Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Iváncsics Barnabás Pál

TURISZTIKAI SÉTÁT KIEGÉSZÍTŐ ALKALMAZÁS KÉSZÍTÉSE FLUTTER KERETRENDSZERREL

KONZULENS

Pásztor Dániel

BUDAPEST, 2022

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Iváncsics Barnabás Pál**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2022. 12. 09.

...…….

Iváncsics Barnabás Pál

Összefoglaló

Egy mobilalkalmazás tervezésekor mindig felmerül az a kérdés, hogy milyen platformra érdemes fejleszteni azt. Napjainkban a két legnépszerűbb az iOS és az Android. Mindkettőnek hasonló méretű a picai részesedése, így nélkülözhetetlen, hogy elkészítsük az alkalmazásunkat mindkét felületre. Ugyanaz az alkalmazás elkészítése kétszer viszont nagyon költséges lehet. Erre a problémára nyújt megoldást a cross-platform technológia, aminek a segítségével közös kódbázisból lehet egyszerre több platformra fejleszteni.

Szakdolgozatom keretein belül elkészítettem egy Flutter alkalmazást, ami Android és iOS rendszereken is futtatható. Az applikáció egy turisztikai sétához készült. Az alkalmazás, a séta állomásai közötti navigáció segítését és az állomásokon felállított installációk művészi kiegészítését szolgálja. Az alkalmazáshoz készítettem backendet is, amihez a Firebase több szolgáltatását is igénybe vettem.

Dolgozatomban először bemutatom a Flutter keretrendszert és a Firebase szolgáltatásait. Utána megmutatom az elkészült alkalmazást és leírom, hogy hogyan valósítottam meg az adatelérést, a felhasználói hitelesítést, az állapotkezelést, a térképes navigációt és a többi technológiát, amit felhasználtam az applikáció elkészítése során.

Abstract

When designing a mobile application, there is always the question of which platform it should be developed for. Nowadays, the two most popular are iOS and Android. Both have a similar market share, so it is necessary to develop the application for both platforms. However, building the same application twice can be very expensive. The cross-platform technology provides a solution to this problem. By using it, you can develop from a common code base to several platforms at the same time.

As part of my thesis, I created a Flutter application that can be run on Android and iOS systems. The application was created for a tourist walk. In the application, you can navigate between the stations of the walk. Furthermore, the mobile app implements art supplements (for example playing audios). I also created a backend, for which I used Firebase applications.

In my thesis, I first present the Flutter framework and Firebase services. Afterwards, I show the completed mobile application and then I describe how I implemented data access, user authentication, state management, map navigation, and the other technologies I used during the creation of the application.

1 Bevezetés

Napjainkban a legtöbb embernek van már okostelefon a zsebében. Ezek a készülékek felfoghatatlan tempóban fejlődtek az elmúlt 15 év alatt és nagyban hozzájárultak a társadalmunk változásához. Ezekre a változásokra sokan negatívan tekintenek, de én úgy gondolom, hogy nagyon sok pozitív dolgot hoztak az életünkbe. Számos feladatot, tevékenységet megkönnyítettek, valamint új lehetőségeket tártak fel előttünk. A készülékek terjedése és az egyre inkább felhasználóbarát telefonos operációs rendszerek megjelenésével párhuzamosan, több cég is elkezdett különböző mobilos programokat fejleszteni.

Az egyre okosabb rendszerek és szoftverek hatására a telefonok elkezdték kiszorítani az egyéb informatikai eszközöket, mint például az asztali számítógépeket és laptopokat. Sok olyan ismerősöm van, akinek nincs saját laptopja és nem is gondolkozik annak a vásárlásában. Ehhez a trendhez alkalmazkodva, a különféle szolgáltatásokat biztosító cégek is prioritásba vették (egyes területeken) a mobilalkalmazások fejlesztését a számítógépes programokkal szemben. Ma már ha egy boltnak, banknak, szállást foglaló oldalt üzemeltető cégnek, vagy akár a közösségi közlekedéssel foglalkozó vállalatnak nincs applikációja telefonra, akkor azzal sok felhasználót veszít és népszerűsége, piaci versenyképessége is romlik.

Egy mobilalkalmazás fejlesztését számos tervezési folyamat előzi meg. Az egyik ilyen legfontosabb a célközönsége meghatározása. Számos kutatási módszer létezik, amiknek a segítségével meg tudjuk határozni a célközönségünket. A legtöbb esetben a felhasználók a két legismertebb platform: iOS [1] és Android [2] valamelyikét használják. Mindkét operációs rendszernek hasonló méretű a piaci részesedése, ami azt eredményezi, hogy egy új alkalmazást érdemes mindkettőre implementálni. Funkcióját tekintve ugyanazt az applikációt kétszer lefejleszteni viszont időigényes és nagyon költséges lehet. Ennek elkerülésére ad lehetőséget a Kereszt-Platform technológia.

Ennek a lényege az, hogy a fejlesztők egyszerre tudjanak kettő, vagy akár több platformra fejleszteni. Vannak olyanok, amiknél egyszerre lehet mobil, web és számítógépes operációs rendszerekre is fejleszteni. A Flutter [3] keretrendszer is ezek közé sorolható. Használatával közös kódbázisból lehet egyszerre mindkettő: Android és iOS rendszerre programozni. Ezen kívül lehetőség lenne webre és számítógépes

rendszerekre is fejleszteni, de a szakdolgozat projekt keretein belül ezekkel nem foglalkoztam.

„Térvers – Jelfák az úton” [4] névre hallgató turisztikai sétát egy önkéntes csoport hozta létre Wekerletelepen [5]. Az elkészült terv keretein belül több, összesen 10 helyszínen lett felállítva egy-egy installációs állomás. Mindegyik állomáson egy jelfa található, aminek a tetején egy acélból készült nézőkébe belenézve a városrész egy-egy egyedi szépségű részletét tekinthetjük meg.

A sétát megvalósító csoportban merült fel az az ötlet, hogy egy olyan programot lehetne készíteni a projekthez, ami művészi kiegészítést szolgálna hanganyagok lejátszásával és segítené a stációk közötti navigálást. Az önkéntes csoport megkért rá, hogy fejlesszem le az alkalmazást, amit én szívesen el is vállaltam, mert többek közt úgy gondoltam, hogy ez egy izgalmas szakdolgozat téma lenne.

Korábbi években már több egyetemi tárgyam keretein belül is foglalkoztam telefonos alkalmazások implementálásával Android-os rendszerre. Előző félévben az Önálló laboratórium keretein belül már elkezdtem ezzel a projekttel foglalkozni. A szakdolgozatom keretein belül az alkalmazásnak a korábbi verziójának javításával, továbbfejlesztésével foglalkoztam. Ezen kívül a dolgozatban részletesen bemutatom a Flutter alapjait és az alkalmazásba történő bejelentkezéshez szükséges azonosítási folyamatot. Továbbá bemutatom, hogyan valósítottam meg az applikációt kiszolgáló felhő alapú adatbázist és beszámolok a forráskódomban felhasznált plugin-ekről is.

2 Felhasznált technológiák bemutatása

Fontos szempont volt, hogy az elkészült alkalmazás iOS és Android rendszert használó telefokon is telepíthető legyen. Ugyanakkor nem volt követelmény, hogy valamilyen platform specifikus dolgot kelljen implementálni, ami nehézkes lenne nem natív fejlesztéssel. Adott volt tehát, hogy valamelyik keresztplatformos technológiát fogom használni. Az elkészült alkalmazást Flutter keretrendszerrel valósítottam meg, valamint a szerveroldalon a Firebase [6] felhő alapú szolgáltatást vettem igénybe. Ebben a fejezetben ezeket a technológiákat mutatom be, valamint egy-két érvet is megemlítek, hogy miért ezeket választottam.

2.1 Flutter

A Flutter egy nyílt forráskódú UI szoftverfejlesztő készlet, amelyet a Google készített. Cross-platform alkalmazások fejlesztéséhez használják. Egyetlen kódbázisból lehet egyszerre Android, iOS, macOS, Windows és Linux operációs rendszerekre programokat fejleszteni. 2015-ben mutatta be a Google, majd az első stabil verzióját (1.0) 2018-ban adta ki. Akkoriban a Windows, macOS és Linux rendszerekre történő fejlesztés még béta verzióban volt. Újdonság lehet azóta, hogy mára már mind a három desktop platform teljes támogatása megérkezett. Először a Windows (2022. februárjában, 2.10.0 verzió [7]), majd pár hónappal később a Linux és macOS is megkapta a stabil kiadást (2022. május, 3.0.0 verzió [8]).

2.1.1 Dart nyelv

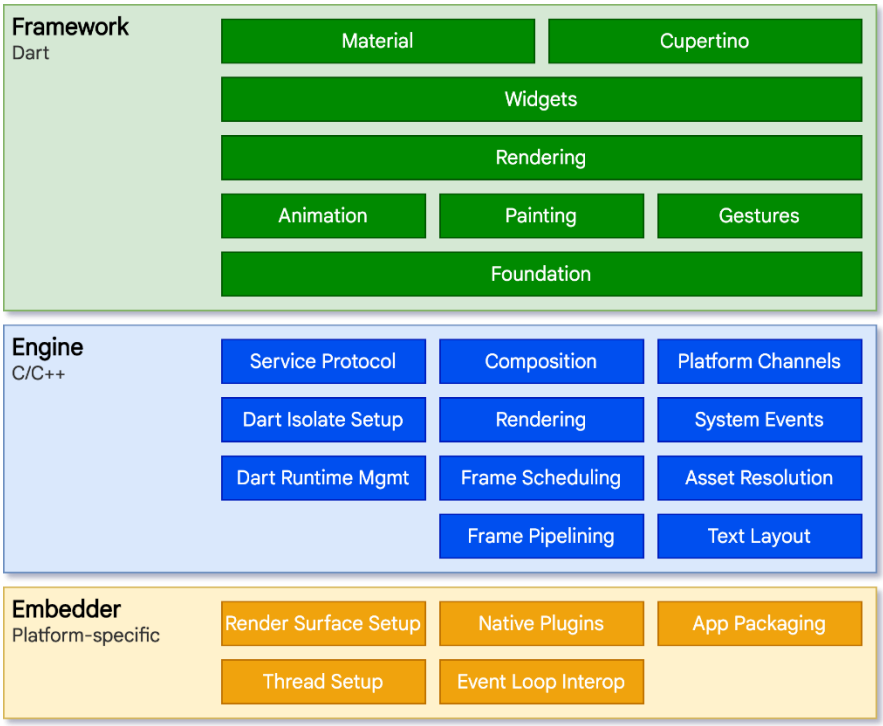
Flutter alkalmazások készítésekor Dart nyelven kell írni a kódunkat. A Dart egy kliensre optimalizált nyelv, amely lehetővé teszi a gyors alkalmazások fejlesztését a legtöbb platformon. A Dart [9] is Google kezdeményezés. 2011-ben mutatta be a Google, majd az első stabil verziót 2013-ban adta ki. Eleinte webre, a Javascript [10] leváltására szánták. A nyelv objektumorientált, osztályalapú, és van benne garbage collector (szemétgyűjtő) is. Szintaxisa a C nyelvhez hasonlítható.

2.1.1.1 A Dart nyelv előnyei

A Flutter nyelvének a kiválasztásakor több, már létező nyelvet is figyelembe vettek a fejlesztők. A Dart az összes fontos szempontnak megfelelt. Ezek közül párat most megemlítenék: Egyik nagy előnye, hogy fordítható Javascriptre, natív bájtkódra, valamint

AOT (Ahead Of Time) és JIT (Just In Time) módon is. A JIT fordítás a fejlesztés közben hasznos a nagyon gyors fordítási idő miatt. Ennek köszönhető az úgynevezett Hot Reload funkció is, ami futás közbeni gyors újra fordítást tesz lehetővé anélkül, hogy az egész alkalmazást újra kelljen fordítani. Ezzel szemben az AOT fordítás lassabb, de kiszámíthatóbb futásidőt eredményez. A nyelv előnyei közé sorolható még a gyors memóriafoglalás, a magas teljesítmény, valamint a korábban említett objektum orientált felépítés. A végére hagytam az egyik legmeghatározóbb érvet ami az, hogy Dart is a Google ökoszisztéma része. A Dartot a Flutter elött nem igazán használták, ami azt eredményezte, hogy könnyű volt a platformhoz igazítani a nyelvet. A nyelvet fejlesztő csapat sok új funkciót vezetett be, amit kifejezetten a Flutternek szántak. Egyik ilyen például az AOT mód, korábban (amíg főleg webre szánták) nem lehetett natív kódra fordítani, de ahhoz, hogy a Flutter egy hatékony keresztplatform legyen a Dart csapata beépítette ezt a funkciót is.

2.1.2 Flutter keretrendszer felépítése



1. ábra: A Flutter keretrendszer felépítése

A Fluttert architektúrálisan három külön részre lehet felosztani. A legalsó réteg az Embedder. Ez a réteg minden platformra külön meg van írva azon a nyelven, melyet a platform használ. Felelőssége összekötni a Fluttert az operációs rendszerrel. Feladatai

közé tartozik még többek között a szálkezelés (szálakat kér) és a natív API-k elérhetővé tétele.

A második réteg az úgynevezett motor, ami C/C++ nyelven van megírva. Ez maga a Flutter magja. Biztosítja a Flutter működéséhez szükséges alacsony szintű API-kat, például: grafika, szövegelrendezés, fájl és hálózati hozzáférés. A Dart futásidejű és fordítási eszközei is ide sorolhatóak. Ezen a rétegen fut maga a magas szintű alkalmazás is.

A harmadik réteg a Flutter keretrendszer. A fejlesztők általában ezen a rétegen keresztül lépnek kapcsolatba a Flutterrel. A réteg egy modern keretrendszert biztosít Dart nyelven, ami platformok, alapkönyvtárak és elrendezések készletét biztosítja. Ilyenek például az animációk, a renderelő réteg, a widget rétegek, vagy a Material és Cupertino könyvtárak.

2.1.3 Döntés a Flutter mellett

A 2. fejezet (9. oldal) elején már említettem, hogy miért egy keresztplatformos technológiát választottam az alkalmazás elkészítéséhez. A következő pár mondatban kitérnék arra, hogy miért pont a Fluttert választottam közülük.

A legtöbb keresztplatformos technológiát úgy valósították meg, hogy a már meglévő natív UI elemeket tették elérhetővé. Ez olyan technológia megoldásokat követelt, amik teljesítményvesztéshez vezettek. A Flutter ezzel szemben saját UI elemeket kínál, amik nagyban hasonlítanak az eredeti elemekre. Ezeket az elemeket pixelesen rajzolja ki a keretrendszer az összes platformon, ami azt eredményezi, hogy mindenhol ugyanúgy fog kinézni és biztosan meg fog jelenni. Ezeknek az elemeknek a használata nem jár teljesítménycsökkenéssel, valamint mindenre lehetőséget adnak, amire egy natívos alkalmazásban lehetőség van.

Ez számomra egy elég kényelmes megoldást biztosított az alkalmazás kinézetének a megvalósításakor. Emellett a fejlesztés egészét Android emulátoron végeztem macOS-t futtató eszköz hiánya miatt. Csak a végén kértem kölcsön egy olyan gépet, amin tudtam iOS-re is fordítani. A korábban említett megoldás miatt számítottam arra, hogy nem lesz kinézetbéli különbség az Androidos rendszeren futó programhoz képest, így ezzel sem kellett külön foglalkozom a fejlesztés alatt.

Egy másik érv a 2.1.1.1. bekezdésben (9. oldalon) említett fordítási technológiák. Ennek köszönhetően gyorsan és kényelmesen tudtam fordítani az elkészült kódrészleteket fejlesztés közben, valamint a végső gépi kódra fordított alkalmazás futási teljesítménye is teljesen megfelelő lett.

2.2 Firebase

Szerveroldali fejlesztéshez a Firebase-t és annak szolgáltatásait használtam. A Firebase egy úgynevezett Backend as a Service (BaaS) technológiát valósít meg. Használatával nem szükséges egy külön szervert létrehozni, üzemeltetni, mert annak a funkcióit is megvalósítja maga a Firebase. A továbbiakban bemutatok röviden három olyan Firebase szolgáltatást, amit igénybe vettem az alkalmazás elkészítésének során.

2.2.1 Authentication

A Firebase hitelesítéssel foglalkozó szolgáltatása a Firebase Authentication [11]. Ez az azonosításhoz nagyon sok különböző lehetőséget kínál. A legegyszerűbb email-jelszó páros módszertől, a telefonszámon keresztül, akár más szolgálgatók azonosításait is igénybe vehetjük (például: Google, Facebook, Microsoft, Apple, Twitter, stb) ezen keresztül. Ezeken kívül lehetőségünk van egyedi azonosító szolgáltatást is beállítani.

A legtöbb platformra van SDK-ja a szolgáltatásnak így könnyen integrálható. A hozzá tartozó webes felületen szabályozhatjuk az adott projekthez tartozó felhasználók jogait. Lehet általános szabályokat is megfogalmazni, illetve a felhasználókat is tudjuk egyesével kezelni. Ezen kívül jelszó emlékeztető e-mail küldést is be lehet állítani. A bejelentkezés után kapott visszatérési értékekben különböző információkat kapunk annak sikerességről, sikertelenségéről, vagy akár a sikertelen belépés okáról. Ha sikerült bejelentkezni, akkor lekérhetjük a bejelentkezett felhasználó azonosítóját, aminek a segítségével a felhasználó az egyedi adatbázisához hozzá tud férni, vagy bármi máshoz, amihez engedélyt adtunk neki.

2.2.2 Realtime Database

Az adatok tárolásához a Firebase kettő különböző lehetőséget is biztosít. Az egyik a Realtime Database [12], a másik pedig a Cloud Firestore [13]. Mindkettő valós időben szinkronizálja az adatokat és mindkettőnél lehetőségünk van webes felületen keresztül is szerkeszteni azokat. Az elsőben az adatokat JSON struktúra szerint rendezhetjük,

tárolhatjuk, frissíthetjük, míg a másodikban már lehetőségünk, van dokumentumokat, kollekciókat is létrehozni.

Én a Realtime Database megoldást vettem igénybe, mert az alkalmazásom adatbázis leírásához nem volt szükség bonyolultabb felépítést, vagy kapcsolatokat megvalósítanom. Elég volt egy egyszerű struktúrát megvalósítanom, ahol a felhasználói adatokat, valamint az állomások megtekintett állapotait tárolom.

Ahogy az összes Firebase szolgáltatást, ezt is nagyon egyszerűen lehet implementálni a projektünkbe a részletes dokumentációk alapján. Erről később, a 5.3.3. fejezetben (25. oldal) részletesebben be is számolok.

2.2.3 Cloud Storage

A Firebase Cloud Storage [14] tartalmak feltöltését, tárolását és letöltését teszi lehetővé. Az SDK telepítése után, a konzolos felületen megadhatjuk a bucket-ot, amiben több mappát (akár egymásba ágyazva) is létrehozhatunk a fájlok rendszerezésére.

Használatához elég lekérni az elérni kívánt fájl, vagy könyvtár referenciáját, majd a beépített metódusok segítségével fel és letölthetjük azokat, vagy akár a letöltési linket is kérhetünk, így akár másik függvényekkel akár streamelhetjük, vagy cache-be is betölthetjük azokat.

2.2.4 Döntés a Firebase mellett

Ebben a fejezetben rövidem összefoglaltam, hogy miért a Firebase-t használtam a backend megvalósításához.

Az első szempont itt is a Google ökoszisztéma volt. Számos technológiát használtam fel a projektemben, amiket a Google valósított meg. Ezek együttes alkalmazása nagyban megkönnyítette a fejlesztési folyamatokat. Az alkalmazásban található térképhez a Google Cloud [15] szolgáltatásait vettem igénybe. Erről a későbbiekben részletesen beszámolok a 5.15.5 fejezetben (27. oldalon). A Google a Cloud és a Firebase szolgáltatáshoz tartozó projekteket közösen tudja kezelni, így nem szükséges külön-külön létrehozni azokat. Ennek eredményeként a projekten már elvégeztem korábban (a térkép megvalósításakor) az alapbeállításokat és lépéseket ahhoz, hogy használni tudjam a Firebase szolgáltatásokat, így ezzel nem kellett már foglalkoznom.

A következő érvem az az egyszerű és átlátható használat. A Firebase-t egy Google fiók segítségével egyszerűen igénybe lehet venni. A konzolos felületével pedig felhasználóbarát módon tudjuk beállítani a szükséges beállításokat. Ezeken kívül lehetőségünk van statisztikákat és elemzéseket is megnézni a projektünkben felhasznált szolgáltatásokról (például: napi felbontásban nézhetjük meg a felhasznált adatmennyiséget). Az élesben futó alkalmazáshoz pedig dinamikusan árazza be a szolgáltatás költségeit, így mindig csak a forgalommal arányosan kell fizetnünk érte.

További előnye, hogy Android, iOS és webes rendszerekhez is megfelelő támogatást nyújt. Végül pedig számomra nagyon nagy segítség volt, hogy elérhető külön a Flutterhez is SDK és részletes dokumentáció is.

3 A feladatkiírás pontosítása

A 2. fejezetben (9. oldal) bemutatott technológiák ismertetése mellett feladatom volt a *Bevezetés* fejezetben már röviden ismertetett alkalmazás elkészítése. Az ezzel szembeni funkcionális követelményeket a konzulensemmel egyeztetve én határoztam meg a félév elején. Ezeket a következő felsorolásban ismertetem:

- Rövid leírás, fogadó képernyő
  - Az alkalmazás elindítása után egy főképernyő jelenjen meg, ahol a sétáról és a projektről olvashatunk pár rövid információt
- Az összes állomás megtekintése
  - Egy külön oldalon lehessen megnézni az összes állomást. Tetszőleges állomásra kattintva egy másik képernyő jelenjen meg. A megjelent oldalon a választott állomásról plusz információkat lehessen olvasni.
- Navigálás az állomások között
  - Egy térkép segítségével meg lehessen nézni, hogy az egyes állomások hol találhatók és a felhasználó (azaz a készüléke) hol található ahhoz képest.
- Menü
  - Legyen egy menü, ahonnan könnyedén elérhetjük az alkalmazás különböző funkcióit.
- Felhasználókezelés
  - Az alkalmazásba be lehessen jelentkezni, valamint legyen legalább egy olyan funkció, amit csak bejelentkezve tud a kliens igénybe venni.
- Backend
  - Az alkalmazáshoz készüljön a Firebase szolgáltatások igénybevételével backend. A szerver képes legyen adatok, tartalmak tárolására és azok lekérésének a kezelésére.
- Információk, érdekességek bemutatása a projekttel kapcsolatban
- Kapcsolatok megjelenítése egy külön oldalon
- Platformok
  - Az alkalmazásnak fordulnia kell Android és iOS platformra is.

4 Elkészült alkalmazás áttekintése

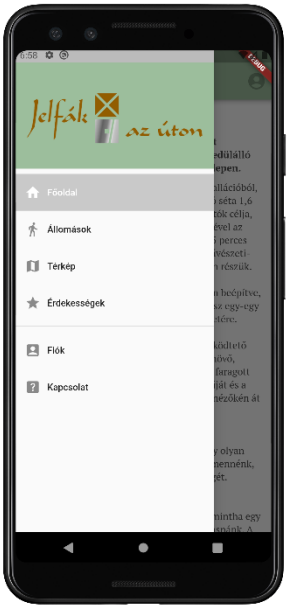
Ebben a fejezetben rövidem bemutatom az elkészült alkalmazást. Először a betöltő képernyőt és a kezdő képernyőt mutatom be. Utána a menüről és az onnan elérhető többi oldalról, valamint az ott megvalósított funkciókról írok. A következő fejezetekben több képi illusztráció is lesz az alkalmazásról, emellett további képernyőképeket is készítettem a különböző állapotokról, amik az utolsó, *Függelék* fejezetben (39. oldalon) tekinthetőek meg.

4.1 Betöltő képernyő

Az alkalmazás indulása után egyből megjelenik egy úgynevezett splash screen (betöltő képernyő, 4. ábra). Ennek az a funkciója, hogy amíg az alkalmazás teljesen fel nem áll, addig ne egy üres képernyőt kelljen nézni, hanem már ekkor legyen valamilyen interakció a felhasználó felé. A képernyőn a Térvers projekt logója jelenik meg egy fehér háttérrel.

4. ábra: Splash screen

3. ábra: Kezdő képernyő



2. ábra: Oldalsó menü

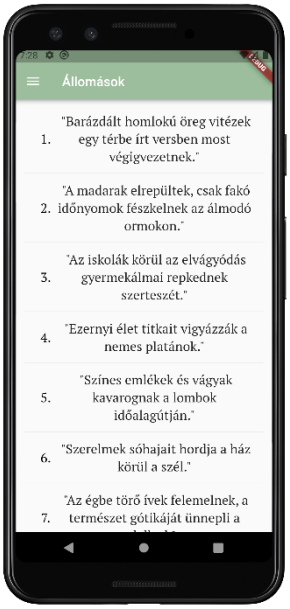
4.2 Kezdő képernyő, menü

Miután a háttérben megtörtént az inicializálás megjelenik a Kezdő képernyő (3. ábra). Ezen az oldalon a Térvers projektről, valamint magáról a sétáról olvashatunk pár információt. Ahogy az ábrán is látható az oldal tetején található appBar-on megjelenik bal oldalt egy hamburger menü, jobb oldalon pedig egy felhasználói profilt ábrázoló kis

ikon. Ez utóbbi szürke színű, ha az alkalmazásban nincs bejelentkezve a kliens, ellenkező esetben pedig fehér színű (lásd: Függelék).

A bal oldali ikonra nyomva (kattintva) megjelenik egy oldalsó menü (2. ábra), ahonnan lehetőségünk van a különböző oldalak közötti navigálásra (azok megjelenítésére). Látható továbbra az is, hogy az éppen aktív oldal ki van emelve, meg van különböztetve a többi menüponthoz képest. Az egyes menüpontokra kattintva az alábbi képernyőket tudjuk kiválasztani

- Főoldal: Kezdő képernyő (3. ábra)
- Állomások: Állomások képernyő (6. ábra)
- Térkép: Térkép képernyő (8. ábra)
- Érdekességek: Érdekességek képernyő (10. ábra)
- Fiók: Ha be van jelentkezve a felhasználó, akkor a Felhasználói fiók képernyő (12. ábra), egyébként a Bejelentkezés képernyő (11. ábra)
- Kapcsolat: Kapcsolat képernyő (13. ábra)



6. ábra: Állomások képernyő

7. ábra: Állomás részletei képernyő 1. (messze)

5. ábra: Állomások részletei képernyő 2. (közel)

4.3 Állomások képernyő

Az állomások oldalon (6. ábra) egy lista látható. Összesen annyi elem található a listában amennyi állomás van. Egy listaelemen az adott állomás sorszáma, illetve az ahhoz tartozó versszak, idézet látható. Abban az esetben, ha a felhasználó be van jelentkezve, akkor a listában halványabb, szürke színnel vannak formázva a már

megtekintett állomások sorszámai, idézetei. Tetszőleges elemre (állomásra) kattintva megjelenik az adott állomás részleteit megjelenítő képernyő (7. ábra, 5. ábra).

4.4 Állomások részletei képernyő

Az állomás részleteit megjelenítő képernyőnek két elrendezése van. Az első amikor a telefon nincs az állomás közelében (7. ábra). Ekkor alul egy térkép jelenik meg, ahol látható az adott állomás helyzetét jelölő marker és az eszköz aktuális lokációja is. A második amikor a telefon 5 méteren belül van az adott állomáshoz képest (5. ábra). Ekkor alul a térkép helyett egy hanglejátszó vezérlő jelenik meg. Az indítás gombra kattintva elindul a háttérben egy hanganyag, a vissza gombra kattintva pedig az elejéről kezdi el annak a lejátszását. A képernyő nézet a két elrendezés között dinamikusan változik a telefon helyzetének változása esetén. Ez azt jelenti, hogy az állomáshoz közeledve a térkép eltűnik és a hangvezérlő jelenik meg, távolodva pedig fordítva.

4.5 Térkép képernyő

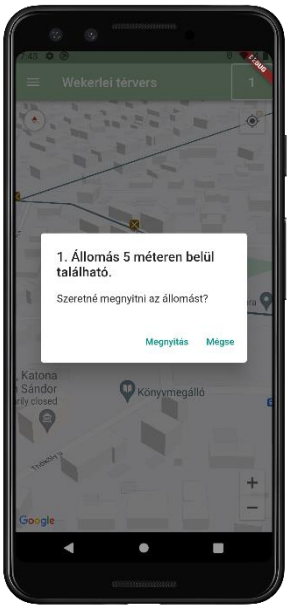
A térkép oldalon az egész képernyőn (az appBar alatt) egy térkép jelenik meg (8. ábra). A térképen 10 marker található, ami a 10 állomás helyszínét jelöli. A markerek az útvonalat szimbolizáló egyenesekkel össze vannak kötve. A markerre kattintva megjelenik felette az állomás sorszáma. Emellett a kattintás hatására lefut egy animáció, aminek a végén a térkép egy olyan nézetbe kerül, ahol középen az adott állomás van, egyenes (a térkép tetején) pedig a következő állomás. Így a felhasználó könnyebben meg tudja állapítani, hogy merre van a következő állomás. A markerre kattintva az adott sorszámú állomás részletező nézete töltődik be (5. ábra, 7. ábra). A térkép nézet közben, ha 5 méteren belülre érünk egy állomáshoz, akkor felugrik egy dialógus ablak (9. ábra). Az ablakon két gomb található. A megnyitás gombra kattintva megjelenik az adott állomáshoz tartozó részletező nézet (5. ábra). Ha a felhasználó nem nyitja meg azonnal, akkor később is van rá lehetősége. Ez esetben az appBar jobb oldalán található, az állomás sorszámát kirajzoló gombra kell kattintani. Ez a gomb csak abban az esetben látható, ha a készülék 5 méteren belül van az adott állomáshoz.

4.6 Érdekességek képernyő

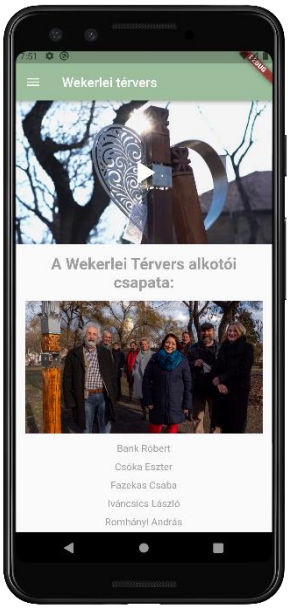
Az érdekességek képernyő tetején egy videólejátszó található (10. ábra). Amikor a képernyő megjelenik, akkor a videólejátszóban betöltődik, majd elindul egy előre

megadott YouTube videó. A lejátszót ki lehet tenni teljes képernyőre is. A videólejátszó alatt egy csoportkép látható a projekt alkotóiról. Alatta pedig egy lista a nevükről.

8. ábra: Térkép képernyő 1.



9. ábra: Térkép képernyő 2. (dialógus ablak)



10. ábra: Érdekességek képernyő

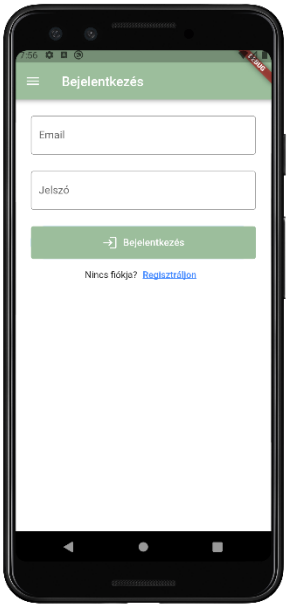
4.7 Bejelentkezés, Regisztráció képernyő

Bejelentkezés képernyőn van lehetősége a kliensnek bejelentkezni a fiókjába. Ezt egy egyszerű e-mail cím – jelszó párossal teheti meg (11. ábra). A bemeneti mezőkbe való gépelés közben az alkalmazás jelzi, ha még nem jó valamelyiknek a szintaktikája. A Bejelentkezés gombra kattintva a Felhasználói fiók oldalra jut a kliens abban az esetben, ha létező fiókhoz tartozó helyes adatokat adott meg. Ellenkező esetben a képernyő alján megjelenik egy üzenet, amiben a sikertelen bejelentkezés indoka van leírva. Ezek a következők lehetnek:

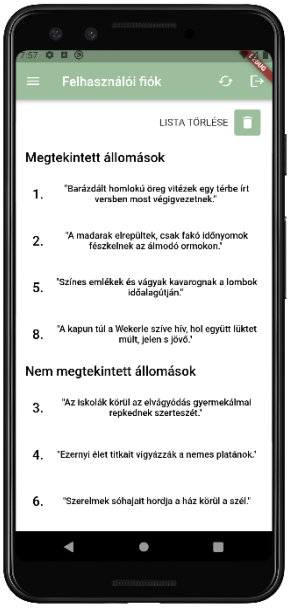
- Hibás email!: Nincs ilyen e-mail címmel felhasználó regisztrálva.
- Hibás jelszó: A megadott e-mail címmel regisztrált felhasználói fiókhoz hibás jelszó lett megadva.
- Minden egyéb más esetben fiókokat kezelő szervertől kapott üzenetek jelennek meg angolul, de érthetően, tömören fogalmazva.

Ha felhasználónak egy vagy több sikertelen bejelentkezése volt már, akkor a Bejelentkezés gomb alatt megjelenik egy gomb „Elfelejtette jelszavát?” szöveggel. Erre kattintva az Email szövegdobozba írt e-mail címre küld a felhasználókat kezelő szerver egy levelet, benne egy jelszó helyreállító linkkel.

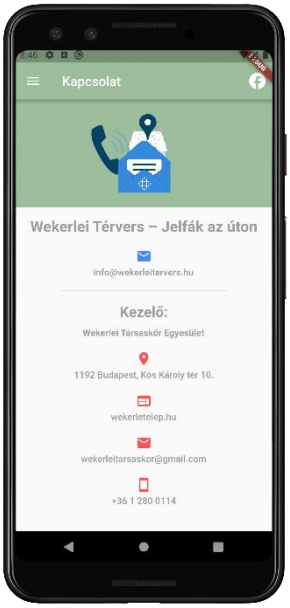
Ha a felhasználó nem rendelkezik még fiókkal, akkor a Bejelentkezés gomb alatti „Regisztráljon” feliratra kattintva a Regisztráció oldalra jut. Itt ugyanúgy, mint a Bejelentkezés oldalon egy egyszerű e-mail cím – jelszó párossal regisztrálhat. A bemeneti mezők itt is ellenőrzik a helyes szintaktikákat.



11. ábra: Bejelentkezés képernyő



12. ábra: Felhasználói fiók képernyő



13. ábra: Kapcsolat képernyő

4.8 Felhasználói fiók képernyő

Ezen az oldalon a bejelentkezett kliens megtekintheti két külön listába rendezve a már megtekintett, és a még meg nem tekintett állomásokat (12. ábra). A listák felett van egy kuka ikont ábrázoló gomb. Ennek a megnyomásával a felhasználó törli a megtekintett állomásokat. Ha az összes állomást már megtekintette a kliens, akkor a listák helyén egy gratulációs szöveg és animáció jelenik meg (lásd: Függelék). Ezen a képernyőn az appBar jobb oldalán található két gomb. Az elsővel a listákat tudjuk frissíteni, a második gomb megnyomásával pedig a felhasználó ki tud jelentkezni a fiókjából.

4.9 Kapcsolat képernyő

Ezen az oldalon lehet megtekinteni a Térvers, valamint a projektet kezelő társaskör elérhetőségeit (13. ábra). A kapcsolati adatokra, vagy a hozzájuk tartozó ikonokra kattintva az alkalmazás megnyitja az alapértelmezett levelezőt, böngészőt, vagy tárcsázót.

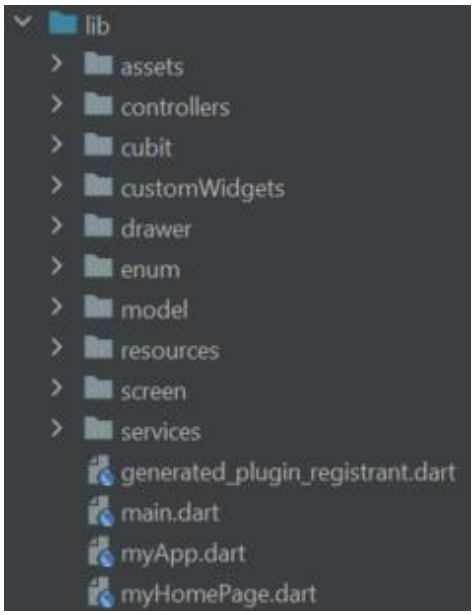
5 Alkalmazás megvalósításának bemutatása

Ebben a fejezetben bemutatom, hogy hogyan valósítottam meg magát az alkalmazást. Először a projekt struktúráját, az oldalak közötti navigálást, majd az adatelérés és a hitelesítés megvalósításának lépéseit írom le. Továbbá vázolom a térkép és a térképes navigációs funkciók létrehozását. A projektben található modellről, erőforrásokról és állapotkezelésről is írok. Végül pedig az oldalak, képernyők kinézetének a kialakítását, a felhasznált külső könyvtárakat és a projekt verziókezelését mutatom be.

5.1 Projekt könyvtár struktúra

Az alkalmazás elkészítése során az alábbi könyvtár hierarchiát alakítottam ki a projekthez (14. ábra):

- *assets*: Erőforrás fájlok.
- *controllers*: Térkép, YouTube lejátszó és hanganyag lejátszó működéséhez szükséges osztályok.
- *cubit*: Cubit és hozzájuk tartozó State osztályok.
- *customWidgets*: Widget-ek, amelyeket én írtam és a kódban több helyen is példányosítom.
- *drawer*: Menü megjelenítéséhez szükséges osztályok.
- *enum*: Definiált enumok.
- *model*: Adatmodellek.
- *resources*: Állomások, útvonalak megjelenítéséhez szükséges adatok, hosszab sztringek.
- *screen*: Képernyőket leíró Widgetek.
- *services*: Adatbázist, hitelesítést és egyéb szerverrel való kommunikációt megvalósító osztályok.
- *main.dart*: Az alkalmazás belépési pontja.
- *myApp.dart*: Az alkalmazás kezdő widget-ét írja le.
- *myHomePage.dart*: Az első oldalt megjelenítő widget-et valósítja meg.



14. ábra: Projekt könyvtár hierarchia

5.2 Oldalak közötti navigáció

Az oldalak közötti navigálást, lapozást úgy valósítottam meg, hogy a menüből elérhető oldalakról visszafelé mindig a Főképernyőre jut a felhasználó. Erre azért volt szükség, mert ha a kliens egymás után nyitja meg sokszor mondjuk a Térkép oldalt és az Állomások oldalt felváltva, akkor miután vissza szeretne menni, egy darabig csak a két oldal között fog tudni ugrálni. Több létező alkalmazásban is megnéztem, hogy hogyan működik ez a navigáció és a legtöbb esteben az általam is megvalósított koncepciót használták. Ez alól egy kivétel van az állomás részletei oldal. Innen a visszafelé navigáció esetén az előző oldalra lehet visszajutni, ami vagy az Állomások (4.3 fejezet), vagy a Térkép (4.5 fejezet) oldal.

Az oldalak közötti navigáláshoz a keretrendszer által biztosított Navigator osztályt és egy Cubit állapotkezelőt használtam. A cubit működését a 5.8 fejezetben (29. oldalon) fejtettem ki bővebben.

5.2.1 Pages enum

Az aktuális oldal állapot meghatározásához csináltam egy enum-ot, amiben a következő értékeket adtam meg: *main, locations, map, curiosities, detailScreen, detailStation, videoFullScreen, login, accountMain*. Így amikor az aktuális oldal állapotát állítom be, vagy kérdezem le, biztosan csak olyan értéket fog visszaadni, ami létező oldalt reprezentál.

5.2.2 Navigator

A Navigator osztály *push* metódusával vettem fel új context-et a navigációs stack panelre, a *pop* metódussal meddig levettem arról legfelső elemet. Ezen kívül még használtam a *popUntil* metódust is, amiben megadtam, hogy melyik oldalig menjen vissza a stack-ben.

5.2.3 WillPopScope Widget

Sok widget *build* függvénye *WillPopScope* widget-el tér vissza. A widget *onWillPop* paraméterében megadtam, hogy mi történjen akkor, amikor a felhasználó vissza gombbal az előző oldalra szeretne menni. Erre azért volt szükség, hogy ne csak az eggyel korábban lévő képernyőre lehessen visszalépni. Ilyen eset lehet például a teljes képernyős videónézetből való kilépés.

5.2.4 Bejelentkezés és Felhasználói fiók oldalak váltása

Ahogyan már a 4.2 fejezetben is említettem a menüből a Fiók lehetőséget választva, vagy a Kezdő képernyőn a jobb felső ikonra kattintva a Bejelentkezés oldalra jut a felhasználó, ha még nincs bejelentkezve. Ellenkező esetben pedig a Felhasználói fiók oldalra. Ebben a fejezetben rövidem összefoglalom, hogy ezt hogyan valósítottam meg.

Függetlenül, hogy melyik helyről nyitotta meg a kliens a fent említett oldalakat, ugyanaz az esemény következik be annak hatására. Egy olyan StatelessWidget kerül a meghívásra, amiben egy *SreamBuilder* van. Ennek használatával az alkalmazás különböző állapotaiban az egyik vagy a másik oldalt fogja betölteni. Ez a lenti kódrészletben is látható. A FirebaseAuth osztály segítségével lekérem, hogy van-e bejelentkezett felhasználó. Amíg a csatlakozás történik a Firebase-hez addig egy betöltést jelző kör jelenik meg az oldalon. Ha valamilyen probléma akadt a lekérdezés során, akkor egyik oldalt sem tölti be, hanem egy hibaüzenetet jelenik meg az oldalak helyén. Ha be van jelentkezve a felhasználó, akkor a *VisitedStationScreen* widget-et jeleníti meg, ami Felhasználói Fiók oldalt reprezentálja. Ha pedig nincs bejelentkezve, akkor a *SignInUpScreen* widget-et jeleníti meg, ami pedig a Bejelentkezés oldalt valósítja meg.

```
Widget build(BuildContext context) => Scaffold(
    body: StreamBuilder<User?>(
      stream: FirebaseAuth.instance.authStateChanges(),
      builder: (context, snapshot) {
        if(snapshot.connectionState == ConnectionState.waiting){
          return const Center(child: CircularProgressIndicator());
        }
        else if(snapshot.hasError){
          return const Center(child: Text('Error: Something went wrong!'));
        }
        else if(snapshot.hasData){
          return const VisitedStationScreen();
        }
        else{
          return const SignInUpScreen();
        }
      }),
  );
```

5.3 Adatelérés

Az applikációban három különböző adatelérést valósítottam meg. Az egyiket lokálisan, a másik kettőt pedig a Firebase szolgálatatásainak a felhasználásával. A Firebase szolgáltatások regisztrációját, beüzemelését a Firebase konzolos felületén [16] végeztem el. Ebben a fejezetben bemutatom a fent említett adatelérések megvalósítását.

5.3.1 Lokális adatelérés

Az alkalmazásban használt állomások, útvonalak megjelenítéséhez szükséges adatokat lokálisan constans értékekben tároltam el. Azért választottam ezt a megoldást, mert mindegyik állomás helyhez kötött, nem változik, így nem szükséges azok értékeit lekérdezni máshonnan. A widget-ekben található hosszabb szövegeket is külön tárolom constans string tömbökben.

Az összes ilyen adat a /lib/resources/values mappában található.

5.3.2 Firebase Storage megvalósítása

Az alkalmazásban található hanganyagok tárolását, letöltését a Firebase Storage szolgáltatás igénybevételével valósítottam meg. Azért volt szükség arra, hogy a hanganyagok felhő alapú tárolóból legyenek elérhetőek, mert a legtöbb állomáshoz még nem készült el a hanganyag. Ettől függetlenül már lehet használni az alkalmazás többi funkcióját, mint például a navigációt, illetve az elérhető hanganyagokat is le lehet játszani. Így, hogy a „felhőben” vannak tárolva a fájlok, bármikor fel tudom tölteni utólag oda a később elkészült hanganyagokat, vagy akár ki is tudom cserélni azokat. A Firebase Storage technológiát részletesebben 2.2.3 fejezetben (13. oldalon) mutattam be.

A Firebase szolgáltatások igénybevételének első lépése az, hogy beregisztráljuk a Firebase online felületén az alkalmazásunkat a Flutter projekt azonosítójával. Ezt nekem nem kellett megtennem, mert korábban, a Google Térkép implementálásánál már beregisztráltam az alkalmazást a Google Cloud oldalán, aminek hatására a Firebase oldalán is megjelent a projektem. Ennek köszönhetően, már csak le kellett töltenem a Core és Storage SDK-t, majd utána beállítottam a Rules oldalon, hogy minden felhasználó (akár kijelentkezett állapotban is) elérje az itt tárolt fájlokat. A lenti kódrészletben bemutatom egy mp3 kiterjesztésű fájlhoz tartozó letöltési URL lekérdezését:

```
Future<String?> downloadUrlSound(String soundID) async{
String? downloadUrl;
try {
     downloadUrl = await storage
     .ref('sounds/sound$soundID.mp3').getDownloadURL();
    } on FirebaseException catch (e) {
       if(e.code == "object-not-found"){
       downloadUrl = "";
       }
     }
     return downloadUrl;
}
```

A fenti kódrészletben található függvény bemeneti paraméterben megadtam, hogy melyik azonosítójú számhoz kell az elérhetőségi út (referencia). A függvény, ha sikeres volt a lekérdezés, akkor visszaadja a fájlhoz a referenciát, ha nem, akkor egy üres string-el tér vissza. A hang lejátszás többi részét pedig már az azért felelős osztályokban írtam meg.

5.3.3 Firebase Realtime Databese megvalósítása

Az appban bejelentkezés után a kliens a Felhasználói oldalon meg tudja tekinteni azt, hogy melyik állomásoknál járt már. Ezeket az adatokat nem lokálisan, hanem egy szerveren tárolom, hogy mindig csak az aktuális felhasználó adatai jelenjenek meg, illetve egy esetleges újratelepítés esetén ne vesszenek el ezek az adatok. További előnye, hogy ezeket az adatokat az üzemeltető is át tudja állítani távolról, ha szüksége lenne rá.

Ezt az adattárolást, adatelérést a Firebase Realtime Database alkalmazás felhasználásával valósítottam meg. A Firebase konzolos felületén az alkalmazás projekt hozzáadását és alap beállításait már a Firabase Storage megvalósításánál elvégeztem (5.3.2 fejezet, 24. oldal), így ezzel itt nem kellett foglalkoznom.

A fenti lépések után először letöltöttem a Realtime Database SDK-t és a konzolos felületen hozzáadtam a Realtime Database alkalmazást a Térvers projekthez. Ezután a *Rules* oldalon egy egyszerű szabály megfogalmazásával megadtam, hogy a felhasználók csak a saját adataikhoz férjenek hozzá:

```
{
    "rules": {
        ".read": false,
        ".write": false,
        "users":
        {
            "$userId":
            {
                ".read": "auth.uid === $userId",
                ".write": "auth.uid === $userId"
            }
        }
    }
}
```

Végül a *services* mappában található osztályokban implementáltam a *getFirebaseDatabaseReference* függvényt, amiben a szervertől elkérem a referenciát a bejelentkezett felhasználó saját adatainak az útvonalához. Ezen kívül írtam egy *saveData, readData* és *updateDatabase* metódust. Az elsőben létrehozom és elmentem az először bejelentkezett felhasználók adatait és a hozzájuk tartozó alapértelmezett értékeket az

adatbázisban. A másodikban megvalósítom az adatok lekérését a szerverről, a harmadikban pedig az adatbázisban található értékek frissítését.

A következő kód bemutatja azt, hogy hogyan frissítem az állomások látogatottsági állapotát. A függvény első paraméterben megkapja az adatbázis elérhetőségi útját (referenciát), a második paraméterben egy rendezett listát a módosítandó állomások azonosítóiról, a harmadikban pedig egy rendezett listát az új értékekről. A metódus a második és harmadik paraméterből Map típusokat hoz létre és átadja azokat a Firebase-nek az *update* függvényen keresztül:

```
Future updateDatabase(DatabaseReference reference, List<String> iDList, List
valueList) async{
    if (iDList.length != valueList.length) {
      return;
    }

    await reference.update(Map.fromIterables(iDList, valueList));
  }
```

5.4 Hitelesítés megvalósítása

Az alkalmazásban a bejelentkezés és regisztrációnál lévő hitelesítést a Firebase Authentication használatával oldottam meg. A technológiát részletesebben a 2.2.1. fejezetben (12. oldalon) mutattam be. Ebben a fejezetben röviden leírom, hogy az elkészült alkalmazásban hogyan használtam fel ezt a technológiát.

Ahogyan a Realtime Database beüzemelésénél (5.3.3 fejezet) sem, itt sem kellett a Firebase oldalán megcsinálnom a projekt hozzáadását és a többi beállítást, mivel az már a Storage implementálásánál megtörtént.

A szolgáltatás beüzemelésekor először is a Firebase oldalán beállítottam, hogy a projekten elérhető legyen ez a szolgáltatás. Utána a *Sign-in method* tabon beállítottam az Email/Jelszó szolgáltatást. Utána a *Templates* oldalon megadtam, hogy mi szerepeljen a jelszó helyreállító e-mailben.

Ezek után, telepítettem az SDK-t és megvalósítottam a kódban a szükséges metódusokat. A *services* mappában található osztályokban implementáltam a *signUp* függvényt, ami a regisztrációt valósítja meg és a *signIn* függvényt, ami pedig a belépést. Ezen kívül létrehoztam egy *forgotPw* metódust, ami a jelszó helyreállító e-mail küldését valósítja meg. A kódban a *FirebaseAuth* osztály instance tulajdonságán hívott különböző függvények segítségével tudtam lekérdezni azt, hogy az alkalmazáson belül van-e bejelentkezett felhasználó és ha van, akkor ahhoz lekérni a felhasználó referenciáját.

5.5 Térkép megvalósítása

Ebben a fejezetben bemutatom az alkalmazás fő funkcióját, a térképes navigációt. A térkép megvalósításához, a rajta lévő útvonal és markerek megjelenítéséhez a Google által fejlesztett Google Map API-t [17] (5.10.1 fejezet, 31. oldal) használtam. A helymeghatározáshoz, távolságok kiszámításához és a helymeghatározás engedélyezéséhez külső könyvtárakat (*geolocator, permission_handler)* is felhasználtam. Ezeket a 5.10.1 fejezetben (31. oldalon) ismertetem. Ezeken kívül létrehoztam különböző osztályokat és segédosztályokat is.

5.5.1 Google Map API

Ebben a fejezetben rövidem leírom, hogy hogyan implementáltam a Google térképeket az alkalmazásban.

Először is a Google Cloud webes, konzol felületén egy új projektet kellett létrehoznom, amiben meg kellett adnom a Flutter projekt számát és azonosítóját. A Google szolgáltatások ezen adatok alapján ellenőrzik és kezelik az API-khoz való hozzáférést. Utána az *APIs and services* menüpont alatt ki tudtam választani, hogy milyen API-kat szeretnék használni az alkalmazásomban. A Credentials menüpont alatt létrehoztam az API kulcsot, amit kódban a megfelelő helyre beillesztettem. Ezzel a szerver oldali rész már készen is állt. Ezután telepítettem a Google Maps Flutter plugint, amiről 5.10.1 fejezetben (31. oldal) bővebben írok. A telepítés után már használhatóak is voltak a plugin által kínált widget-ek és metódusok.

A Google térképet a *GoogleMap* widget-el hoztam létre. A widget létrehozásakor be lehet állítani a térkét típusát, a kiinduló kamera pozícióját. Meg lehet adni a kirajzolandó markereket és alakzatokat. Valamint be lehet állítani azt is, hogy milyen vezérlők (saját helyzet, nagyítás/kicsinyítés stb.) legyenek elérhetőek a térképen.

5.5.2 MapCalculator segédosztály

A projektben létrehoztam egy *MapCalculator* segédosztályt, ami a két különböző térkép megjelenítéséhez és kirajzolásához szükséges funkciókat látja el. A következőkben röviden ismertetem az osztályban található függvényeket.

Az *updateLocations* nevű metódus lekéri az eszköz aktuális pozícióját a *Geolocator* osztályon keresztül. Ha valamelyik állomás közel van az eszközhöz, akkor

meghívja a paraméterében megkapott *nearStation* metódust. Ha egyik sincs az eszközhöz közel, akkor pedig az *allStationFar* metódust.

Az *_ifNearState* függvény megnézi, hogy a paraméterben kapott állomás és a saját lokáció között mekkora a távolság. Ha az kisebb, mint ami a *maxDistanace* attribútumban meg van határozva, akkor igazzal tér vissza, ha nagyobb akkor hamissal.

A *cancelLocationStream* leállítja az aktuális pozíciót lekérő stream-et.

Az *initMyLocation* ellenőrzi, hogy a helymeghatározás engedélyezve van-e. Ha igen, akkor meghívja a *setMyLocation* metódust, ha nem akkor megjelenít egy dialógus ablakot, amiben megkéri a felhasználót, hogy engedélyezze azt. A függvény *permission_handler* pár beépített metódusait is megvalósítja arra, hogy az alkalmazás helymeghatározás használati engedélyt kérjen az operációs rendszertől.

A *getMarkers* függvény a paraméterben kapott *Station* lista alapján létrehozza a Markereket, majd visszaadja azokat egy listában.

A *_setMyLocationInitCameraPos* beállítja a térkép alap nézetét, a saját lokáció és a paraméterben kapott állomás alapján. Végeredményben az állomásnak és a saját helyzetnek is látszódnia kell a térképen.

A *_midpoint* visszaadja a paraméterben kapott két pozíció által meghatározott középső pontot.

Az utolsó metódus az osztályban a *refreshScreen*, ami frissíti az alkalmazás képernyőjét. Erre azért van szükség, mert a GoogleMap nem tudja frissíteni önmagában a myLocationEnable paraméterét, amikor elérhetővé válik az eszköz lokációja.

5.6 Modell leírása

Az alkalmazásban egy adatmodellt hoztam létre, ami egy állomást reprezentál. Az adatmodellben az *id* reprezentálja az állomás azonosítóját, a *location* a pozícióját, *name* a nevét, *description* a hozzá tartozó leírást, *bearing* a térkép kameranézet értékeit, a *near* az állomás és az eszköz közötti közeli, illetve messzi távolságot, a *visited* pedig azt, hogy az adott állomást megtekintette-e már a felhasználó.

5.7 Erőforrások

Az erőforrás fájlok olyan fájlok, amiket nem szeretnénk, vagy nem tudunk .dart fájlokban tárolni. A flutter keretrendszer az ilyen fájlokhoz biztosít egy erőforrás bundle-

t. A bundle becsomagolja a megadott elérési útvonalon található fájlokat. Ezt futás időben el tudjuk kérni a keretrendszertől és utána ki tudjuk belőle olvasni az adott fájlokat.

Ezeket az erőforrás fájlokat a projektkönyvtárban található pubspec.yaml fájlban flutter szekcióban kell megadni egy assets szekciót, majd ebben kell felsorolni az erőforrás fájlok elérési útvonalát.

Az alkalmazásban az erőforrás fájlokat a /lib/assets könyvtárba helyeztem el. A következő listában felsorom a felhasznált erőforrásokat:

- Összes állomást megtekintett gratulációs animációt megvalósító gif
  - lib/assets/images/congrats.gif
- Alkotók csoportkép:
  - elérési útvonal: lib/assets/images/creaters.jpg
- Elérhetőségek oldal tetején található illusztráció
  - mail_illustration.png
- Térkép marker ikon:
  - elérési útvonal: lib/assets/images/marker_icon.png
- Menü fejléc kép
  - elérési útvonal: lib/assets/images/nav_header.png
- Betöltő képernyő kép
  - elérési útvonal: lib/assets/images/splash_image.png
- Indítógomb ikon
  - elérési útvonal: lib/assets/launcher_icon.png

5.8 Állapotkezelés

A flutter egyik sajátossága, hogy minden létrehozott widget immutable, azaz változhatatlan. Ezért is van az, hogy a fordító udvariasan jelzi nekünk, ha egy konstruktor nem const. Ez nem egy szigorú előírás, de a keretrendszert fejlesztők ezt tanácsolják. Ha minden widget változhatatlan, akkor felmerül az a kérdés, hogy az állapotkezelést hogyan lehet megoldani.

Erre a kérdésre nagyon sok technológia megoldás létezik. Én ezekből két különböző módszert használtam. Az első a StatefulWidget. A második a keretrendszer által biztosított Cubit (Block) állapotkezelési technológia.

5.8.1 StatefulWidget

Azt a widget-et, amelyikben állapotot definiáltam, leszármaztattam a StatefulWidget osztályból. A widget-ben az ős *createState* metódusát felüldefiniálva létrehoztam egy *State* objektumot. A State objektum leszármazik ebből a widget-ből,

majd megvalósítja az ős osztály build metódusát. Ezzel értem el azt, hogy a fent említett widget immutable (változhatatlan) marad, miközben különböző állapotokat tudtam meghatározni a létrehozott State objektumban.

5.8.2 Cubit

Ez a módszer leginkább az MVVM (Model-View-ViewModel) mintára hasonlít. Hasonlóan működik, mint a Bloc osztály, de annak egy egyszerűsített változata.

Működéséhez egy *State* és egy *Cubit* osztályt implementáltam. A *State* osztályt az *Equatable* ősből származtattam le, aminak a *get* listáját felüldefiniáltam. Ebben a listában megadtam azt, hogy az adott állapotnak milyen paramétereit szeretném elérni kívülről. A *Cubit* osztály ebből az osztályból származik le. A *Cubit* osztályban különböző metódusokat írtam, amik a *State* osztály állapotát változtatják meg.

Használata hasonlóan működik, mint egy Provider-é. A különböző állapotokat a context-ből ki lehet olvasni, vagy akár az állapotváltozásokra is fel lehet iratkozni. A következő listában felsorolom a kódban létrehozott Cubit-okat és röviden leírom, hogy milyen állapotváltozásra használtam fel azokat:

- Navigation Cubit
    - Az oldalok közötti navigálásban segít.
- StationScreenCubit
    - Különböző állapotokat definiáltam arra az esetre, amikor a mobilkészülék egy állomáshoz képest közel, illetve távol van.
        - képernyő elrendezésének változtatása
            - zenelejátszó megjelenítése/elrejtése
            - térkép megjelenítése/elrejtése

5.9 Oldalak kinézetének megvalósítása

Az alkalmazás során a legtöbb widget-nél a keretrendszer által biztosított Material design-t használtam. Így az alkalmazás kinézete egységes és letisztult maradt.

A képernyők, oldalak megtervezésénél alap stílusként a sétához tartozó weboldal stílusát vettem. Az alkalmazásban található szöveges leírások nagy részét is innen vettem át. Ezt a weboldalt nem én terveztem, kódoltam.

5.10 Külső könyvtárak

A mobilalkalmazás elkészítése során több külső könyvtárat is használtam, amik nagyban segítették a fejlesztési folyamatot. Ezek nagyrészét a pub.dev [18] oldalon

kerestem. Ez egy hivatalos oldal, ahonnan le lehet tölteni a Google és mások által készített Dart és Flutter könyvtárakat. Ebben a fejezetben ezeket a könyvtárakat mutatom be.

5.10.1 Térkép és lokáció könyvtárak

A térképes navigáció implementálás során 3 külső könyvtárat is felhasználtam.

Az első a google_maps_flutter (v. 2.2.1) [19]. Ez egy Flutter plugin, ami Google Térképes widget-eket deklarál. Ennek a segítségével hozom létre a térképeket, azokon a markereket és az útvonalat imitáló egyeneseket. Valamint ennek a segítségével jelenítem meg a saját helyzetemet a térképen.

A második a *geolocator* (v. 9.0.2) [20]. Ez egy olyan plugin, aminek a segítségével könnyen el lehet érni a platform specifikus location service-eket. A projektben a lokáció frissítésére és két pont közötti távolság meghatározására használtam fel.

A harmadik pedig a *permission_handler* (v. 10. 2.0) [21]. Ennek a segítségével le tudtam kérdezni az operációs rendszer engedélyeit, valamint be tudtam azokat állítani (felhasználó engedélyezésével). Ezt a kódban az alábbi funkció megvalósítására használtam fel: eszköz hely adatainak engedélyezési állapotának elkérése a rendszertől, és annak engedélyezésének a kérése a felhasználótól.

5.10.2 Betöltő képernyő és indító ikonhoz használt könyvtárak

A betöltő képernyő (splash screen) elkészítéséhez az alábbi könyvtárat használtam fel: *flutter_native_splash* (v. 2.1.6) [22]. Ez splash Screen-t tud generálni a különböző platformokra. Fontos, hogy a generált képernyőt native-an hozza létre külön-külön. Ez azért szükséges, hogy az alkalmazás indulásakor egyből és ne csak a flutter inicializálása után jelenjen meg a betöltő képernyő.

Az indító ikon szerkesztéséhez a *flutter_launcher_icons* (v. 0.9.2) [23] csomagot használtam fel. Ez egy olyan parancssori eszközt biztosít, aminek a segítségével könnyedén tudtam frissíteni az alkalmazás indító ikonját.

5.10.3 Állapotkezelési könyvtárak

A 5.8. fejezetben (29. oldalon) leírt állapotkezeléshez három könyvtárat használtam fel.

Az első a *bloc* (v. 7.2.1) [24]. Ez egy olyan állapotkezelési könyvtár, amely segít megvalósítani a BLoC tervezési mintát.

A második a *flutter_bloc* (v. 7.3.3) [25]. Olyan Flutter widget-eket tartalmaz, amelyek megkönnyítik a BLoC tervezési minta használatát.

A harmadik pedig az *equatable* (v. 2.0.3) [26]. Egy dart package, amely segít az értékalapú egyenlőség megvalósításában.

5.10.4 Hang és videó lejátszáshoz használt könyvtárak

Az elkészült alkalmazásban található hanganyagok lejátszásához, és a lejátszó vezérlőjének megjelenítéséhez a *just_audio* (v. 0.9.20) [27] plugin-t használtam. Ez különböző forrásokból képes betölteni és lejátszani hanganyagokat, amihez testre szabható lejátszót is biztosít.

Az érdekességek oldalon található videó lejátszót a *youtube_player_flutter* (v. 8.0.0) [28] segítségével valósítottam meg. Ez egy olyan plugin, ami YouTube videók lejátszását és streamelését biztosítja. A lejátszáshoz a hivatalos IFrame Player API-t használja [29].

5.10.5 Google betűtípus könyvtár

Több szöveg formázásánál is használtam a Google által biztosított betűtípus készletet. Ezeknek az eléréséhez, felhasználásához a *google_fonts* (v. 3.0.1) [30] könyvtárat vettem igénybe. Ez egy Flutter package Google betűtípusok [31] használatához.

5.10.6 Firebase könyvtárak

Ebben a fejezetben rövidem leírom a Firebase szolgáltatásainak a megvalósításához felhasznált könyvtárakat.

Az első a *firebase_core* (v. 2.3.0) [32]. Ez egy Flutter plugin, amely lehetővé teszi több Firebase-alkalmazáshoz való csatlakozást. A Storage, Authentication és Realtime Database szolgáltatásoknál is szükséges ennek a használata.

A második a *firebase_storage* (v. 11.0.6) [33] plugin, ami egy beépülő modul a Firebase Cloud Storage API használatához. A harmadik a *firebase_auth* (v. 4.2.1) [34]. Ez egy plugin a Firebase Authentication API használatához. A negyedik pedig a

firebase_database (v. 10.0.6) [35] plugin, ami a Firebase Authentication API használatát teszi lehetővé.

5.10.7 Url indító és e-mail cím validáló könyvtár

Webböngésző, levelező, tárcsázó, és egyéb külső alkalmazások elindításához az *url_launcher* (v. 6.1.7) [36] plugint használtam fel, ami támogatja a telefonos, SMS- és e-mail-sémákat.

Az e-mail címek helyes formázási ellenőrzését az *email_validator* (v. 2.0.1) [37] könyvtár segítségével valósítottam meg. Ebben a könyvtárban egy egyszerű Dart osztály található a mail címek ellenőrzéséhez.

5.11 Verziókezelés

A fejlesztés folyamata során verziókezelésre a GitHub felületét használtam. A GitHub felületén a projekt egyelőre még privát, többek között azért, mert vannak benne olyan szenzitív adatok, amiket nem lenne célszerű publikussá tenni.

6 Értékelés és továbbfejlesztési lehetőségek

Az alkalmazást a feladatkiírás alapján sikeresen megterveztem, megvalósítottam és leteszteltem. Az alkalmazást kipróbáltam Android és iOS emulátoron is. Ezek mellett egy Androidos telefont használva a helyszínen is volt alkalmam kipróbálni. Az élő helyszíni tesztelés után elmondható, hogy a funkciók jól működnek, de az állomások koordinátáin még pontosítani kell. Megemlíteném, hogy az alkalmazás már készen áll a mobilalkalmazások áruházaiba való feltöltésre, de amíg a társaskör (a megrendelő) nem látja a végleges verziót, addig ezt a lépést nem szerettem volna megtenni (költségek és egyéb indokok miatt).

Úgy gondolom, hogy a megvalósított funkciók mellett még számos módon lehetne kiegészíteni az alkalmazást, hogy a felhasználók még jobb élményben részesülhessenek használata közben. Ebben a fejezetben ezekről a lehetőségekről és a fejlesztés alatt szerzett tapasztalataimról írok.

6.1 Továbbfejlesztési lehetőségek

A felhasználó saját profiljába belépve egy dolgot tud megtekinteni: egy listát, amiben a megtekintett, illetve a még nem megtekintett állomások láthatóak egymás alatt, külön rendezve. Ezen kívül érdemes lenne egy külön lapot csinálni, ahol a saját profiljának az adatait tudja megadni és szerkeszteni.

Az alkalmazás, mint ahogy már korábban is írtam, le lett tesztelve élesben (a helyszínen) és megfelelően is működik, de rendes tesztek még nem készültek a kódhoz. Mielőtt a különböző mobilalkalmazások áruházaiba kerülne, mindenképpen szeretném ezt pótolni és ezzel csökkenteni a későbbi hibalehetőségeket.

Hosszú távú tervem a projekttel kapcsolatban az, hogy az alkalmazást átalakítsam egy sokkal több funkciót ellátó, „Wekerle” vagy hasonló névre hallgató applikációra. A településen több ehhez hasonló séta is van már, amiket hasonló módon lehetne integrálni az új alkalmazásba. Ezen kívül lenne egy külön oldal, ahol a wekerlei programok között lehetne böngészni, illetve egy másik oldalon a helységgel kapcsolatos híreket is el lehetne olvasni. A rendezvények ideje alatt pedig különböző interaktív vetélkedők kiegészítését szolgálná a telefonos program. Ezekről a lehetőségekről mindenképpen szeretnék egyeztetni a Wekerlei Társaskör Egyesülettel.

7 Irodalomjegyzék

[1] Apple Inc., „Apple,” [Online]. Available: https://www.apple.com/ios/ios-16/. [Hozzáférés dátuma: 08 12 2022].

[2] Google, „Android,” [Online]. Available: https://www.android.com/. [Hozzáférés dátuma: 08 12 2022].

[3] Google, „Flutter,” [Online]. Available: https://flutter.dev/. [Hozzáférés dátuma: 08 12 2022].

[4] Wekerlei Társaskör Egyesület, „Wekerlei Térvers,” [Online]. Available: http://wekerleitervers.hu/. [Hozzáférés dátuma: 08 12 2022].

[5] Wikipédia, „Wekerletelep - Wikipédia,” 08 12 2022. [Online]. Available: https://hu.wikipedia.org/wiki/Wekerletelep.

[6] Google, „Firebase,” [Online]. Available: https://firebase.google.com/. [Hozzáférés dátuma: 08 12 2022].

[7] C. Sells, „What’s New in Flutter 2.10. Windows stable, performance… | by Chris Sells | Flutter | Medium,” [Online]. Available: https://medium.com/flutter/whats-new-in-flutter-2-10-5aafb0314b12. [Hozzáférés dátuma: 08 12 2022].

[8] . C. Kevin Jamaul, „What’s new in Flutter 3. Deep dive into our latest release… | by Kevin Jamaul Chisholm | Flutter | Medium,” [Online]. Available: https://docs.flutter.dev/development/tools/sdk/release-notes. [Hozzáférés dátuma: 08 12 2022].

[9] Google, „Dart programming language | Dart,” [Online]. Available: https://dart.dev/. [Hozzáférés dátuma: 08 12 2022].

[10] Pluralsight team, „JavaScript.com,” [Online]. Available: https://www.javascript.com/. [Hozzáférés dátuma: 08 12 2022].

[11] Google, „Firebase Authentication,” [Online]. Available: https://firebase.google.com/docs/auth. [Hozzáférés dátuma: 08 12 2022].

[12] Google, „Firebase Realtime Database,” [Online]. Available: https://firebase.google.com/docs/database. [Hozzáférés dátuma: 08 12 2022].

[13] Google, „Firestore | Firebase,” [Online]. Available: https://firebase.google.com/docs/firestore. [Hozzáférés dátuma: 08 12 2022].

[14] Google, „Cloud Storage for Firebase,” [Online]. Available: https://firebase.google.com/docs/storage. [Hozzáférés dátuma: 08 12 2022].

[15] Google, „Cloud Computing Services | Google Cloud,” [Online]. Available: https://cloud.google.com/. [Hozzáférés dátuma: 08 12 2022].

[16] Google, „Firebase console,” [Online]. Available: https://console.firebase.google.com/. [Hozzáférés dátuma: 08 12 2022].

[17] Google, „Google Maps Platform | Google Developers,” [Online]. Available: https://developers.google.com/maps. [Hozzáférés dátuma: 08 12 2022].

[18] Google, „Dart packages,” [Online]. Available: https://pub.dev/. [Hozzáférés dátuma: 08 12 2022].

[19] Google, „google_maps_flutter | Flutter package,” [Online]. Available: https://pub.dev/packages/google_maps_flutter. [Hozzáférés dátuma: 08 12 2022].

[20] Google, „geolocator | Flutter package,” [Online]. Available: https://pub.dev/packages/geolocator. [Hozzáférés dátuma: 08 12 2022].

[21] Google, „permission_handler | Flutter package,” [Online]. Available: https://pub.dev/packages/permission_handler. [Hozzáférés dátuma: 08 12 2022].

[22] Google, „flutter_native_splash | Flutter package,” [Online]. Available: https://pub.dev/packages/flutter_native_splash. [Hozzáférés dátuma: 08 12 2022].

[23] Google, „flutter_launcher_icons | Flutter package,” [Online]. Available: https://pub.dev/packages/flutter_launcher_icons. [Hozzáférés dátuma: 08 12 2022].

[24] Google, „bloc | Flutter package,” [Online]. Available: https://pub.dev/packages/bloc. [Hozzáférés dátuma: 08 12 2022].

[25] Google, „flutter_bloc | Flutter package,” [Online]. Available: https://pub.dev/packages/flutter_bloc. [Hozzáférés dátuma: 08 12 2022].

[26] Google, „equatable,” [Online]. Available: https://pub.dev/packages/equatable. [Hozzáférés dátuma: 08 12 2022].

[27] Google, „just_audio | Flutter package,” [Online]. Available: https://pub.dev/packages/just_audio. [Hozzáférés dátuma: 08 12 2022].

[28] Google, „youtube_player_flutter | Flutter package,” [Online]. Available: https://pub.dev/packages/youtube_player_flutter. [Hozzáférés dátuma: 08 12 2022].

[29] Google, „YouTube Player API Reference for iframe Embeds | YouTube IFrame Player API | Google Developers,” [Online]. Available: https://developers.google.com/youtube/iframe_api_reference. [Hozzáférés dátuma: 08 12 2022].

[30] Google, „google_fonts | Flutter package,” [Online]. Available: https://pub.dev/packages/google_fonts. [Hozzáférés dátuma: 08 12 2022].

[31] Google, „Browse Fonts - Google Fonts,” [Online]. Available: https://fonts.google.com/. [Hozzáférés dátuma: 08 12 2022].

[32] Google, „firebase_core | Flutter package,” [Online]. Available: https://pub.dev/packages/firebase_core. [Hozzáférés dátuma: 08 12 2022].

[33] Google, „firebase_storage | Flutter package,” [Online]. Available: https://pub.dev/packages/firebase_storage. [Hozzáférés dátuma: 08 12 2022].

[34] Google, „firebase_auth,” [Online]. Available: https://pub.dev/packages/firebase_auth. [Hozzáférés dátuma: 08 12 2022].

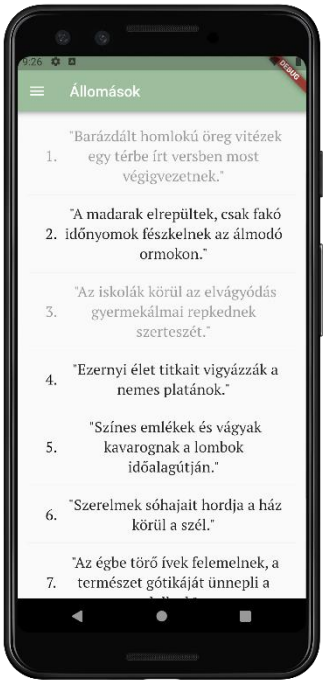
[35] Google, „firebase_database,” [Online]. Available: https://pub.dev/packages/firebase_database. [Hozzáférés dátuma: 08 12 2022].

[36] Google, „url_launcher,” [Online]. Available: https://pub.dev/packages/url_launcher. [Hozzáférés dátuma: 08 12 2022].

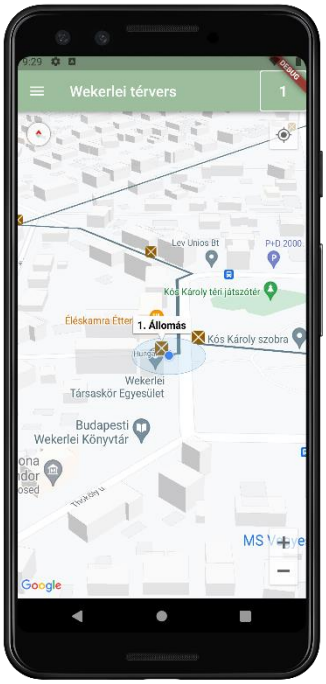
[37] Google, „email_validator | Flutter package,” [Online]. Available: https://pub.dev/packages/email_validator. [Hozzáférés dátuma: 08 12 2022].

8 Függelék

17. ábra: Kezdő képernyő bejelentkezett felhasználói állapotban



18. ábra: Állomások képernyő, 1. és 3. állomás megtekintett állapotban



15. ábra: Térkép képernyő, első állomás közelben, kamera a következő állomás irányába néz

16. ábra: Videó lejátszása teljes képernyőn

21. ábra: Regisztráció képernyő

22. ábra: Felhasználói fiók képernyő, mindegyik állomás nem megtekintett állapotban

19. ábra: Felhasználói fiók képernyő, mindegyik állomás megtekintett állapotban

20. ábra: Állomás részletei képernyő, készülék közelében, az adott állomáshoz nincs még hanganyag