egyéb fő funkciók oldalai érhetőek el.



9.5 Menü

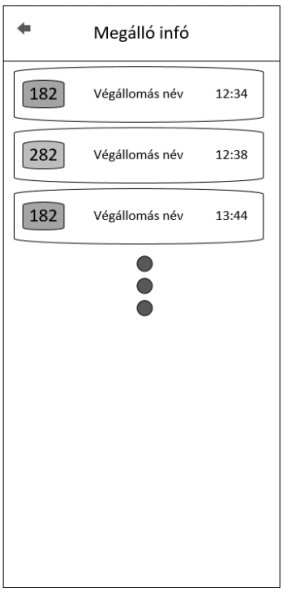
A főképernyő oldalsó sávjában megjelenő menü nyújt navigációs lehetőséget a főképernyő különböző oldalai között. Itt válthat a felhasználó a térképes nézet között, kezdhet útvonaltervezést, végezhet keresést, illetve megnyithatja a beállítások menüt. Az oldalsó menüt a 9.5-ös ábra mutatja be



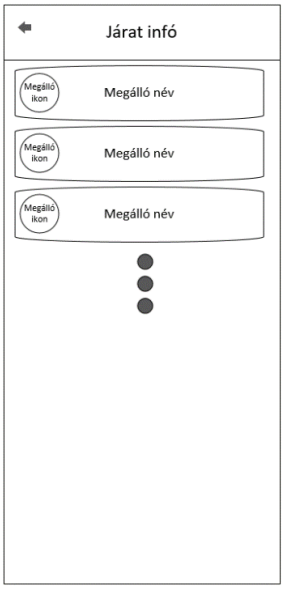
9.6 Keresés

A keresés fület megnyitva a kijelző tetején található egy szöveges beviteli mező, ahova a keresési értéket írhatja a felhasználó. Ez alá kerülnek csoportosítva a megállók, a járatok, az egyéb térképi pontok és a kereséshez kapcsolódó BKK által kiadott riasztások, például ha egy járat elterelve közlekedik. Minden elem a riasztásokat kivéve

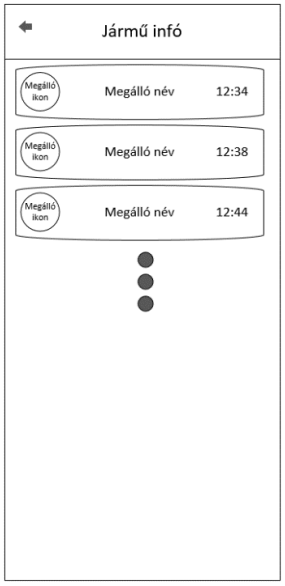
rendelkezik egy gombbal, amely segítségével a felhasználó elmentheti az adott elemet a kedvencei közé. Amennyiben a kijelző tetején lévő keresési mező üres, az alkalmazás az elmentett kedvenceket jeleníti meg. A felület terve a 9.6-os ábrán látható.



9.7 Megálló információk

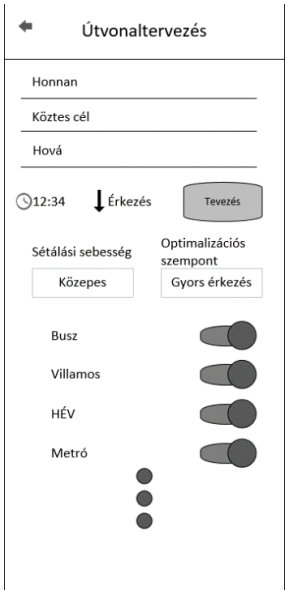


9.8 Járat információk



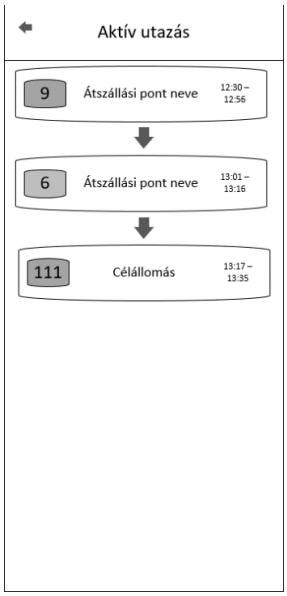
9.9 Jármű információk

A keresési felületről egy járatot, vagy a térképről egy megállót vagy járművet kiválasztva a felhasználó bővebb információhoz juthat az adott elemről. Ez általában egy felsorolást jelent, amely időrendben, vagy járat esetén érkezési sorrendben mutatja a releváns adatokat. Megálló esetében a 9.7-es, járat esetében a 9.8-as, jármű esetében pedig a 9.9-es ábrán látható a felület terve.



9.10 Útvonaltervezés

Az útvonaltervezést indító felületen a felhasználó részletesen beállíthatja az útvonallal kapcsolatos preferenciáit. A kiinduló és célállomás mellett megadhat egy köztes célt is, amit szeretne érinteni utazása közben. Személyre szabhatja a sétálási sebességét, ami átszállásoknál lehet fontos, illetve az útvonal optimalizációjának szempontját két legördülő menüből. Emellett megjelölhet kivételeket kapcsolók segítségével, amelyek segítségével meghatározhatja, milyen járműveket szeretne, illetve nem szeretne igénybe venni utazása közben. Az útvonaltervező felület tervezett megjelenését a 9.10-es ábra mutatja be.



9.11 Utazást leíró nézet

Utazás közben nyomon követheti útvonaltervének állapotát, ahol valós idejű információkat kaphat arról, ha egy járat késik, vagy esetleg lekésett egy átszállását. Az utazást leíró nézet a 9.11-es ábrán tekinthető meg.

10. Implementáció

10.1. Retrofit

A Retrofit [29] egy típus biztos REST kliens Java és Kotlin nyelvekhez. A könyvtár használatával egy olyan keretrendszert kapunk, amely jelentősen könnyíti az API-kkal való kommunikációt. A hálózati kommunikáció lebonyolítására az OkHttp [30] klienst használja. A válaszként kapott JSON vagy XML formátumú adatokat egyszerű Java objektumokká alakítja, de kiegészíthetjük az adatok szerializálását és deszerializálását végző könyvtárakkal is. Én erre a feladatra a Gson [31] könyvtárat választottam, amely a JSON adatokat az általam elkészített Kotlin adat osztályok elemeivé alakítja.

10.2. FUTÁR API

A FUTÁR API [13] egy egységes lekérdező felület, ahol ingyenesen hozzáférhet bárki minden menetrendi és jelen idejű adathoz, amire a BudapestGO alkalmazás épít. Ebbe beletartozik az összes útvonal, az összes megálló adata, minden menetrendi járat, de még az egyes járművek adatait is el lehet érni (ez alól kivétel a metró szerelvények). A megadott kérés paraméterekre az API egy JSON formátumú választ ad.

Egy útvonal tervezés kérés mintája:

```
https://futar.bkk.hu/api/query/v1/ws/otp/api/where/
    plan-trip.json
    ?showIntermediateStops=true
    &fromPlace=Tarkő utca::BKK:CSF03863
    &toPlace=Nagyvárad tér::BKK:CSF01244
    &mode=SUBWAY,SUBURBAN_RAILWAY,TRAM,TROLLEYBUS,BUS,RAIL,WALK
    &version=4
```

A kérési paraméterek még kiegészíthetőek maximum átszállások számával is, illetve a keresés optimalizációjának módjával (kevés séta, gyors érkezés, kevés átszállás). Az indulási és érkezési helyek BKK kódja helyett koordinátákat is használhatunk.

Az útvonaltervezés kérés válaszának mintája, a lényeges részekre tömörítve:

```
data: Object {
 limitExceeded: false
 entry: Object {  …
  plan: Object {
   date: 1647612160000
   from: Object { name: "Tarkő utca", stopId: "BKK_F03864", … }
   to: Object { name: "Nagyvárad tér", stopId: "BKK_F01253", … }
   itineraries:   [
    0: Object {
     duration: 2042
     startTime: 1647612178000
     endTime: 1647614220000
     …
     legs: [
      0: Object {
             startTime: 1647612178000,
             endTime: 1647613438000,
             departureDelay: 118,
             …
      }
      1: Object { … }
      2: Object { … }
      3: Object { … }
      … ]
     patternFrequency: Object { min: 11, max: 15, text: "11 - 15" }
     patternDuration: Object { min: 34, max: 38, text: "34 - 38" }
    }
    1: Object { duration: 2280, startTime: 1647612660000, … }
    2: Object { duration: 4642, startTime: 1647612213000, … }
   ]
  }
 }
 references: Object { … }
}
```

A válasz tartalmazza a választható útvonalterveket az *itineraries* listában, amelyeken belül az egyes szakaszok is külön részletezésre kerülnek a *legs* listában. Általában a terv szakaszainak későbbi ismétlődései is felsorolásra kerülnek, illetve ezek gyakorisága, a *pattern* elemekben. Amennyiben az API hívásban kérjük, a válasz tartalmaz referencia elemeket is a *references* objektumban, amelyek bármely kapcsolódó elemről nyújtanak információt, például érintett megállók részletes adatai.

Az én alkalmazásom a hálózati csomagon keresztül végzi a kommunikációt az API-val. A csomagban található egy Singleton osztály, amely a Retrofit segítségével http kéréseket állít össze, ezeket saját háttér szálon elindítja, majd a válaszokat feldolgozza. A megvalósításom pontos működésének egy példája:

1. A Futár API-tól adatot igénylő osztály meghívja a hálózati feladatokat ellátó NetworkManager singleton osztály megfelelő függvényét. Ez a mi esetünkben legyen az útvonaltervezés.

```
private fun loadPlanData() {
        NetworkManager.planTrip(
            fromPlace,
            toPlace,
            date,
            time,
            mode,
            optimization,
            walkSpeed,
            arrive,
            ::displayPlanData,
            ::showError
        )
    }
```

2. A függvénynek átadott paramétereket használva a Retrofit, az általam összeállított interfész segítségével, egy http kérést állít össze. Ez az interfész határozza meg, milyen paramétereket igényel a hálózati kérés.

```
@GET("query/v1/ws/otp/api/where/plan-trip.json")
    fun planTrip(
        @Query("key") key: String?,
```

```
        @Query("version") version: String?,
        @Query("appVersion") appVersion: String?,
        @Query("includeReferences") includeReferences: String?,
        @Query("mode") mode: String?,
        @Query("optimize") optimize: String?,
        @Query("walkProfile") walkProfile: String?,
        @Query("fromPlace") fromPlace: String?,
        @Query("toPlace") toPlace: String?,
        @Query("date") date: String?,
        @Query("time") time: String?,
        @Query("arriveBy") arriveBy: Boolean?
    ): Call<PlanTrip>
```

3. Ezt a kérést egy háttér szálon elindítom. Válasz érkezésekor, a kérést indító osztály paraméterként átadott függvénye kerül meghívásra. Ez dolgozza fel a Retrofit által adatosztályokká alakított JSON választ.

```
private fun displayPlanData(receivedPlanTrip: PlanTrip) {
        planTrip = receivedPlanTrip
        if (planTrip?.data?.entry?.plan?.itineraries?.isNotEmpty() == true){
            selectedTrip = planTrip?.data?.entry?.plan?.itineraries?.first()
            findViewById<ImageView>(R.id.ivLoading).visibility =
                View.INVISIBLE
            val tripsPagerAdapter =
                TripsPagerAdapter(supportFragmentManager, this)
            findViewById<ViewPager>(R.id.mainViewPager).adapter =
                tripsPagerAdapter
        } else {
            findViewById<ImageView>(R.id.ivLoading).background = null
            findViewById<ImageView>(R.id.ivLoading)
                .setImageResource(R.drawable.no_route_found)
        }
    }
```

A függvény feladata, hogy a kapott választ eltárolja, és amennyiben a válaszban található legalább egy útvonalterv, elrejti a töltő animációt és a megjelenítő felülethez beállítja az azt adatokkal feltöltő adaptert. Ha üres választ adott a futár, a töltő animációt lecseréli egy hibát jelző grafikára.

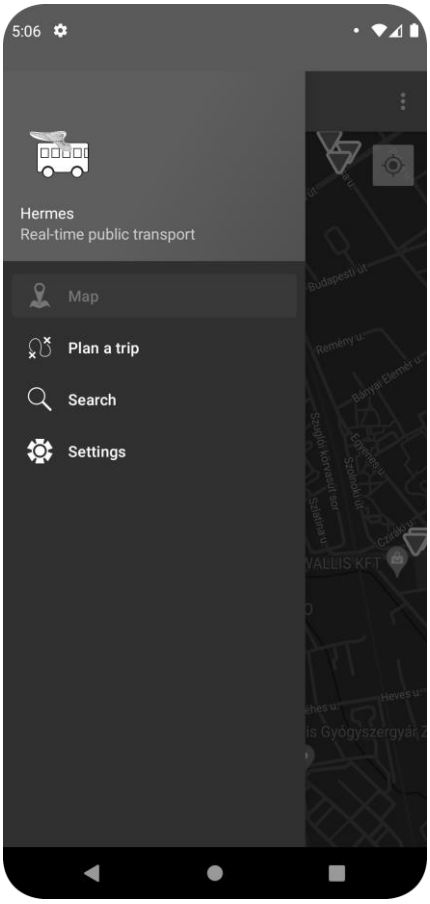
10.3. Room adatbázis

A Room [32] könyvtár egy absztrakciós réteget nyújt az SQLite adatbázis használatához. Az SQLite közvetlen használatával szemben olyan előnyöket nyújt a Room, mint az SQL kérések ellenőrzése és az ismétlődő kód mennyiségének csökkentése annotációk segítségével. A Room három fő komponensből áll. Az adatbázis osztályából, ami a perzisztens adatokat kezeli, az adat entitásokból, amik a táblákat reprezentálják és az ún. adat elérési elemekből (DAO-k), amik függvényein keresztül az alkalmazás kezelni tudja a perzisztens adatokat.

Az én megvalósításomban a Room osztályai az adatbázis csomagban találhatóak. A csomagban található a négy adat elérési elem interfésze, amelyek annotációk segítségével meghatározzák, melyik metódus végez el egy adott feladatot, például a beszúrást. Az adatbázis osztály feladata, hogy jelölje a migrációkat verziók között, és az adat elérési elem interfészek alapján elérhetővé tegye a perzisztens adatbázist a program felé. Ehhez nekünk csak absztrakt függvényeket kell definiálnunk, a konkrét megvalósítást a Room végzi.

10.4. Megjelenítő felületek és mögöttes logikájuk

10.4.1. Főmenü csomag



10.1 Főmenü

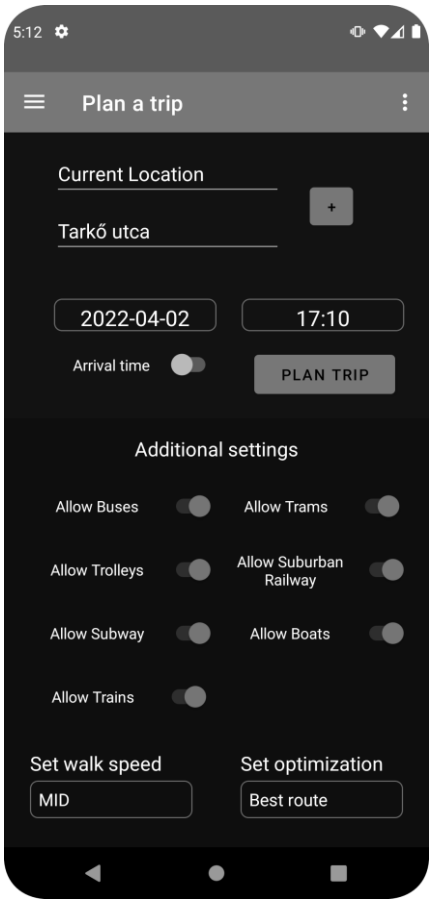
A főmenü egy oldalsó sávban található a kezdőoldalon, melynek felépítését a 10.1-es ábra mutatja be. A csomag négy részből áll. Az első a kezdőoldalként funkcionáló térképes nézet, ahol a felhasználó a környezetében lévő megállókat és járműveket láthatja. Ezekre, vagy a térkép bármely pontjára kattintva megjelenik egy felugró ablak, amely segítségével tervezést indíthatunk az adott pontból, vagy pontba. A térképes felület a 10.2-es ábrán látható.

Második eleme a csomagnak a keresés ablak, amelyet a 10.3-as ábra mutat be. Itt láthatjuk az elmentett kedvenceket, illetve végezhetünk keresést az oldal tetején lévő szöveges mező segítségével. A felhasználó által elmentett kedvenc elemeket akkor láthatja, ha a keresési mező üres. A kedvencek mellett a jelenlegi pozícióját is kiválaszthatja a felhasználó, mint tervezési pont.

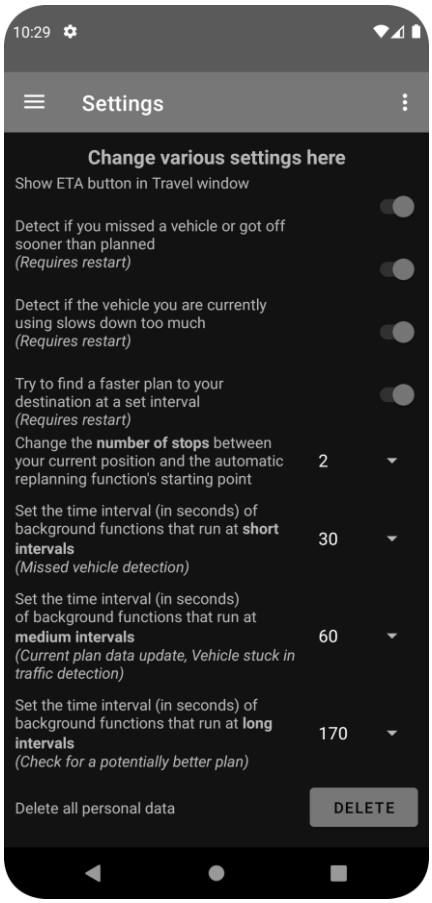
10.2 Fő térkép nézet

10.3 Kereső ablak

A másik két eleme a főmenü csomagnak a beállítások és az útvonal tervezés fül. Az útvonal tervezés fülön kezdhetjük megtervezni utazásunkat. Megadhatjuk kiinduló és célpontunkat is, illetve egy köztes megállót is megnevezhetünk. Ezen felül három optimalizációs szempont közül választhatunk, amellyel beállíthatjuk, hogy a gyors érkezés, a kevés átszállás, vagy a lehető legkevesebb séta a fő szempont számunkra. Meghatározhatjuk továbbá azt is, hogy milyen járműtípusokat szeretnénk igénybe venni, illetve, hogy milyen sebességgel sétálunk. Ezek adják az alapjait annak, hogy valóban számunkra optimális útvonaltervvel tudjon szolgálni az alkalmazás. Ez a felület a 10.4-es ábrán található.



10.4 Útvonaltervezés

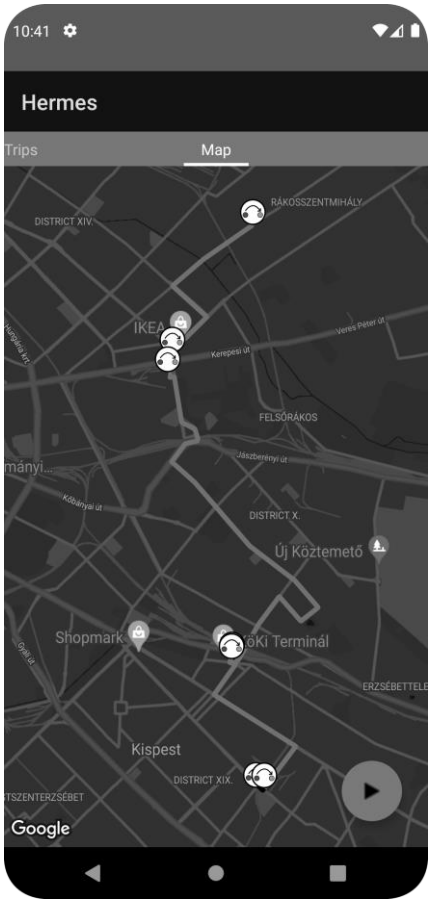


10.5 Beállítások

A beállítások fülön (10.5-ös ábra) a felhasználó pedig személyre szabhatja, mely háttérben futó szolgáltatásokat szeretné igénybe venni, illetve ezeknek futási gyakoriságát is módosíthatja. Emellett minden személyes adatot is törölhet, beleértve a beállított preferenciákat és a kedvencnek mentett elemeket is. A preferenciák az Android operációs rendszer által szolgáltatott *SharedPreferences* fájlban tárolódnak, ahonnan a különböző aktivitások kiolvashatják azokat.

10.4.2. Útvonalválasztó csomag

10.6 Útvonaltervek listája

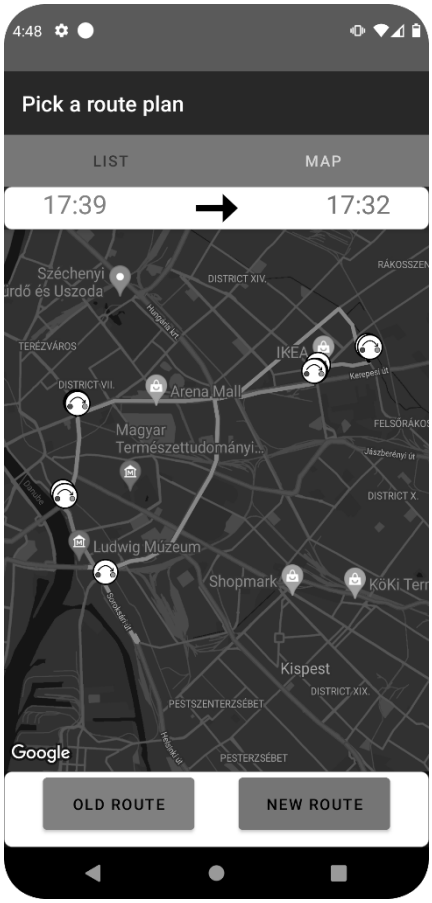


10.7 Útvonalterv térképen

A keresés gombra kattintva az alkalmazás kilistázza a Futár API által javasolt útvonalakat. Az útvonal folyamán igénybe vett járatok, illetve az útvonal várható időtartamát is ez a felület jelöli. Az útvonalakat akár térképes kirajzolásban is megtekinthetjük, ha rákattintunk az egyik útvonalra. Miután kiválasztjuk az útvonalat, a jobb alsó sarokban lévő indítás gombra kattintva megkezdhetjük utazásunkat. Ekkor indul el az utazás nyomon követés aktivitás, és ezzel együtt indulnak el a későbbiekben részletezett utazás nyomon követését szolgáló funkciók is. Az aktivitás listás felületét a 10.6-os, a térképes felületét pedig a 10.7-es ábra mutatja be.

10.4.3. Útvonalváltó csomag

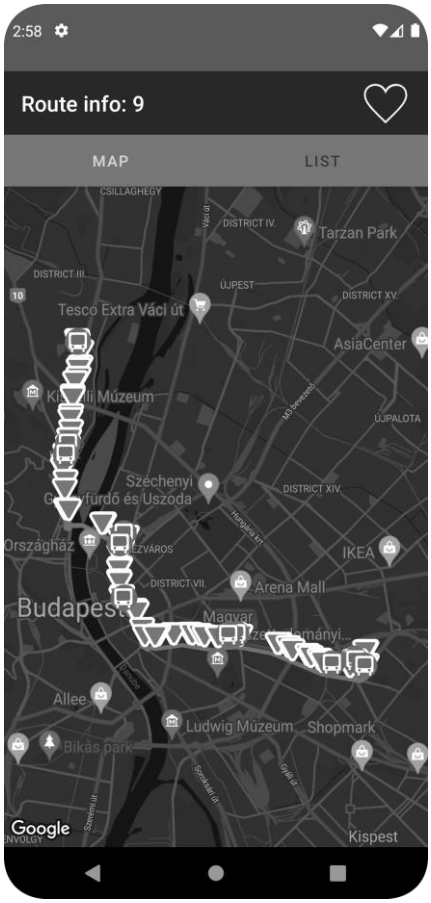
10.8 Útvonalváltás lista



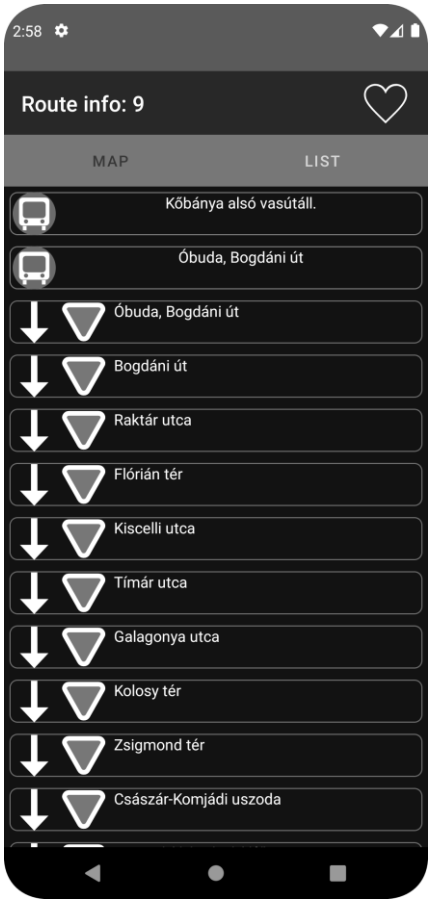
10.9 Útvonalváltás térkép

Amennyiben utazás közben az alkalmazás jobbnak vélt útvonalat talál, ezt értesítésben jelzi a felhasználó felé. Ha erre az értesítésre kattint a felhasználó, megjelenik az útvonalváltó felület. Itt a jelenlegi és az új útvonalat térképen hasonlíthatjuk össze, ahol a régi útvonal szürke színben, míg az új színesen jelenik meg. Erre mintát a 10.9-es ábra mutat. Emellett listás nézeten az új útvonal részletei is láthatóak a nyomonkövetett útvonal listás felületéhez hasonló módon. Ezt a 10.8-as ábra demonstrálja.

10.4.4. Járatról információt nyújtó csomag



10.10 Útvonal információs térkép

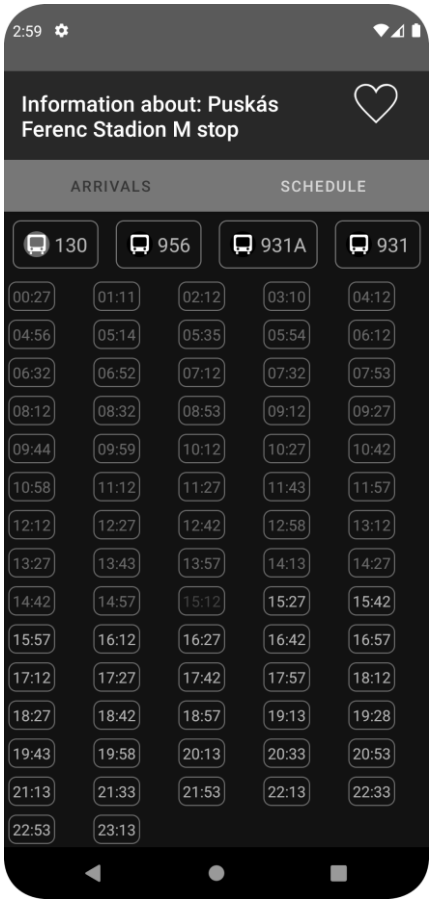


10.11 Útvonal információs lista

Ha a felhasználó a térképen, vagy a kereső felületről kiválaszt egy járatot, és a bővebb információk megtekintését választja, ez a csomag indul el. Két felületen is tájékozódhatunk a járatról. Az egyik egy térképes nézet (10.10-es ábra), ahol az összes megálló és a járatot éppen kiszolgáló járművek is megtalálhatóak. A másik egy listás nézet (10.11-es ábra), ahol a járat különböző irányait kiválasztva egy listát kapunk a megállókról, haladási sorrendben fentről lefelé. Mindkét felületen rákattinthatunk a megállókra, illetve a térképen a járművekre is, és indíthatunk akár utazás tervezést, vagy kaphatunk bővebb információkat a kiválasztott elemről. A címsorban található szív gombbal elmenthetjük vagy törölhetjük a kedvenceink közül az adott járatot.

10.4.5. Megállóról információt nyújtó csomag

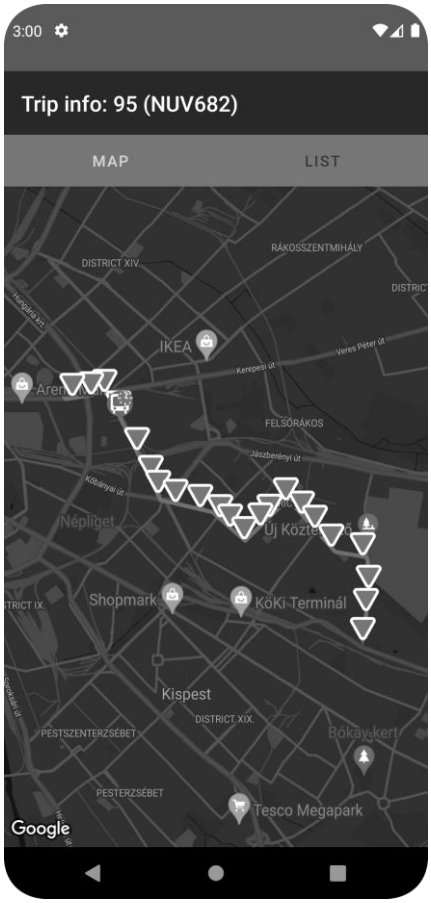
10.12 Megállóba érkező járatok



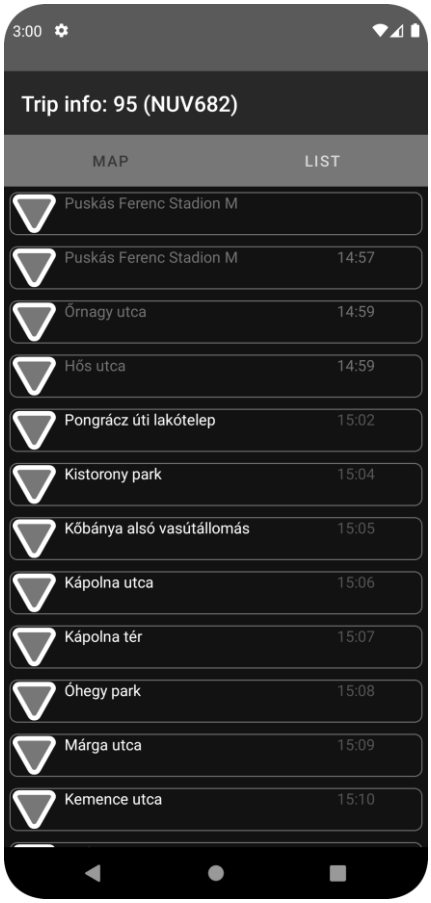
10.13 Megálló menetrend

Egy megállót kiválasztva bővebb információhoz juthatunk a megállót érintő járatokról, ahol a járatokhoz hasonlóan szintén két nézet közül választhatunk. Az egyik a hamarosan érkező járatokat listázza az érkezésük időpontjával, ami a 10.12-es ábrán látható. A másik menetrendi információt nyújt, amelyet a felhasználó szűrhet a járatok szerint. Ez a felület a 10.13-as ábrán látható. Mindkét fülön türkiz színnel jelöli az alkalmazás, ha a kiírt adat valós idejű információkra alapul, és a járat nem késik. Amennyiben késik, piros színre vált. A címsorban itt is található egy szív gomb, ami szintén a kedvencnek való jelölést, illetve törlést végzi.

10.4.6. Járműről információt nyújtó csomag



10.14 Jármű információs térkép



10.15 Jármű megállóinak listája

Ha a felhasználó megtekinti egy jármű részletes adatait, abban az esetben ez a csomag indul el. Ezen két módon tájékozódhat a jármű helyzetéről: térképen, amit a 10.14-es ábra mutat, és egy listás nézeten, ami a 10.15-ös ábrán látható. A térkép jelöli a jármű pozícióját és útvonalát, megállókkal együtt, míg a listás nézetben a megállók találhatóak sorrendben. A már elhagyott megállók szürke színnel jelöltek, míg a hátralévő megállókhoz írt érkezési idő, amennyiben nem késik a járat, türkiz, ha késik, piros színnel.

10.4.7. Utazás jelen idejű nyomon követésére szolgáló csomag

10.16 Utazás nyomon követése térképen

10.17 Utazás nyomon követése listán

Az utazás megkezdése után ezen a felületen tudja a felhasználó az utazási tervet nyomon követni. Egy térképes felületen láthatja az utazásban részt vevő járműveket, az útvonalat és az átszállási pontokat. Ezt a felületet a 10.16-os ábra mutatja be. A listás nézetben felsorolásra kerülnek az utazás szakaszai kezdő és végpontjukkal, valamint a szakasz időintervallumával. Az időintervallum színe jelzi, ha az adat valós idejű, illetve ha a járat késik. Amennyiben egy jármű késése miatt a felhasználó egy átszállásnál lekési a következő járművet, a listás nézet frissül a következő, a megtervezett útvonalba illő járműre. A listás nézet a 10.17-es ábrán látható.

10.5. Utazás jelen idejű nyomon követésének háttérben futó folyamatai

Az utazás megfigyelését egy szolgáltatáson (service) keresztül valósítom meg. Ez egy olyan komponens, amely hosszan tartó, háttérben futó feladatokat képes elvégezni, akár azután is, hogy a felhasználó az alkalmazást bezárja. A tervezés után a felhasználó

kap egy listát a lehetséges útvonalakról, majd miután kiválasztja a számára legmegfelelőbbet, az alkalmazás elindítja a szolgáltatást. Ennek feladatai a következők:

- **Érkezés előtti SMS értesítés küldése:** A felhasználó utazása megkezdése után bekapcsolhatja azt a funkciót, amely egy kiválasztott névjegyzék számára üzenetet küld 5 perccel célba érkezés előtt. Ez a funkció, és a bekapcsolását vezérlő gomb kikapcsolható a beállítások menüben.
  (Kódrészlet a 16.1-es számú mellékletben)
- **Járműelhagyás, járműlekésés észlelése:** Ez a függvény 30 másodpercenként fut le. Megvizsgálja az utazási terv szerint jelenleg használandó jármű pozícióját és a felhasználó pozícióját. Ezután megvizsgálja a jármű haladási irányát, és ehhez képest a felhasználó elhelyezkedését. Amennyiben a felhasználó a jármű mögötti 180 fokos szögben van, és legalább 100 méterrel hátrébb, arra következtet a program, hogy a felhasználót „elhagyta" a jármű. Ekkor egy értesítést küld a felhasználónak, amivel felajánlja, hogy újratervezhet a jelenlegi pozíciójából az eredeti úti célba.
  (Kódrészlet a 16.2-es számú mellékletben)
- **Frissen tartja az utazás adatait:** Ez a függvény 60 másodpercenként kerül futtatásra. Feladata egyszerű, az utazáshoz tartozó járatok adatait lekéri a hálózati csomagon keresztül a Futár API-tól, az eredményeket pedig egy listába tölti.
  (Kódrészlet a 16.3-as számú mellékletben)
- **Eltárolt utazás adatainak frissítése:** Ez a függvény 5 másodperccel az adatfrissítés után fut le. Feladata, hogy a felhasználó által kiválasztott, jelenleg aktív utazást tároló adatelem tulajdonságait frissítse a kapott adatokkal. Ilyen változások például a járművek megállóba érkezési ideje, vagy a várható érkezési idő. Amennyiben egy késő jármű miatt egy átszállást lekésne a felhasználó, két művelet közül az egyiket elvégzi:
  - Amennyiben elérhető az elmentett útvonal adatai között a lekésett járat következő érkezése az adott megállóba, a terv végéig az összes elemet átváltja a következő, még le nem késett járművek adataira, és ennek megfelelően frissíti az érkezési időpontot is.
  - Ha nem érhető el ilyen adat, értesítést küld a felhasználónak, hogy lekéste az átszállását, és újratervezés szükséges. Erre az értesítésre kattintva az alkalmazás automatikusan újratervez.

  (Kódrészlet a 16.4-es számú mellékletben)
- **Lassuló, dugóba került jármű érzékelése:** Ez a függvény is 5 másodperccel az adatfrissítés után fut le. Feladata, hogy a jelenleg használt jármű késési értékét megállapítsa, és amennyiben ez több mint két perc, összehasonlítsa az előző vizsgálat értékével. Ha az előző vizsgálathoz képest több mint 80 másodperccel nagyobb ez az érték, arra következtethetünk, hogy forgalmi elakadásba került a jármű. Ebben az esetben az alkalmazás küld egy értesítést erről a felhasználónak, amire kattintva új útvonalat választhat.

(Kódrészlet a 16.5-ös számú mellékletben)

- **Jobb útvonalterv keresése:** Ez a függvény ritkábban, 3 percenként fut le. Ez a függvény indít egy útvonaltervezést a jelenlegi járat későbbi megállójából. Az, hogy pontosan hányadik megállót használ ilyenkor a kereséshez, beállítható a beállítások menüben a felhasználó által, az alapértelmezett érték viszont kettő. A tervezés eredményét összehasonlítja az előzőleg említett függvény által aktuálisan tartott jelenlegi útvonaltervvel, és amennyiben 3 perccel kedvezőbb célba érést jelentene a tervváltás, értesíti erről a felhasználót. Ekkor egy erre készített felületen, térképes és listás felsorolás segítségével a felhasználó eldöntheti, megtartja-e a jelenlegi tervet, vagy átvált a javaslatra. Ez a függvény végzi a beérkező útvonaltervek feldolgozását dugóban ragadt jármű érzékelése esetén.
  (Kódrészlet a 16.6-os számú mellékletben)

11. TESZTELÉS

11.1. Tesztelési terv

Alkalmazásom publikus függvényeinek nagy része olyan függvény, amely aktivitások és a felettük futó töredékek közötti adatcserét végzik, illetve az Android rendszer által meghívott alapvető függvények, mint az aktivitás szüneteltetésekor lefutó függvény. Ez alól kivételt képez a fő aktivitás néhány függvénye, melyek tesztelési terve a 11.1-es táblázatban látható:

11.1 Tábla: Unit tesztek

TESZTESET	ELVÁRT EREDMÉNY
Megálló felugró ablak – Részletes információk megtekintése	Az operációs rendszer elindítja a paraméterként átadott megállóról információt nyújtó aktivitást
Megálló felugró ablak – Útvonaltervezés a kiválasztott megállóból	Az alkalmazás átvált az útvonaltervező fülre, és az aktivitásban eltárolt kiinduló állomás neve és koordinátája megegyezik a paraméterként átadott megálló adataival
Megálló felugró ablak – Útvonaltervezés a kiválasztott megállóba	Az alkalmazás átvált az útvonaltervező fülre, és az aktivitásban eltárolt célállomás neve és koordinátája megegyezik a paraméterként átadott megálló adataival
Térképi pont felugró ablak – Útvonaltervezés a kiválasztott helyről	Az alkalmazás átvált az útvonaltervező fülre, és az aktivitásban eltárolt kiinduló pont neve és koordinátája megegyezik a paraméterként átadott pont adataival
Térképi pont felugró ablak – Útvonaltervezés a kiválasztott helyre	Az alkalmazás átvált az útvonaltervező fülre, és az aktivitásban eltárolt célpont neve és koordinátája megegyezik a paraméterként átadott pont adataival
Indulási pont kiválasztása kereső nézetben	Az alkalmazás átvált az útvonaltervező fülre, és az aktivitásban eltárolt adatok

	megegyeznek a paraméterként átadott indulási ponttal.
Köztes pont kiválasztása kereső nézetben	Az alkalmazás átvált az útvonaltervező fülre, és az aktivitásban eltárolt adatok megegyeznek a paraméterként átadott köztes ponttal.
Célpont kiválasztása kereső nézetben	Az alkalmazás átvált az útvonaltervező fülre, és az aktivitásban eltárolt adatok megegyeznek a paraméterként átadott célponttal.

Az alkalmazásom fő tesztelési módszertana a manuális funkcionális tesztek, amelyek a hibamentes felhasználói élményt vizsgálják. Olyan szélső eseteket vizsgálnak, amelyek ritkán fordulnak elő, de hibás működésre vezethetnek. Ezen tesztesetek tervezete a 11.2-es táblázatban találhatóak.

11.2 Tábla: Funkcionális tesztek

TESZTESET	ELVÁRT EREDMÉNY
Általános tesztek	
A felhasználó internet kapcsolat nélkül indítja el az alkalmazást	• Felugró üzenetben értesíti az alkalmazás a felhasználót, hogy nincs internetkapcsolata • Amennyiben a kedvencek használatával mégis megpróbál útvonaltervezést indítani, a tervezés nézet jelzi, hogy nincs internetkapcsolat
Megszakad a hálózati kapcsolat használat közben	Amikor az alkalmazás a háttérben megpróbál a Futár API-val kommunikálni, felugró szöveges üzenettel értesíti a felhasználót az internetkapcsolat hiányáról
Az operációs rendszer energiatakarékos módra vált	Ha ennek következtében leáll a szolgáltatás, értesítést küld a felhasználónak a nem rendeltetésszerű leállásról, amire kattintva újraindíthatja azt.

Váltás sötét és világos mód között	Az android operációs rendszer ilyenkor újraindítja az aktuális aktivitást, így ha épp útvonal választás közben vált a telefon, újratervezés szükséges. Egyéb esetekben a nézet gond nélkül újraindul
Útvonaltervezés specifikus tesztek	
Azonos induló és célpont tervezésnél	A Futár API üres választ nyújt, így az eredményt mutató ablak hibás tervezést jelentő grafikát mutat
Budapesten és agglomerációs körzetén kívüli tervezési pont megadása	A Futár API üres választ nyújt, így az eredményt mutató ablak hibás tervezést jelentő grafikát mutat
BKK által támogatott dátumon kívüli tervezés	A dátum választó felületen nem elérhető olyan dátum, ami nem támogatott. Ha a felhasználó mégis olyan dátumra tervez valamilyen módon, a hibás tervezést jelentő grafika jelenik meg
Személyre szabási lehetőségeknél minden jármű tiltottnak jelölése	Amennyiben a távolság megtehető gyalog, a tervezés szokásos módon történik. Ellenkező esetben a hibás tervezést jelentő grafika jelenik meg
A felhasználó nem engedélyezi a helymeghatározást, és így próbál a jelenlegi pozícióját felhasználva tervezni	Az alkalmazás a hibás tervezést jelentő grafikát jeleníti meg.
Keresési specifikus tesztek	
A felhasználó a kereső mezőbe karakterektől eltérő tartalmat próbál beszúrni	Az alkalmazás egy felugró szöveges üzenettel figyelmezteti a felhasználót, hogy nem támogatott a formátum
Beállítások menü tesztek	
Beállítások módosítása, majd az alkalmazás rögtöni bezárása	A beállítások a nézet szüneteltetése esetén történnek, így amíg ezen a fázison átesik az alkalmazás, a beállítások mentésre kerülnek
Utazás nyomon követés tesztek	

Helymeghatározás nélkül utazás nyomon követés	Azon funkciói a szolgáltatásnak, amelyekhez szükséges a felhasználó pozíciója nem futnak le, de hibára nem kerül sor
A háttérszolgáltatás leállításra kerül az operációs rendszer által utazás közben	Nem rendeltetésszerű leállás esetén a szolgáltatás küld a felhasználónak egy értesítést, amire ha rákattint, újraindul
Járat információs nézet tesztek	
Olyan járat vizsgálata, amelyhez nem tartozik útvonal, vagy aktív jármű (pl.: ASP jelölésű útvonal nélküli járművek)	Útvonal hiányában a térkép Budapest középpontjára mutat, és amennyiben vannak járművek, azokat gond nélkül megjeleníti. A listás nézet üresen marad
Több végállomással rendelkező járat, vagy egy végállomásos körjárat kiválasztása	A BKK Futár API jelenlegi állapotában minden járatot két végállomásra transzformál, amennyiben ez változtatásra kerül, a listás nézet képes több mint két irányt feldolgozni
Jármű információs nézet tesztek	
Olyan járműt választ ki a felhasználó, amihez nem tartozik útvonal	Az alkalmazás a térképen megjeleníti a járművet, a listás nézet üresen marad
A jármű vizsgálat közben lecsatlakozik a Futár API szolgáltatásról (áramtalanítás)	Az alkalmazás az utolsó aktuális adatot mutatja a felületeken.
Megálló információs nézet tesztek	
Olyan megálló vizsgálata, ahol menetrend szerint egy járat sem áll meg	Az érkező járatok és a menetrendet listázó nézet is üres marad, de a program nem fut hibára

11.2. Tesztelési eredmények

A tesztek eredményeit a 11.3-as táblázat tartalmazza, kategóriákra bontva:

11.3 Tábla: Teszteredmények

TESZT CSOPORT	EREDMÉNY
Unit tesztek	Minden teszt az elvárt eredményt hozta
Általános tesztek	Minden teszt az elvárt eredményt hozta
Útvonaltervezés specifikus tesztek	Minden teszt az elvárt eredményt hozta
Keresési specifikus tesztek	Minden teszt az elvárt eredményt hozta
Beállítások menü tesztek	Minden teszt az elvárt eredményt hozta
Utazás nyomon követés tesztek	Minden teszt az elvárt eredményt hozta
Járat információs nézet tesztek	Minden teszt az elvárt eredményt hozta
Jármű információs nézet tesztek	Minden teszt az elvárt eredményt hozta
Megálló információs nézet tesztek	Minden teszt az elvárt eredményt hozta

11.3. Konklúzió

Az alkalmazásom a tesztelési tervnek megfelelt, a specifikációban foglaltakat teljesíti, a piacon elérhető konkurens tömegközlekedési útvonaltervező alkalmazásokkal szemben több előnyt is nyújt. A Futár API által nyújtott valós idejű adatokat szélesebb körben felhasználva az utazás adatai frissen tarthatóak és az utazás optimalizálására lehetőség van, ezek által a felhasználói élmény javítható.

12. TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

- **Az alkalmazás lefordítása több nyelvre:**
  Magyar, illetve angol nyelvű felületekkel készítettem el az alkalmazást, hogy nemzetközi felhasználók számára is elérhető legyen, amit egyéb nyelvekkel is kiegészítve növelhetném a felhasználói kényelmet.

- **A felhasználó szokásainak figyelembe vétele:**
  Például gyakran látogatott megállók felajánlása tervezésnél, vagy gyakran igénybe vett járatokat tartalmazó útvonaltervek előnyben részesítése. Ehhez valamilyen statisztika gyűjtő szolgáltatásra lenne szükség, ami tanul a felhasználó viselkedéséből. Kiegészíthető lenne akár egy olyan funkcióval is, amely megfigyeli a felhasználó rendszeres tervezési időpontjait, például munkába indulás előtti tervezés, és az ilyen időpillanatokban előre felajánlhatna egy útvonalat értesítésen keresztül.

- **Várható forgalmi torlódások előzetes figyelembe vétele tervezésnél:**
  A forgalmi adatokról szükséges lenne egy adatgyűjtemény, ami alapján vagy előnyben részesítené az alkalmazás a Futár API által nyújtott olyan útvonalakat, amelyek a várhatóan forgalmas szakaszokat elkerülik, vagy egy saját API implementációra lenne szükség, amelynek az útvonaltervező algoritmusa ezeket az adatokat is figyelembe veszi tervezésnél.

- **Okosóráról való tervezés indítása, útvonalválasztás és utazási terv nyomon követés:**
  Viselhető okoseszközről az alapvető funkciók kezelése, például kedvencnek elmentett megállók közötti útvonaltervezés és az utazás nyomon követését szolgáló listás felület megjelenítése.

- **Útvonaltervezés kezdeményezése egy jármű következő megállójából:**
  Útvonaltervezés kezdeményezésénél a jelenlegi jármű észlelése és felajánlása, mint kiindulási pont.

13. Összefoglaló

Szakdolgozatom célja egy olyan mobilalkalmazás megtervezése és elkészítése volt, amely Budapest tömegközlekedési hálózatán az útvonal megtervezése után a célba érkezés pillanatáig figyeli az útvonaltervet, és valós idejű adatok alapján megpróbálja a felhasználót minél optimálisabb módon célba juttatni. Ennek érdekében megvizsgáltam több algoritmust, melyek a gráfelmélet legrövidebb út problémájára nyújtanak megoldást. Vizsgálataim során arra jutottam, hogy a modernebb algoritmusok, mint a RAPTOR (Round bAsed Public Transit Optimized Router) jelentős előrelépést csak nagyméretű gráfoknál nyújtanak, kisebb méretű keresési tereknél például az A* algoritmus is megfelelő megoldást adhat. Figyelembe vettem az útvonaltervezés legmegfelelőbb megvalósítási módját is, miszerint a tervezést a mobileszközön, egy saját implementálású szerveren, vagy a BKK Futár API-n keresztül érdemes végezni. A tervezés a felhasználó eszközén való elvégzése több súlyos problémát is magával vonna, mint a jelentős hálózati és erőforrás igény, így csak a szerver oldali megvalósításokat vizsgáltam részletesebben. Ezen vizsgálatok eredménye alapján megállapítottam, hogy ahhoz, hogy egy RAPTOR alapú szerver megvalósítás jelentős tervezési gyorsulást eredményezzen, legalább egy országos méretű, komplex tömegközlekedési hálózatra van szükség. Budapest tömegközlekedési hálózata ilyen szempontból kisméretűnek számít, tehát elegendő egy egyszerűbb algoritmus is, így a Futár API használatát választottam, amely tervezéshez egy módosított A* keresést alkalmaz. Az okos telefonok piaci helyzete alapján a felhasználók jelentős része Android operációs rendszerű eszközt használ Magyarországon, ezért az alkalmazásomat ezen a platformon implementáltam. Megvalósításom perzisztens adattárolásához a Room adatbázist választottam, ami egy SQLite alapú könyvtár. A valós idejű adatok lekérdezéséhez és az útvonaltervezéshez is a BKK Futár API-t használtam, a beérkező adatok feldolgozásához pedig a Retrofit könyvtárat. Alkalmazásom fő funkciói a járatokról, járművekről, megállókról való információ lekérés, útvonaltervezés, megállók és egyéb tájékozódási pontok elmentése kedvencek közé, és az utazás jelen idejű nyomon követése célba érésig. Ezen funkciókhoz többnyire társul egy-egy megjelenítő felület, ahol általában egy térképes és egy listás felületen tud a felhasználó a kért adatokhoz hozzájutni. Az utazás nyomon követését szolgáló funkciók egy háttérben futó szolgáltatás keretében működnek. Ez a szolgáltatás az utazás megkezdésekor indul el és célba éréskor, vagy a felhasználó kérésére áll le. Ilyen háttérben futó funkció például a jármű elhagyás/lekésés érzékelése. Ekkor a felhasználó egy értesítésre kattintva tervezhet újra jelenlegi pozíciójából. Egy másik funkció a tervezett útvonal adatainak frissen tartása, késő járművek és lekésett átszállások jelzése. A szolgáltatás időnként megpróbál gyorsabb útvonalat is keresni, amennyiben talál, erről is értesíti a felhasználót, aki ezután eldöntheti, elfogadja-e az új útvonaltervet. Ezekkel a funkciókkal az alkalmazásom sikeresen megvalósítja a specifikációban foglaltakat.

14. SUMMARY

The goal of my thesis was to design and develop a smartphone application for public transport route planning in Budapest, which would observe the planned route until the user reached their goal, and try to optimize it further during travel using real-time data. To achieve this I examined several algorithms designed to solve the shortest path problem of graph theory. Through my examinations, I found that modern algorithms, like RAPTOR (Round bAsed Public Transit Optimized Router) only provide a noticeable increase to planning speed in large enough networks. On a smaller scale, a simpler algorithm like A* can give similar results. I also considered the best way to implement route planning in my application, whether directly on the device, through a custom server implementation, or using BKK's Futár API. Planning on-device would impose serious problems, such as heavy mobile data usage and high system requirements for optimal speeds. Thus, I only examined server side implementations more thoroughly. The results of my tests supported my previous findings, that a RAPTOR based server application would only provide noticeably better results on countrywide networks. For an application that plans only on Budapest's public transport network, a simpler algorithm would be sufficient. This led me to use the Futár API, which uses a modified A* for route planning. Based on market statistics, the majority of Hungarians uses Android based devices, which is why I chose to develop my app on Android as well. For persistent storage, I chose the Room database library, which is an SQLite-based library. For querying real time data, and route planning, I utilised the Futár API, and processed the response data with the Retrofit library. The main functions of my application include querying information detailing routes, vehicles, and stops, route planning, favoriting stops, routes, and other locations, and real-time tracking of the user's planned route. For most of these functions, there is a dedicated user interface, where the user can usually view the requested information on a map and a list based layout. The real-time tracking functions are contained in a background service, which automatically starts after the user selects a route, and ends once they reach their goal, or manually stop the service. Such a function is the detection of missing or getting off a vehicle too soon. In this case, the user can plan again from their current location by tapping on a notification sent by the service. Another such function is the updating of the chosen travel plan to show delays and missed connections. The service also tries to find a faster plan at certain intervals, and if it does, it notifies the user, who can decide to change to the new plan or keep the current one. With these functions in place, my application satisfies the criteria outlined in this document's specifications.

15. FORRÁSOK

[1] L. Rónyai, G. Ivanyos és R. Szabó: Algoritmusok. *Typotex*, 2005.

[2] E. W. Dijkstra: „A note on two problems in connexion with graphs.” *Numerische Mathematik,* Vol. 1. Iss. 1., 1959, 269-271 old.

[3] T. Nagy: „Gráfalgoritmusok,” (https://ntibi.web.elte.hu/programozas/elmelet/Gr%E1falgoritmusok.pdf), utoljára megtekintve: 2020. 10. 24.

[4] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner és R. F. Werneck: „Route Planning in Transportation Networks,” Algorithm engineering, *Springer*, 2016, pp. 19-80.

[5] G. B. Dantzig: „Linear Programming and Extensions,” *Princeton University Press*, 1963.

[6] M. Hilger, E. Köhler, R. H. Möhring és H. Schilling: „Fast point-to-point shortest path computations with arc-flags,” *The Shortest Path Problem: Ninth DIMACS Implementation Challange,* Vol. 74., 2009, 41-72 old.

[7] R. Geisberger, P. Sanders, D. Schultes és C. Vetter: „Exact routing in large road networks using contraction hierarchies,” *Transportation Science*, Vol. 46. Iss. 3., 2012, pp. 388-404.

[8] D. Delling, T. Pajor és R. F. Werneck: „Round-Based Public Transit Routing,” *Transportation Science,* 49. Vol. Iss. 3. 2015, pp. 591-604.

[9] OpenTripPlanner: „OpenTripPlanner2,” (http://docs.opentripplanner.org/en/latest/), utoljára megtekintve: 2022. 03. 13.

[10] Google: „GTFS Realtime Overview,” Google, (https://developers.google.com/transit/gtfs-realtime/), utoljára megtekintve: 2011. 11. 1.

[11] OpenStreetMap: „OpenStreetMap,” OpenStreetMap Foundation, (https://www.openstreetmap.org/about), utoljára megtekintve: 2022. 03. 13.

[12] OpenTripPlanner: „Comparing OTP2 and OTP1," (http://docs.opentripplanner.org/en/latest/Version-Comparison/), utoljára megtekintve: 2022. 03. 13.

[13] Apiary.io: „BKK FUTÁR Utazástervező API," (https://bkkfutar.docs.apiary.io/), utoljára megtekintve: 2020. 11. 1.

[14] Google, „Develop Android apps with Kotlin," (https://developer.android.com/kotlin), utoljára megtekintve: 2020. 11. 27.

[15] Apple, „Programming with Objective-C," (https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html), utoljára megtekintve: 2020. 11. 27.

[16] Apple Inc., „About Swift," (https://docs.swift.org/swift-book/index.html), utoljára megtekintve: 2020. 11. 27.

[17] Google, „Android NDK," (https://developer.android.com/ndk/), utoljára megtekintve: 2020. 11. 27.

[18] Microsoft, „Xamarin," (https://dotnet.microsoft.com/apps/xamarin), utoljára megtekintve: 2020. 11. 27.

[19] Microsoft, „Experimental Mobile Blazor Bindings," (https://docs.microsoft.com/en-us/mobile-blazor-bindings/), utoljára megtekintve: 2020. 11. 27.

[20] StatCounter GlobalStats, „Mobile Operating System Market Share Hungary," (https://gs.statcounter.com/os-market-share/mobile/hungary), utoljára megtekintve: 2020. 12. 06.

[21] Gartner, „Gartner Says Worldwide Sales of Smartphones Grew 9 Percent in First Quarter of 2017," (https://www.gartner.com/en/newsroom/press-releases/2017-05-23-gartner-says-worldwide-sales-of-smartphones-grew-9-percent-in-first-quarter-of-2017), utoljára megtekintve: 2020. 12. 06.

[22] Google, „<uses-sdk>," (https://developer.android.com/guide/topics/manifest/uses-sdk-element), utoljára megtekintve: 2020. 12. 6.

[23] JetBrains, „IntelliJ IDEA,” (https://www.jetbrains.com/idea/), utoljára megtekintve: 2020. 12. 7.

[24] Google, „Android Studio,” (https://developer.android.com/studio), utoljára megtekintve: 2020. 12. 7.

[25] JetBrains, „Tutorial: Create your first Android application,” (https://www.jetbrains.com/help/idea/create-your-first-android-application.html), utoljára megtekintve: 2020. 12. 7.

[26] Google, „Maps SDK for Android,” (https://developers.google.com/maps/documentation/android-sdk/overview), utoljára megtekintve: 2021. 01. 19.

[27] Operations Working Group, „API Usage policy,” OpenStreetMap Foundation, (https://operations.osmfoundation.org/policies/api/), utoljára megtekintve: 2021. 01. 19.

[28] Menetrend.app, „Adatforrások,” (https://menetrend.app/data-sources), utoljára megtekintve: 2020. 11. 1.

[29] Square Inc., „Retrofit,” (https://square.github.io/retrofit/), utoljára megtekintve: 2021. 06. 30.

[30] Square, „OkHttp,” (https://square.github.io/okhttp/), utoljára megtekintve: 2022. 03. 18.

[31] Google, „Gson,” (https://github.com/google/gson), utoljára megtekintve: 2022. 03. 18.

[32] Google, „Save data in a local database using Room,”: (https://developer.android.com/training/data-storage/room), utoljára megtekintve: 2022. 03. 31.

[33] M. Müller-Hannemann és M. Schnee: „Finding all attractive train connections by multi-criteria Pareto search.,” *Algorithmic Methods for Railway Optimization, Springer,* 2007, pp. 246-263.

16. MELLÉKLETEK

16.1. Érkezés előtti SMS értesítés küldése kódrészlet

```
private fun sendETA() {
        if (shouldSendETA) {
            val localCal = Calendar.getInstance()
            val fiveBefore = selectedTrip?.endTime!! - 300000 //5 minutes
            if (localCal.timeInMillis > fiveBefore) {
                try {
                    val smsManager = SmsManager.getDefault()
                    smsManager.sendTextMessage(
                        phoneNumber,
                        null,
                        getString(R.string.eta_default_text),
                        null,
                        null
                    )
                    shouldSendETA = false
                } catch (e: SecurityException) {
                    makeToast("Could not send ETA text message")
                }
            }
        }
    }
```

16.2. Járműelhagyás, járműlekésés észlelése kódrészlet

```
private fun isUserOnVehicle() {
        if (tripDetails.isNotEmpty()) {
            var isUserOnRide = false
            val localCal = Calendar.getInstance()
            for (leg in selectedTrip?.legs!!) {
                val currentTrip = tripDetails.find { x ->
                    x.data.entry.tripId == leg.tripId }
                if (currentTrip != null && localCal.timeInMillis <
                        leg.endTime && localCal.timeInMillis>leg.startTime){
                    if (isUserOnRide) break
                    if (currentTrip.data.entry.vehicle != null &&
                            mLastLocation != null &&
                            localCal.timeInMillis < leg.endTime &&
                            localCal.timeInMillis > leg.startTime) {
                        val vehicleLatLng = LatLng(
                            currentTrip.data
                                .entry.vehicle.location.lat.toDouble(),
                            currentTrip.data
                                .entry.vehicle.location.lon.toDouble()
                        )
                        val userLocation = LatLng(
                            mLastLocation!!.latitude,
                            mLastLocation!!.longitude
                        )
                        val vehicleHeadingRelUser =
                            (vehicleLatLng.sphericalHeading(userLocation)+
                                360) % 360
                        val userAngle =
                            ((vehicleHeadingRelUser -
                                currentTrip.data
                                .entry.vehicle.bearing) + 360) % 360
                        if (userAngle > 90 && userAngle < 270) {
                            isUserOnRide = userLocation
                                .sphericalDistance(vehicleLatLng) < 100
                            if (!isUserOnRide) {
                                … (Értesítés összeállítása és megjelenítése)
        }}}}}}}
```

16.3. Utazás adatainak frissen tartása kódrészlet

```
private fun refreshTripDetails() {
        tripDetails.clear()
        val localCal = Calendar.getInstance()
        for (leg in selectedTrip?.legs!!) {
            if (leg.tripId != null) {
                NetworkManager.tripDetails(
                    leg.tripId,
                    String.format(
                        "%04d%02d%02d",
                        localCal.get(Calendar.YEAR),
                        localCal.get(Calendar.MONTH) + 1,
                        localCal.get(Calendar.DAY_OF_MONTH)
                    ),
                    ::addTripDetailsToList,
                    ::showError
                )
            }
        }
    }
```

```
private fun addTripDetailsToList(receivedTripDetails: TripDetails) {
        tripDetails.add(receivedTripDetails)
    }
```

16.4. Eltárolt utazás adatainak frissítése kódrészlet

```
private fun updateCurrentPlanTimes() {
    for (leg in selectedTrip?.legs!!) {
        if (previousTrip != null) {
            if (leg.mode == "WALK") {
                leg.startTime += timeDifference
                leg.endTime += timeDifference
            } else {
                … (jelenleg vizsgált útvonal adatainak kikeresése)
                if (startStop.predictedDepartureTime * 1000 <
                        previousLeg?.endTime!! && null vizsgálatok){
                    if (selectedTrip?.patternItineraries?.size!!>1){
                        … (átlépteti a fentmaradó szakaszokat a
                           következő időpontjukra)
                    } else {
                       isMissedConnection = true
                       checkForBetterRoute()
                    }
                    timeDiffernce = 0
                } else {
                    timeDifference +=
                        if (endStop?.predictedArrivalTime != null) {
                            ((endStop.predictedArrivalTime!! * 1000) -
                                leg.endTime) – timeDifference
                        } else {
                           -timeDifference
                        }
                }
                leg.endTime += timeDifference
            }
            previousLeg = leg
        }
        … (első iterációban a szakasz megjelölése mint „előző szakasz”, és a
           kezdő késési idő [timeDifference] kiszámolása)
    }}
```

16.5. Lassuló, dugóba került jármű érzékelése kódrészlet

```
private fun isVehicleTooSlow() {
        if (tripDetails.isNotEmpty()) {
            val localCal = Calendar.getInstance()
            … (aktuálisan használt jármű megkeresése)
                    if (currentTrip?.data?.entry?.vehicle != null) {
                        val nextStop =
                            currentTrip.data
                            .entry.stopTimes[
                                currentTrip.data.entry.vehicle.stopSequence
                            ]
                        if (nextStop.predictedArrivalTime != null) {
                            val currentDifference =
                                nextStop.predictedArrivalTime!! -
                                    nextStop.arrivalTime!!
                            if (currentDifference > 120
                                    && currentDifference
                                    > previousDifference + 60) {
                                … (útvonaltervezés kezdeményezése, ami majd
                                 az értesítésre kattintás után megjelenik)
                            }
                            previousDifference = currentDifference
                        }
                    }
                }
            }
        }
    }
```

16.6. Jobb útvonalterv keresése kódrészlet

```
private fun receivePlanTrip(receivedPlanTrip: PlanTrip) {
        var shortestAlternative: Itineraries =
            receivedPlanTrip.data.entry.plan?.itineraries?.get(0)!!
        for (item in receivedPlanTrip.data.entry.plan?.itineraries!!) {
            if (item.endTime < shortestAlternative.endTime) {
                shortestAlternative = item
            }
        }
        … (értesítés előkészítése)
        when {
            selectedTrip?.endTime!! >
                    (shortestAlternative.endTime + 180000) ->
            {
                … (ha az új útvonalak közül a leggyorsabb 3 perccel jobb
                 érkezés jelent, mint a jelenlegi útvonal, értesítést küld a
                 felhasználónak)
            }
            isCongested -> {
                … (ha dugó érzékelés után tervezett újra, értesítés küldése
                 a felhasználónak)
            }
            isMissedConnection -> {
                … (ha a felhasználó lekésett egy átszállást, értesítés
                 küldése a felhasználónak)
            }
        }
    }
```