**Eötvös Loránd Tudományegyetem**

**Informatikai Kar**

**Programtervező Informatikus szak**

**Parkoló helyeket nyilvántartó rendszer**

**Témavezető:**

Dr. Nikovtis Tibor

adjunktus, Ph.D.

**Szerző:**

Hováth Kristóf

Programtervező informatikus BSc.

Budapest, 2020

Tartalomjegyzék

[1 Bevezetés 4](#_Toc40964161)

[2 Felhasználói dokumentáció 5](#_Toc40964162)

[2.1 Telepítés 5](#_Toc40964163)

[2.2 Webes felület használata. 5](#_Toc40964164)

[2.2.1 Bejelentkezési képernyő. 6](#_Toc40964165)

[2.2.2 Parkolóházak listáját tartalmazó képernyő 7](#_Toc40964166)

[2.2.3 Parkolóház adatait tartalmazó képernyő 8](#_Toc40964167)

[2.2.4 Parkoló adatait tartalmazó képernyő 9](#_Toc40964168)

[2.2.5 Saját adatok 10](#_Toc40964169)

[2.2.6 Nyilvántartás 12](#_Toc40964170)

[2.2.7 Napló 13](#_Toc40964171)

[2.3 Mobil alkalmazás 14](#_Toc40964172)

[2.3.1 Bejelentkezés 14](#_Toc40964173)

[2.3.2 Parkolóházak listáját tartalmazó képernyő 15](#_Toc40964174)

[2.3.3 Saját adatok képernyő 16](#_Toc40964175)

[3 Fejlesztői dokumentáció 17](#_Toc40964176)

[3.1 Architektúra 17](#_Toc40964177)

[3.2 Háttérszolgáltatás 17](#_Toc40964178)

[3.2.1 Maven 18](#_Toc40964179)

[3.2.2 Adatbázis 19](#_Toc40964180)

[3.2.3 Modellek 20](#_Toc40964181)

[3.2.4 Repository 26](#_Toc40964182)

[3.2.5 Servicek 26](#_Toc40964183)

[3.2.6 Kontrollerek 26](#_Toc40964184)

[3.2.7 Autentikáció 27](#_Toc40964185)

[3.2.8 Tesztelés 29](#_Toc40964186)

[3.3 Webes kliens 29](#_Toc40964187)

[3.3.1 Futtatás 30](#_Toc40964188)

[3.3.2 Függőségek 30](#_Toc40964189)

[4 Összefoglalás 31](#_Toc40964190)

[5 További fejlesztések 32](#_Toc40964191)

# Bevezetés

Bevezetés. Parkoló rendszerekre szükség van.

# Felhasználói dokumentáció

A Parkoló Tároló egy parkoló helyeket nyilvántartó és kezelő rendszer, amely a felhasználóknak segít felmérni az egyes parkolóházakban található szabad parkolóhelyeket, segít megtalálni hova lehet beparkolni és megjegyzi, hogy hova parkoltunk. Minden felhasználó kezelhet legfeljebb öt autót, amikkel beparkolhat, illetve kiparkolhat a nyilvántartott parkolóhelyekből.

A rendszert két féle felhasználók használhatják. Az egyszerű felhasználók, akik hozzáférnek a parkolókkal kapcsolatos adatokhoz és használhatják a parkolókat, vagyis a saját autóikkal ki- és beparkolhatnak, és az adminisztrátor jogkörrel rendelkező felhasználók, akik ezeken felül hozzáférnek egy naplóhoz, amin keresztül a rendszerben történő eseményeket visszanézhetik, más felhasználók nevében cselekedhetnek, valamint szerkeszthetik is a rendszerben lévő adatokat. Hozzáadhatnak új parkolóhelyeket, törölhetnek parkolóházakat, megváltoztathatják azok neveit stb.   
(Ezen kívül van még egy féle felhasználó, aki csak kis mértékben tér el az adminisztrátoroktól. Ez a fajta jogkör később kerül kifejtésre a Nyilvántartás felület részletezésénél.)

A rendszerrel való interakcióra egy webes, böngészőből elérhető felület, valamint egy mobil applikáció nyújt lehetőséget.

A következőkben a rendszer használatának részletes leírása kerül tárgyalásra.

## Telepítés

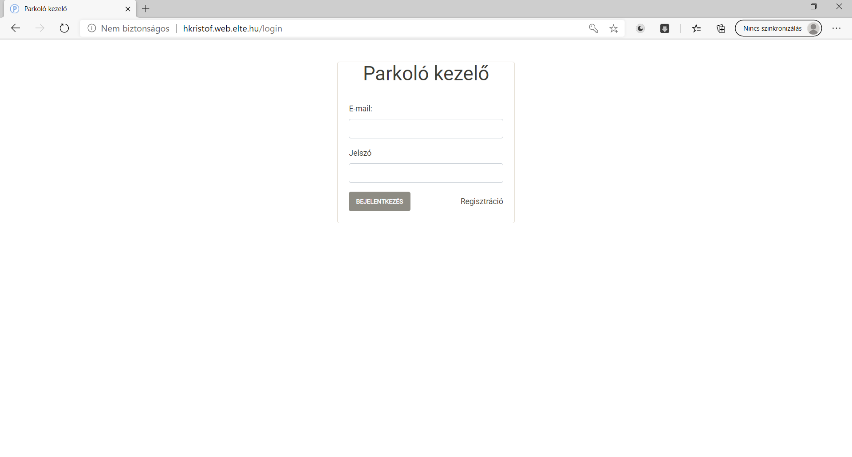
Telepítési útmutató…

## Webes felület használata.

A böngészőben elérhető alkalmazás elsősorban az adminisztrátoroknak szól. Csak itt kérdezhetőek le a naplóbejegyzések, és innen lehet más felhasználók nevében tevékenykedni.

A következőkben menüpontról menüpontra kerül részletes kifejtésre melyik funkció mire alkalmas és hogyan érhető el.

### Bejelentkezési képernyő.

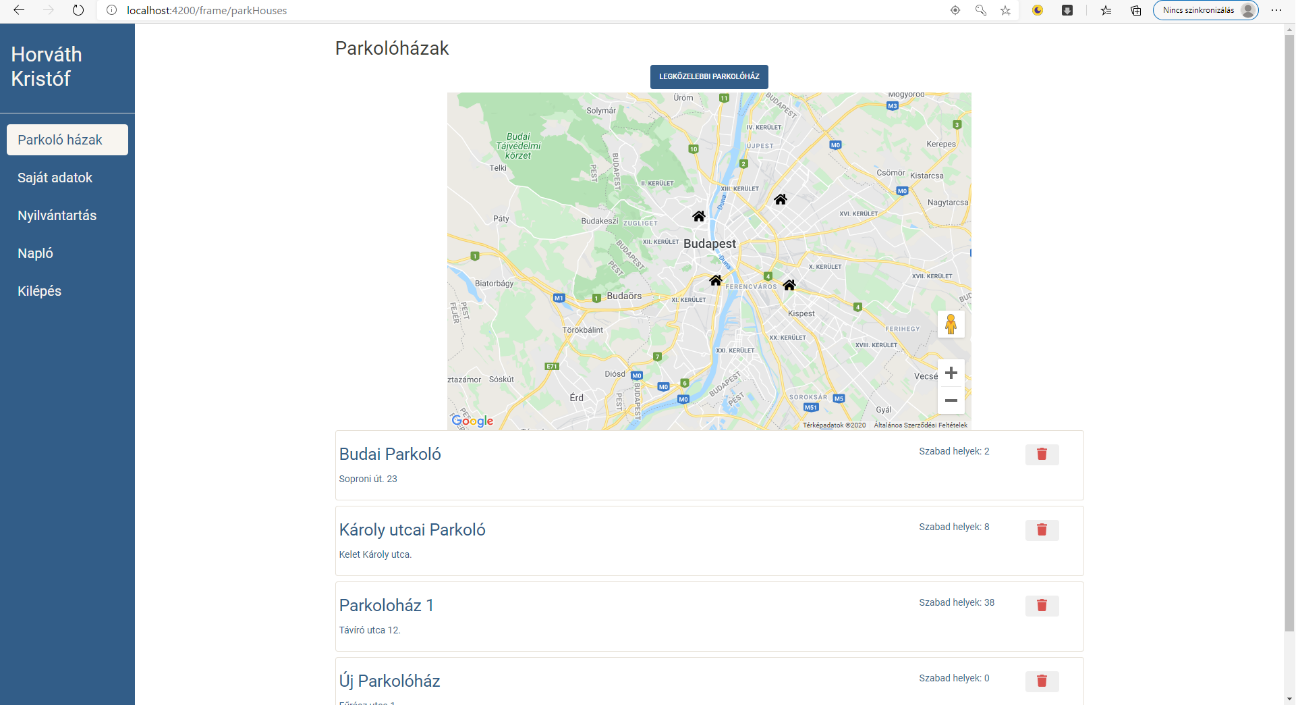


1. ábra Bejelentkező képernyő

Az oldalt megnyitva legelőször egy hagyományos bejelentkező felület jelenik meg. A bejelentkező űrlap egy email címet és egy jelszót kér a felhasználótól az azonosításhoz. Ha a felhasználó még nem regisztrált be a rendszerbe akkor a jobb alul lévő „Regisztáció” feliratú gombra kattintva az űrlap ált alakul a regisztrációs műveletet elősegítő űrlappá. Az itt megjelenő mezők segítségével be lehet vinni vezeték- illetve keresztnevet, emailt és jelszót (amit kétszer kell beírni a két jelszó mezőbe). Ha mind a két jelszó mezőben szereplő szöveg megegyezik akkor a „Küld” gombra kattintva rögzülnek a bevitt adatok a rendszerben, majd az űrlap ismét a bejelentkezéshez használatos formát veszi fel. Itt ha a felhasználó egy létező email cím és a hozzá tartozó érvényes jelszót megadva rányom a „Bejelentkezés” gombra akkor a rendszer belépteti és átirányítja az alkalmazás kezelőfelületére. Ellenkező esetben piros hiba üzenettel tájékoztatja az alkalmazás a felhasználót, hogy nem sikerült belépni.

Bejelentkezés után függ az adott felhasználó jogaitól, hogy mi jelenik meg. Bizonyos funkciók le vannak tiltva az egyszerű felhasználók elől. Ezek mindig az adott rész tárgyalásánál fognak kifejtésre kerülni.

### Parkolóházak listáját tartalmazó képernyő



*2. ábra Parkolóház lista*

Minden felhasználó esetében a kezdő oldal a jelenleg a rendszerben tárolt parkolóházak listája, valamint egy térkép, ami megjeleníti a parkolóházakat földrajzi elhelyezkedésük szerint. Az adminisztrációs jogkörrel rendelkező felhasználók itt tudnak új parkolóházat felvenni a rendszerbe a lista alatt található plus jelet ábrázoló gombbal. A gombra rákattintva megjelenik egy felugró ablak az új parkolóház létrehozásához szükséges űrlappal. Ha kitöltésre került az összes szükséges mező akkor a „Ok” gomb megnyomásával eltűnik a felugró ablak és a listában megjelenik az új parkolóház.   
Az adminisztrátor joggal bíró felhasználók számára elérhető minden parkolóház listaelemének a végén egy szemetest ábrázoló gomb. Ezzel értelemszerűen el lehet távolítani az adott parkolóházat a rendszerből, annak minden parkolójával együtt. A törlés gombra kattintva megjelenik egy felugró ablak, ami rákérdez meg egyszer, hogy biztosan ki akarja e törölni a felhasználó a parkolóházat. Igenre kattintva eltűnik az ablak, valamint a parkolóház a listából.

Az összes felhasználó képes rákattintani az egyes parkolóházakra ezzel egy új oldalra lépve az adott parkolóház részletesebb adataival.   
Ha a térképen található „ház” ikonra kattintanak akkor egy kis felugró ablak jelenik meg a parkolóház nevével és címével. Ha dupla kattintás történik valamelyik ikonra akkor ugyanúgy átnavigál a rendszer az adott parkolóház részletesebb adatait megjelenítő oldalra.  
A térkép felett található a „Legközelebbi parkolóház” feliratú gomb, melynek megnyomására a térképen felugrik a felhasználó eszközének helyadatai szerint a legközelebbi parkolóház. (A helyadatok nem mindig érhetőek el. Vagy nem támogatja az eszköz ezt a funkciót vagy a felhasználó is letilthatja. Ebben az esetben a le van tiltva, és a legközelebbi parkolóház kiválasztásának funkciója nem érhető el.)

### Parkolóház adatait tartalmazó képernyő

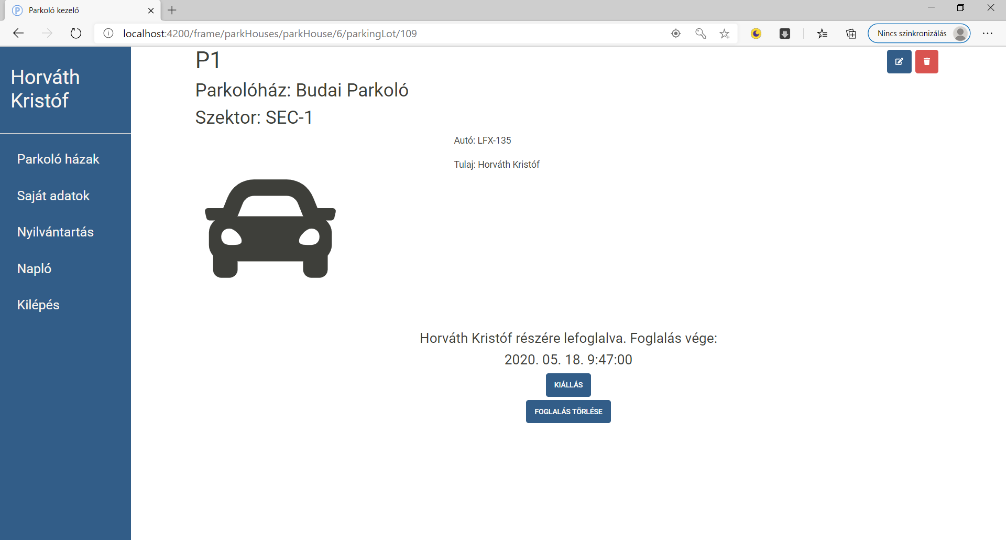


*3. ábra: Parkolóház adatai képernyő*

Ezen a felületen a kiválasztott parkolóház adatai találhatók. A felső részen látható a parkolóház neve, címe, valamint a benne megtalálható elemetek számozásának terjedelme.   
Egy kördiagram jelzi, hogy menyire telített a parkolóház.  
Adminisztrációs jogkörrel rendelkező felhasználóknak itt lehetőségük van szerkeszteni a parkolóház nevét, címét a jobb felül található ceruza ikonnal ellátott gomb segítségével. A gomb megnyomásával felugró ablakban megjelenik az űrlap, amin keresztül megváltoztathatóak az adatok. A szerkesztés gomb mellett található egy törlés gomb, aminek segítségével innen is törölhető a parkolóház. Ebben az esetben törlés után vissza navigál az alkalmazás a parkolóházak listájához.

Ezek alatt szerepelnek a megtalálható szektorok. Ezek emeletek szerint sorba rendezve jelennek meg. A szektorra rákattintva egy legördülő panel jelenik meg ami tartalmazza az adott szektorhoz tartozó parkolóhelyeket, ezek szintén név szerinti sorrendben jelennek meg. Amelyik parkoló el van foglalva azon egy autó ikon található, valamint ha le van foglalva akkor szürkítetten jelenik meg. Egy parkolóra kattintva át navigál a rendszer annak a parkolónak a részletes adatainak oldalára.

### Parkoló adatait tartalmazó képernyő



*4. ábra Parkoló adatai képernyő*

Ezen a felületen az adott parkoló részletes adatai láthatóak. Itt van lehetőség beállni az adott parkolóba vagy éppen kiállni, ha már foglaljuk azt. Az adminisztrátorok beállhatnak más felhasználók nevében és ki is álhatnak helyettük. Sima felhasználók csak a saját autójukkal tevékenykedhetnek.

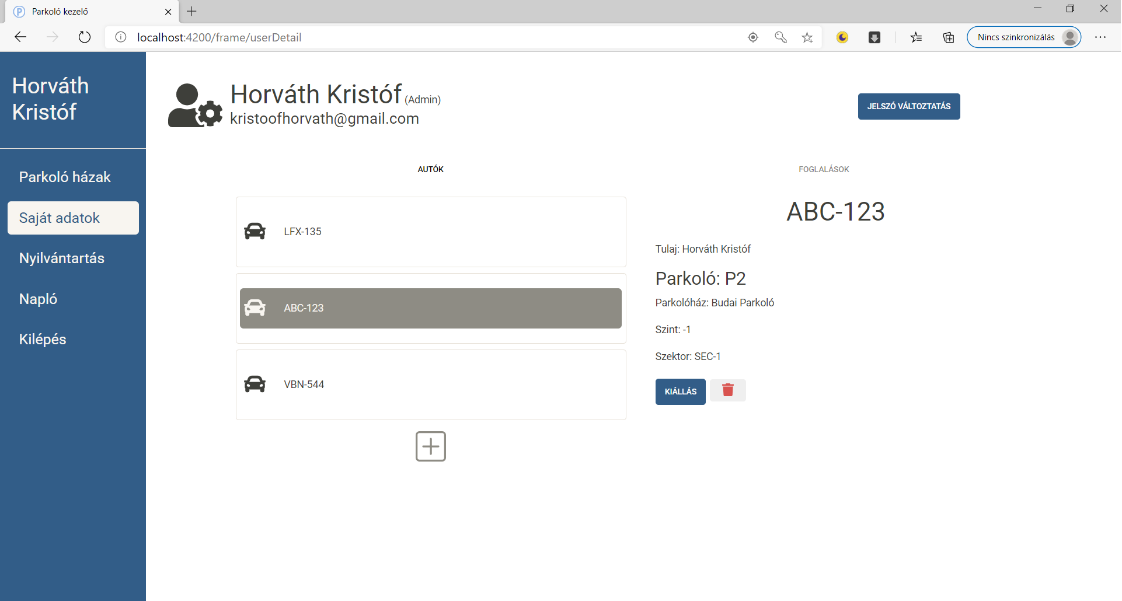
Jobb felül találhatóak a parkoló szerkesztésére és törlésére alkalmas gombok. Szerkeszteni a parkoló nevét lehet. A ceruza ikonnal ellátott gombra kattintva megjelenik egy felugró ablakban az erre alkalmas szövegmező.

Ha a parkolóban nem áll autó akkor középen mindössze egy „Üres” felirat látható, alatta egy „Beállás” feliratú gombbal. Arra rákattintva, ha a felhasználó egyszerű felhasználó akkor megjelennek a saját autóinak a rendszámai egy felugró listában. Az egyikre rányomva eltűnik a felugró ablak és a parkoló el lesz foglalva az által az autó által. Ha a felhasználó egy adminisztrátor akkor először a felhasználók listája jelenik meg (ebben a listában abc sorrend szerint vannak rendezve a felhasználók, de mindig a bejelentkezett felhasználó van legfelül). Ha rányom a listában található egyik névre akkor annak a felhasználónak az autói jelennek meg. Ha nincs autója a választott felhasználónak akkor azt kiírja az alkalmazás. Ezután ugyanúgy, mint az egyszerű felhasználók esetében rá lehet nyomni valamelyik autóra, amelyet ennek hatására beparkoltnak fog tekinteni a rendszer.

A „Beállás” gomba alatt található a „Foglalás” gomb. Ennek segítségével a felhasználók lefoglalhatják saját maguknak az adott parkolót. A gombra kattintva felugró ablak jelenik meg ahol kiválaszthatjuk mennyi időre szeretnénk lefoglalni a parkolót. (Legfeljebb 24 órára lehet lefoglalni.) Ha a parkoló le van foglalva akkor csak a foglalás birtokosa állhat be és ki a parkolóból. A többi egyszerű felhasználónk nem jelennek meg a gombok, amikkel ezeket a műveleteket végre tudnák hajtani. Az adminisztrátoroknak is először meg kell szüntetniük a foglalást és csak utána állhatnak be vagy ki saját vagy más nevében.

Ha a parkolóban áll autó akkor azt egy nagy autó ikonnal jelzi a rendszer, valamint kiírja mellé az autó tulajdonosát is. Egyszerű felhasználók ilyenkor, ha a saját autójuk áll a parkolóban akkor van lehetőségük kiállni a „kiállás” feliratú gomb segítségével. Ha nem az ő autójuk áll bent akkor nem tudnak semmi változtatást csinálni ezen a felületen. A „kiállás” gomb nem jelenik meg nekik. Adminisztrátoroknak ellenben megjelenik akkor is, ha nem az ő autójuk áll bent.

### Saját adatok



*5. ábra Saját adatok képernyő*

A saját adatok felületet a bal oldalon található menüben tudjuk elérni. Ez az oldal tartalmaz minden információt a bejelentkezett felhasználóról. Látható a neve, email címe, hogy adminisztrátor e vagy sem, illetve az autóinak a listája és a foglalásai is itt szerepelnek.

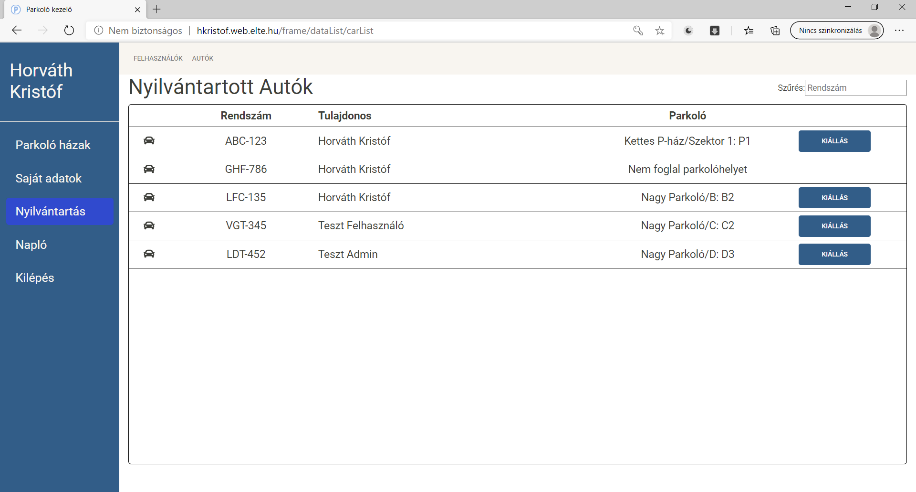
Jobb felül a „Jelszó változtatás” gombbal lehetőség van jelszót változtatni. Rá kattintva megjeleni az ehhez szüksége űrlap egy felugró ablakon keresztül. Itt meg kell adni a régi jelszót és kétszer az újat. Ha nem egyeznek a jelszavak vagy nem helyes a régi jelszó akkor piros szöveggel jelzi ezt a rendszer. Ellenkező esetben megtörténik a változtatás és bezáródik a felugró ablak.

Ki lehet választani az egyes autókat felhasználó autóinak listájából. Ha rákattintunk az egyikre akkor a jobb oldali felületen megjelennek a hozzá tartozó információk. Hogy ha bent áll egy parkolóban akkor le lehet olvasni, hogy melyik parkolóház melyik szektorában található. Ilyenkor megjelenik itt is egy kiállást lehetővé tevő gomb, melynek funkciója és működése megegyezik a parkoló adatait tartalmazó képernyőn lévő „Kiállás” gombbal. Ugyan itt van lehetőség törölni is az adott gépjárművet, a szemetes ikonnal ellátott gomb segítségével. Ha az autó nem foglal éppen parkolóhelyet akkor ezt közli velünk a felület, és nem jelenik meg a „Kiállás” gomb.

Ha a felhasználónak nincsen összesen öt autója akkor van lehetősége új autót felvennie az autók listája alatti plusz jellel ellátott gomb segítségével. Rákattintva megjelenik a felugró ablak, ahol az új autó rendszámát megadva hozzá adásra kerül az új autó. Ha a felhasználó autóinak a száma elérte az ötöt akkor nincs lehetősége többet hozzáadni, az ehhez szükséges gomb nem jelenik meg többé.

Van lehetőség átnavigálni a foglalások listájához is. Itt a felhasználó a saját jelenlegi foglalásait látja, amiből legfeljebb három lehet. Itt helyben le is lehet mondani őket a lista elemek végén található „Lemondás gombbal”.

### Nyilvántartás



*6. ábra*

*Nyilvántartás képernyő*

Ehhez a felülethez csak az adminisztrátorok férnek hozzá. Itt érhetőek el a rendszerben nyilván tartott felhasználók és autók.

#### Felhasználók

Idenavigáláskor elsőként a felhasználók listája jelenik meg. Leolvasható innen az illetők nevei, email címei, és hogy mennyi autót kezelnek. Jobb felül van lehetőség konkrét felhasználó nevére is keresni. Rákattintva valamelyik felhasználóra egy a Saját adatok-hoz hasonló felületre irányít át a rendszer. Itt meg lehet nézni a tényleges autókat és hogy esetlegesen melyik parkolóban állnak. Más felhasználó autójának törlésére adminisztrátorként sincs lehetőség.

Jogosultságokat szerkeszteni a „Jogosultság változtatása” feliratú gombbal lehet, ami felhasználók listájának egyes elemeinek a végén található. Ha az adott felhasználó egyszerű felhasználó akkor gombra nyomáskor a felugró ablak arra fog rákérdezni, hogy adminisztrátorrá szeretné e tenni, ha már adminisztrátor akkor pedig, hogy el akarja e venni ezt a titulust? „Ok” gombra kattintva eltűnik a felugró ablak és végbe megy a változtatás.

Az adminisztrátorok között van egy kitüntetett jogkör, ez a Fő Adminisztrátor. Ő ugyanazokkal a jogokkal rendelkeznek, mint a sima adminisztrátorok azt leszámítva, hogy nekik nem lehet szerkeszteni a jogosultságát. Ő maga, ha át kívánják ruházni ezt a titulust valaki másra azt megteheti. Ennek következtében mindig csak pontosan egy ilyen titulussal rendelkező felhasználó létezik. Átruházni úgy lehetséges, hogy ha rákattint az adott felhasználóra és a megjelenő részletes adatokat tartalmazó felületen, jobb felül helyezkedik el a „Fő Adminisztrátor jog átadása” feliratú gomb, ami csak akkor jelenik meg ha a bejelentkezett felhasználó éppen a Fő Adminisztrátor.

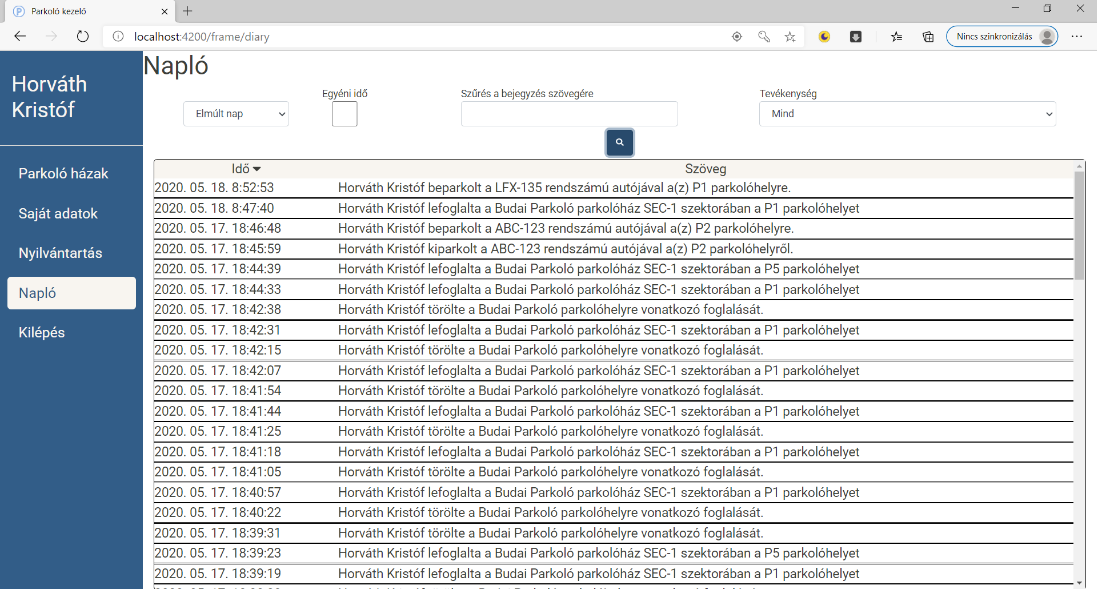
#### Autók

A felső navigáló menü segítségével át leget navigálni az autók listájára.

Ez a felület nagyon hasonló a felhasználók felületéhez. Az autók szintén egy listában jelennek meg, amiben leolvasható a rendszám, a tulajdonos neve, és hogy melyik parkolóban áll (vagy ha nem áll parkolóban). Lehet keresni rendszámra a jobb felül elhelyezkedő szövegmezővel. Ha valamelyik autó épp parkolóban áll akkor van lehetőség erről a felületről is végrehajtani a kiállást, a listaelemek végén szereplő „Kiállás” feliratú gombbal. (Ez persze csak abban az esetben látható a gomb, ha az adott autó parkol valahol.)

Foglalások  
Itt jelennek meg a rendszerben található jelenlegi foglalások. Az adminisztrátorok ezen a felületen lemondhatják a foglalásokat bárki nevében. Az egyes foglalásokra kattintva átnavigál a rendszer a foglalt parkoló felületére.

### Napló



*7. ábra Napló felülete*

Ezt a menüpontot is csak az adminisztrátori jogkörrel rendelkező felhasználók látják és férnek hozzá. Itt tudják lekérdezni a rendszerben történő eseményeket, hogy ki mikor hova parkolt be és honnan parkolt ki. A bejegyzések időrendi sorrendben jelennek meg, és alapértelmezés szerint legújabb időponttal rendelkező van legfelül. Ez a sorrend megcserélhető a fejlécben található „idő” feliratra kattintva. A bejegyzések listája feletti bevitelimezők segítségével lehetőség van időpontra a bejegyzés szövegében található bármelyik kifejezésre és a bejegyzés kategóriákra szűrni. A bejegyzések listája alatt egy grafikon is látható, ami a kiválasztott időintervallum alatti változásokat mutatja az egyes parkolóházakban. A grafikon feletti gombok a parkoló házakat reprezentálják, az egyikre kattintva az ahhoz tartozó adatok rajzolódnak ki a grafikonon. Minden pont egy adott időben mutatja, hogy akkor mennyi volt a szabad helyek száma, illetve a foglalt helyek száma. Azt, hogy melyik adatokat jelenítse meg a grafikon feletti szövegekre kattintva tudjuk állítani. Am át van húzva az nem fog megjelenni.

## Mobil alkalmazás

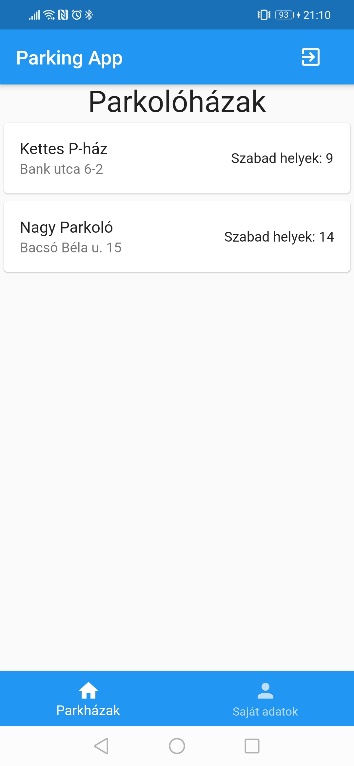
A mobil applikáció elsősorban az egyszerű felhasználók részére van. Az adminisztrációs funkciók nagy része nem elérhető benne. A mindenki számára elérhető funkciók hasonlóképpen működnek, mint a webes alkalmazás esetében.

### Bejelentkezés

Az alkalmazás megnyitásakor egy bejelentkező felület jelenik meg. Helyes emailcím és jelszó megadása után a „Bejelentkezés” gombra kattintva be lehet lépni az alkalmazásba. Be regisztrálni is van lehetőség a „Belépés” gomb alatti regisztráció feliratra kattintva. Ekkor a webes felülethez hasonlóan átalakul az űrlap és megjelennek plusz mezők, mint a „Vezetéknév” és „Keresztnév”, illetve a „Jelszó ismét”. A mezőket megfelelően kitöltve a „Regisztráció” gombra kattintva rögzülnek az adatok a rendszerben és vissza navigál az alkalmazás a bejelentkező űrlaphoz.

Az alkalmazás elmenti a belépési adatokat a mobil eszköz saját tárhelyén ezért ameddig ki nem jelentkezik a felhasználó addig automatikusan bejelentkezik a további indításoknál.

### Parkolóházak listáját tartalmazó képernyő

Belépés után a parkolóházak listája fogadja a felhasználót. Jobb felül a vissza nyíl ikonnal ellátott gombbal lehet kijelentkezni, aminek hatására az applikáció visszanavigál a bejelentkező képernyőre. A lista alatt a webes felületen látottakhoz hasonlóan egy térkép jelenik meg, ami házikó ikonokkal jelzi a rendszerben található parkolóházak pontos földrajzi helyeit.

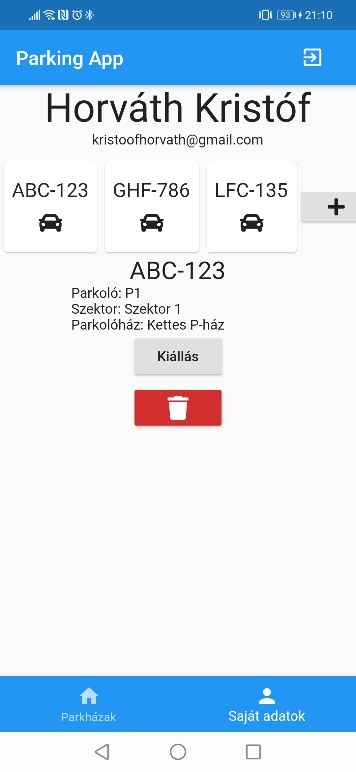
*8. ábra*

*Parkolóház lista*

Az egyes parkolóházakra nyomva a listában vagy a térképen, megjelennek annak a parkolóháznak a szektorai. A szektorokra rányomva a webes felülethez hasonlóan legördülő panel jelenik meg ami tartalmazza a szektorban elhelyezkedő parkolókat. Itt a foglalt parkolók egy autó ikonnal vannak jelölve.

Ha rányomunk egy parkolóra akkor megjelenik a parkoló részletes információit tartalmazó felület. Itt lehetőség van kiparkolni a parkolóból, ha a felhasználó autója parkol bent éppen, ha üres a parkoló akkor be lehet állni az egyik birtokolt autóval. Ha tíz órán belül nem áll ki a parkolóból a felhasználó akkor kap egy emlékeztetőt a mobil készülékére. Ha adminisztrátor a bejelentkezett felhasználó akkor nem kell, hogy a saját autója álljon a parkolóban, bárkit kiállíthat.

### Saját adatok képernyő

A parkolóházak listájának felületéről átnavigálhatunk - a képernyő alján lévő navigációs menü segítségével - a Saját adatok felületre. Itt a felhasználó nevét és email címét lehet megtekinteni, valamint a birtokolt autóit. A „plusz” gomb segítésével lehet új autót felvenni amennyiben ötnél kevesebbel rendelkezik a felhasználó.

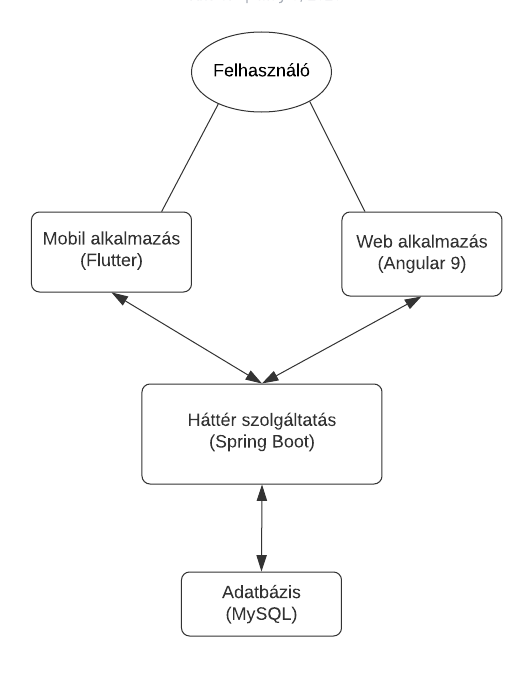
*9. ábra*

*Saját adatok képernyő*

Egy autóra rányomva megjelenik annak információi. Ha parkol valahol akkor a parkolóhelyet tartalmazó parkolóház és szektor lesz leolvasható, valamint a kiállást elősegítő gomb is megjelenik. Ha nem parkol sehol akkor csak a „nem foglal parkolóhelyet” felirat jelenik meg. Az autó törlésére is lehetőség van ezen a felületen a piros, szemetes ikonnal ellátott gomb segítségével.

# Fejlesztői dokumentáció

## Architektúra

A rendszer három fő komponensből tevődik össze.

*10. ábra  
Architektúra*

A megjelenítésért és a felhasználóval való kommunikációért két felület a felelős. Az egyik egy Android eszközökön futtatható mobil alkalmazás, a másik pedig egy böngészőkből elérhető webes felület. A vizuális felületek kiszolgálását a háttérszolgáltatás végzi. Itt történik az adatbázissal való kommunikáció és az adatok előállítása a felületek számára. A kliens alkalmazások és a háttérszolgáltatás REST API-n[[1]](#footnote-1) keresztül http protokoll használatával kommunikálnak egymással. Az üzenetek, amikkel a kommunikáció folyik Json[[2]](#footnote-2) formátumúak.

Az adatok tárolására MySQL[[3]](#footnote-3) relációs adatbázis szerver került kialakításra.

## Háttérszolgáltatás

A háttér szolgáltatás Java nyelven íródott és a Spring Boot[[4]](#footnote-4) keretrendszert használja. Ez lehetővé teszi, hogy a lefordított alkalmazás az egyetlen .jar kiterjesztésű állományból álljon, ami tartalmaz egy beépített Tomcat[[5]](#footnote-5) webszervert és ennek segítségével http kéréseket fogadjon.

### Maven

A fordítást és a fordított állományok előállítását a Maven[[6]](#footnote-6) projekt építő eszköz végzi, és ez gondoskodik a fejlesztéshez szükséges külső függőségek kezelésről is.

#### Függőségek

Ezeket a függőségeket a háttérszolgáltatás projektjének gyökérkönyvtárában található pom.xml-ben (Project Object Model) kell megadni, melynek következtében a következő futtatásnál minden hiányzó könyvtárat letölt egy központi tárhelyről. A Parkoló Kezelő rendszer háttérszolgáltatása a következő fontosabb függőségeket használja:

* spring-boot-starter-data-jpa
  + Java osztályoknak az adatbázis táblákba való transzformációért felelős könyvtárakat tartalmazza.
* spring-boot-starter-web
  + A legfontosabb Spring Boot szolgáltatásokat tartalmazza.
* spring-boot-starter-security
  + Autentikáció megvalósításához való könyvtárak.
* mysql-connector-java
  + MySQL adatbázis szerverhez való kapcsolódáshoz van rá szükség.

#### Futtatható állomány előállítása.

Ahhoz, hogy a Maven minden függőséget, amire a Spring Boot keretrendszernek szüksége van belepakoljon egy darab .jar kiterjesztésű állományba, szükség van, a *spring-boot-maven-plugin* nevezetű bővítményre. Ezt szintén a pom.xml-ben tudjuk a projekthez adni. A *configuration* tagek között lehet beállítani a main osztályt, illetve a futtatható állomány nevét.

*11. ábra*

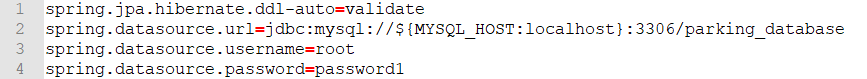
*Sprint-boot-maven-plugin*

Ezek után a projekt gyökérkönyvtárában kiadva a   
„mvn clean package” parancsot, előáll a /target mappa, amiben megtalálhatóak a .class kiterjesztésű fileok, a .jar kiterjesztésű futtatható állomány, amit a „java -jar ParkingApp.jar” parancs kiadásával lehet futtatni.

### Adatbázis

#### Konfiguráció

Ahhoz, hogy a háttérszolgáltatás kommunikálni tudjon a MySQL szerverrel néhány konfigurációs adatot meg kell adnunk. Ezt Spring Boot alkalmazások esetében az *src/main/resources/* mappában lévő *application.properties* nevű fájlban lehet megtenni. Az adatbázisszerverhez való kapcsolódás érdekében itt meg kell adnunk az szerver elérhetőségét, felhasználónevet és a jelszót, amikkel hozzáférhetünk az adatbázishoz, ahogy azt a *12. ábra* mutatja. Az 1. sorban lévő beállítás azt mondja meg, hogy a háttérszolgáltatás minden indításkor ellenőrizze az adatbázis helyességét. Ezt fejlesztés alatt érdemes update-re átírni ugyanis ekkor, ha az osztály modellekben változás történik akkor a következő indításkor a Hibernate frissíti az adatbázis táblákat is. A 2. sorban adjuk meg az adatbázis szerver elérési útvonalát és hogy melyik adatbázist érje el az alkalmazás. Ezt az adatbázist nekünk kell létrehoznunk, nem generálódik automatikusan. Ezt a következő SQL utasítással hajthatjuk végre az adatbázis szerveren: *CREATE DATABASE parking\_database;* Természetesen a „parking\_database” mint az adatbázis neve, változtatható de akkor az application.properties állományban is írjuk át ennek megfelelően.



*12. ábra*

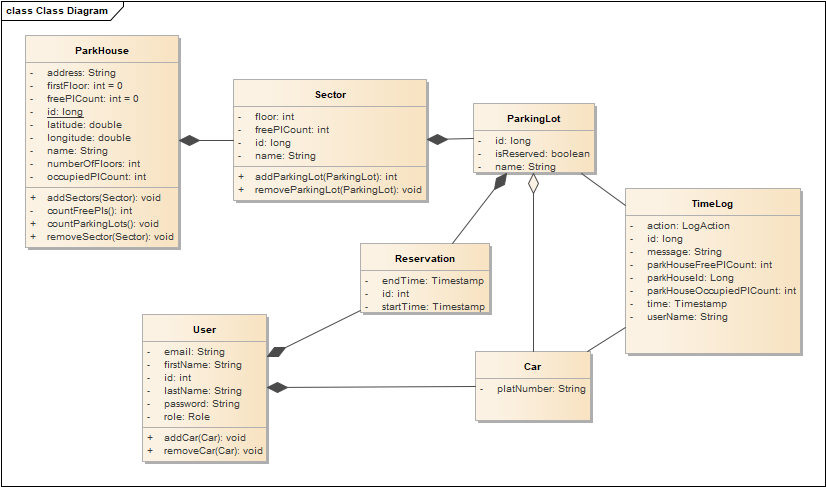
*Adatbázis konfiguráció*

#### Egyed-kapcsolatok

*13. ábra  
 Adatbázis egyedkapcsolat diagramja*

### Modellek

A háttérszolgáltatás hét modellt (saját típust) határoz meg amik a megvalósított funkciók alapjait képzik. Ezeknek a modelleknek megfelelően generálódnak le az adatbázis táblák. Minden új rekord beillesztésekor az elsődleges kulcs automatikusan generálódik. Az elsődleges kulcs a Car esetében a plateNumber (rendszám) attribútum, a többi modell esetében pedig az id.



*14. ábra*

*Entitások osztálydiagramja*

User

A rendszerben szereplő felhasználókat a User osztály reprezentálja. Az osztály adattagjai a következők:

* id: Automatikusan generált egyedi azonosító.
* firstName: A felhasználó vezetékneve.
* lastName: A felhasználó keresztneve.
* email: A felhasználó emailcíme, érvényes emailnek, és egyedinek kell lennie.
* password: A felhasználó jelszava, amivel be tud jelentkezni. Legalább hat karakter hosszúnak kell lennie. Az adatbázisba kódolva kerül elmentésre. A kódolást a Spring Boot keretrendszer által biztosított *BCryptPasswordEncoder[[7]](#footnote-7)* osztály végzi.
* role: A felhasználó jogköre. Ez egy három értékből álló enumerációs típus. A következő értékekkel:
  + ROLE\_USER: Alap felhasználó
  + ROLE\_ADMIN: Adminisztrációs jogkörrel rendelkező felhasználó.
  + ROLE\_FIRST\_USER: A rendszer felállítása után elsőként beregisztráló felhasználó ezt a jogkört kapja. Az ROLE\_ADMIN jogkörhöz képest annyi plusz funkcióval bír, hogy ezt a felhasználót nem lehet megfosztani az adminisztrációs jogkörétől.
* ownedCars: A felhasználó tulajdonában lévő autók listája.
* reservations: A felhasználó aktív foglalásainak listája. Ha nem foglal jelenleg egy parkolót sem akkor üres a lista.

Tagfüggvények:

* addCar(Car): Hozzáadja a paraméterben kapott autót a felhasználó autóihoz.
* removeCar(Car): Eltávolítja a paraméterben megadott autót a felhasználó autói közül.

ParkHouse

A parkolóházak, amik tartalmazzák a parkolóhelyeket. A következő attribútumokat tartalmazza:

* id: egyedi azonosító szám.
* name: A Parkolóház neve, nem lehet üres.
* address: A Parkolóház címe. Ez sima szöveg, nincs semmilyen megszorítás, üresen is lehet hagyni.
* freePlCount: A parkolóházban jelenleg üresen álló parkolók száma, az értéke automatikusan kerül kiszámításra a countFreePls privát tagfüggvény segítségével.
* occupiedPlCount: A parkolóházban a jelenleg autó által elfoglalt vagy felhasználó által lefoglalt parkolóhelyek száma.
* firstFloor: A parkolóház legalsó szintjének a száma. A parkoklóházban lévő szektorok mind egy-egy szinten helyezkendek el.
* numberOfFloors: A parkolóházban lévő szintek száma.
* secotrs: A parkolóházban fellelhető szektorok listája.

Tagfüggvények:

* countParkingLots() : Ez a függvény @Postload annotációval van ellátva. Ez gondoskodik róla, hogy mindig lefusson miután betöltöttük az adatbázisból az objektumot. A metódus lefutásának hatására a freePlCount és az occupiedPlCount attribútumok kitöltésre kerülnek.
* addSectors(Sectors): A paraméterben kapott szektor listát hozzáfűzi a jelenlegi szektorokhoz.
* removeRestor(Sector): A paraméterben kapott szektort eltávolítja a szektorok közül.

#### Sector

A parkolóházakon belül szektorokba csoportosítva tárolódnak a parkolók. Egy parkolóházban több szektor is lehet és nincs elméleti felső korlátja a szektorok számának.

* id: Automatikusan előállított egyedi azonosító.
* name: A szektor megnevezése. Sima szöveg típus, nem lehet üres.
* floor: Az emeletnek a száma, amin elhelyezkedik a szektor a parkolóházon belül.
* freePlCount: A szektorban fellelhető üres parkolóhelyek száma. @Formula annotációval van ellátva az attribútum, ami azt teszi lehetővé, hogy SQL lekérdezéssel kerüljön kitöltésre. Itt összeszámolja a parking\_lots táblából azokat a rekordokat amelyiknek az id mezője megegyezik a szektor objektum id attribútumával (tehát az adott szektorhoz tartozik a rekord) és ahol a plate\_number mező null (tehát nem parkol benne épp autó).
* parkHouse: A parkolóház, aminek a része a szektor.
* parkingLots: A szektorban lévő parkolóhelyeknek a listája.

Tagfüggvények:

* addParkingLot(ParkingLot): Hozzáad egy új parkolót a szektorhoz.
* removeParkingLot(ParkingLot): Törli a paraméterben megadott parkolót a szektor parkolói közül.

#### ParkingLot

A parkolóhelyeket reprezentálja, amikbe be lehet parkolni autókkal. Attribútumai a következők:

* id: Automatikusan előállított egyedi azonosító
* name: A parkoló neve, sima szöveg, nem lehet üres.
* occupyingCar: Az autó amelyik épp a parkolóban áll. Ha null akkor nem áll bent autó.
* sector: A szektor amelyikben megtalálható a parkoló.
* isReserved: Logikai attribútum, ami azt mondja meg, hogy le van e foglalva valamelyik felhasználó által a parkoló vagy sem.
* reservation: A foglalás, ami éppen a parkolón érvényben van. Ha nem foglalja senki a parkolót akkor ez az érték null.

#### Reservation

Ez az osztály reprezentálja a rendszerben történő foglalásokat. Ha egy parkoló le van foglalva egy felhasználó által akkor csak annak a felhasználónak az autói parkolhatnak be egészen addig ameddig a foglalás érvényben van.

Adattagok:

* id: Automatikusan generált egyedi azonosító.
* user: A felhasználó, aki birtokolja a foglalást.
* parkingLot: A parkolóhely amelyik le van foglalva.
* startTime: Az az időpont, amitől kezdve érvényben van a foglalás. (Ez mindig automatikusan a foglalás létrehozásakor éppen aktuális időpont.)
* endTime: Az az időpont amikor automatikusan lejár a foglalás és a parkoló felszabadul.

#### Car

Az autókat a Car nevezetű osztály reprezentálja.

Adattagok:

* plateNumber: Az autó rendszáma. Egyben elsődleges kulcsként is szolgál az adatbázisban
* owner: Az autót birtokló felhasználó.
* occupiedParkingLot: A foglalt parkoló. Ha épp nem foglal el parkolóhelyet, akkor ez az érték null.

#### TimeLog

A naplóban eltárolt bejegyzéseket a TimeLog osztály reprezentálja. Egy ilyen objektum automatikusan keletkezik az egyes (felhasználók által) kiváltott műveletek közben. Az adattagjai a következők:

* id: Automatikusan generált egyedi azonosító.
* time: Az időpont amikor az esemény történt, percre pontosan.
* action: A bejegyzett esemény típusa. Három féle eseményt ment el a napló. Felhasználó beregisztrálását, parkolóból való kiállást, illetve beállást. Ezeket a LogAction nevű enumerációs típus tartalmazza.
* userName: A bejegyzett eseményt kiváltó felhasználó neve (vezetéknév és keresztnév szóközzel elválasztva).
* message: A bejegyzet esemény szöveges reprezentációja. Ezt a szöveget látják az adminisztrátorok a naplót megjelenítő felületen.
* parkiHouseId: Annak a parkoló háznak az egyedi azonosítója, amiben az elnaplózott művelet végrehajtásra került.
* parkHouseFreePlCount: A művelet utáni szabad parkolóhelyek száma a parkolóházban.
* parkHouseOccupiedPlCount: A művelet utáni nem elérhető parkolóhelyek száma a parkoló házban.

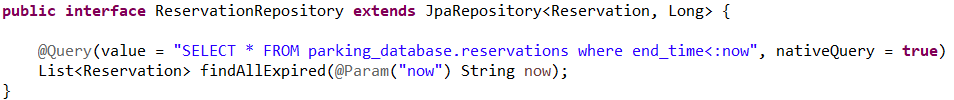
#### LofFilter

Az alap modelleken kívül létezik még egy LogFilter osztály is. Ez a naplóban való keresés funkció miatt létezik. Az adat tagjai a LogFilter osztálynak azok a paraméterek, amik mentén lehet szűrni az adatbázisban eltárolt TimeLog objektumoknak megfelelő rekordokat. Ezek az adattagok a következők:

* userName: Az elnaplózott eseményt kiváltó felhasználó neve.
* action: Az a bejegyzés típus amire szűrni szeretne a felhasználó.
* startTime: Annak az időintervallumnak a kezdeti ideje amire szűrni szeretnénk.
* endTime: Annak az időintervallumnak a vége amire szűrni szeretnénk.

### Repository

Minden entitáshoz, aminek a példányát adatbázisba kell menteni tartozik egy Repository interfész, ami a JpaRepository[[8]](#footnote-8) interfészből öröklődik. Ez tartalmazza az alapvető adatbázis műveleteket, mint például a lekérdezések id szerint vagy új entitás mentése. Minden entitásnak a Repository interfészét ki lehet egészíteni saját lekérdezésekkel. Ezt egy függvény deklarálásával és az a fölé elhelyezett @Querry annotáció paramétereként megadott SQL kifejezéssel tehetjük meg. A *15. ábra* mutat egy példát, ami mutatja, hogy történik a lejárt foglalások lekérdezése.



*15. ábra*

*ReservationRepository findAllExpired lekérdezés*

### Kiszolgáló réteg

A fejlesztést elősegítve a könnyebb átláthatóság érdekében az alkalmazás üzleti logikái a különböző kiszolgáló (service) osztályokba lettek kiszervezve. Ezek az osztályok a Repositorik felhasználásával kommunikálnak az adatbázissal és a követelményeknek megfelelően kezelik az adatokat, és előállítják a kliens alkalmazások felé küldeni kívánt válaszokat. A legtöbb végpont esetében egy entitás objektum vagy egy entitás objektum listája kerül visszaküldésre válaszként. Azonban van olyan eset amikor egyedi válasz objektumra van szükség. Ezek az osztályok a *hu.hkristof.parkingapp.responsetypes* package-ben találhtóak.

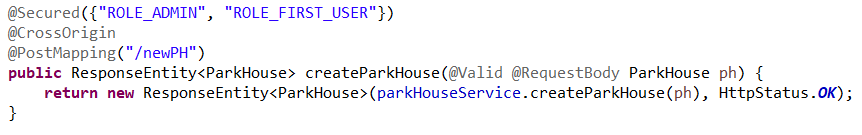
Minden entitáshoz tartozik egy kiszolgáló osztály, amiben ahhoz az entitáshoz köthető üzleti logikák foglalnak helyet. Például a ParkHouseService osztályban történik az új parkolóház létrehozása, a parkolóházak lekérdezése, szerkesztése, törlése.

### Kontrollerek

A kliensekkel való kommunikációhoz szükséges http végpontokat a kontroller osztályok definiálják. Ezek az osztályok @RestController annotációval vannak ellátva, hogy a Spring Boot keretrendszer automatikusan konfigurálja őket. Az egyes tagfüggvények maguk a végpontok, amik a servicekben definiált függvényeket felhasználva előállítják és elküldik a választ a kérést küldő klienseknek. Ezen függvények annotációkkal vannak ellátva, amik definiálják azt, hogy milyen URI-n[[9]](#footnote-9) keresztül hívhatóak meg, valamint, hogy azt, hogy a végpont milyen fajta műveletet hajt végre a kért erőforráson. Ezek az annotációk lehetnek:

* @GetMapping: Erőforrás lekérdezésére
* @PostMapping: Új erőforrás létrehozására
* @PutMapping: Erőforrás módosítására
* @DeleteMapping: Erőforrás törlésére.

Ezen kívül némelyik végpont el van látva @Secured annotációval is, amivel a jogosultság szerinti hozzáférést lehet beállítani. A *16. ábra* mutat egy példát. Ezen a végponton egy új parkolóház létrehozása a végrehajtandó művelet.



*16. ábra*

*Új parkoló ház létrehozásának a végpontja*

### Autentikáció

A háttérszolgáltatás autentikálása a SpringSecurity[[10]](#footnote-10) keretrendszer segítségével lett megvalósítva, Basic autentikáció[[11]](#footnote-11) használatával. Ez azt jelenti, hogy minden levédett végpontot csak a http kérés fejélcében található helyes bejelentkezési adatok kódolt verziójának ellenében lehet elérni. A WebSecurityConfig osztály felelős az autentikációs működés konfigurálásáért. Többek között azt van itt beállítva, hogy minden végpont, aminek az url-je „/auth/” szöveggel kezdődik, legyen levédve és csak autentikálva lehessen elérni őket.



*17. ábra WebSecurityConfig osztály*

Minden API híváskor a http kérés fejlécében található kódolt e-mail cím és jelszó páros segítségével beazonosítja a rendszer a felhasználót, majd az adatbázis segítségével meghatározza a jogkörét is. Valamint a kérés feldolgozásának idejére az AuthenticatedUser osztályban eltárolásra kerül a bejelentkezett felhasználó. Ezt mind a Spring Boot által szolgáltatott UserDetailService[[12]](#footnote-12) interfészt implementáló MyUserDetailService osztály végzi.

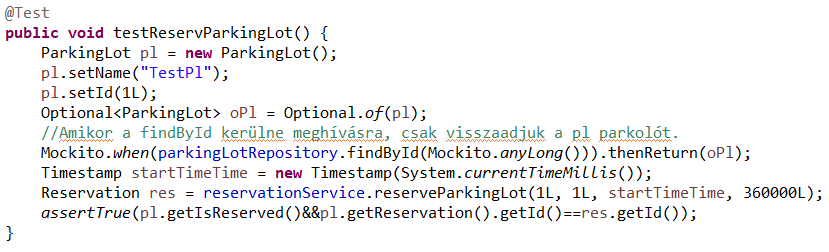


*18. ábra: MyUserDetailService osztály*

### Tesztelés

A háttérszolgáltatás tesztelését automatizált egységtesztek végzik, amik a Junit[[13]](#footnote-13) keretrendszer segítségével lett megvalósítva. A tesztek minden alkalommal lefutnak amikor a Maven előállítja a futtatható állományokat, de az „*mvn test”* paranccsal külön is futtathatóak. Fontos, hogy ha a teszteket tartalmazó osztályok nevei nem felelnek meg a Maven által várt konvencióknak - miszerint a „Test” vagy „TestCase” szöveggel kell kezdődniük, vagy végződniük - akkor ez a parancs nem fogja felismerni és nem fogja lefuttatni azokat. A testek forráskódjai az *src/test/java* mappában találhatóak.

A teszteléskor nem előnyös ténylegesen adatbázisba menteni az adatokat, ezért ezekben az osztályokban a különböző Service-k és Repository-k csak szimulálva vannak. A szimulációt a Mockito[[14]](#footnote-14) keretrendszer biztosítja, amely lehetővé teszi a @Mock és az @InjectMocks annotációk használatát. Az ezekkel megjelölt objektumok nem fognak ténylegesen megkonstruálódni, viszont Mockito.when() függvény használatával definiálhatunk nekik saját, szimulált működést. A *19. ábra* mutat egy példát egy egységtesztre.



*19. ábra*

*Parkoló foglalásának egyik egységtesztje*

## Webes kliens

A webes felület az Angular[[15]](#footnote-15) keretrendszer segítségével lett fejlesztve. Ez lehetővé tette, hogy a megjelenítést különböző komponensekre bontva lehessen implementálni. A programozási nyelv a TypeScript[[16]](#footnote-16) amely lényegében a JavaScript nyelv típusos változata.

### Futtatás

Az Angular CLI[[17]](#footnote-17) eszköz segítségével van lehetőség futtatni a projektet. Az „ng serve” parancs hatására az alapértelmezett 4200-as porton elindul a webszerver, és ott tesztelhető a program. Maga a projekt is a CLI segítségével lett generálva az „ng new <projektnév>” parancs kiadásával.

### Függőségek

Az alkalmazás függőségei a package.json file-ban vannak felsorolva. Hogy ezeket a projektben használni tudjuk be kell importálni az *app/app.module.ts* file-ban. Azonban nem szükséges ezeket manuálisan megtenni ugyanis az npm[[18]](#footnote-18) csomag menedzser segítségével egyetlen paranccsal (npm install <package-name>) el lehet végezni ezeket a műveleteket.

Az Angular alkalmazás a következő külső csomagokat használja:

* Fontawsome: A felületen megjelenő ikonok grafikáját szolgáltatja.[[19]](#footnote-19)
* Bootstrap: A html komponensek különböző stilizálását és rendezését segíti elő.[[20]](#footnote-20)
* RxJS: Aszinkron függvények kezelésére szolgáló könyvtár.[[21]](#footnote-21)
* ng2-charts: A különböző grafikonok megjelenítéséért és kezeléséért felel.[[22]](#footnote-22)
* Agm: A google térkép megjelenítését és az eszköz koordinátáinak lekérdezését teszi lehetővé. [[23]](#footnote-23)

### Modellek

A webalkalmazásban használt saját típusok tükrözik a háttérszolgáltatás entitásait. A háttérszolgáltatáshoz küldött kérések válaszaiban szerepel minden szükséges információ, hogy le lehessen képezni ezeket a típusokat.

### Komponensek

A különböző komponense lényegében a felületnek az építőköveit reprezentálják. Az app.module.ts file-ban minden komponenst be kell importálni a külső csomagokhoz hasonlóan. Ezt szintén automatizálni lehet az Angular CLI segítségével. A következő paranccsal lehet új komponenst létrehozni: „ng generate component <component-name>”. Egy komponens tipikusan három állományból épül fel.

* Egy .html állományból, ami a komponens html szerkezetét írja le.
* Egy .css állományból, ami a komponens stílus leírását tartalmazza.
* És egy .ts állományból, ami a komponens logikai működését leíró TypeScript kódot tartalmazza.

Ez a három állomány a parancs lefuttatásakor automatikusan létrejön és beimportálásra kerül az app.module.ts file-ban.

#### Navigáció

Az AppComponent osztály tölti be a gyökér komponens szerepét. Az útvonalválasztás következtében ebbe komponensbe töltődnek be az átirányított útvonalhoz tartozó komponensek tartalmai. Az alkalmazás minden képernyője gyakorlatilag egy komponens, ami tartalmaz kisebb (nem oldal méretű) alkomponenseket is. Az, hogy melyik komponens melyik útvonalhoz tartozzon az az *app-routing.module.ts* file-ban *routes* tömbben kerül beállításra.

Amikor a „/” url-re navigál a felhasználó akkor automatikusan átirányítja a rendszer a „/login” url-re. A többi útvonalat csak autentikálva lehetséges elérni. Az url-el levédését az AuthGuard osztály végzi, amelynek a canActivate metódusa le ellenőrzi, hogy van e bejelentkezett felhasználó és van e megfelelő jogosultsága. Ha nincs bejelentkezve akkor átirányítja a bejelentkezési oldalra, különben tovább engedi. Ha olyan útvonalat akar elérni a felhasználó, ami nem létezik akkor „/notFound” oldalra irányít át a rendszer.

### Kiszolgálók

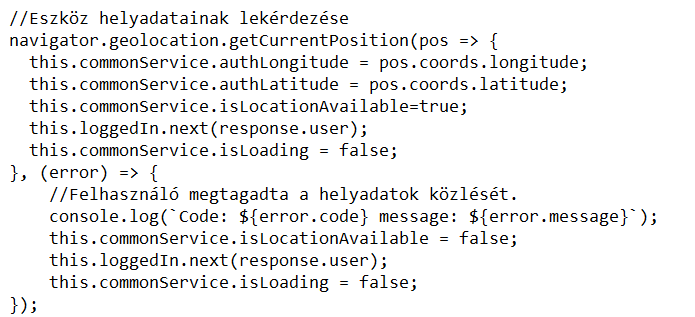
Az üzleti logika azon részei, amelyeket az alkalmazás több pontjából is el kell tudni érni és amik nem kapcsolódnak szorosan a komponensekhez, a kiszolgálókban (Service) kaptak helyet. Itt történnek a háttérszolgáltatáshoz küldött http kérések összeállítása és az adatok kezelése. A projektben lévő servicek a CLI segítségével lettek generálva az „ng generate component <component-name>” parancs segítségével.

#### Bejelentkezés, regisztráció, kijelentkezés

A be- és kijelentkezés, valamint a regisztráció folyamatiéiért az AuthService osztály felelős.

A bejelentkezés a háttérszolgáltatáshoz küldött post hívással történik. Ennek a hívásnak a fejlécében szerepelnie kell az emailcím és jelszó páros elkódolt változatának. Sikeres hívás után megkapjuk a bejelentkezett felhasználó adatait. (Autók, foglalások, jogosultságok stb.) A kódolt jelszó és emailcím ezután mentésre kerül a memóriába, hogy a többi http kérésnél fel lehessen használni.

A bejelentkezési folyamat részeként történik az eszköz helyadatinak lekérdezése. Ez az Agm által szolgáltatott geolocation segítségével kerül végrehajtásra. A *20. ábra* mutatja ennek a kódját. Ha hiba keletkezik, (például, ha nem engedélyezett a helyadatok lekérdezése) akkor ezt tudomásul veszi a rendszer és tovább lép.



*20. ábra*

*Helyadatok lekérdezése*

A regisztráció szintén egy post kéréssel történik. Itt értelemszerűen nem kell a fejlécben küldeni bejelentkezési adatokat, ez a végpont nincs levédve autentikációval.

A kijelentkezésnél eldobásra kerül a memóriában tárolt kódolt emailcím-jelszó páros és elnavigálunk a bejelentkező képernyőre.

#### Modellek kiszolgálói

A háttérszolgáltatáshoz hasonlóan itt is modellek szerint vannak csoportosítva a service-k. Ennek megfelelően például a parkolóházakkal kapcsolatos műveletek a ParkHouseService osztályban találhatóak. A parkolóházakkal kapcsolatos háttérhívások is itt történnek, mint például, az összes parkolóház lekérdezése, parkolóház módosítása, hozzáadása és törlése. A UserService osztályban hasonlóan az összes felhasználó lekérdezése, jogosultságok módosítása, illetve az autók hozzáadása és törlése található meg.

### Webes felület tesztelési jegyzőkönyv

Az alábbiakban néhány felhasználási eset tesztelési jegyzőkönyve van részletezve.

Bejelentkezés, regisztráció

## Mobil alkalmazás

A rendszer mobil alkalmazás kliense elsősorban Android rendszerekre készült. A fejlesztés a Flutter[[24]](#footnote-24) keretrendszer felhasználásával történt, ami a Dart programozási nyelvet támogatja. A webes felülethez hasonlóan itt is különböző építő elemekből (úgynevezett Widget-ekből) lehet összerakni a felületet.

### Építés és futtatás

A projekt a „flutter create <app-name>” paranccsal lett létrehozva. Fejlesztői változatot futtatni a „flutter run” parancsot kiadva lehetséges. Android esetében mobil eszközre telepíthető állományt (APK) létrehozni a „flutter build –prod” kiadásával történik.

### Függőségek

A Flutter projekt függőségeit a *pubspec.yaml* file-ban, a „dependecies” alatt lehet megadni. Általában ennek a file-nak a szerkesztése utáni első futtatásnál letöltődnek az újonnan hozzáadott függőségek, de ezt külön paranccsal („flutter packages get”) is megtehetjük.

Az alkalmazás a következő külső könyvtárakat használja:

* provider: Központosított adat tárolást tesz lehetővé.
* http: A háttérszolgáltatással való kommunikációhoz használt http hívásokat teszi lehetővé
* font\_awsome\_flutter: FontAwsome ikonok megjelenítésére szolgál.
* enum\_to\_string: Enum típusok szöveggé és vissza konvertálására szolgál.
* shared\_preferences: A mobil eszközre történő adatmentést segíti elő.
* flutter\_local\_notifications: Emlékeztető üzenetek értesítések küldésére alkalmas.
* google\_maps\_flutter: A google térkép megjelenítéséhez és menedzseléséhez szükséges.
* geolocator: A készülék helyadatait lehet lekérdezni.
* date\_format: Dátumok szöveggé alakítására különböző paraméterek alapján használatos.

### Widgetek

A Flutter alkalmazás minden építőkövei a Widget osztályok. A teljes alkalmazás ezeből épül fel. A legtöbbet használt widget-eket a Flutter keretrendszer szolgáltatja, de sajátot is létre lehet hozni. Két féle widget létezik, a StateLessWidget és a StatefulWidget. Saját widget létrehozásánál e kettő osztály valamelyikéből kell leszármaztatni. A StateLessWidget-ek egyfajta statikus widgetek amelyek létrehozás után nem rajzolódnak újra. A StatefulWidget-ek ellenben többször is újra rajzolódhatnak.

Amikor egy Widget megjelenítésre kerül akkor lefut a build() metódusa aminek a visszatérési értéke szintén egy Widget. Minden Widget-nek van egy child adattagja aminek segítségével egymásba lehet ágyazni több Widgetet így egy Widget-fát létrehozva. A build() metódus futásakor ez a szerkezet fog kirajzolódni a képernyőre. StatefulWidget esetében ez többször is lefuthat.

# Összefoglalás

Összefoglalás.

# További fejlesztések

Jobb program készítése.

1. REST: <https://restfulapi.net/> [↑](#footnote-ref-1)
2. Json: <https://www.json.org/json-en.html> [↑](#footnote-ref-2)
3. MySQL: <https://www.mysql.com/> [↑](#footnote-ref-3)
4. Spring Boot: <https://spring.io/projects/spring-boot> [↑](#footnote-ref-4)
5. Apache Tomcat: <http://tomcat.apache.org/> [↑](#footnote-ref-5)
6. Apache Maven: <http://maven.apache.org/> [↑](#footnote-ref-6)
7. ByCryptPasswordEncoder: <https://docs.spring.io/spring-security/site/docs/4.2.14.RELEASE/apidocs/org/springframework/security/crypto/bcrypt/BCryptPasswordEncoder.html> [↑](#footnote-ref-7)
8. JpaRepository: <https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html> [↑](#footnote-ref-8)
9. Uniform Resource Identifier, egységes erőforrás-azonosító [↑](#footnote-ref-9)
10. SpringSecurity: <https://spring.io/projects/spring-security> [↑](#footnote-ref-10)
11. Basic Authentication: <https://en.wikipedia.org/wiki/Basic_access_authentication> [↑](#footnote-ref-11)
12. UserDetailService: <https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/userdetails/UserDetailsService.html> [↑](#footnote-ref-12)
13. JUnit: <https://junit.org/junit5/> [↑](#footnote-ref-13)
14. Mockito: <https://site.mockito.org/> [↑](#footnote-ref-14)
15. Angular: <https://angular.io/> [↑](#footnote-ref-15)
16. Type Script: <https://www.typescriptlang.org/> [↑](#footnote-ref-16)
17. Angular CLI: <https://cli.angular.io/> [↑](#footnote-ref-17)
18. npm: <https://www.npmjs.com/> [↑](#footnote-ref-18)
19. FontAwsome: <https://fontawesome.com/> [↑](#footnote-ref-19)
20. Bottstrap: <https://getbootstrap.com/> [↑](#footnote-ref-20)
21. RxJS: <https://rxjs-dev.firebaseapp.com/> [↑](#footnote-ref-21)
22. ng2-charts: <https://valor-software.com/ng2-charts/> [↑](#footnote-ref-22)
23. Agm: <https://angular-maps.com/> [↑](#footnote-ref-23)
24. Flutter: <https://flutter.dev/> [↑](#footnote-ref-24)