**Budapesti Műszaki és Gazdaságtudományi Egyetem**
Villamosmérnöki és Informatikai Kar

# Community Parking

Közösségi parkolást segítő portál

Szoftverarchitektúrák házi feladat

Készítették: Fodor Árpád, Gyönki Bendegúz

KONZULENS
Gazdi László

BUDAPEST, 2020

# Tartalom

# 1. Követelményspecifikáció

## 1.1. Feladatkiírás

A hallgatók feladata egy olyan térkép alapú online rendszer kifejlesztése, amelyik lehetővé teszi felhasználóinak, hogy megosszák az általuk ismert ingyenes/előnyös parkolási lehetőségeket a többi felhasználóval. A rendszer legyen elérhető mobil készülékekről. Az alkalmazás tegye lehetővé fényképek feltöltését és a cím alapú keresést.

## 1.2. A fejlesztői csapat

| **Csapattag neve** | **Neptun-kód** | **E-mail cím** |
|---|---|---|
| Gyönki Bendegúz | IZZT5E | gyonkibendeguz@gmail.com |
| Fodor Árpád | S4AZIE | arpadfodor01@gmail.com |

A csapatban a következőképpen osztottuk fel a feladatokat:

- **Bendegúz** a backend (szerveralkalmazás) fejlesztésével foglalkozik
- **Árpád** a frontend (Android kliens) fejlesztését végzi

## 1.3. Részletes feladatleírás

Célunk a projekt keretében egy olyan rendszer készítése, amellyel a felhasználók bárhol képesek parkolási lehetőségek bejelentésére és megtekintésére. Ezáltal adott területek monitorozására nyílik lehetőség, ami megkönnyítheti a városban autóval járók mindennapjait.

Az alábbiakban ismertetjük az Android operációs rendszerre készített kliens és a szerveroldali alkalmazás tervezett funkcióit.

### 1.3.1. Android alkalmazás

Egy bejelentéshez a GPS koordináták és a szükséges adatok megadásán felül (pl. fizetős/ingyenes parkolás) lehetőség van képek feltöltésére is, amelyek révén a többi felhasználó könnyebben találhatja meg az adott helyet. A képek a gyors bejelentés céljából az alkalmazásban készülhetnek, de lehetőség van eszközön tárolt fotók feltöltésére is.

A felhasználók többféle módon böngészhetnek a bejelentések között. Egyrészt egy interaktív térképen vizuálisan láthatják a bejelentéseket, illetve lehetőség van cím alapján történő keresésre is. A kiválasztott parkolóhely részletes adatai (pozíció, típus, bejelentés ideje, kép) megtekinthetők, illetve navigáció indítása is lehetséges az adott helyre.

Egy felhasználó az alábbi műveleteket hajthatja végre egy potenciális parkolóhely esetén:

- Megadja a tulajdonságait (pl. fizetős/ingyenes)
- Képet készít hozzá
- Bejelenti

Egy már bejelentett parkolóhelyen a felhasználók az alábbi műveleteket végezhetik:

- Lefoglalás (egy felhasználó jelzi, hogy az adott helyre tart, oda fog parkolni)
- Foglalás levétele
- Módosítás (pl. egy felhasználó jelzi, hogy az adott parkolóhelyen már áll valaki; vagy képet fűz hozzá)

### 1.3.2. Szerveralkalmazás

A szerveroldali alkalmazás az alábbi, parkolóhelyekkel kapcsolatos műveleteket támogatja:

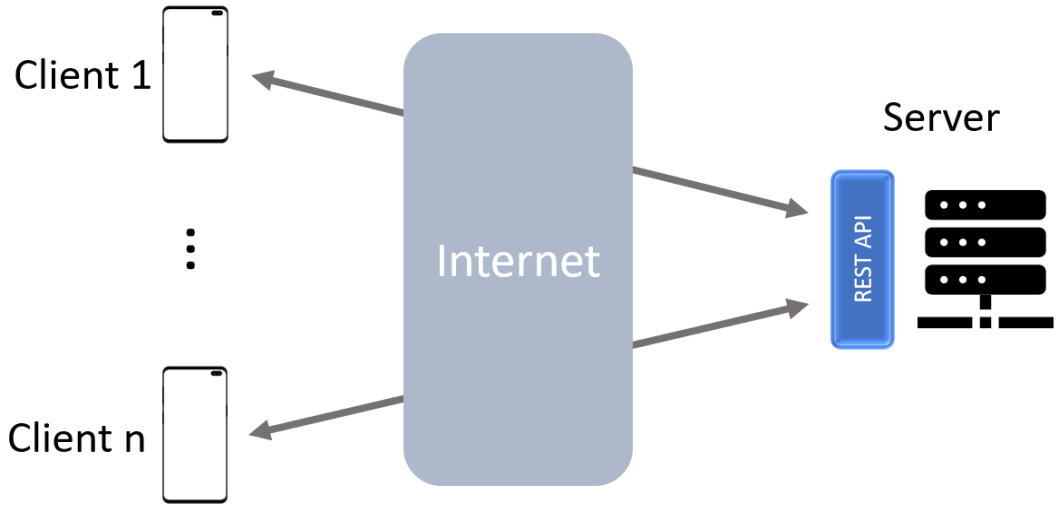
- Új parkolóhely hozzáadása az adatbázishoz
- Meglévő parkolóhely adatainak módosítása az adatbázisban
    - Parkolóhely lefoglalása
    - Foglalás levétele
- Meglévő parkolóhely törlése az adatbázisból
- Parkolóhely keresése

- GPS koordináták alapján és a kliens által megadott sugarú körben

A szerveroldali alkalmazás felel a felhasználók nyilvántartásáért is. A szerver a felhasználók alábbi adatait tárolja: felhasználónév, email cím, jelszó (titkosított formában). A szerver lehetővé teszi új felhasználó regisztrálását, illetve a felhasználói adatok módosítását.

## 1.4. Architektúra

A rendszert kliens-szerver architektúra alapján tervezzük megvalósítani. A kliensek Android alkalmazások, a szerver pedig egy REST API-t biztosít a kliensek számára. A bejelentett parkolóhelyek, képek, koordináták a szerveren tárolódnak, a kliensek ide tudnak bejelenteni új dolgokat, vagy lekérdezni az aktuális állapotot. A kommunikáció HTTPS protokollon keresztül történik, a felhasználók azonosítása pedig HTTP Basic authentikációval történik.



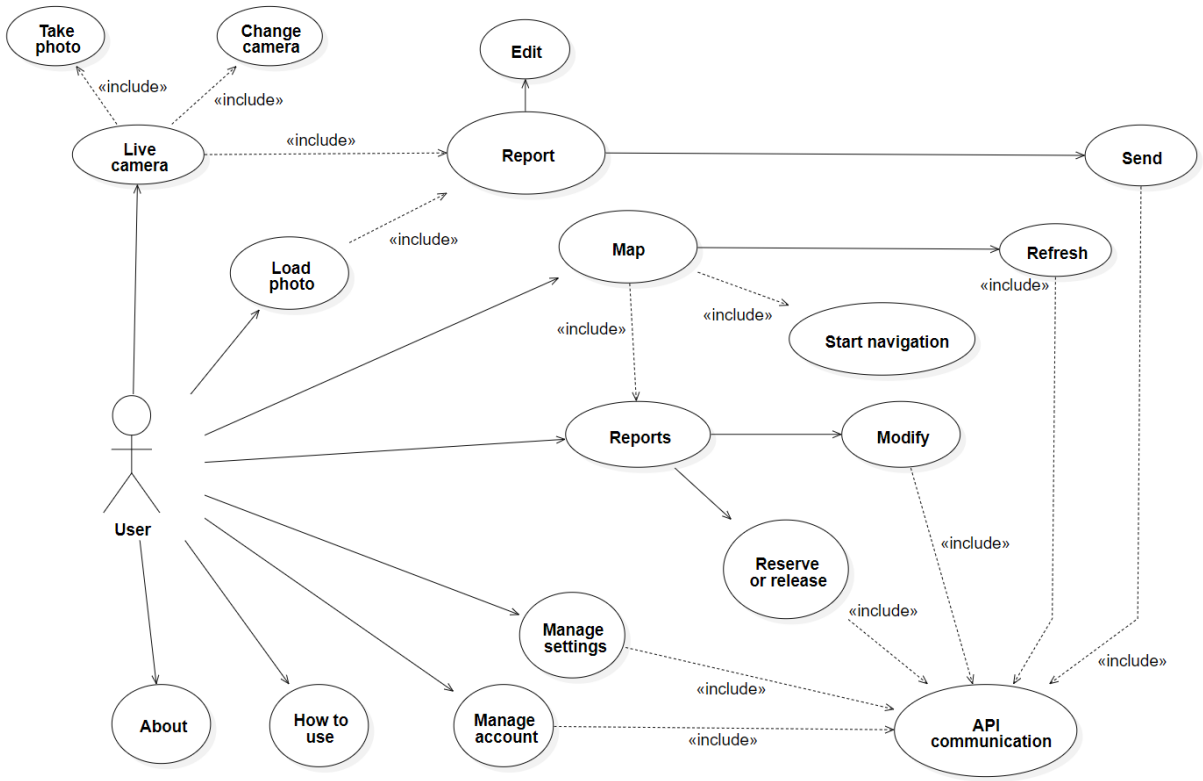

ábra 1: Kliens-szerver architektúra

## 1.5. Technikai paraméterek

A rendszer kliensoldali része Android platformra készül. Az okostelefonok napjainkra széles körben elterjedtek, a legdominánsabb operációs rendszer pedig jelenleg az Android, ezért ezt a platformot célozva potenciálisan széles lehet a felhasználók köre. A fejlesztés Kotlin nyelven történik. Az alkalmazás specifikus perzisztens adatok SQLite alapú Room adatbázisban kerülnek tárolásra. A kamerakezelés a CameraX API segítségével kerül megvalósításra. A térkép megjelenítéséhez a Google Maps API segít, míg a szerverrel történő hálózati kommunikációhoz a Retrofit kerül felhasználásra.

A szerveralkalmazás fejlesztése is Kotlin nyelven történik, a Ktor keretrendszer felhasználásával. A Ktor egy aszinkron keretrendszer mikroszolgáltatások és webalkalmazások fejlesztéséhez. A szerver az adattároláshoz egy PostgreSQL adatbázist használ, az adatbázisműveletek pedig a Jetbrains által készített Exposed ORM keretrendszer használatával kerülnek megvalósításra.

## 1.6. Use-case diagram




ábra 2: Android app use-case

# 2. Rendszerterv

## 2.1. Android alkalmazás

A kliens egy Android alkalmazás. Fő funkciója a parkolóhelyek bejelentése, azok megtekintése, és felhasználói engedélyhez kötött módosítása (lefoglalás, törlés, megjegyzés hozzáfűzése). Az eszközön tárolt kép mellett lehetséges az élő kameraképen látható parkolóhely bejelentése is. Az élő kameraképen lehetséges a nagyítás és a fókusz változtatása is. Az egyes helyek az eszközön is eltárolásra kerülnek, hogy internetkiesés esetén is használható maradjon az alkalmazás. Frissítés/bejelentés/törlés azonban csak az API elfogadásával történhet, megelőzve az inkonzisztens állapotot.

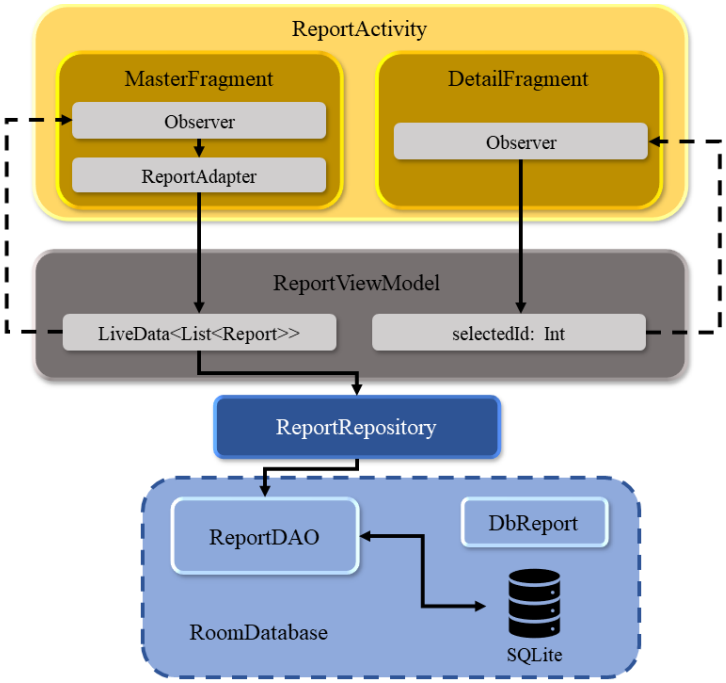
### 2.1.1. Architektúra

A Model View ViewModel (MVVM) felhasználói felület tervezési minta lett felhasználva az Android alkalmazás készítése során. Ez egy eseményvezérelt modell, amelyet a Microsoft talált ki az adatkötési képességek kihasználására. Az MVVM-ben a View UI felületleíró kódot tartalmaz, általában deklaratív módon (XML, XAML, HTML). A kapcsolat a ViewModellel explicit adatkötés segítségével valósul meg. Ezért kevesebb klasszikus kódolási feladat van a View-ban, és az üzleti logika komponensek könnyedén elválaszthatók.


*1. ábra: MVVM komponensek és azok kapcsolatai*

Az alkalmazás Model rétegében további alkomponensek találhatók. A magasszintű hierarchiát egy parkolóhely (Report) adatbázisban való frissítésén keresztül, az alábbiakban szemléltetem. A ReportActivity DetailFragment-je jeleníti meg az adatokat, ahol például lefoglalható egy parkolóhely. Egy Report elem listában kerül tárolásra a ViewModel-ben, ami LiveData objektumba van csomagolva (ez a UI elemek számára megfigyelhető, mert adatkötésben használható). Mikor a felhasználó frissít egy elemet a DetailFragment-ben, az adott elem a listában frissítésre kerül, mikor a Model a műveletet elfogadja. Ekkor a változás a MasterFragment-ben is azonnal megjelenik.
A ReportViewModel változás esetén értesíti a Model-ben található RepositoryService-t. Ebben található a ReportRepository, ami a Report-okhoz tartozó műveleteket (adatbázisba írás, API hívások kezelése) rejti el a külvilág számára. Egy Report változása esetén először az APIService-en keresztül a szerver a változásról értesítve lesz. Ha a backend ezt elfogadta, a Report az adatbázisban tárolt formátumra alakításra kerül (DbReport), és a ReportDAO (data access object) segítségével perzisztálva lesz. Így van biztosítva, hogy csak az kerül elmentésre, ami az API-n is érvényre jut, megelőzve az

inkonzisztenciát. A Model-ben történő műveletek (API kommunikáció, adatbázis tranzakciók) háttérszálon futnak, így a felhasználói felület nem fagy be, érvényre jutáskor viszont az adatkötés miatt azonnal frissítésre kerül.

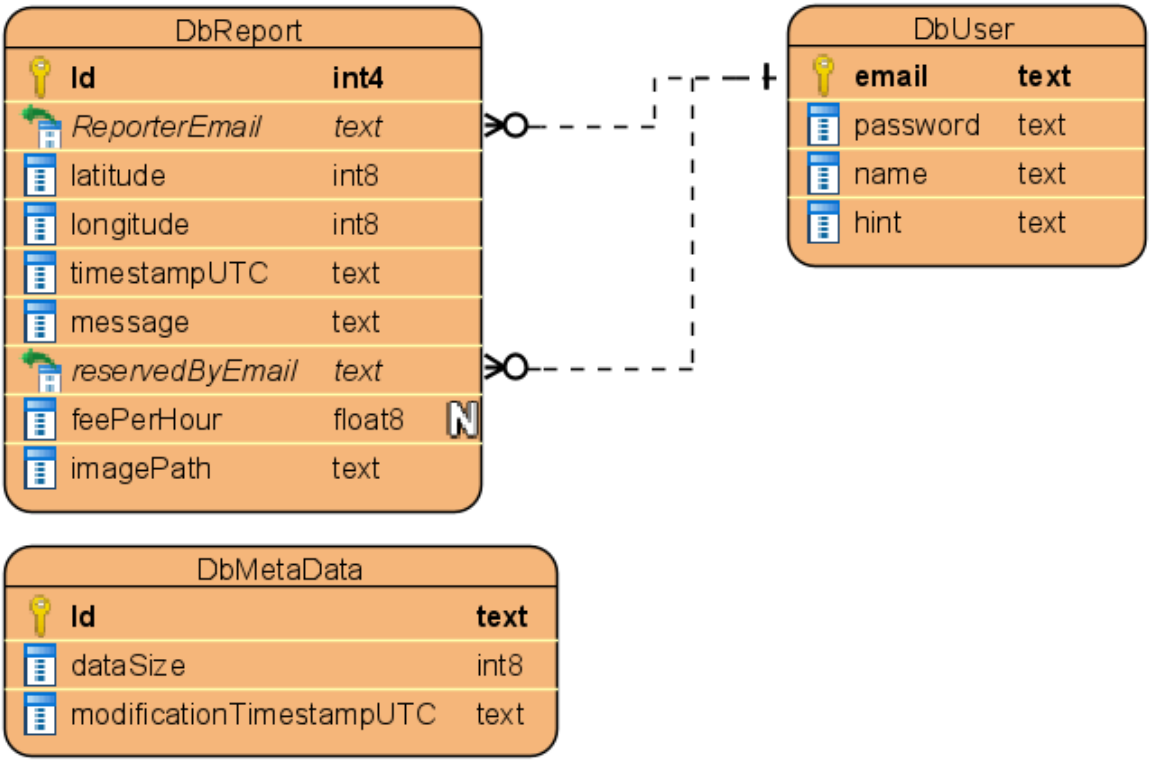



*2. ábra: parkolóhely adatok adatbázisba írásának folyamata az egyes rétegeken keresztül*

A parkolóhelyek adatai mellett az alkalmazás tárolja a felhasználó adatait is, amelyekkel például az automatikus bejelentkezés, illetve a fiókadatok frissítése valósíthatók meg. Ezen kívül az API-tól lekért metaadatok is eltárolásra kerülnek (időbélyeg, parkolóhelyek száma), amit minden API hívás előtt ellenőriz az alkalmazás. Ha az API által lekért metaadatok alapján annak adatai változtak az app legutóbbi frissítése óta, adatletöltés indul, ha azonban nincs változás, nincs hálózati forgalom sem.

### 2.1.2. Perzisztens adattárolás

Az alkalmazás tartalmaz egy SQLite alapú adatbázist, ami a Room könyvtár segítségével lett kialakítva. Hogy korlátozott módon internetkiesés esetén is használható legyen az app (parkolóhelyek megtekintése, térképen böngészés) a bejelentések DbReport entitásokban vannak az eszközön tárolva. A DbMetadata arra szolgál, hogy nyilvántartsa, mikor történt az utolsó lekérés. Ha az API válaszában az adat legutóbbi módosítás lekérésekor egy újabb UTC timestamp szerepel, mint mikor a kliens legutóbb lekérte őket, frissít. Ha azonban a két időbélyeg megegyezik, felesleges a lekérés, mert a lokális SQLite adatbázisban már naprakész adatok szerepelnek. A DbUser a felhasználói adatokat tartalmazza, ami bejelentkezés esetén van használva. Lehetőség van megadni, hogy az alkalmazás megjegyezzen egy felhasználót, így automatikusan be tudjon lépni.




*3. ábra: perzisztens módon tárolt adatok modellje*

## 2.2. Szerver alkalmazás