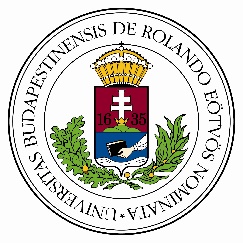
**Eötvös Loránd Tudományegyetem  
Informatikai Kar  
Média- és Oktatásinformatika Tanszék**

**Valós idejű elektromos autó töltőállomás kereső alkalmazás**

Heizlerné Bakonyi Viktória Neue Károly  
műszaki tanár nappali tagozat  
 programtervező informatikus

Budapest, 2018

Téma bejelentő…

Tartalomjegyzék

**Nincsenek tartalomjegyzék-bejegyzések.**

1. Bevezetés
   1. Villanyautó

Napjainkban rohamosan növekszik az villanyautók (más néven elektromos autók) piaca, a világon egyre többet adnak el belőlük. Egy 2017 novemberi felmérés alapján a világszerte már több mint 3 millió elektromos autó van használatban.

Az elektromos autó egy villanymotorral meghajtott gépkocsi. Ez egy jó alternatíva a belső égésű motorok helyett, amelyek többsége benzin és gázolaj alapú. Léteznek úgynevezett hibridautók is, amelyek egyszerre két motorral is rendelkeznek, egy elektromotorral, illetve többnyire még egy benzinmotorral.

A villanyautók nagy előnye az alacsony üzemanyagköltség, ezt több ingyenes töltési lehetőség még olcsóbbá teszi. Továbbá nincs zaj- illetve légszennyezés.

1.2. Elektromos autó töltőállomás

Problémát jelent, hogy az elektromos autók hatótávolsága még jóval kisebb a benzinnel vagy gázolajjal működő társainál. Ezeket az autókat lemerülésük ellőtt vagy egy az erre a célra kihelyezett töltőállomásoknál vagy az otthoni áramot fogyasztva kell feltölteni. Természetesen az előbbi megoldás jóval gyorsabb és Magyarországon ezek a töltőállomások többsége ingyen használható jelenleg. Szerencsére hazánkban is egyre több töltőállomást építenek országszerte a frekventált helyekre, de ezekből Budapesten található meg a legtöbb.

1.3. Az alkalmazás célja

Az általam írt „*eHubSharing*” névre hallgató androidos alkalmazás célja, a Magyarországon található elektromos autó töltőállomások összegyűjtése, megjelenítése egy térképen, illetve, hogy azokat a felhasználók előre le tudják foglalni. Minden töltőállomáshoz tartozik egy adatlap, amelyen a töltő főbb tulajdonságai találhatóak. Továbbá a felhasználó a kedvencei közé teheti a neki megfelelő töltőállomásokat.

Ezenfelül egy külön menüpontban meg lehet tekinteni az aktuális foglalásokat, azokat itt törölni is lehet. Az alkalmazás lehetőséget nyújt a töltőállomások térképen való manuális kikereséséhez, valamint egy kereső is található benne.

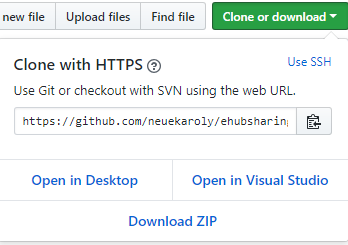
1. Felhasználói dokumentáció

A következőkben ismertetjük a program telepítését, összes funkcióját, azoknak pontos használatát. Továbbá részletezük a felhasználó számára a szükséges előfeltételeket az applikáció használatához.

2.1 Az alkalmazás telepítése

Az alkalmazás használatára két lehetőségünk is van. Ha rendelkezünk megfelelő Androidos készülékkel akkor könnyebb dolgunk van, de használhatunk helyette egy emulátort is számítógépen keresztül. A fejlesztés során végig emulátort használtam megfelelő mobilkészülék hiányában.

Az applikációhoz szükséges teljes projektet megtaláljuk a szakdolgozathoz mellékelt CD-n, vagy a <https://github.com/neuekaroly/ehubsharing> weboldalról is letölthetjük a „Clone or download” gombra kattintva azon belül pedig a „Downlod ZIP”-re. Miután letöltődött csomagoljuk ki.



2.1 ábra. A teljes projekt letöltése GitHubról

2.1.2 Alkalmazás telepítése Android Studio használata nélkül

Ha nem szeretnénk az Anroid Studio-t használni és rendelkezünk egy saját androidos készülékkel, akkor lehetőségünk van az *ehubsharing\app\release* mappában lévő ehubsharing.apk-t használni a telepítéshez. Ehhez a következő lépések szükségesek:

1. A készülék beállításaiban a Security fül alatt engedélyezzük az Unknow sources funkciót
2. Csatlakoztassuk a számítógéphez USB kábelen keresztül a készüléket és várjuk meg amíg az felismeri
3. Készítsünk egy üres mappát a mobilon, amit nevezünk el tetszőlegesen
4. Az APK file-t másoljuk az mappába
5. Navigáljunk a mobilon az elkészített mappába
6. Nyomjunk a felmásolt APK file-ra és engedélyezzük a telepítést
7. Végül a Settings-ben deaktiváljuk az Unknow sources funkciót a készülék biztonsága érdekében.

2.1.2 Alkalmazás telepítése Android Studioval

Az alkalmazás elindításához az Android Studio-ra és egy saját Androidos készülékre van szükségünk vagy a Gennymotion nevű Android emulátorra.

Az Android Studiot töltsük le a <https://developer.android.com/studio/> weboldalról, majd telepítsük. Indítás után a jobb felső sarokban lévő File > Open opcióra kattintva tallózzuk be a korábban letöltött és kicsomagolt eHubSharing projektet. Ezek után, ha lehetséges csatlakoztassunk egy Androidos készüléket a számítógéphez. Majd a Anroid Studio-ban a Run > App menüpontra kattintva válaszuk ki a csatlakoztatott eszközt. Ez után az OK gombra kattintva az applikáció feltelepül és automatikusan elindul a készüléken.

Ha nem rendelkezünk Androidos készülékkel, használjuk a Gennymotion emulátort. Ezt letölthetjük <https://www.genymotion.com/> weboldalról ingyenesen, Trial verzióban. Fontos, hogy a használatához a BIOS-ban engedélyeznünk kell a VT-x vagy AMD-V funkciót, attól függően, hogy milyen típusú videókártyát használunk.

Ha megtettük a szükséges lépéseket nincs más dolgunk, mint a Gennymotion-on belül rákattintani az Add menüpontra, kiválasztani a nekünk megfelelő szimulálandó készüléket majd a Star-tal indítani azt. Ezek után ugyanúgy használhatjuk azt, mint egy valós Androidos készüléket és az Android Studio is felismeri.

Fontos, hogy bizonyos szimulált készülékeknél be kell kapcsolnunk a GApps funkciót a Google Maps szolgáltatásokhoz. Ehhez a szimuláció elindítása során a jobb felső sarokban található GAPPS logóra kell kattintanunk. Ez után be kell regisztrálnunk magunkat, majd az emulátor felajánlja, hogy letölti számunkra a Google Play Services szolgáltatásokat, ezt hagyjuk jóvá, és már készen is állunk az alkalmazás telepítéséhez.

2.2 A program funkciói

Ebben a fejezetben bemutatjuk a felhasználó számára, hogy mely funkciókat hogyan érhet el és azokat hogyan használhatja. Részletezük az alkalmazáson belüli térkép használatát, a foglalási rendszert, a töltőállomások között való keresést és a kedvencek közé adást, illetve onnan való törlést.

2.2.1 A program indulása

A program indulásakor a felhasználót egy betöltő képernyő fogadja, amelyen az alkalmazás logója és neve látható. Ez 3 másodperc után automatikusan eltűnik és betöltődik a térkép. Az alkalmazás legelső indulása esetén ez több időt is igénybe veheti, hiszen ekkor történek az alapvető adatok betöltése az adatbázisba.

Az alkalmazás az indulását követően a töltőképernyőnél jelzi a felhasználó számára, ha esetleg nincs megfelelően beállítva a készülék, vagy valamilyen hibát észlelt a térkép betöltése közben. Ha előugrik egy külön ablak, akkor csak nincs megfelelően beállítva a Google szolgáltatások a készüléken, az itt megjelenő utasításokat követve megoldható a probléma. Viszont, ha a „You can’t make map requests” feliratot olvassuk akkor sajnos a készülékünk nem alkalmas az alkalmazás futtatására, ennek okai lehet szoftveres vagy hardveresek is. Például GPS hiánya, vagy nincs minimum API 15 verziójú Android operációs rendszer a készüléken.

2.2.2 Térkép

Az töltőképernyő után a felhasználót egy térképet látt, illetve azon megjelölve a Magyarországon található töltőállomások. Alapesetben a térkép betöltése után, egész Magyarországot láthatjuk. Ennek oka, hogy a felhasználó hosszabb útvonal esetén egyből átláthassa, hogy hol lehet megállni útközben feltölteni az villanyautót.

A térképen navigálni a Google Maps-en megszokott módon lehetséges. A képernyőt húzogatva lehet mozgatni, illetve nagyítani és kicsinyíteni a két ujjunkkal lehetséges. Utóbbira a jobb alsó sarokban lévő „+” és „-„ jelek segítségével is lehetőség nyílik.

A térképen lévő kis fekete jelzések mindegyike egy-egy töltőállomást takarnak. Erre rányomva megjelenik kiemelten a töltőállomás rövid neve, ez, ha nincs külön neve, akkor a címére utal. Az ezalatt lévő jelzés a töltőállomás nyitvatartásáról add tájékoztatást.



2.2 ábra. Töltő állomást jelző ikon

Erre a leírásra nyomva előjön a töltőállomás adatlapja ahol bővebb leírást kaphatunk róla és a foglalását is ott készíthetjük el. Bővebb leírás a használatáról a 2.1.3 fejezetben található.

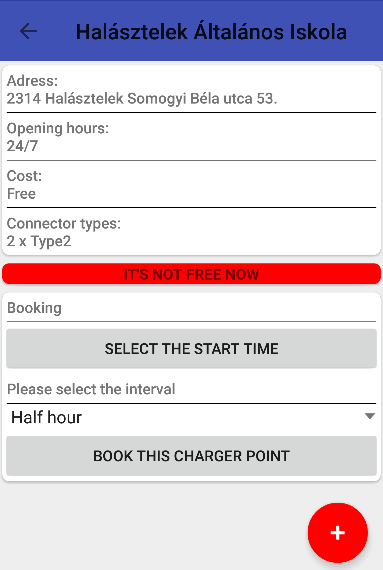
Az útvonal tervezéshez a jobb alsó sarokban lévő nyílra kell nyomni, ekkor megnyílik a Google Maps hivatalos térképe az adott töltőállomásra való navigálással együtt.

A térkép legalján található egy navigációs menü, négy menüponttal, ennek részletes bemutatása 2.1.7 fejezetben olvasható.

2.2.3 Töltőállomás adatlapja

A térképen feltűntetett rövid leírásra nyomva, juthatunk el az adott töltőállomás adatlapjára.

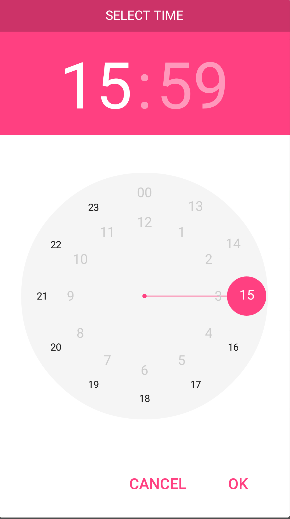
Az ablak tetején találja a felhasználó a töltő részletes adatait. Az Adress-nél a pontos címet, az Openin hours alatt a nyitva tartást, a Cost-nál, hogy az adott töltő ingyenes (ez az alkalmazáson belül Free-vel van jelölve) vagy fizetni kell érte illetve a Connector types alatt csatlakozó típusok találhatók és, hogy hány darab van belőlük.



2.3 ábra. Töltőállomás adatlapja

Az adatok alatt látható kártya a töltő foglaltságát jelzi. Ha éppen foglalt valaki által az adott töltőállomos, akkor a kártya pirosra vált és az „IT’S NOT FREE NOW” felirat jelenik meg rajta, ha szabad akkor zöld színű és az „IT’S FREE NOW” olvasható rajta.

A „Booking” felirat alatt láthatók a foglalási rendszerért felelős beállítási lehetőségek. A „Select the start time” gombra nyomva egy új ablak jelenik meg. Ebben az ablakban választhatjuk ki a foglalás kezdő időpontját. Itt az aktuális időhöz képest korábbi időpontot nem lehet lefoglalni, illetve az este 11 óra után való foglalás is le van tiltva. Először az órát kell kiválasztanunk majd a percet, amit az „OK” feliratra nyomva fogadhatunk el. Ha meggondolnánk magunk akkor a „CANCEL” gombra kattintva visszaléphetünk.



2.4 ábra. Időpontválasztó ablak

Ha kiválasztottuk a kezdési időpontot az intervallum kiválasztása következik. A „Please select the interval” alatt egy lenyíló ablakban választhatjuk, hogy egy („One hour”) vagy fél órára („Half hour”) szeretnénk-e lefoglalni a töltőt. Alapértelmezetten a fél óra van kiválasztva. Ha ezzel is megvagyunk akkor a „Book this charger point” gombra nyomva lefoglalhatjuk a töltőállomást. Ezután egy felirat jelenik meg a képernyő alján a foglalás eredményéről, az alábbiakban ezek 3 fajtáját ismertetjük a felhasználóval:

* „Please select another time, this is already taken!”: ekkor a kiválasztott időpont pont egybe esik egy másik foglalással, nyomjunk a „Select the start time” feliratú gombra és válaszunk ki egy másik megfelelő időpontot
* „Please select the start time”: ekkor nem választottuk ki a kezdési időpontot, nyomjunk a „Select the start time” feliratú gombra és válaszuk ki a megfelelő időpontot
* „Succesfull reservation”: ekkor sikeres volt a foglalás, a rendszer elmentette azt

A töltőállomás adatlapjának jobb alsó sarkában találunk egy piros hátterű „+” ikonú gombot. Erre nyomva az adott töltőállomás bekerül a kedvencek közé, amiket a kedvencek menüpontban tekinthetünk majd meg. A gombra rányomva eltűnik az. Ha megnyitjuk a töltőállomás adatlapját és nem található ez a gomb, akkor az már korábban hozzá lett adva a kedvencek közé.

2.2.4 Foglalások

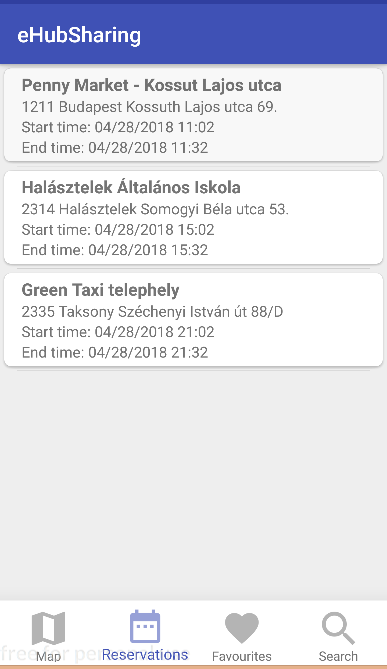
A felhasználó a foglalásait megtekintheti egy külön ablakban, ez a térkép, a kedvencek és kereső menüjéből is elérhető. A foglalások közül csak az éppen aktuálisakat, illetve a jövőben lévőket láthatjuk, a már megtörtént foglalások automatikusan törlődnek az adatbázisból.

Mindegyik foglaláshoz négy információ tartozik. Félkövér betűtípussal olvasható a lefoglalt állomás neve, alatta a pontos címe. A „Start time” a foglalási kezdési időpontját jelzi, míg az „End time” a végét.

Ha egy foglalás kártyájára rányomunk az automatikusan az alkalmazás térképét nyítja meg, amin közelebbről is megnézhetjük a pontos címet, illetve egyből navigációt is indíthatunk ide.

Ha a foglalást balra elhúzzunk az újunkkal, akkor törölni tudjuk azt az adatbázisból és többé nem lesz elérhető, csak ha újra lefoglaljuk azt az adott töltőállomás adatlapján.

Az ablak legalján található egy navigációs menü, négy menüponttal, ennek részletes bemutatása 2.1.7 fejezetben olvasható.



2.5 ábra. Foglalások ablaka

2.2.5 Kedvencek

A korábban kedvencek közé tett töltőállomásokat szintén megtekinthetjük egy külön ablakban, amit a térkép, a foglalások és a kereső menüjéből is elérhetünk.

Mindegyik kedvenc töltőhöz 3 adat van megadva, felülről lefelé olvasva ezek a töltő neve, a pontos címe, illetve nyitvatartása.

Működése hasonló a foglalások menüponthoz, ha egy kedvenc kártyájára rányomunk az automatikusan az alkalmazás térképét nyitja meg, amin közelebbről is megnézhetjük a pontos címet, illetve egyből navigációt is indíthatunk ide.

Ha pedig a kedvencet balra elhúzzunk az újunkkal, akkor törölni tudjuk azt az adatbázisból és többé nem lesz elérhető, csak ha újra hozzáadjuk a kedvencekhez az adott töltő adatlapján.

Az ablak legalján található egy navigációs menü, négy menüponttal, ennek részletes bemutatása 2.2.7 fejezetben olvasható.

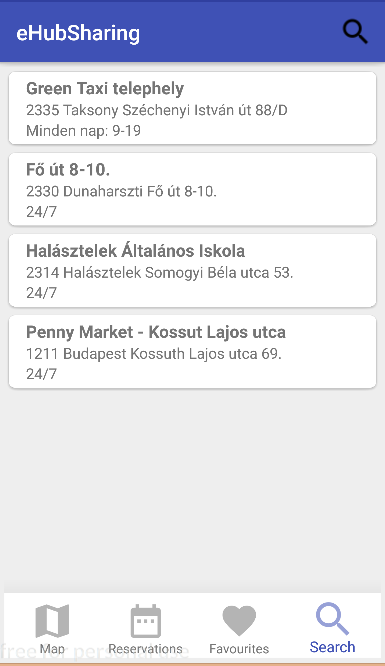
2.2.6 Kereső

Az alkalmazáson belül lehetőség van a töltőállomások címe és neve alapján keresni azok között. Erre egy külön kereső ablak van a programban, amit a térkép, a foglalások és a kedvencek menüjéből nyithatunk meg.

Alapértelmezetten keresés nélkül az összes adatbázisban lévő töltőállomást kilistázza. Az ablak jobb felső sarkában található fekete nagyító gombra kattintva jelenik meg a kereső mező, ahova a keresendő töltő címét vagy nevét kell gépelnünk. Ahogy gépeljük a szöveget az egyből leszűri a keresésnek megfelelő állomásokat.

Ha valamelyik kilistázott keresőre rányomunk azt egyből betölti az alkalmazás térképét, amin közelebbről is megnézhetjük a pontos címét és egyből navigációt indíthatunk.

Az ablak legalján található egy navigációs menü, négy menüponttal, ennek részletes bemutatása 2.2.7 fejezetben olvasható.



2.6 ábra. Kereső ablaka

2.2.7 Navigációs menü

A térkép, a foglalások, a kedvencek és a kereső ablak alján található egy navigációs menü négy darab menüponttal. Ennek használata igen egyszerű. Az éppen aktuálisan megnyitott ablak menüpontja ki van emelve kék színnel ilyenkor a többi 3 szürke színű.

A „Map” megnyomása esetén a térkép ablak nyílik meg, a „Reservations”-nél a foglalásokat tekinthetjük meg, a „Favourites” a kedvenceket takarja, míg a „Search” a keresőt.



2.7 ábra. Navigációs menü

2.3 Követelmények

2.3.1 Szoftveres követelmények

* Git
* Android Studio (opcionális)
* Genymotion (opcionális)
* a készüléken minimum API 15 Android operációs rendszer

2.3.2 Hardveres követelmények

Amennyiben nem közvetlenül egy készülékre szeretné a felhasználó telepíteni az alkalmazást APK kiterjesztésű file-ját, szüksége van az Android Studio fejlesztő környezetre. Ennek használatához a következőkre van szükségünk Windows használata esetén:

* Microsoft Windows 7/8/10 (32 vagy 64 bites verzió)
* 3 GB RAM minimum, 8GB RAM az ajánlott plusz 1 GB az, ha emulátort használunk
* minimum 2 GB szabad tárhely, 4 GB az ajánlott
* minimum 1280 x 800 pixeles képernyőfelbontás

Linux vagy Mac használata esetén a követelmények változhatnak, ezekről bővebben a <https://developer.android.com/studio/> oldalon lehet olvasni a „System requirements” fejezetben.

Továbbá, ha nem rendelkezünk megfelelő androidos készülékkel, akkor helyettesíthetjük azt egy emulátor segítségével, ahogy én is tettem a fejlesztés során. Ajánlott a Genymotion-t használni. Az ehhez tartozó követelmények az alábbiak:

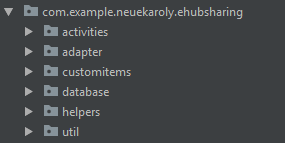
* Microsoft Windows 7/8/10 (32 vagy 64 bites), MacOS 10.9 vagy efölött, illetve Linux Ubuntu 16.04, Debian 9 és Fedora 27
* OpenGL 2.0-val rendelkező videókártya, friss driver verzióval
* 64 bites CPU, VT-x vagy AMD-v lehetőséggel, amit külön a BIOS-ban lehetséges beállítani
* minimum 2 GB RAM
* minimum 1024 x 768 pixeles képernyőfelbontás
* legalább 100 MB szabad tárhely a merevlemezen

1. Fejlesztői dokumentáció

Ebben a fejezetben az alkalmazás részletes specifikációját, a felhasznált módszerek részletes leírását, adatbázistervét és modulfelbontását ismertetjük. Továbbá kitérünk az alkalmazás teljes tesztelési folyamatára, illetve azok eredményére. Az alkalmazás az Android Studio fejlesztő környezetben készült, JAVA programozási nyelven.

3.1 Az alkalmazás szerkezete

Az eHubSharing alkalmazás package szerkezete az általánosan elfogadott android fejlesztési szabályoknak megfélően épül fel. Az androidos alkalmazások fejlesztése során fontos a gondosan rendezett csomagszerkezet, így a későbbiek folyamán könnyebé teszi az eligazodást az osztályok között és az esetleges továbbfejlesztési munkálatokat is előkészíti.

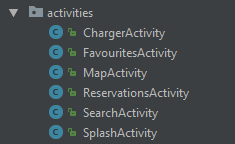


3.1 ábra. Az alkalmazás package struktúrája

A java class fileok a szokásos módon UpperCamelCase módszerrel lettek elnevezve. A következőkben a java-csomagok tartalmát részletezzük, viszont a bennük lévő osztályok tartalmát és részletes működését későbbi fejezetekben tárgyaljuk.

3.1.1 Activities mappa

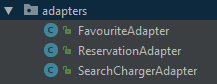
Az activities package-ben található az alkalmazás összes Activity-je. Az Activity az android fejlesztés mondhatni legfontosabb komponense. Ez egy felhasználói felülettel rendelkező képe a programnak. Egy alkalmazásban, mint itt is látható több Activity is található, ezek mind különböző, jól elhatárolt célokat szolgálnak., de együtt adják az alkalmazás teljes funkcionalitását. Az Actitvity osztályok nevére használt általános konvenció, hogy Activity-re kell végződnie. Az Activityk-ről bővebben a 3.2 fejezetben olvashatunk.



3.2 ábra. Activities package

3.1.2 Adapters

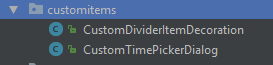
Ebben a mappában találhatóak az alkalmazásban található különböző RecylerView-khoz tartozó adapterek. A RecyclerView az 5.0-ás Android verzióban lett először bemutatva. Ez egy jóval rugalmasabb megoldás az egymás után lévő elemek felsorolására és azok között való görgetésre, mint a régebbi ListView vagy GridView. Az alkalmazáson belül a SearchActivity-ben, ReservationsAcitvity-ben és a FavouritesActivity-ben található egy-egy RecyclerView. A RecyclerView pontos működését a három osztály leírásában részletezzük.



3.3 ábra. Addpters package

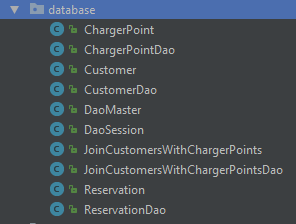
3.1.3 CustomItems

Az itt található osztályok az alkalmazás felhasználói felületén megjelenő olyan elemek, amelyeket valamilyen módon a program számára testre lettek szabva.

  
3.4 ábra CustomItems package

3.1.4 Database

A database mappában az adatbázishoz tartozó entitásokat, illetve azokat kezelő generált osztályokat találjuk. Mindegyik osztályhoz tartozik egy „Dao” osztály, amiket az alkalmazásban használt GreenDao generált, amely egy ORM adatbázis kezelő rendszer. Ennek működését a későbbiekben részletesen kifejtjük az olvasó számára.

  
3.5 ábra. Database package

3.1.5 Helpers package

Ebben a csomagban található az összes olyan osztály, amely nem kapcsolódik logikailag más packagekhez, de az alkalmazás működéséhez feltétlenül szükséges.

  
3.6 ábra. Helpers package

3.1.6 Util package

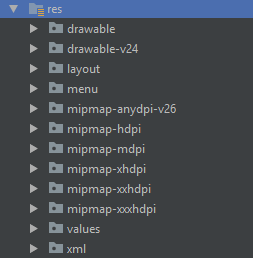
Az itt található StringUtils osztály az olyan String adattípúshoz tartozó függvényeket valósítja meg, amelyek az alap java packagek-ben nem találhatóak meg.

  
3.7 ábra. Util package

3.1.2 Res mappa

A res mappában több almappa is található. Ezekben található fileok mind az alkalmazás vizuális kinézetéért felelősek. A nagy részük XML-kiterjesztésűek, de van közöttük png képfile is.

A drawable és drawable-v24 mappákban az ikonok képei találhatóak. Az összes activity-hez tartozik egy XML file, ezeket tárolja a layout mappa. A menu-ben és az XML -ben a navigációs menühöz szükséges XML-ek vannak. A különböző végződésű mipmap csomagokban képek találhatóak, több méretben is a reszponzívitás megtartása érdekében. A values-ben az XML-k által használt színek, stringek, stílusok és számértékek vannak tárolva.

  
3.8 ábra. Res mappa

3.2 Activity

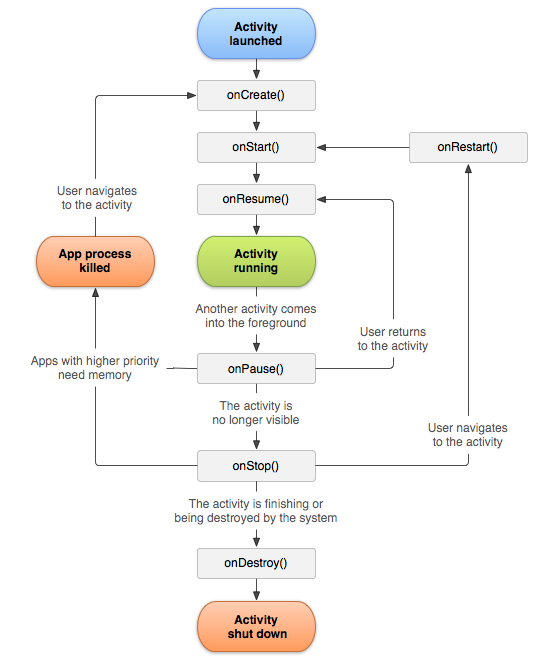
Az Activity egy alkalmazás komponens, ami egy képernyőt biztosít, amelyen keresztül a felhasználó kezelheti az alkalmazás szolgáltatásait. Minden activity-hez tartozik egy XML file, amely az ablak pontos kinézetét definiálja.

Egy alkalmazás általában több Activityt tartalmaz. Ezek közül van egy, amelyet main activityként kezel, ez jelen esetben a SplashActivity. Ez indul el, mikor a felhasználó először futtatja az alkalmazást. Minden activity indíthat másik activityt. Amikor egy új activity indul, az előző megáll és ezt a rendszer egy verembe teszi. Ez a verem LIFO mechanizmus alapján működik, így, ha a felhasználó visszatér az új activityből, akkor az előző fog folytatódni. Az ilyen állapotváltozásokat különböző callback metódusok hívásával jelzi a rendszer az activitynek.

A fejlesztés során fontos figyelembe venni az activity életciklusát, és hogy melyik tevékenységet melyik callback metódusban tesszük.

Az alábbi ábra részletezi, hogy egy activity az indulásától egészen a végleges törléséig milyen folyamatokon esik át. A mi alkalmazásunk legtöbbször az onCreate() illetve az onResume() metódusokat írja felül. Előbbi az adott activity legelső indulása esetén fut le, ezt kötelező minden activity-nél implementálni és ebben történik a felhasználói felület inicializálása. Míg utóbbi, akkor hívódik meg, ha egy másik activityből visszatérünk a jelenlegire.

Általános konvenció, hogy az activityk-ben található nem publikus és nem statikus változó neveket kis „m” -el kezdjük.



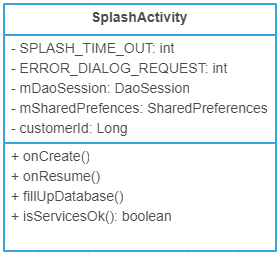
3.9 ábra. Egy Android acitvity életciklusa

3.2.1 SplashActivity

A SplashActivity felel az alkalmazás indításakor megjelenő töltőképernyőért, ahol egy logó és az alkalmazás neve jelenik meg. A háttérben több fontos inicializálási folyamat is folyik, amiket a felhasználó nem észlel.

A program itt ellenőrzi, hogy legelsőnek fut-e az adott készüléken az alkalmazás, ha letöröltük azt majd újratelepítettük, az szintén legelső futásnak minősül. Az alkalmazás első futása esetén meghívódik a fillUpDatabase() metódus amely az assets mappában lévő chargers.txt-ben található adatokkal feltölti az adatbázist. Ebben a szöveges file-ban egy JSON formátumú szöveg található, amely a töltőállomások adatait reprezentálja.

Függetlenül, hogy hányadik indulása a programnak az isServicesOk() metódus mindig lefut, ez ellenőrzi, hogy a térkép szolgáltatásokhoz szükséges beállításokat eltudta-e végezni az alkalmazás. Amennyiben bármilyen probléma merült fel azt a felhasználó számára jelzi a képernyőn. Ha mindent rendben talált, akkor automatikusan elindítja a következő activity-t, amely a térképet tartalmazza.

  
3.10 ábra. SplashActivity UML diagramja

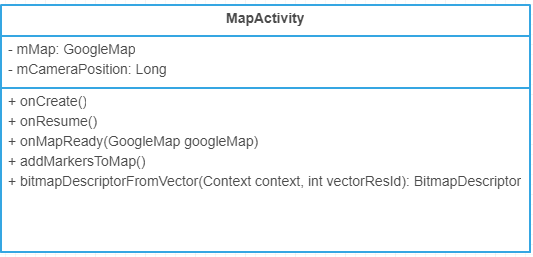
3.2.2 MapActivity

A MapActivity a térkép szolgáltatásokért felelős. Az indulásakor lefutó onCreate() metódusban ellenőrzi, hogy kapott-e az őt meghívó előző activity-től Intent-ben extra Stringet. Amennyiben igen, akkor az egy töltőállomást reprezentál, és a térkép kamerája egyből a kapott töltőre fókuszál, ha nem akkor egész Magyarországot mutatja felülről.

Az előbb említett Intentek futási időben kötik össze a különálló komponenseket. Jelen esetben két Acitivityt, amelyek képesek egymásnak adatokat átadni az Intent-en keresztül. Ennek pontos lebonyolítását az operációs rendszer végzi a háttérben.

Az addMarkersToMap() rendeli hozzá adatbázisban lévő töltőállomásokat a térképhez a megadott hosszúsági és szélességi fokok alapján. A térképen előre megjelölt helyszíneket markereknek nevezzük. Ezekhez a markerekhez lehetőségünk van saját ikont is hozzárendelni, viszont ehhez a Google Maps API saját objektum típusára van szükségünk, a BitmapDescriptor-ra. Ezért volt szükség a bitmapDescriptorFromVector függvény megírására. Ez tetszőleges vektorgrafikákat képes átalakítani BitmapDescriptor típusúvá.

A MapActivity implementálja a Maps API OnMapReadyCallback interfészét, ezért felül írjuk az onMapReady metódust, amely akkor hívódik meg, ha sikeresen betöltődött a térkép.



3.11 ábra. MapActivity UML diagramja

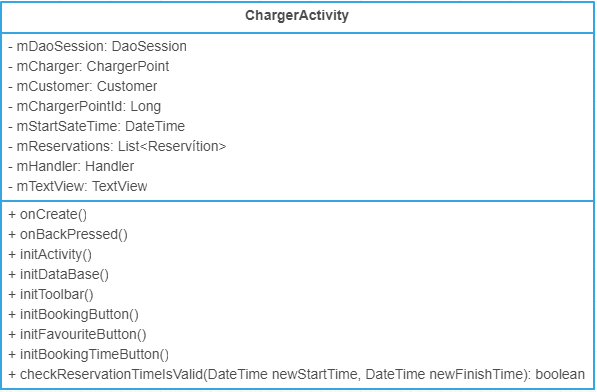
3.2.3 ChargerActivity

A ChargerActivity a töltőállomások adatlapját jeleníti meg. Itt található a szakdolgozat címében is olvasható valós idejűségért felelős rendszer. Ennek működésére egy külön alfejezetet szentelünk a későbbiekben.

Több inicializáló metódus is található az osztályban, amelyek beállítják az UI elemeket, illetve lekérik a szükséges adatokat az adatbázisból. A többi activityhez képes különbség, hogy itt megtalálható egy initToolbar() nevű eljárás, amelyben beállítjuk, hogy az activity fejlécében az adott töltőállomás neve legyen olvasható.

Ez az activity felelős a foglalási rendszer lebonyolításáért is. Itt használjuk a TimePickerDialog-ot a kezdési idő kiválasztására, amelyben kiválasztott időpontot a checkReservationTimeIsValid függvény ellenőriz, hogy nem-e ütközik már meglévő foglalással. Ha ez igazzal tér vissza, akkor a foglalást elfogadja a rendszer, és feljegyzi azt az adatbázisban.

A kedvencek közé adáshoz egy FloatingActionButton-t használunk, amely hide() metódusát használjuk fel az eltűntetésére, ha már a kedvencek között van az adott töltőállomás.



3.12 ábra. ChargerActivity UML diagramja

3.2.4 FavouritesActivity

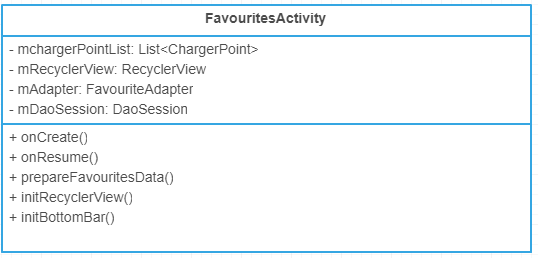
A FavouritesActvity a felhasználó kedvenc töltőállomásait sorolja fel, hozzáadási sorrendben.

Az activity alapja egy RecyclerView. Erre azért van szükség, hogy dinamikusan lehessen futási időben hozzáadni, illetve elvenni elemeket, és azok között görgetni. Mindegyik RecyclerView-hoz tartozik egy Adapter osztály az adapters package-ben. Ehhez konkrétan a FavouriteAdapter kapcsolódik, amely egy kedvenc töltőállomás elemet reprezentál.

Az initRecyclerView metódus inicializálja a RecyclerView-t, és beállítja, hogy az elemek balra való elhúzása esetén töröljük azt az adatbázisból, illetve az elemre való rányomással megnyitjuk azt a térképen. Utóbbi esetén a korábban már tárgyalt Intent objektumot használjuk a szükséges adatok átadatáshoz.

Az initBottomBar metódus mindegyik navigációs menüt tartalmazó Activity-ben megtalálható, ez állítja be azt.

A prepareFavouritesData pedig a szükséges adatokat kéri le az adatbázistól.

  
3.13 ábra. FavouritesActivity UML diagramja

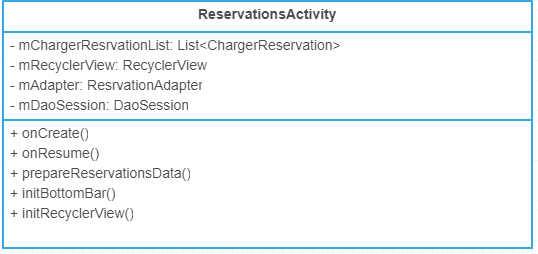
3.2.5 ReservationsActivity

A ReservationsActivity a foglalások megjelenítéséért felelős és azokon végzet műveletekért.

Működése hasonló a FavouritesActivity-hez csak itt foglalások jelenek meg a RecyclerView-ban. Az ehhez tartozó adapter a ReservationAdapter.

Megtalálhatóak a szokásos inicializáló metódusok, amelyek a felhasználó felülethez tartozó kinézetért, illetve működéséért felelős. Ilyen az initBottomBar és az initRecyclerView, utóbbi valósítja meg az elemek törlését és a térképre való navigálását is.

A prepareReservationData metódus kéri le az adatbázistól a szükséges adatokat és egyben ez törli a már lejárt foglalásokat is az adatbázisból.

  
3.14 ábra ReservationActivity UML diagramja

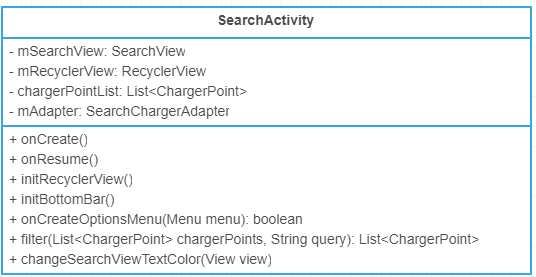
3.2.6 SearchActivity

A SearchActivity a kereső ablak megjelenítését és annak kezelését végzi. Szintén található benne egy RecyclerView amihez található egy adapter osztály, a SearchChargerAdapter.

A szokásos initRecyclerView és initBottomBar inicializáló metódusok és onCreate, onResume működése hasonló a ReservationActivity és FavouritesActivity-hez.

A nagy különbség, hogy itt a keresés miatt a RecyclerView elemei szűrhetőek. Ehhez segítség a SearchView osztályhoz tartozó setOnQueryTextListener, amely folyamatosan figyeli a kereső mezőt, hogy éppen gépel-e valamit a felhasználó, így egy betű lenyomása, illetve törlése esetén egyből szűrhetőek is a megfelelő találatok.

Másik fontos metódus a filter függvény, amely minden egyes kereső szöveg változásnál meghívódik. Ez a kapott kereső szöveg alapján leszűri a felhasználó számára megfelelő találatokat és csak azokat jeleníti meg az ablakban.

  
3.15 ábra. SearchActivity UML diagramja

3.3 Egyéb fileok

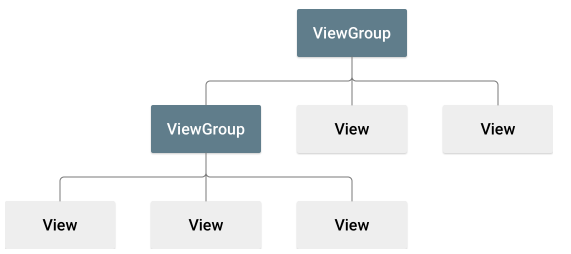
3.3.1 XML file

XML file kiterjesztésűek az alkalmazás pontos kinézetét leíró, illetve a stringeket és egyéb értékeket tároló fileok. Minden activity-hez tartozik egy-egy saját XML file. Ezekben írjuk le a gombok, képek, szövegek és egyéb UI elemek pontos kinézetét, tulajdonságát és azoknak egymáshoz való elhelyezkedését.

Az XML (Extensible Markup Language, Kiterjeszthető Jelölő Nyelv) a W3C által fejlesztett általános célú leíró nyelv. Az elsődleges célja strukturált szöveg és információ megosztása volt az interneten keresztül.

Az Android programozás során használt XML fileok nagy előnye, hogy mind ember és mind gép számára jól strukturált. olvasható formátum. Ehhez természetesen a fejlesztés során ügyelni kell a jólformáltságra és helyes használatra. Egy helyesen formázott XML dokumentum megfelel minden XML szintaxis szabálynak.

Az XML-ek többségében megtalálhatóak az úgynevezett Layout-tok, magyarul elrendezések. Egy Layout meghatározza az alkalmazás felhasználó felületének struktúráját. Az elrendezés összes eleme a View és ViewGroup objektumok hierarchiája alapján készül. A View-k többnyire olyan dolgokat mutat be, amelyeket a felhasználó konkrétan láthat és használhatja őket. A ViewGroup egy láthatatlan konténer, amely meghatározza a View és más ViewGroup objektumok elrendezési struktúráját. Ennek működését az alábbi ábrán látható példa mutatja.



3.16 ábra. A View hierarchia, amely meghatározza az UI felület elrendezését

Ezeket a Layoutokat kézzel is lekódolhatjuk, vagy az Android Studio lehetőséget nyújt egy drag-and-drop szerkesztőre. A fejlesztés során csak időhiányában ajánlatos az utóbbit használni, hiszen, ha jól ismerjük az XML programozást, akkor jobban testre tudjuk szabni az elemeket saját kezűleg.

A két legismertebb és leggyakrabban használt Layout a Relative Layout és a Linear Layout. A Relative Layout olyan nézetcsoport, amelyben az elemeket egymáshoz képest lehet elrendezni. Az egyes nézetek a testvérelemekhez vagy a szülő (Relative Layout) területéhez képest lehet a megfelelő pozícióba helyezni. Míg a Linea Layout esetén soronkénti vagy oszloponkénti elrendezés teljesül. Azaz minden vagy horizontálisan vagy vertikálisan jelenik meg egymáshoz képest. Ez kevésbé nyújt szabad testreszabhatóságát, mint a Relative Layout.

A következőkben felsorolunk pár az alkalmazásban használt és egyben elég gyakori úgynevezett XML View-t. Mindegyik View külön testre szabható az attribútumain keresztül.

* TextView: Szöveget képes megjeleníteni a felhasználó számára. Több saját attribútuma is van, amelyeken keresztül a szöveg kinézetet lehet változtatni. Az alkalmazásunkban ilyet használunk például a töltőállomás adatlapjánál a fontosabb információk felsorolásához.
* Button: Egy gomb, amelyet a felhasználó megnyomhat és ennek hatására használhatja az alkalmazás adott funkcióját. Ismert az ImageButton féle verziója is, itt nem egy szöveg jelenik meg a gombon, hanem egy kép. A sima szöveges fajtáját használjuk például a foglalások megerősítése során.
* Spinner: Olyan nézet, amely csak egy elemet jelenít meg egyszerre, de lehetővé teszi a felhasználó számára, hogy egy lenyíló ablakban a többi lehetőség közül is választhasson. A Spinner elemei az ehhez a View-hoz társított array XML-ben vannak felsorolva. Ilyen látható a foglalási időtartam kiválasztásánál.
* CardView: A CardView valójában egy FrameLayout lekerekített sarkokkal és árnyékkal. A CardView-n belül egyszerre több másik View elemet is felhasználhatunk. Az alkalmazásban ilyen kártya nézet található az összes RecyclerView esetén, illetve a ChargerActivity-ben is.
* FloatingActionButton: Ez a felhasználói felület felett lebegő kerekített ikon, amely egyben egy nyomható gomb is. Kiválasztható a rajta található minta, illetve annak és hátterének a színe. A ChargerActivity-ben a kedvencek közé adást egy ilyen FloatingActionButton végzi.
* RecyclerView: Erről a nézetről már korábban is esett szó. Ez egy rugalmas nézet amely segítségével sok adatot tudunk megjeleníteni egy korlátozott méretű ablakban a görgetés segítségével.

3.3.2 AndroidManifest

Az AndroidManifest szintén egy XML kiterjesztésű file, amely a JAVA gyökérkönyvtárában található. Ezt a filet minden alkalmazásnak kötelezően tartalmazni kell, mivel ez szolgáltat olyan információkat a rendszernek, amiket ismernie kell minden futás előtt.

Ez nevezi meg a JAVA csomag nevét, amely egyedi azonosítóként szolgál az alkalmazásnak. Továbbá tartalmazza az alkalmazás acitvity-jeit, és add hozzájuk esetleges különleges jogokat. Itt lehet például beállítani, hogy melyik activity legyen a main acitvity, azaz indulás esetén melyik jelenjen meg először a képernyőn.

A uses-permission részben adhatjuk meg az alkalmazáshoz szükséges jogosultságokat, ahhoz, hogy hozzáférjenek a rendszer vagy más alkalmazások védett részeihez. Jelen esetben az internet hozzáférésre és a pontos helyzetünkre van szüksége az applikációnak a helyes működéshez.

Itt van feljegyezve a Google Map API-hoz szükséges API\_KEY is, amiről a későbbiekben még esik szó.

3.3.3 Build.gradle

Az Android Studio a fejlesztéshez a Gradle projektépítő eszközt használja. Az Android építő rendszer ezt használja az alkalmazáserőforrások és forráskódok összeállítására, majd ezeket összecsomagolja ezeket egy úgynevezett APK-ba, amelyet tesztelhet, telepíthet és terjeszthet.

A Gradle automatizálja és kezeli a build folyamatot, miközben lehetővé teszi a rugalmas testreszabott konfigurációk meghatározását. A Graddle és az Android független az Android Studiótól. Ez azt jelenti, hogy az alkalmazást akár a gép parancssorából is buildelhetjük.

Két build.gradle file található a könyvtárszerkezetben. A gyökérkönyvtárban lévő legfelső színtű build.gradle határozza meg a projekt össze moduljára vonatkozó beállításokat. Ebben lehet a megfelelő verziójú Gradle-t megadni, amely kihat az alkalmazás összes moduljára.

Egyéni build-konfigurációk és külső könyvtárak használatához módosítani kell a másik build.gradle filet. Ezt modul-szintű build.gradle filenak is szokás nevezni és a fejlesztés során többnyire ezt kell szerkeszteni. A következőkben az ebben található beállítási lehetőségeket taglaljuk.

A compielSdkVersion megadja az Android API szintjét, amelyet a Gradle-nek kell alkalmazni. Ez azt jelenti, hogy az applikáció felhasználhatja adott API szinten és az alatti lévő szinteken található összes funkciót.

A minSdkVersion leírja, hogy minimum milyen API szintű Android rendszerrel kell rendelkezni az adott készüléknek az alkalmazás futtatásához. Ez jelen esetben 15.

A targetSdkVersion meghatározza az alkalmazás teszteléséhez használt API-szintet.

A dependency blokk az egyik legfontosabb része gradle filenak. Itt található az összes alkalmazásban használt függőség. A függőségeknek csak az azonosítóját írja le, hogy az azokhoz tartozó külső könyvtárakat a későbbiekben használni tudjuk a fejlesztés során.

Az alkalmazásban összesen 19 darab külső függőséget használunk. Ezeknek egy részét a következő fejezetben tárgyaljuk.

3.4 Felhasznált módszerek

A következőkben az alkalmazás során felhasznált módszereket ismertetjük. Ezek többsége külső könyvtár amiknek használatához az előbb bemutatott build.gradle-be dependency-ként hivatkozni kellett.

Több esetben előfordul olyan, hogy egy külső alkalmazás szolgáltatásait szeretnénk felhasználni. Ezekre szolgálnak az API-k, azaz az alkalmazásprogramozási felület (Application Programming Interface). Ezek távolról hívható komponensek, amelyeket anélkül tudunk használni, hogy azoknak belső működését ismernünk kellene.

3.4.1 Google Maps API

A Google 2005-ben indította el internetes térképszolgáltatását a Google Maps-et Akkoriban ez a többi webes térképhez képest újdonságnak számított az interaktív felülete miatt.

A Mapsen jelenleg a Föld 99%-a barangolható be és naponta 25 millió frissítést végeznek rajta, hogy mindig valós-idejű információkkal szolgáljon.

Ezeket a térkép szolgálgatásokat érhetjük el a Maps SDK segítségével Androidon is az alkalmazásunkban. Az API automatikusan kezeli a Google Maps-kiszolgálókhóz való hozzáférést, az adatok letöltését, a térkép-kijelzést és a térképen végzett kézmozdulatokat. API hívásokat használhatunk továbbá a markerekhez és módosíthatjuk a felhasználó rálátását a térkép egy részére. A Markerek további információkat nyújtanak a térképen való eligazodáshoz és lehetővé teszi a felhasználónak a térképpel való interakciót.

A Google Maps API használatához az Android Studioban a projekthez hozzá kell adni a Google Play szolgáltatásokat. Továbbá mind két build.gradle fileban be kell állítani a megfelelő modul elérést.

Továbbá a Maps használatához az applikációnak egy API-kulcsra van szüksége. A kulcs igénylése teljesen ingyenes és azt bármilyen olyan alkalmazás használhatja, amely az Android Maps SDK-t használja. Ez korlátlan számú felhasználót támogat. Ezt az API kulcsot a korábban már bemutatott AndroidManifestben kell eltárolni metaadatként.

A Map az acitvity\_map.xml fileban egy fragmentként van elhelyezve. Erre azért volt szükség, hogy magát a térképet tartalmazó ablakot activityként tudjuk kezelni és ne fragmentként.

A MapActivity-nek implementálni kell az OnMapReadyCallback interfészt. Erre azért van szükség, hogy az alkalmazás tudja a térkép mikor áll készen a helyes működésre és a Markerek hozzáadására.

Fontos, hogy az onCreate metódusban beállítsuk a megfelelős layout az activity számára tartalmi nézetként. Ezt a FragmentManager.findFragmentById segítségével tehetjük meg. Ezután a getMapAsync() metódussal be kell regisztrálni a térképet a adott activity-hez.

A töltőállomásokat a térképen Markerekkel jelöljük meg. Ezt pin-nek is neveznek és egy adott helyszín pontos megjelölésére alkalmas. A GoogleMap objektumhoz az addMarker() segítségével adhatunk hozzá új pint. Itt beállítjuk a pozícióját, amelyet a hosszúsági és szélességi koordinátájával adhatunk meg. Majd adhatunk neki nevet, egy rövid leírást és saját ikont is beállíthatunk rá.

Lehetőség van beállítani, hogy a térkép megnyitásakor melyik helyszínre fókuszáljon a kamera. Ezt szintén a GoogleMap objektumon lehet, az animateCamera függvény hívásával. Ahol a kívánt területet koordinátákkal adhatjuk be.

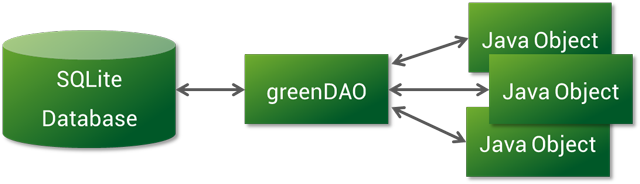


3.17 ábra. A Google Maps API logója

3.4.2 GreenDAO

A GreenDAO egy ORM hozzáférés az SQLite adatbázisokhoz, ennek az API-nak a segítségével lett létrehozva az adatbázis, illetve ezzel lehet rajta műveleteket végrehajtani. Az ORM azaz Object-Relational Mapping (magyarul objektum-relációs leképezés) egy programozási technika adatok konvertálására nem típusos rendszerek és objektumorientált programozási nyelvek között. Jelen esetben az SQL és JAVA nyelvek között. Lényegében egy virtuális objektum-adatbázist hoz létre, amit programozási nyelvekben használhatunk.

Az adatmódosításokat egy objektumorientált programozási nyelvben általában az objektumokon végezzük el, amelyek szinte soha nem skalár változók, hiszen egy objektumon belül legtöbbször több adattag is található. Azonban a legtöbb népszerű adatbázis, mint például az SQLite csak skalár értékeket fogad el, stringeket, számokat. Ennek a rendszernek az összekapcsolását végzi el a GreenDAO. Azaz a programozónak nem kell sajátkezűleg az objektumokat egyszerű értéké konvertálni, elvégzi ezt helyette az ORM eszköz.



3.18 ábra. GreenDAO kapcsolata a JAVA objektumokkal és az SQL adatbázissal

A GreenDAO teljesen nyílt forráskódú, így ingyen használható API. Ennek segítségével a leegyszerűsíthető a fejlesztés, nem kell az alacsony színtű adatbázis követelményekkel foglalkozni, így sok időt takarítható meg a fejlesztés során. Egyszerűen az API segítségével tárolhatunk, frissíthetünk, törölhetünk és lekérhetünk adatokat az SQLite adatbázisból.

Több nyílt forráskódú ORM rendszer is létezik, azonban az alábbi okok miatt került a fejlesztés során a választás mégis a GreenDAO-ra.

* Maximális teljesítmény, a leggyorsabb ORM az Android alkalmazások számára a benchmarkok alapján
* Könnyen használható és kezelhető nagy teljesítményű API
* Minimális memória felhasználás
* Kis könyvtár méretek, így build idő alacsonyan marad
* Adatbázis titkosítás, a GreenDAO támogatja az SQLCipher-t, hogy biztonságban tartsa a felhasználó adatait
* Erős közösség az esetleges segítségnyújtás számára, a GitHub-on az egyik legismertebb open source ORM API

Az egyed más néven entitás, az, amit le akarunk írni, amelynek adatait tároljuk és gyűjtjük az adatbázisban. Ilyen például az alkalmazásban a ChargerPoint, ez egy töltőállomást reprezentál a saját adataival.

Egy entitás létrehozásához a GreenDAO-ban az adott osztály egy @Entity annotációval kell ellátni. Ezután felsorolhatjuk az egyedhez tartozó adatokat, mintha egy JAVA osztálynak az adattagjai lenénnek.

Az adattagoknak külön-külön is megadhatunk tulajdonságokat egy-egy annotációval. Minden annotációt kukac jellel kell kezdeni.

Az @Id (autoincrement = true) paranccsal megszabjuk, hogy ez az adattag legyen az adott tábla ID-ja, amelyet automatikusan növelünk új eleme beszúrása esetén.

A @Property lehetővé teszi, hogy ne alapértelmezett oszlopnevet használjon az adatbázis, hanem az általunk megadottat.

A @NotNull-al jelölhetjük, hogy az adott adattag nem hagyható üresen, kötelező kitölteni az adatbázisban való beszúrásához.

Az elsődleges kulcsnak minden esetben long vagy Long típusú adattagnak kell lennie. Emellé lehetőségünk van indexeket létrehozni a táblában, ha az @Index annotációt ellátjuk az unique = true paranccsal, akkor az index minden értékének egyedinek kell lennie. Erre lehetőségünk van sima adattagoknál is, ehhez a @Unique annotációt kell használnunk.

Az adatbázis táblák a szokásos módon egymáshoz kapcsolódhatnak ONE-TO-ONE, MANY–TO–ONE, vagy MANY–TO–MANY módon. A GreenDAO-ban nem a szokásos módon kell ezeket megadni, a következőkben ezt mutatjuk be.

A @ToOne egy másik entitáshoz való kapcsolatot ír le. Ebben az esetben a GreenDAO-nak szüksége van a másik entitás ID-ja, ezért ezt az annotáció után zárójelben kell megnevezni joinProperty néven. Az egy-egy kapcsolat esetén első lazy módon tölti be az adatbázisból a kért objektumot, a későbbi hívások már azonnal visszaadják a korábban lekért adatot.

A @ToMany egy másik entitás kapcsolatot határozz meg. Ez a célentitások egy listáját jelöli meg. Ebben az referencedJoinProperty adja meg, hogy mely entitás mely idegen kulcsa alapján történjen a kapcsolat. Lehetőség van a listát egy @OrderBy-al is megjelölni. Ekkor a listát a zárójelben megadott adattag alapján és csökkenő vagy növekvő módon rendezi.

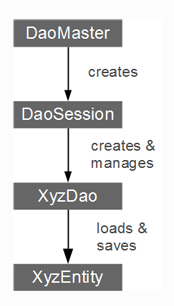
Ezek kombinációjával és a @JoinEntity segítségével van lehetőség sok-sok kapcsolat létrehozására. Ekkor a GreenDAO-nak egy kapcsoló-tábla entitást is létre kell hozni. A JoinEntityben kell megadni a kapcsoló entitás osztály nevét és a két összekapcsolandó egyed külső kulcsait.

Ha elkészítettük a szükséges entitásokat a megfelelő kapcsolatokkal, akkor Android Studio-ban a build gomra nyomva a GreenDAO minden entitásnak legenerálja a metódusait és mindegyikhez létrehoz egy Dao osztályt is. Továbbá egy DaoMaster és egy DaoSession class-t is generál.

A DaoMaster úgymond egy belépési pont a GreenDAO használatához. Ez rendelkezik az adatbázis objektummal és kezeli a DAO osztályokat. Statikus metódusokkal rendelkezik a táblák eldobásához és létrehozásához. Belső osztályai az OpenHelper és a DevOpenHelper amelyek SQLiteOpenHelper implementációk.

A DaoSession minden rendelkezésre álló DAO objektumot kezel egy adott séma számára. Ennek a segítségével lehet beilleszteni, betölteni, frissíteni és törölni az entitásokat.

Az entitások DAO filejai lekérdezéseket végeznek az adott egyeden. Rendelkezik count, loadAll és insert metódusokkal is. Ennek a segítségével történek a tényleges munk a fejlesztés során.



3.19 ábra. A GreenDAO felépítése

3.4.3 Gson

A Gson egy nyíltforráskódú JAVA könyvtár, amely JSON formátummá való konvertálást és visszakonvertálást végez. A JSON egy kis méretű, szöveg alapú szabvány, amely ember és gép által is jól olvasható.

A Gson használata nagyon egyszerű, bármilyen saját osztály képest JSON formátummá alakítani. Ehhez csak az osztály nevére van szüksége, és a konkrét adatra. Az alkalmazásban ennek a segítségével töltjük be az adatbázisban a charger.txt file-ból a töltőállomásokat.

3.4.4 Joda-Time

A dátumok kezelése az utóbbi időben JAVA-ban nehezebbé vált, hiszen több Date objektumon végezhető metódust is elavultnak minősítettek. Ezért a könnyebb fejlesztés érdekében a dátumok tárolásához a JodaTime API-t hívtuk segítségül.

A Joda-Time-al könnyen kicserélhetjük a nehézkesen használható JAVA dátum és idő osztályokat az API objetumaira. Nagyon könnyen kezelhető sok hasznos metódussal rendelkezik, mint például a getYear vagy get DayOfWeek.

Sajnos a GreenDAO csak Date objektumot képes kezelni a Joda-Time saját osztályait nem, így az adatbázisban való beszúráskor azokat át kell konvertálni Date típusúvá, majd kiolvasásuk esetén visszakonvertálni.

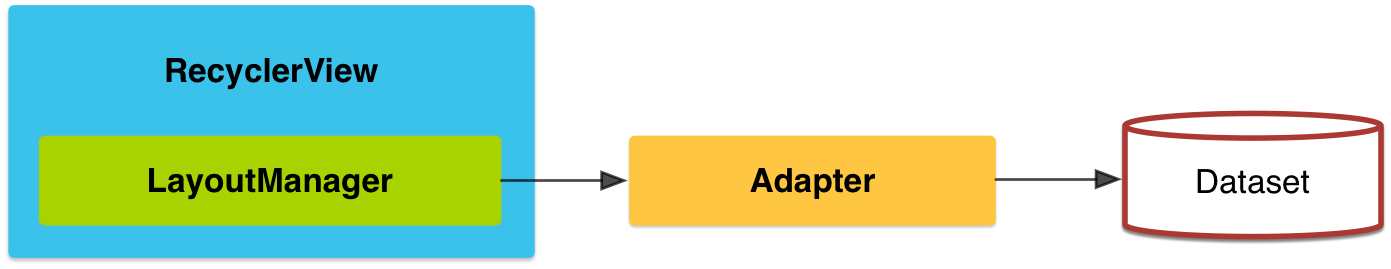
3.4.5 RecylerView

A RecyclerView is egy külső függőségként lett behúzva a build.gradle-be. Ez már a korábbi fejezetkben említett rugalmas nézet, amelyben nagy számú adatokat vagyunk képesek megjeleníteni egy korlátozott méretű képernyőn. A RecyclerView a ListView egy fejlettebb verziója, amely jobb teljesítményt és más előnyökkel is jár.

Ehhez szükségünk van minden RecyclerView esetén egy Adapter osztályra, amely felelős az adatkészlet elemeit megjelenítő nézetekért.

Fontos, hogy az Adapter osztály mellé kell egy saját list item layout XML-et írni, amely meghatározza, hogy egy elem pontosan, hogy nézzen ki.

Recycler nevet (magyarul újrahasznosítás) onnan kapta, hogy az adatok megjelenítéséhez használt nézetet el lehet helyezni egy gyorsítótárba a későbbi újra felhasználás céljából.



3.20 ábra. A RecyclerView működési struktúrája

3.4.6 Stetho

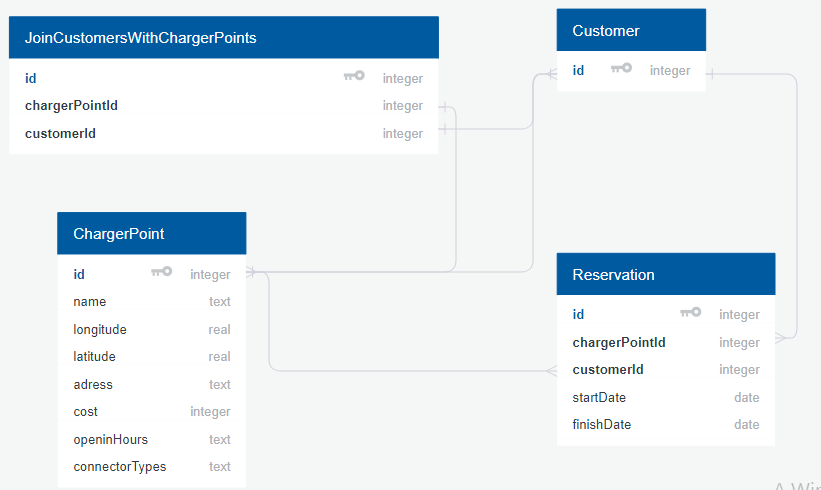
A Stetho egy kifinomult hibakereső API Android alkalmazásokhoz. Az engedélyezés után a fejlesztők hozzáférhetnek a Chrome fejlesztői eszközeihez.

Esetünkben a fejlesztés során az adatbázis helyes működésének való ellenőrzésére lett használva. Ennek segítségével az SQLite adatbázisok adatai vizuálisan láthatók.

3.5 Adatbázis

Ebben a fejezetben az adatbázis felépítését, entitásokat és a közöttük lévő kapcsolatokat mutatjuk be.

Néhány típus nem tárolható az SQLite adatbázisban, ezért át kell konvertálni. Ilyen például a Long, ami integerré konvertálódik, vagy a double ami pedig real típusúvá.



3.21 ábra. Az adatbázis terv diagramja

3.5.1 ChargerPoint entitás

Ez az egyed egy töltőállomást reprezentál az összes adatával. Két kapcsolata is van az egyik egy sok-sok kapcsolat a Customer táblával, ez a kedvencek közé tételhez szükséges, míg a másik a Reservation táblával egy egy-sok kapcsolat.

* id: a tábla elsődleges kulcsa, amely Long típusú
* name: töltő nevét írja le, String típusú adattag
* longitude: hosszúsági fok, double típusú
* latitude: széleségi fok, double típusú
* adress: a töltő címe, string típusú
* cost: a töltő költsége, integer típusú
* openingHours: a töltő nyitvatartása, string típusú
* connectorTypes: a töltőnél található konnektor típusók, string típusú

3.5.2 Customer

Ez egy felhasználót tárol, jelenleg csak egy oszlopa van. Későbbi továbbfejlesztési lehetőség egy teljes adatlap készítése a felhasználó számára. Egy-sok kapcsolata van a Reservation táblával és sok-sok a ChargerPoint-tal.

* id: elsődleges kulcs, Long típúsú

3.5.3 Reservation

Egy foglalást reprezentáló entitás. Sok-egy kapcsolata van a Customer táblával és sok-egy a ChargerPoint-tal.

* id: elsődleges kulcs, Long típusú
* chargerPointId: idegen kulcs a töltőállomás számára, Long típusú
* customerId: idegen kulcs a felhasználó számára, Long típusú
* startDate: a foglalás kezdetének ideje, Date típusú
* finishDate: a foglalás végének ideje, Date típusú

3.6 Valós idejűség

Az szakdolgozat nevében is megtalálható, hogy az applikációnak bizonyos rsézei valós időben történnek kiértékelésre.

A valós idejű alkalmazások lényege, hogy bizonyos funkciói olyan időkereten belül változik, amelyet a felhasználó azonnal és aktuálisan érzékel. A késleltetésnek kisebbnek kell lennie, mint egy meghatározott érték, amelyet általában másodpercben mérnek.

Az eHubSharing alkalmazásban a foglaltsági rendszerben találhatók valós idejű megoldások. A felhasználó minden töltőállomás adatlapján nyomon követheti, hogy az adott töltő éppen foglalt-e. Amennyiben meg van nyitva a töltő adatlapja és lefoglalásra kerül az, a foglaltságat jelző kártya azonnal pirosra vált.

Ha egy korábbi foglalás lesz éppen aktív, akkor a felhasználó maximum öt másodpercen beül értesül, hogy éppen foglalt lett, vagy felszabadult az adott töltő. Ezt az értéket természetesen le lehetne csökkenteni, de mivel egy autó feltöltése egy hosszabb folyamat, ezért arányaiban megfelelő az öt másodperc.

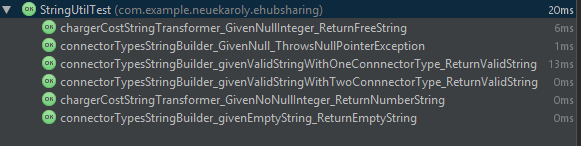
3.22 ábra. Bal oldalon éppen szabad, jobb oldalt foglalt a töltőállomás

3.7 Tesztelés

Egy androidos alkalmazást többféle módon lehetséges tesztelni, mi most, amely függvényeken lehetséges az alap JUNIT teszteket használjuk, illetve az alkalmazás helyes működéséének ellenőrzését manuálisan ellenőrízük. Továbbá android tesztelés esetén lehetőség van egy automatizáló rendszert is használni például az Espresso-t, de ezt mi manuális tesztelésre cseréljük le, hiszen így könnyebben észrevehetők az esetleges hibák

3.7.1 JUNIT tesztek

A StringUtils-ban található metódusok jól tesztelhetők JUNIT technikával. A két metódusra összesen hét tesztesetet írtunk, mely mindegyike helyes futással végződött. A tesztelés során felhasználtunk hibás bemeneti eseteket is. A tesztesetek eredményei a következő ábrán láthatók.



3.23 ábra. StringUtils osztály JUNIT tesztjei

3.7.2 Manuális tesztelés az UI számára

A manuális tesztelést androidos készülék hiányában Genymotion emulátorral végeztem. Ennek segítségével egy teljes értékű Samsung Galaxy S6 mobiltelefont szimuláltunk, amelyen a 5.0-as, Android operációs rendszer található. A képernyő felbontása 1440x2560 pixel.

Az üdvözlő képernyő tesztelését többszöri egymásután való elindítással végeztük. Az applikáció újra-telepítése esetén is sikeresen betölti a szükséges adatokat az adatbázisba.

Kikapcsolt internet hozzáférés esetén majdnem teljes értékűen használható az alkalmazás, csak a térképen az utcákat nem jeleníti meg megfelelően, de hibát nem dob és nem is fagy le.

A Google MAP tesztelése során figyeltünk a kicsinyítő, nagyító gomb helyes működésére, illetve a térképen való navigálásra. Az összes marker végig-nyomkodása során, helyesen megnyíltak a hozzájuk tartozó töltőállomás-adatlapok. A markerre való kattintás esetén sikeresen megnyílik a Map applikáció az útvonaltervezéshez.

Kedvencek közé adás esetén a FloatingActionButton egyből eltűnik és a Favourites ablak megnyitása esetén egyből megtalálható a kedvencek között az adott töltőállomás. Onnan való kitörlése esetén újra megjelenik a hozzáadó gomb az adatlapján.

A töltőállomás adatlapján helyesen, jól strukturáltan jelenek meg az adatok egymás alatt minden alkalommal.

A töltő foglaltságát jelző kártya valós időben átvált pirosra, ha foglalttá válik. Akkor is ha éppen a jelenlegi időpontra végezzük a foglalást. A foglalás törlése esetén egyből szabaddá válik az állomás.

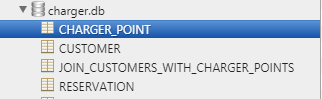
A foglalási rendszer helyesen működik, régebbi időpontokat nem lehetséges kiválasztani, az adatok helyes megadása esetén sikeresen foglalható a töltő. Egyből megjelenik a Reservations ablakban, ahonnan sikeresen törölni is lehet azt az adatbázisból. Ugyanarra az időpontra nem lehetséges két foglalást készíteni, ilyenkor a rendszer felhívja a figyelmünket, hogy sikertelen a foglalás és adjunk meg egy másik időintervallumot.

A foglalás hosszának választása sikeresen működik, fél és egy óra esetén is.

A töltőállomás kereső pontosan és gyorsan működik. Amint új betűt gépelünk vagy törlünk azonnal újra szűri a keresési eredményeket. Semelyik töltőre nem illeszkedő szöveg begépelése esetén üres ablak jelenik meg.

3.7.3 Adatbázis tesztelés

Az adatbázis helyes működését a korábban már említett Stetho keretrendszerrel végeztem. Ezen nyomon-követhető az adatbázisban való beszúrások és törlések.



3.24 ábra. Stetho használata során bármelyik táblát megtekinthetjük az adatbázisból

3.8 Verziókezelés

A folyamatos verziókezelő használata megkönnyíti és átláthatóvá teszi a fejlesztés menetét, illetve ennek segítségével elkerülhető az esetleges adatvesztés.

Több verziókezelő rendszer is ismert a programozás világában, de ezek közül a legismertebb és az egyik legjobban használhatóra esett a választásom, a Git rendszere. A Git egy nyílt forráskodú, elosztott verziókezelő szoftver, amely a sebességre helyezi a hangsúlyt. Minden Git munkamásolat, ez esetben az eHubSharing alkalmazás is, egy teljes értékű úgynevezett repository, amely rendelkezik egy verziótörténettel, ami nem függ a hálózat eléréstől vagy egy központi szervertől.

A Git használatához először le kell tölteni a hivatalos weboldaláról a hozzá tartozó szoftvert majd telepíteni. Ezekután ki kell választani a számunkra legszimpatikusabb online verzió-kezelő tárhelyet, ez az én esetemben a GitHub volt.

GitHub-on létrehozhatunk repositorykat, amelyek egy-egy teljes projektnek felelnek meg. Miután ezt a gépünkre klónoztuk, kezdődhet is a fejlesztés menete. Minden logikailag egybefüggő egész programrész esetén ajánlott elmenteni a munkánkat lokálisan és online is. Ennek menete, hogy először a git add –all paranccsal hozzáadjuk a verziókezelőhöz az összes módosítást, majd a git commit -m-el egy rövid, lényegre törő üzenetben összefoglaljuk, a módosítások lényegét. E két művelet után lokálisan sikerült elmenteni, a módosításokat. Ahhoz, hogy GitHubon online is megjelenjenek a módosítások, adjuk ki a git push origin <branch neve> parancsot. Amennyiben nem akarunk brancheket használni, helyettesítjük azt a master vagy head kulcsszót helyette. A összes parancsot a parancssorban kell kiadnunk a fejlesztés főkönyvtárában.

A branchek úgynevezett fejlesztői ágak. Ezt akkor szokás használni, ha egy új funkciót szeretnénk implementálni és ezt nem a fő ágban (master) szeretnénk megtenni.

Az eHubSharing applikációhoz tartozó kódok és commit-ok megtekinthetőek a <https://github.com/neuekaroly/ehubsharing> weboldalon.

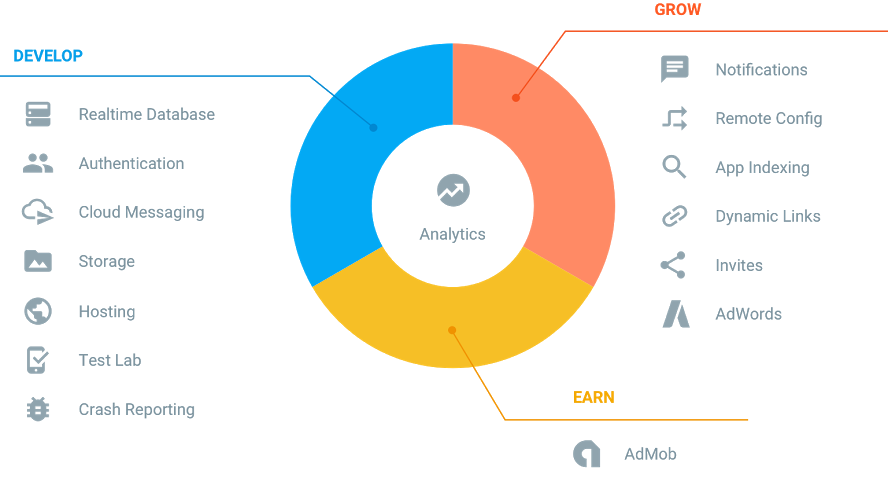
3.9 Továbbfejlesztési lehetőségek

3.9.1 Google Firebase

Ahhoz, hogy több felhasználó is tudja egyszerre használni az applikációt egy közös szerverre és backendre lenne szükség. Ehhez nyújtana segítséget a Google Firebase szolgáltatása, amely 100 felhasználóig és 1GB tárhelyig ingyenes használható. Ennek segítségével a felhasználók bejelentkezhetnek, így más készülékeken is nyomon tudják követni a foglalásukat, értesítéseket tudnának kapni időben bizonyos tartalmakról.

A Firebase használata leegyszerűsíti a fejlesztés folyamatát, enélkül akár több hétig tartó, komplex több résztvevős backend-fejlesztés venné kezdetét.

A Firebase nagyon sok szolgáltatással könnyíti meg a fejlesztés menetét több platformon is. Ezeket az alábbi bára mutatja be.



3.25 ábra. Firebase funkciói

3.9.2 Fragmentek használata

A szebb design érdekében az applikáció kinézetét újra lehetne tervezni. A négy alap activity helyett lehetne fragmenteket használni. A fragmentre úgy kell gondolni, mint egy moduláris szakaszra egy activityn belül, amely saját életciklussal és saját bemeneti eseményekkel rendelkezik. A fragmenteket mindig egy létező acitvitybe kell ágyazni. Ezek jobb animációkkal rendelkeznek, ezáltal szebbé tehetik az alkalmazás UI részét.

3.9.3 Teljes adatlap

A felhasználónak lehetősége lenne a főbb adatait megadni. Ezek közé tartozna, hogy milyen típusú töltőállomásokat keres, így az alkalmazás alapból csak a neki megfelelőket listázná ki. Továbbá lehetőség lenne Facebook és Google profillal is regisztrálni, így megkönnyítve a felhasználó számára ezt a procedúrát.