**Eötvös Loránd Tudományegyetem**

**Informatikai Kar**

**Programtervező Informatikus szak**

**Parkolóhelyeket nyilvántartó rendszer**

**Témavezető:**

Dr. Nikovtis Tibor

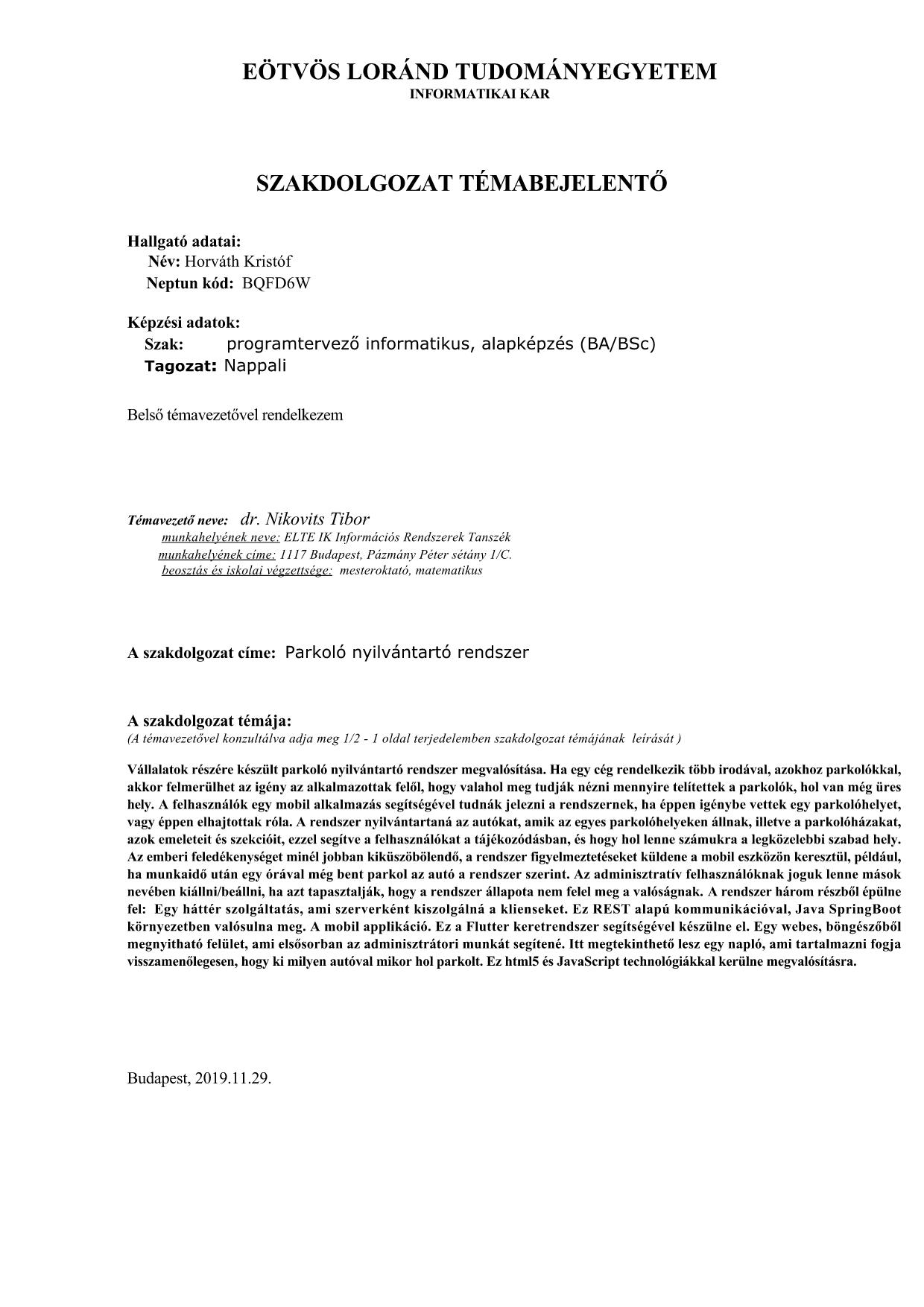
mesteroktató.

**Szerző:**

Hováth Kristóf

Programtervező informatikus BSc.

Budapest, 2020



Tartalomjegyzék

[1 Bevezetés 5](#_Toc41815093)

[2 Felhasználói dokumentáció 6](#_Toc41815094)

[2.1 Rendszer követelmények 6](#_Toc41815095)

[2.2 Telepítés 7](#_Toc41815096)

[2.3 Webes felület használata. 7](#_Toc41815097)

[2.3.1 Bejelentkezési képernyő. 7](#_Toc41815098)

[2.3.2 Parkolóházak listáját tartalmazó képernyő 8](#_Toc41815099)

[2.3.3 Parkolóház adatait tartalmazó képernyő 9](#_Toc41815100)

[2.3.4 Parkoló adatait tartalmazó képernyő 10](#_Toc41815101)

[2.3.5 Saját adatok 12](#_Toc41815102)

[2.3.6 Nyilvántartás 13](#_Toc41815103)

[2.3.7 Napló 15](#_Toc41815104)

[2.3.8 Kijelentkezés 15](#_Toc41815105)

[2.4 Mobil alkalmazás 16](#_Toc41815106)

[2.4.1 Bejelentkezés 16](#_Toc41815107)

[2.4.2 Parkolóházak listáját tartalmazó képernyő 16](#_Toc41815108)

[2.4.3 Saját adatok képernyő 17](#_Toc41815109)

[3 Fejlesztői dokumentáció 18](#_Toc41815110)

[3.1 Architektúra 18](#_Toc41815111)

[3.2 Háttérszolgáltatás 18](#_Toc41815112)

[3.2.1 Maven 18](#_Toc41815113)

[3.2.2 Adatbázis 20](#_Toc41815114)

[3.2.3 Modellek 21](#_Toc41815115)

[3.2.4 Repository 26](#_Toc41815116)

[3.2.5 Kiszolgáló réteg 27](#_Toc41815117)

[3.2.6 Kontrollerek 29](#_Toc41815118)

[3.2.7 Autentikáció 30](#_Toc41815119)

[3.2.8 Tesztelés 32](#_Toc41815120)

[3.3 Webes kliens 33](#_Toc41815121)

[3.3.1 Futtatás 33](#_Toc41815122)

[3.3.2 Függőségek 33](#_Toc41815123)

[3.3.3 Felhasználói felület és navigálási lehetőségek 34](#_Toc41815124)

[3.3.4 Modellek 34](#_Toc41815125)

[3.3.5 Komponensek 34](#_Toc41815126)

[3.3.6 Kiszolgálók 35](#_Toc41815127)

[3.3.7 Webes felület tesztelési jegyzőkönyv 37](#_Toc41815128)

[3.4 Mobil alkalmazás 41](#_Toc41815129)

[3.4.1 Építés és futtatás 41](#_Toc41815130)

[3.4.2 Függőségek 41](#_Toc41815131)

[3.4.3 Mobil alkalmazás felhasználói felülete 42](#_Toc41815132)

[3.4.4 Widgetek 42](#_Toc41815133)

[3.4.5 Kiszolgálók 44](#_Toc41815134)

[3.4.6 Mobil alkalmazás tesztelési jegyzőkönyv 45](#_Toc41815135)

[4 Összefoglalás 48](#_Toc41815136)

[4.1 További fejlesztési lehetőségek 48](#_Toc41815137)

[4.2 Hivatkozások 49](#_Toc41815138)

# Bevezetés

Az emberek igen nagy százaléka jár autóval munkába. Nap mint nap parkolóhelyeket kell keresniük, ahol ott hagyhatják az autóikat. Senki se szeret túl sok időt tölteni üres parkolóhelyek felkutatásával főleg, ha a cég, ahol dolgoznak sok alkalmazottat foglalkoztat és általában zsúfoltak a parkolók. Felmerülhet az igény arra, hogy a keresést megkönnyítve előre lehessen tudni hol találhatóak üres helyek, illetve le is lehessen foglalni munkába menet előtt ezeket a parkolóhelyet, hogy mikor beér az irodába az alkalmazott, biztosan üresen álljon.

Egy vállalat vezetőségének is szüksége lehet rá, hogy valahol nyilvántartsa az irodaházainak vagy a cég birtokában lévő parkolóházaknak a parkolóhelyeit. Igény lehet arra, hogy ez a rendszer tudjon alkalmazkodni a cégben végbemenő változásokhoz, mint például új irodába való költözés, új parkolóház beszerzése, átépítés stb. Az emberi feledékenységet nem lehet soha teljesen kiküszöbölni, így szükséges lehet az is, hogy a vezetőség tagjai mások nevében tudjanak tevékenykedni. Például, ha valaki hazament, de a rendszerben elfelejtette jelezni, hogy kiparkolt a helyéről akkor gyorsabb megoldás kiparkoltatni egyből amikor észreveszi az egyik adminisztrátor, mint felhívni és megkérni, hogy álljon ki maga.

Mind ezen problémák kezelésére jött létre a Parkoló Kezelő parkoló helyeket nyilvántartó rendszer, amely szolgáltat egy mobil alkalmazást a gyors parkolóhely keresés és az azokon való műveletek egyszerű végrehajtására és biztosít egy webes felületet, ami elsősorban adminisztrátorok számára kínál kibővített funkcionalitást, mint a felhasználók kezelése és visszamenőleges naplóbejegyzések áttekintése.

# 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. Ezenkívül minden felhasználó lefoglalhat egyszerre legfeljebb három parkolóhelyet, amelyekbe a foglalás lejártáig csak a saját autóik állhatnak be.

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.

## Rendszer követelmények

Alkalmazásszerver:

* 280 mb memória
* 42 Mb tárhely
* A gépen futni kell egy MySQL adatbázis szervernek.

Mobil applikáció:

* Android operációs rendszer
* 19,42 MB tárhely

A webes felülethez egy böngésző alkalmazásra van szükség.

## Telepítés

A rendszer webes felülete nem igényel telepítést, böngészőből futtatható, és a *hkristof.web.elte.hu* címen elérhető.

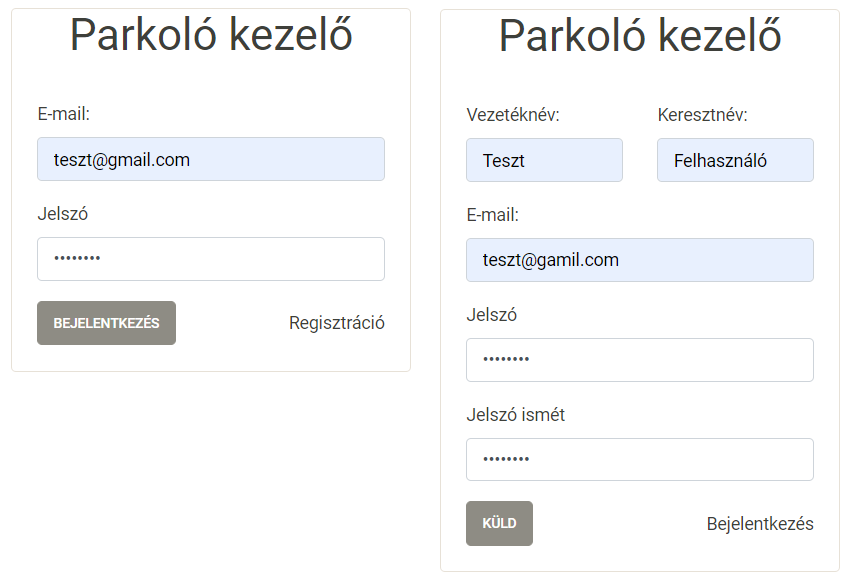
A mobil alkalmazást Android készülékekre telepíteni kell. Ehhez a *parkingapp.apk* állományt fel kell másolnunk egy mobiltelefonra USB kábel vagy Bluetooth segítségével, és egy file kezelő alkalmazás segítségével megnyitni és telepíteni.

## 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.

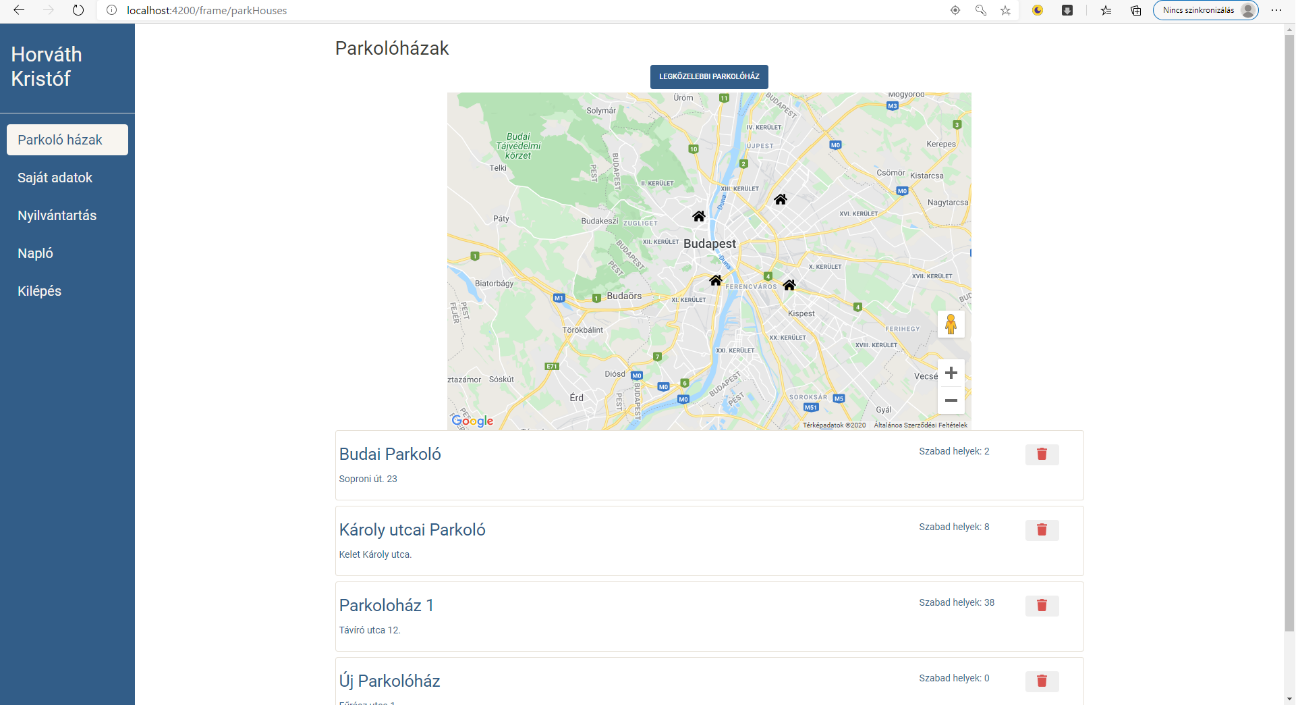


1. ábra Bejelentkező és regisztrációs űrlap.

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 „Regisztráció” 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 a listát felváltja egy űrlap, ami az új parkolóház adatait kéri be. Ha kitöltésre került az összes szükséges mező akkor a „Hozzáad” gomb megnyomásával visszavált a nézet a parkolóházak listájára, amiben szerepel az újonnan hozzáadott parkolóház.

Az adminisztrátor joggal rendelkező 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 felhasználóhoz, az 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 gomb inaktív, és a legközelebbi parkolóház kiválasztásának funkciója nem érhető el.)

### Parkolóház adatait tartalmazó képernyő



*3. ábra: Parkolóház adatai képernyő*

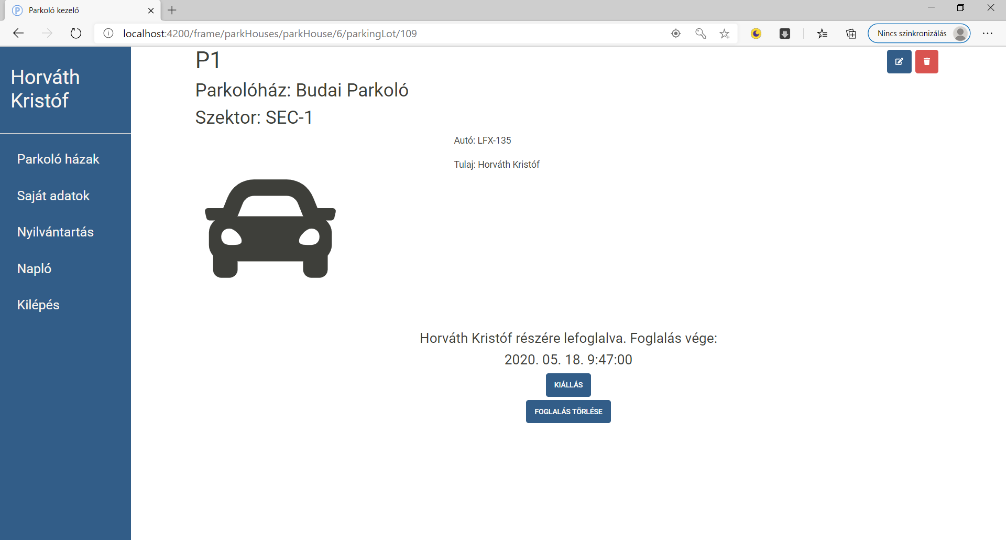
Ezen a felületen a kiválasztott parkolóház adatai találhatók. A felső részen látható a parkolóház neve, címe és hogy mettől meddig vannak számozva az emeletek.

Az információk alatt 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 parkolóházban lévő szektorok, amik 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 a 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.

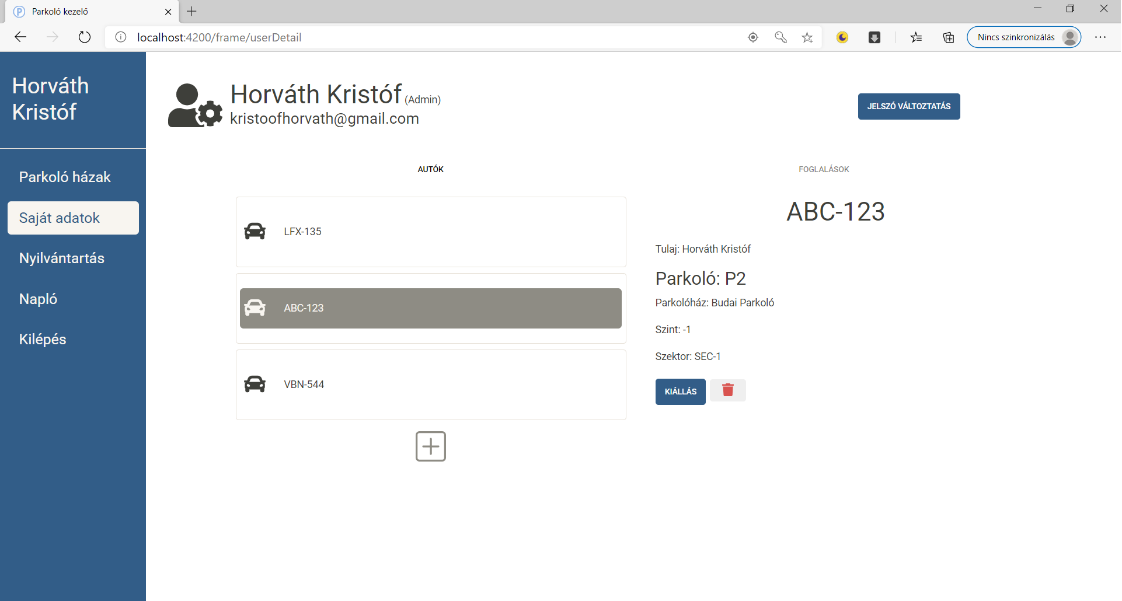
Jobb felül találhatóak a parkoló szerkesztésére és törlésére alkalmas gombok. Szerkeszteni a parkoló nevét lehet. A ceruza ikonnal ellátott gombra kattintva megjelenik egy felugró ablakban az erre alkalmas szövegmező.

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. Sem „kiállás” gomb sem a „Foglalás” gomb nem jelenik meg nekik.

### 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.

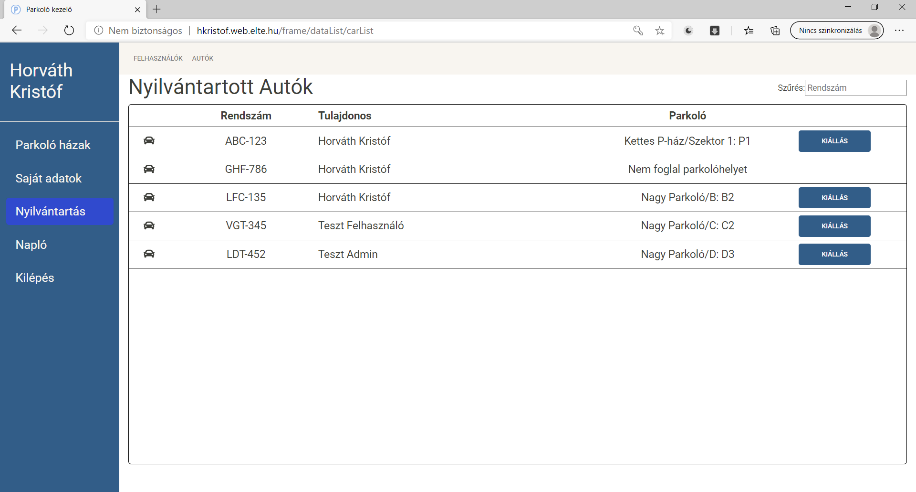
Jobb felül a „Jelszó változtatás” gombbal lehetőség van jelszót változtatni. Rá kattintva megjeleni az ehhez szüksége űrlap egy felugró ablakon keresztül. Itt meg kell adni a régi jelszót és kétszer az újat. Ha nem egyeznek a jelszavak vagy nem helyes a régi jelszó akkor piros szöveggel jelzi ezt a rendszer. 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. Itt lehetőség van felhasználó eltávolítására is a rendszerből, a jobb felül lévő piros gomb segítségével.

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.

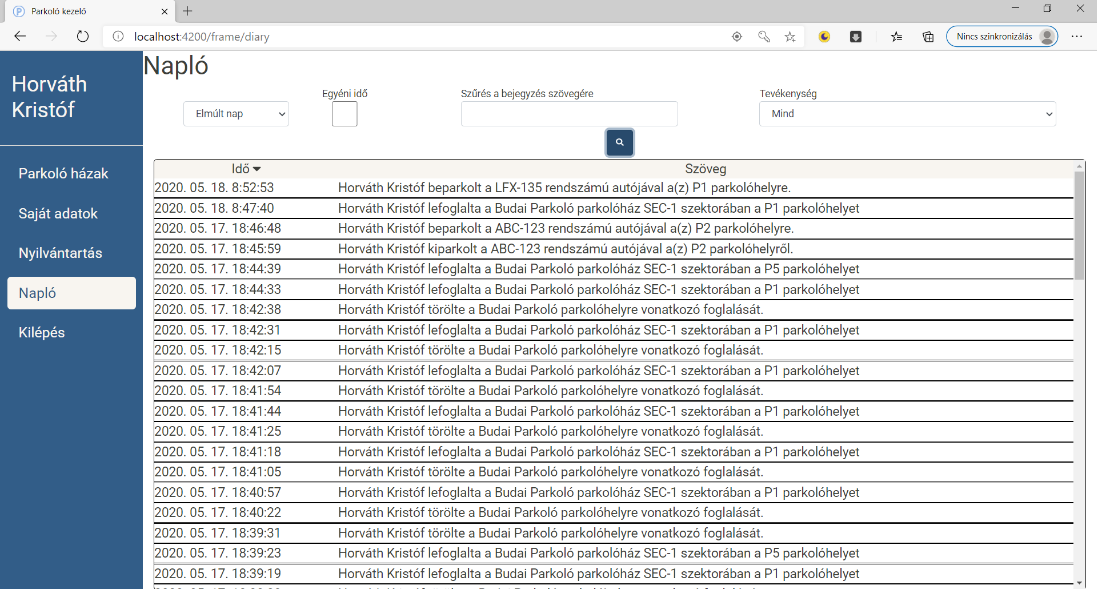
#### Autók

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*

Ehhez a menüponthoz is csak az adminisztrátori jogkörrel rendelkező felhasználók 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 található, 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. Ami át van húzva az nem fog megjelenni.

### Kijelentkezés

Kijelentkezni a jobb oldali menüsáv legalsó menüpontjával van lehetőség. Rákattintva a „Kijelentkezés” menüpontra az alkalmazás átirányít a bejelentkező felületre.

## Mobil alkalmazás

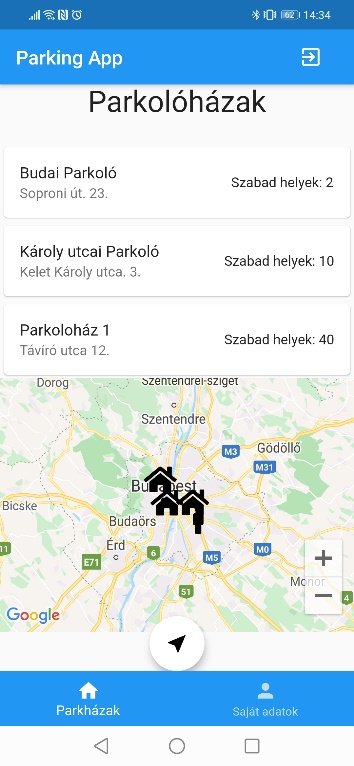
A mobil applikáció elsősorban az alkalmazottak részére lett kitalálva, akik nem rendelkeznek adminisztrációs jogosultsággal. Az adminisztrációs funkciók nagy része nem elérhető benne. A mindenki számára elérhető funkciók hasonlóképpen működnek, mint a webes alkalmazás esetében.

### Bejelentkezés

Az alkalmazás megnyitásakor egy bejelentkező felület jelenik meg. Helyes emailcím és jelszó megadása után a „Bejelentkezés” gombra kattintva be lehet lépni az alkalmazásba. Beregisztrálni 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.

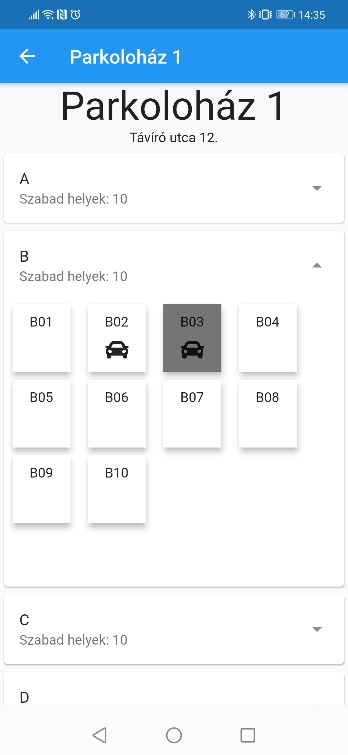
### Parkolóházak listáját tartalmazó képernyő

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 megtalálható a beállást, illetve kiállást elősegítő gomb annak függvényében, hogy áll e autó a parkolóban. Szintén megtalálható itt a gomb, amivel le lehet foglalni a parkolót. Ha a felhasználónak már van három darab foglalása akkor ez a gomb le van tiltva.

Ha tíz órán belül nem áll ki a parkolóból a felhasználó akkor kap egy emlékeztetőt a mobil készülékére.

Az adminisztrátorok itt is kiállhatnak mások nevében és lemondhatják mások helyett a foglalást.

*9. ábra: Szektorok listája*

### 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.

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.

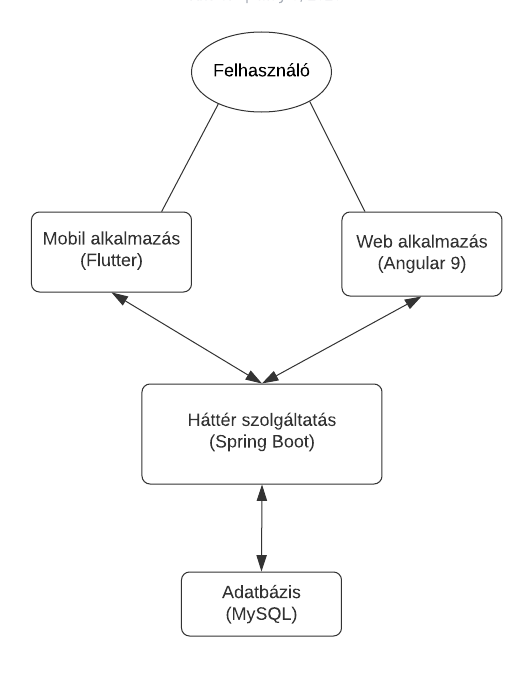
*10. ábra*

*Saját adatok képernyő*

Alul megjelennek a felhasználó aktuális foglalásai amennyiben van neki. Lehetőség van a webes felülethez hasonlóan itt is lemondani az egyes foglalásokat.

# Fejlesztői dokumentáció

## Architektúra

A rendszer négy fő komponensből tevődik össze, és három rétegű architektúra szerint lett megvalósítva.

*11. ábra  
Architektúra*

A felhasználóval való kommunikációért és a grafikus megjelenítésért két felület felelős. Az egyik egy Android eszközökön futtatható mobil alkalmazás, a másik pedig egy böngészőkből elérhető webes felület. A vizuális felületek kiszolgálását egy kiszolgáló 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]](#endnote-1) keresztül http protokoll használatával kommunikálnak egymással. Az üzenetek, amikkel a kommunikáció folyik Json[[2]](#endnote-2) formátumúak.

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

## Háttérszolgáltatás

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

### Maven

A fordítást és a fordított állományok előállítását a Maven[[6]](#endnote-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.

*12. ábra*

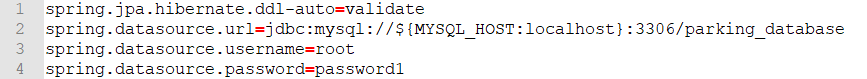
*Sprint-boot-maven-plugin*

Ezek után a projekt gyökérkönyvtárában kiadva a „mvn clean package” parancsot, előáll a */target* könyvtár, amiben megtalálható többek között a ParkingApp.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 adni az szerver elérhetőségét, felhasználónevet és a jelszót, amikkel hozzáférhetünk az adatbázishoz, ahogy azt a *13. 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 kerül megadásra az adatbázis szerver elérési útvonala és hogy melyik adatbázist érje el az alkalmazás. Ezt az adatbázist manuálisan kell létrehozni, 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 át kell írni ennek megfelelően.

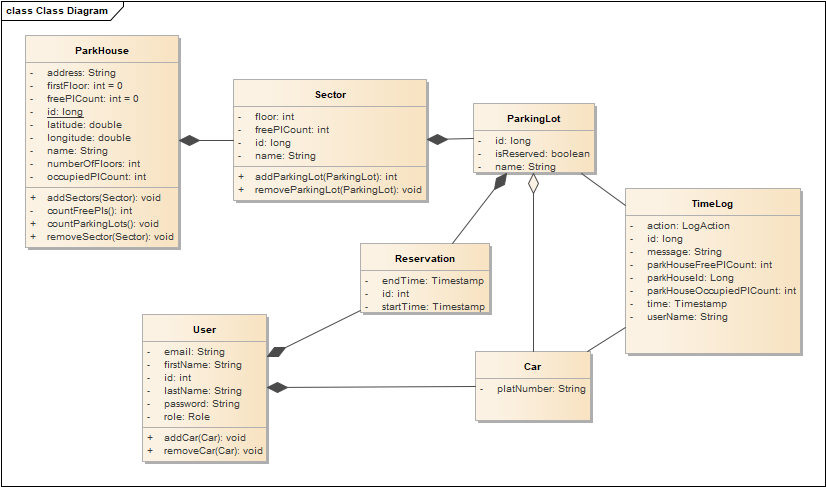


*13. ábra: Adatbázis konfiguráció*

#### Egyed-kapcsolatok

*14. ábra: Adatbázis egyedkapcsolat diagramja*

### Modellek



*15. ábra: Osztály diagrammok*

A háttérszolgáltatás hét modellt (saját típust) határoz meg amik a megvalósított funkciók alapjait képzik. Az osztálydiagrammokat a *15. ábra* mutatja. Ezeknek a modelleknek megfelelően generálódnak le az adatbázis táblák. Minden új rekord beillesztésekor az elsődleges kulcs automatikusan generálódik. Ez alól a Car típus kivétel mivel neki a plateNumber szöveg típusú adattagja az elsődleges kulcs, ami felhasználó által kap értéket.

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]](#endnote-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.

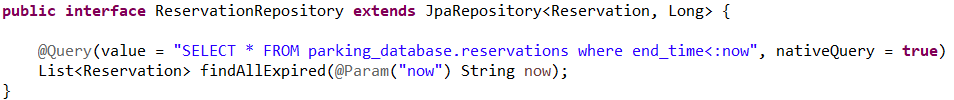
#### LogFilter

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]](#endnote-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 *16. ábra* mutat egy példát, ami mutatja, hogy történik a lejárt foglalások lekérdezése.



*16. ábra: ReservationRepository findAllExpired lekérdezése*

### 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* csomagban találhatóak.

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.

#### Lényeges metódusok

A kiszolgálók metódusait nagyrészben a következő fejezetben kifejtett Kontroller osztályok alkalmazzák a http válaszok megkonstruálására. A működés szempontjából legfontosabb logikát a *ParkingLotService*, ahol a ki- és beparkolás üzleti logikája történik, illetve a *ReservationService* amiben a foglalások kezelése kapott helyet.

##### Beparkolás és kiparkolás folyamata

A *ParkingLotService*-ben található *parkIn* függvény felelős a beparkolás hatására az adatbázisban végbemenő változások végrehajtásáért. A paraméterben kap egy parkoló azonosítót és egy rendszámot. Ezek alapján megkeresi az adatbázisban a megadott entitásokat (ha nem talál hibát dob, a hibakezelés részletesen kifejtve a 3.2.5.2 pontban található) és ha az ellenőrző feltételeken (nem foglalt-e más által a parkoló, ha nem saját az autó akkor adminisztrátor-e a felhasználó) átmegy, módosítja az adatbázist. Visszatérési értéke egy *ParkInResponse* objektum, ami tartalmazza magát a frissült parkolót, a beparkolt autót, valamint a parkolóház frissített statisztikáit miszerint mennyi szabad parkoló és mennyi foglalt parkoló található benne az aktuális beparkolási művelet után.

Szintén a *ParkingLotService*-ben található a *parkOut* metódus, ami egy parkoló azonosítót vár paraméterül és a kiparkolás folyamatát végzi el. A beparkoláshoz hasonlóan ellenőrzi a feltételeket majd végrehajtja az adatbázisban a módosításokat. *ParkOutResponse* típusú objektummal tér vissza, amely ugyanazokat tartalmazza, mint a *ParkInResponse* kivéve, hogy itt nincs autó.

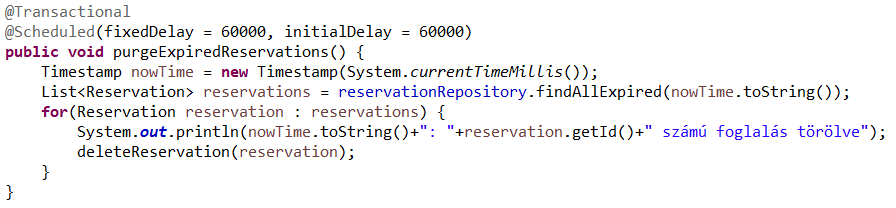
##### Lefoglalás és foglalás törlésének folyamata

A *ReservationService* osztály tartalmazza a foglalások kezelésére vonatkozó logikákat.

A *reserveParkingLot* függvény végzi egy parkolónak a lefoglalását. Paraméterben megkapja a parkoló azonosítóját, a felhasználó azonosítóját, aki le akarja foglalni, a foglalás kezdeti időpontját és a végidőpontját. Itt is végbe mennek az ellenőrzések, hogy nincs e már túl sok foglalása a felhasználónak, nincs- e más által épp lefoglalva a parkoló stb. Ha nem megy át az ellenőrzéseken a kérés akkor kivétel kerül eldobásra. Visszatérési értéke az újonnan létrehozott Reservation objektum.

A *processDeleteReservation* metódus végzi egy foglalás törlését. Egy foglalás azonosítót kap paraméterül, visszatérési értéke pedig a parkolóhely objektum, ami épp felszabadult a foglalás alól. Itt az ellenőrzések után a *deleteReservation* privát függvény kerül meghívásra, ami csak az adatbázisban való változtatásokért felel. Ez a logika külön függvénybe került mert egy másik metódus is használja.

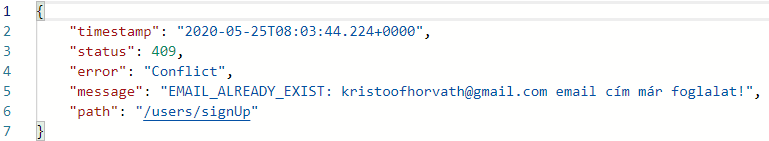
A *ReservationService* osztály *purgeExpiredReservations* void visszatérésű függvénye, szintén a deleteReservation függvényt használja. Ezt a metódust nem kontroller hívja meg hanem bizonyos időközönként fut le. Percenként megnézi a rendszerben lévő foglalásokat és amelyik lejárati ideje korábban van, mint a jelenlegi rendszeridő azokat kitörli az adatbázisból. A periodikus lefutást a @Scheduled annotáció biztosítja.



*17. ábra: Lejárt foglalások törlésének függvénye*

#### Hiba kezelés

Annak érdekében, hogy ha hiba keletkezik egy kérés feldolgozása során, ne álljon le az alkalmazás szerver a lehető legtöbb hibát kivételekkel érdemes kezelni. Az egyedi kivételek a *hu.hkristof.parkingapp.exceptions* csomagban helyezkednek el. Az itt található kivételek el vannak látva a @ResponseStatus annotációval, ami azt definiálják, hogy milyen státuszú hibával válaszoljon az alkalmazásszerver a kliensnek. (Például a *HttpStatus.NOT\_FOUND* érték a 404-es hibakódot jelenti). Ebből adódóan nem kell ezeket a kivételeket *try-catch* blokkal kezelni, amikor kiváltódnak leáll a kérés további feldolgozása és azonnal hibás válasz üzenetet kerül kiküldésre, ami tartalmazza a státusz kódot és a kivétel üzenetét is. A *18. ábra* mutat egy példát egy hibaválaszra amikor már létező emailcímmel szeretne valaki beregisztrálni.



*18. ábra*

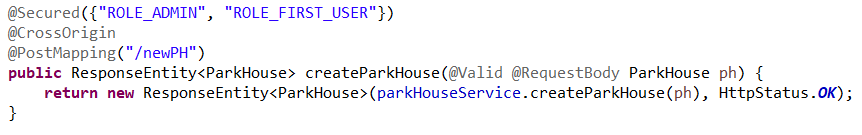
*Hiba válasz példa*

### 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. A végpontokat megvalósító függvények visszatérési értéke minden esetben egy ResponseEntity<T>[[9]](#endnote-9) generikus objektum, aminek segítségével http státuszkóddal fel lehet ruházni a kiküldésre kerülő választ. A válaszok minden esetben a ResponseEntity törzsében megadott objektum Json szöveggé alakított változata. Ezen függvények annotációkkal vannak ellátva, amik definiálják azt, hogy milyen URI-n[[10]](#endnote-10) 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 *19. ábra* mutat egy példát. Ezen a végponton egy új parkolóház létrehozása a végrehajtandó művelet.



*19. ábra: Új parkoló ház létrehozásának a végpontja*

### Autentikáció

A háttérszolgáltatás autentikálása a SpringSecurity[[11]](#endnote-11) keretrendszer segítségével lett megvalósítva, Basic autentikáció[[12]](#endnote-12) 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 az van itt beállítva, hogy minden végpont, aminek az url-je „/auth/” szöveggel kezdődik, legyen levédve és csak autentikálva lehessen elérni őket.



*20. 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[[13]](#endnote-13) interfészt implementáló MyUserDetailService osztály végzi.

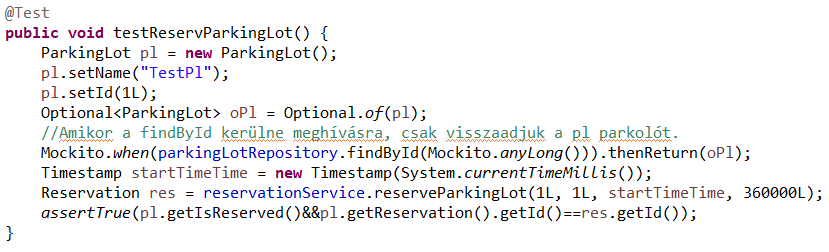


*21. ábra: MyUserDetailService osztály*

### Tesztelés

A háttérszolgáltatás tesztelését automatizált egységtesztek végzik, amik a Junit[[14]](#endnote-14) 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[[15]](#endnote-15) 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 *22. ábra* mutat egy példát egy egységtesztre.



*22. ábra*

*Parkoló foglalásának egyik egységtesztje*

## Webes kliens

A webes felület az Angular[[16]](#endnote-16) keretrendszer segítségével lett fejlesztve. Ez lehetővé tette, hogy a megjelenítést különböző komponensekre bontva lehessen implementálni. A programozási nyelv a TypeScript[[17]](#endnote-17) amely lényegében a JavaScript nyelv típusos változata.

### Futtatás

Az Angular CLI[[18]](#endnote-18) 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.

Az „ng build --prod” paranccsal lehet kész, kiadásra szánt verziót építeni, ami csak a futtatáshoz szükséges minimális állományokat tartalmazza. A parancs hatására generálódott állományok a projekt */dist* könyvtárában találhatóak. Ezeket felmásolva egy webszerverre elérhető lesz a webes felület.

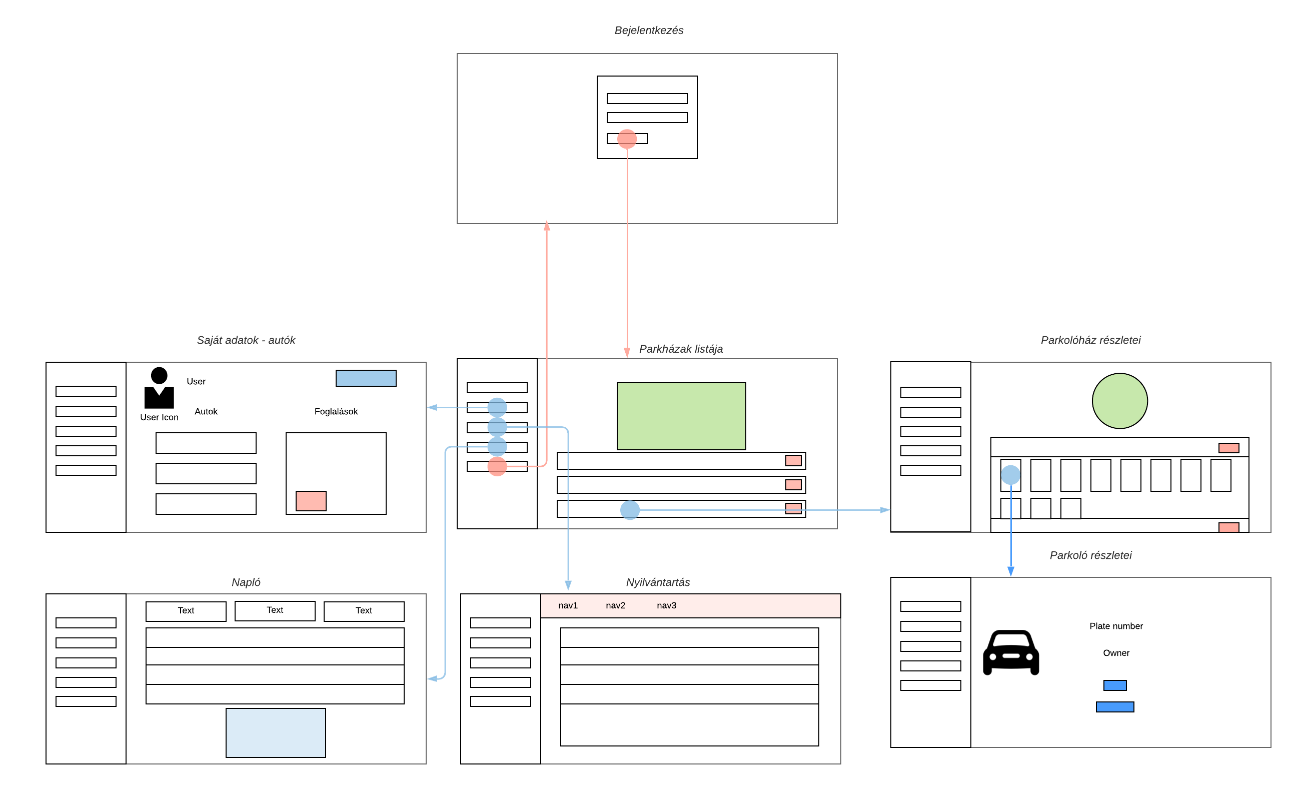
### Függőségek

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[[19]](#endnote-19) 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.[[20]](#endnote-20)
* Bootstrap: A html komponensek különböző stilizálását és rendezését segíti elő.[[21]](#endnote-21)
* RxJS: Aszinkron függvények kezelésére szolgáló könyvtár.[[22]](#endnote-22)
* ng2-charts: A különböző grafikonok megjelenítéséért és kezeléséért felel.[[23]](#endnote-23)
* Agm: A google térkép megjelenítését és az eszköz koordinátáinak lekérdezését teszi lehetővé. [[24]](#endnote-24)

### Felhasználói felület és navigálási lehetőségek



*23. ábra: Képernyő tervek és kapcsolataik*

A felület navigálására egy oldalsávos menü lett megvalósítva. A bejelentkezési képernyő után ez az oldalsó menü mindig látható. Az egyes menüpontokra kattintva a jobb oldali tartalom frissül.

### Modellek

A webalkalmazásban használt saját típusok tükrözik a háttérszolgáltatás entitásait. Itt is megtalálhatóak a *ParkHouse*, *ParkingLot*, *Sector*, *Reservation*, *User* és a *Car* típusok. Az alkalmazás szerverrel való kommunikáció során a webalkalmazás Json formátumban kapja meg a szükséges adatokat, amik alapján létre tudja hozni a saját példányait a szükséges objektumokból.

### 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. Egy komponenst létre tudunk hozni a html címkék használatával bármelyik másik komponensben. Az alkalmazás gyökérkomponense az AppComponent. Ez lényegében csak a navigációt segíti elő, ami a következő pontban kerül kifejtésre.

#### Navigáció

Az útvonalválasztás következtében az AppComponent 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

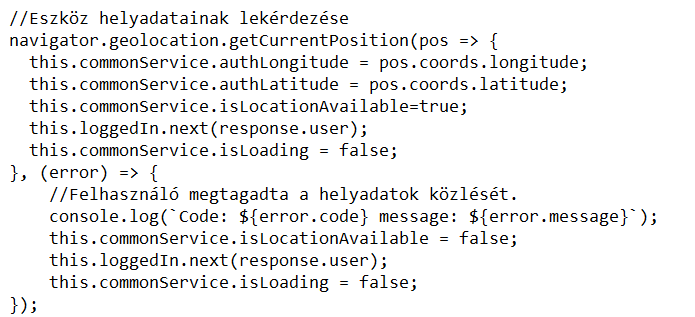
Az üzleti logika azon részei, amelyeket az alkalmazás több pontjából is el kell tudni érni és amik nem kapcsolódnak szorosan a komponensekhez, a kiszolgálókban (Service) kaptak helyet. Itt történnek a háttérszolgáltatáshoz küldött http kérések összeállítása és az adatok kezelése. A projektben lévő servicek 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 *24. ábra* mutatja ennek a kódját. Ha hiba keletkezik, (például, ha nem engedélyezett a helyadatok lekérdezése) akkor ezt tudomásul veszi a rendszer és tovább lép.



*24. 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. GET hívással kerül lekérdezésre az összes parkolóház, POST hívással történik új parkolóház hozzáadása, PUT hívással egy parkolóház szerkesztése és DELETE hívással a törlése. Minden más entitásnál hasonlóan történik ezeknek a műveleteknek a végrehajtása.

A ki- és beparkolás folyamata a ParkingLotService-ben történik az alkalmazásszerver „auth/parkingLots/parkIn/{id}” és az „auth/parkingLots/parkOut/{id}” végpontjaira történő PUT hívással.

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, képernyőképek szerint csoportosítva. A felület tesztelésénél a fő szempont, hogy mennyire kezeli jól a változásokat. A tényleges logika az alkalmazás szerveren történik, aminek helyességét az ottani egység tesztek végzik.

#### Bejelentkezés, regisztráció

A bejelentkezési űrlapon validáció megy végbe, ami nem engedi, hogy üres vagy nem szabványos emailcímet adjon meg, és kötelezővé teszi a jelszó beírását is, aminek minimum hat karakternek kell lennie.

Bejelentkezési űrlap:

|  |  |
| --- | --- |
| **Bemenet/kiváltó esemény** | **Eredmény** |
| A felhasználó helyes adatokat adott meg és rányom a „Bejelentkezés” gombra. | Beléptet a rendszer, megjelenik a Parkolóházak listáját tartalmazó képernyő. |
| Az űrlap validációnak nem felelnek meg a bevitt adatok, és rányom a „Bejelentkezés” gombra. | Piros hibaüzenet jelenik meg, ami beszámol a nem megfelelő mezőkről. |
| Az űrlap validációnak megfelelnek az adatok, de érvénytelenek az alkalmazás szerver szerint. | Piros hibaüzenet jelenik meg: ”Helytelen e-mailcím vagy jelszó!” |

Regisztrációs űrlap:

|  |  |
| --- | --- |
| **Bemenet/kiváltó esemény** | **Eredmény** |
| A felhasználó olyan e-mailcímet adott meg ami már benne van az adatbázisban | Piros hibaüzenet jelenik meg: „Ez az emailcím már regisztrálva van.” |
| A felhasználó az űrlap validáció szerint nem megfelelően tölt ki egy mezőt majd elkattint onnan. | A mező alatt piros hibaüzenet jelenik meg a problémáról. Pl.: Jelszó kevesebb mint hat karakter hosszú -> „Minimum hat karakter” |

#### Parkolóházak listája

|  |  |
| --- | --- |
| **Bemenet/kiváltó esemény** | **Eredmény** |
| A sikeresen bejelentkezett felhasználó ROLE\_USER jogosultsággal rendelkezik. | Csak három menüpont jelenik meg a jobb oldali menüsávban. |
| A sikeresen bejelentkezett felhasználó ROLE\_ADMIN vagy FIRST\_USER jogosultsággal rendelkezik. | Öt menüpont jelenik meg oldalt, valamint a parkolóház elemek végén piros törlésgomb látszik, és egy + gomb jelenik meg a parkolóházak listája alatt. |
| ROLE\_ADMIN felhasználó rákattint egy parkolóház listaelemének a törlés gombjára. | Felugró ablak rákérdez a műveletre. „Ok” gombra kattintva eltűnik a parkolóház a listából |

A + gombra kattintva megjelenik az új parkoló hozzáadásához szükséges űrlap, és a parkolóházak listája eltűnik. A következő tesztek ezen űrlap helyességét ellenőrzik.

|  |  |
| --- | --- |
| **Bemenet/kiváltó esemény** | **Eredmény** |
| Űrlap valamelyik mezője érvénytelen bemenetet tartalmaz. | A „Hozzáad” gomb el van halványítva és nem lehet megnyomni. |
| A felhasználó elkattint egy nem kitöltött mezőről. | Piros hibaüzenet jelenik meg a mező alatt. |
| Felhasználó duplán kattint a térkép valamelyik pontjára. | A hosszúsági- és szélességi fok mezők kitöltődnek az űrlapon. |
| Felhasználó manuálisan olyan hosszúsági- és szélességifok adatokat ad meg amin nincs a térkép területén és rányom a „Hozzáad” gombra. | Piros hibaüzenet jelenik meg az űrlap alatt. |
| Felhasználó helyes adatokat ad meg és rákattint a „Hozzáadás” gombra. | A nézet visszavált a parkolóházak listájára, ami tartalmazza az újonnan létrehozott parkolóházat. A térképen új házikó ikon jelenik meg az új parkolóház számára. |

#### Szektorok listája

|  |  |
| --- | --- |
| **Bemenet/kiváltó esemény** | **Eredmény** |
| A lista alatti + gombra kattint a felhasználó. | Megjelenik egy felugró ablak rajta egy űrlappal. |
| A szektor hozzáadására szolgáló űrlap nincs megfelelően kitöltve, és a felhasználó rányom a „Hozzáad” gombra. | Piros hibaüzenet jelenik meg: „Érvénytelen adatok” |
| A felhasználó olyan emelet számot ír be az emelet mezőbe amilyen emelet nincs a parkolóházban, és elkattint a mezőröl. | Piros hibaüzenet jelenik meg a menü alatt a megfelelő emeletszám tartományról. |

#### Parkoló adatai

|  |  |
| --- | --- |
| **Bemenet/kiváltó esemény** | **Eredmény** |
| ROLE\_USER jogosultsággal rendelkező felhasználó megérkezik a felületre, és a parkoló foglalt más autója által. | A felület kiírja ki áll a parkolóban, de semmilyen gomb nem jelenik meg. |
| ROLE\_ADMIN jogkörrel rendelkező felhasználó megérkezik a felületre, és a parkoló foglalt más autója álltal. | A felület kiírja ki áll a parkolóban és megjelenik egy kiállás gomb. |
| Bármilyen felhasználó megérkezik a felületre, és a parkolóban nem áll autó. | A felület kiírja, hogy üres a parkoló és megjelenik két gomb. Egy „Beállás” gomb és egy „Foglalás” gomb. |
| ROLE\_ADMIN jogosultságú felhasználó rákattint a „Beállás” gombra. | Megjelenik egy felugró ablak a felhasználók listájával. A lista elején maga a bejelentkezett felhasználó szerepel. |
| ROLE\_USER jogosultságú felhasználó rákattint a „Beállás” gombra. | Felugró ablakban megjelenik az autóinak a listája. |
| A felhasználó rákattint a „Foglalás” gombra, úgy, hogy háromnál kevesebb foglalással rendelkezik. | Felugró ablakban megjelenik egy óra választó beviteli mező. |
| A felhasználó három foglalással rendelkezik és rákattint a „Foglalás” gombra. | Felugró ablak jelenik meg, ami tájékoztatja a felhasználót, hogy nem tud háromnál többet foglalni. |
| Beálláskor a felhasználó az autók listájában rákattint az egyik elemre. | Eltűnik a felugró ablak és a felületen megjelenik az autó ikon és a tulajdonosi információk. |
| A foglalási felugró ablakban a felhasználó rákattint a foglalás gombra. | A felugró ablak eltűnik és a felület információt ír ki, hogy ki foglalja és meddig van érvényben a foglalás. |

#### Saját adatok

|  |  |
| --- | --- |
| **Bemenet/kiváltó esemény** | **Eredmény** |
| A felhasználó rákattint a + gombra az autók listája alatt. | Felugró ablak jelenik meg egy beviteli mezővel, ami rendszámot kér. |
| A felhasználó üresen hagyja a rendszám beviteli mezőjét és rányom az „OK” gombra. | Piros hibaüzenet jelenik meg. |
| A felhasználó kitölti a rendszámnak szánt beviteli mezőt és rányom az „OK” gombra. | Az autók listájában megjelenik az újonnan létrehozott autó. |
| A felhasználó rákattintott a „Jelszó változtatás” gombra és a felugró űrlapot üresen hagy vagy rányom az „OK” gombra. | Piros hibaüzenet jelenik meg ami közli, hogy minden mezőt ki kell tölteni. |

#### Nyilvántartás

Ebbe a menüpontba ROLE\_USER jogosultságú felhasználó nem juthat el, ezért innentől kezdve fel tehető, hogy adminisztrátor a rendszert használó személy.

|  |  |
| --- | --- |
| **Bemenet/kiváltó esemény** | **Eredmény** |
| A felhasználó rákattint a felhasználók listájában az egyik elemre. | A felület átnavigál annak a felhasználónak az adataira. |
| A felhasználó rákattint egy ROLE\_USER jogosultságú felhasználó listaelemének végén található „Jogosultság változtatása” gombra | Felugró ablak rákérdez, hogy biztos adminisztrátorrá szeretné e tenni. Igenleges válasz esetén az adott felhasználó ROLE\_USER szerepet kap, és megváltozik a lista elme elején található ikon. |

## Mobil alkalmazás

A rendszer mobil alkalmazás kliense elsősorban Android rendszerekre készült. A fejlesztés a Flutter[[25]](#endnote-25) 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 apk” kiadásával történik.

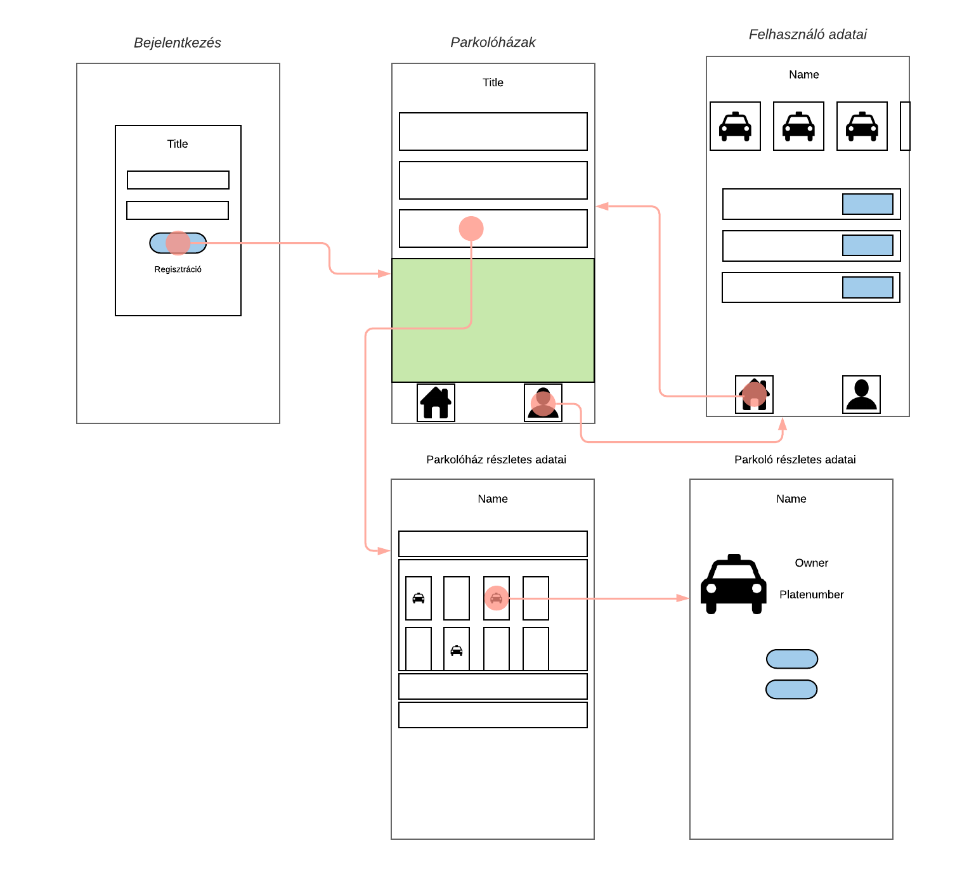
### Függőségek

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.[[26]](#endnote-26)
* http: A háttérszolgáltatással való kommunikációhoz használt http hívásokat teszi lehetővé
* font\_awsome\_flutter: FontAwsome ikonok megjelenítésére szolgál.[[27]](#endnote-27)
* enum\_to\_string: Enum típusok szöveggé és vissza konvertálására szolgál.[[28]](#endnote-28)
* shared\_preferences: A mobil eszközre történő adatmentést segíti elő.[[29]](#endnote-29)
* flutter\_local\_notifications: Emlékeztető üzenetek értesítések küldésére alkalmas.[[30]](#endnote-30)
* google\_maps\_flutter: A google térkép megjelenítéséhez és menedzseléséhez szükséges.[[31]](#endnote-31)
* geolocator: A készülék helyadatait lehet lekérdezni.[[32]](#endnote-32)
* date\_format: Dátumok szöveggé alakítására különböző paraméterek alapján.

### Mobil alkalmazás felhasználói felülete



*25. ábra: Mobil alkalmazás képernyőtervei*

A mobilapplikáció hasonlóan épül, mint a webes felület. A parkolóházak és annak szektorai és parkolói között ugyanúgy kell navigálni. Itt nem volt szükség külön menüsávra mivel csak kettő menüpont érhető el itt a webes felület öt menüpontja közül. Helyette egy alsó navigációs sávon lehet váltakoztatni a parkolóházak és a saját adatok nézeteit.

### Widgetek

A Flutter alkalmazás építőkövei a Widget osztályok. A teljes alkalmazás ezekből épül fel. A legtöbbet használt widget-eket a Flutter keretrendszer szolgáltatja, de sajátot is létre lehet hozni. Két féle widget létezik, a StateLessWidget[[33]](#endnote-33) és a StatefulWidget[[34]](#endnote-34). 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 kapcsolatban áll egy State[[35]](#endnote-35) osztállyal, ami lehetőséget ad arra, hogy többször is kirajzolásra kerüljenek.

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 State osztályaiban ez többször is lefuthat a *setState* függvény hívásának hatására.

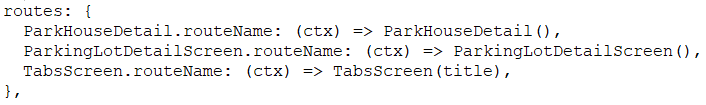
#### Belépési pont és a gyökér Widget

Az alkalmazás belépési pontja a *main()* függvény, a *main.dart* file-ban található. Itt inicializálódnak az értesítés funkcióhoz szükséges objektumok és beállításra kerül, az alkalmazás preferált orientációja. A *runApp(MyApp())* függvény elindítja a widget-fa megjelenítéséért felelős folyamatot.

A MyApp widget az alkalmazás gyökere. Ebben az osztályban történik meg a téma beállítása, a kezdő képernyő kiválasztása és a kiszolgálókra való feliratkozás.

#### Navigáció és útvonal irányítás

Egy Flutterrel készült app navigációja vermes szerkezettel van megvalósítva. A verem tetején lévő widget van kirajzolva a képernyőre, ha átnavigálunk valahova akkor az új képernyőnek szánt widget rárajzolódik az előzőre. A Navigator[[36]](#endnote-36) osztály biztosítja ennek a szerkezetnek a kezelését. a Navigator.pop() hatására visszalépés történik az előző képernyőre, míg a Navigator.push(Widget) segítségével új képernyőt tudunk rakni a verem tetejére. (Bármilyen widgetet berakatunk a verembe, de érdemes Scaffold-ot[[37]](#endnote-37) használni, hogy háttérrel és fejléccel rendelkezzen.) A verembe rakni kívánt widgetre útvonal segítségével is lehet hivatkozni. ezeket a MyApp widget routes paraméterében hell megadni a *26. ábra* szerinti módon. Minden widget elérési útja egy string ami az egyes widgete osztályaiban kaptak helyet statikus változóként.



*26. ábra: Mobil applikáció útvonal beállításai*

### Kiszolgálók

A központi adatokat tároló osztályok és a felülettől elkülönülő logikák az úgynevezett Privder-ek biztosítják. Ezek biztosítják, hogy az általuk szolgáltatott osztályokból csak egy példány jöjjön létre és azt továbbítsák a widgetek felé. A widgetek ezekre az osztályokra fel tudnak iratkozni és bizonyos események bekövetkeztekor frissíteni tudják magukat.

A szolgáltató osztályok ChangeNotifier[[38]](#endnote-38) osztállyal vannak létrehozva, tehát futtatni tudják a *notifyListeners()* függvényt, amely végrehajtásakor az összes feliratkozott widget build metódusa lefut.

#### Autentikáció

Az alkalmazás beléptető logikája az AuthService osztályban található, és az ott létrejövő eredmények függvényében a CommonProvider üzen a widgeteknek amik megfelelően reagálnak az eseményekre.

A bejelentkezés a webes felülethez hasonlóan egy post kéréssel zajlik az alkalmazásszerver felé. A válaszban kapott felhasználói adatok Json üzenet fromályában érkeznek. Ez kikódolásra kerül és betöltődik a CommonProvider *loggedInUser* nevű User típusú adattagjába. (A mobil alkalmazás egyedi típusai megegyeznek a 3.2.3 fejezetben leírtakéval) Sikeres bejelentkezés után az autentikációs token (ami az emailcím és a jelszó kettősponttal elválasztott kódolt változata) elmentésre kerül a mobil eszköz tárhelyén. Következő belépésnél, ha található elmentett token akkor automatikusan belépteti a felhasználót az alkalmazás.

#### Modellek kiszolgálói

A webalkalmazáshoz hasonlóan itt is külön osztályokba kerültek az egyes modellekhez kapcsolódó logikák. Az alkalmazás szerver felé történő adat lekérdező és módosító hívások ezekben az osztályokban történik.

A CarService osztályban történik az autók hozzáadása az *addCar* függvény segítségével, ami egy POST hívást küld a háttérszolgáltatás felé. Paraméterben meg kell adni az új autó objektumát és a felhasználót, akinek az új autója lesz. A removeCar ennek az ellentétes műveletét hajtja végre, ami egy DELETE hívást küld az alkalmazás szerver felé.

A ParkingLotService osztály gondoskodik a ki- és beparkolás folyamatáról, amelyek PUT hívást intéznek az alkalmazás szerver felé.

A ReservationService osztályban pedig a foglalásokkal kapcsolatos műveletek szerepelnek. Ilyen a makeReservation függvény, ami egy POST hívással létrehozza az új foglalást az adatbázisban és sikeres válasz után előállítja a mobil alkalmazásban is az objektumot. Szükséges paraméterben megadni a parkolót, ami lefoglalásra kerül, a felhasználót, aki a foglalást intézi és az időtartamot amennyi ideig tart majd a foglalás órában kifejezve. A deleteReservation DELETE hívást intéz a háttérszolgáltatás felé és eltávolítja a paraméterben megadott parkolón lévő foglalást.

#### Értesítés küldés

A NotificationService gondoskodik az értesítések beállításáról és időzítéséről. Minden beparkolási folyamat lefutásakor lefut a NotificationService.*setUpNotification* függvény is, ami egy egész számot és egy parkolót vár paraméterül. Ez a függvény beállít egy időzített értesítést, ami tíz óra elteltével megjelenik a felhasználó telefonján, és emlékezteti, hogy még mindig ment áll a rendszer szerint a parkolóban. Ha a tíz óra eltelte előtt végrehajtóik a kiparkolás művelet a parkolóból akkor megszakításra kerül az értesítés és nem fog megjelenni. Ezt a megszakító metódust a FlutterNotificationPlugin[[39]](#endnote-39) szolgáltatja cancel() néven.

### Mobil alkalmazás tesztelési jegyzőkönyv

A mobil alkalmazás tesztelése manuálisan történt. Külön oda kellet figyelni arra, hogy hogyan működik az alkalmazás internet elérése nélkül, mivel ezeknél az eszközöknél gyakran előfordul, hogy nincsenek internet közelben.

Az alábbiakban néhány fontosabb teszteset kerül részletezésre.

#### Bejelentkezés

|  |  |
| --- | --- |
| **Bemenet/kiváltó esemény** | **Eredmény** |
| A felhasználó helytelen adatokat ad meg és rányom a „Belépés” gombra. | Felugró ablak jelzi, hogy helytelen e-mailcímet vagy jelszót adott meg. |
| A készüléken nem elérhető az internet vagy az alkalmazás szerver nem fut és a felhasználó rányom a „Belépés” gombra. | Felugró hibaüzenet jelzi, hogy a szerver nem elérhető. |
| Elérhető az alkalmazás szerver és helyes adatokat adott meg a felhasználó, majd rányom a „Belépés” gombra. | Az alkalmazás belépteti és átirányítja a parkolóházak listáját megjelenítő képernyőre. |

#### Legközelebbi parkolóház

|  |  |
| --- | --- |
| **Bemenet/kiváltó esemény** | **Eredmény** |
| A parkolóházak listájának képernyőjén a felhasználó rányom az alul középen található lebegő gombra. | Az applikáció átnavigál a légvonal szerint legközelebbi parkolóház adatait tartalmazó képernyőre. |
| A felhasználó nem engedélyezte az alkalmazás számára a helyadatok elérését. | A lebegő gomb nem jelenik meg, valamint a térkép nem mutatja az emberke ikonnal, a tartózkodási helyet. |

#### Be-kiállás, foglalás

A be- és kiparkolások és folyamatai nagyjából megegyeznek a webes felületével tehát a tesztesetek is azonosak. Annyi különbséggel, hogy itt a ROLE\_ADMIN jogosultsággal rendelkező felhasználóknak is csak a saját autójuk listája jelenik meg, a felhasználók listája nem.

|  |  |
| --- | --- |
| **Bemenet/kiváltó esemény** | **Eredmény** |
| A felhasználó üres parkolóhelyre szeretne beállni. Az eszközön nincs internet elérés. Rányom az autóinak listáján egy autóra. | Piros hibaüzenet jelzi, hogy nem sikerült csatlakozni a szerverhez. |
| A felhasználó internet hozzáférés nélkül rányom a „Kiállás” gombra egy parkolóban, ahol az ő autója áll. | Piros hibaüzenet jelzi, hogy nem sikerült csatlakozni a szerverhez. |

#### Autó törlése, hozzáadása

A Saját Adatok felületen történő + gomb és az autókra nyomva megjelenő piros törlés gombok használatának tesztelése.

|  |  |
| --- | --- |
| **Bemenet/kiváltó esemény** | **Eredmény** |
| A felhasználó a + gomb megnyomásakor megjelenő mezőbe nem ír semmit majd rányom a mentés gombra. | Piros hibaüzenet jelzi, hogy ki kel tölteni a mezőt. |
| A felhasználó kitölti a mezőt, de nincs hozzáférés az internethez | Piros hibaüzenet jelenik meg a szövegmező helyén, amely közli, hogy nem lehetett kapcsolódni a szerverhez. |
| A felhasználó rányom az egyik autót kiválasztva a törlés gombra. | Felugró ablak rákérdez még egyszer és igenlő válasz esetén eltűnik az autó a listából. |
| A felhasználó rányom a törlés gombra internet kapcsolat nélkül. | Megjelenik a felugró ablak és igenlő válasz esetén eltűnik, majd piros szöveg jelenik meg a gomb alatt. |

# Összefoglalás

A Parkoló Kezelő egy olyan rendszer lett, amely segítségével a minden napi parkolás egyszerűsödhet. Ha egy vállalaton belül sok alkalmazott használja akkor sokkal kényelmesebben és átláthatóbb módon lehet megközelíteni a parkolással járó mindennapi kellemetlenségeket.

## További fejlesztési lehetőségek

Mindig lehet további funkciókon gondolkozni és egyéb kiegészítéseket belefejleszteni minden alkalmazásba. A Parkoló Kezelőben is sok lehetőség van további fejlesztésekre.

Jelenleg a rendszer csak és kizárólag magyarnyelvű és budapesti vállalatokra van korlátozva. Több nyelvre le lehetne fordítani és egy nyelv kiválasztási funkciót be lehetne építeni a webes felületre, a mobilalkalmazás pedig ki tudná használni a telefonokban lévő lokalizációs funkciót és aszerint beállítani a saját nyelvét.

A térképet kilehetne terjeszteni nagyobb területre nem csak Budapestre (akár az egész világra), hogy az internacionális vállalatok is tudják használni.

A mobil alkalmazás jelenleg csak Android rendszerekre van építve, de a Flutter keretrendszer biztosít lehetőséget IOS-re való portolásra is. (Ehhez azonban egy Mac[[40]](#endnote-40) számítógépre van szükség, ami az alkalmazás fejlesztése során nem állt rendelkezésre.)

## Hivatkozások

A dokumentumban megjelölt hivatkozások elérése 2020. 05. 30.-ai napon az akkori állapotukra törtlént.

1. REST: <https://restfulapi.net/> [↑](#endnote-ref-1)
2. Json: <https://www.json.org/json-en.html> [↑](#endnote-ref-2)
3. MySQL: <https://www.mysql.com/> [↑](#endnote-ref-3)
4. Spring Boot: <https://spring.io/projects/spring-boot> [↑](#endnote-ref-4)
5. Apache Tomcat: <http://tomcat.apache.org/> [↑](#endnote-ref-5)
6. Apache Maven: <http://maven.apache.org/> [↑](#endnote-ref-6)
7. ByCryptPasswordEncoder: <https://docs.spring.io/spring-security/site/docs/4.2.14.RELEASE/apidocs/org/springframework/security/crypto/bcrypt/BCryptPasswordEncoder.html> [↑](#endnote-ref-7)
8. JpaRepository: <https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html> [↑](#endnote-ref-8)
9. ResponseEntity: <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/ResponseEntity.html> [↑](#endnote-ref-9)
10. Uniform Resource Identifier, egységes erőforrás-azonosító [↑](#endnote-ref-10)
11. SpringSecurity: <https://spring.io/projects/spring-security> [↑](#endnote-ref-11)
12. Basic Authentication: <https://en.wikipedia.org/wiki/Basic_access_authentication> [↑](#endnote-ref-12)
13. UserDetailService: <https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/userdetails/UserDetailsService.html> [↑](#endnote-ref-13)
14. JUnit: <https://junit.org/junit5/> [↑](#endnote-ref-14)
15. Mockito: <https://site.mockito.org/> [↑](#endnote-ref-15)
16. Angular: <https://angular.io/> [↑](#endnote-ref-16)
17. Type Script: <https://www.typescriptlang.org/> [↑](#endnote-ref-17)
18. Angular CLI: <https://cli.angular.io/> [↑](#endnote-ref-18)
19. npm: <https://www.npmjs.com/> [↑](#endnote-ref-19)
20. FontAwsome: <https://fontawesome.com/> [↑](#endnote-ref-20)
21. Bottstrap: <https://getbootstrap.com/> [↑](#endnote-ref-21)
22. RxJS: <https://rxjs-dev.firebaseapp.com/> [↑](#endnote-ref-22)
23. ng2-charts: <https://valor-software.com/ng2-charts/> [↑](#endnote-ref-23)
24. Agm: <https://angular-maps.com/> [↑](#endnote-ref-24)
25. Flutter: <https://flutter.dev/> [↑](#endnote-ref-25)
26. Provder: <https://pub.dev/packages/provider> [↑](#endnote-ref-26)
27. FontAwsome: <https://pub.dev/packages/font_awesome_flutter> [↑](#endnote-ref-27)
28. Enum-to-string: <https://pub.dev/packages/enum_to_string> [↑](#endnote-ref-28)
29. Shared preferences: <https://pub.dev/packages/shared_preferences> [↑](#endnote-ref-29)
30. Notifications: <https://pub.dev/packages/flutter_local_notifications> [↑](#endnote-ref-30)
31. Google-maps: <https://pub.dev/packages/google_maps_flutter> [↑](#endnote-ref-31)
32. Geolocator: <https://pub.dev/packages/geolocator> [↑](#endnote-ref-32)
33. Statless Widget: <https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html> [↑](#endnote-ref-33)
34. Statefull Widget: <https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html> [↑](#endnote-ref-34)
35. State: <https://api.flutter.dev/flutter/widgets/State-class.html> [↑](#endnote-ref-35)
36. Navigator: <https://api.flutter.dev/flutter/widgets/Navigator-class.html> [↑](#endnote-ref-36)
37. Scaffold: <https://api.flutter.dev/flutter/material/Scaffold-class.html> [↑](#endnote-ref-37)
38. ChangeNotifier: <https://api.flutter.dev/flutter/foundation/ChangeNotifier-class.html> [↑](#endnote-ref-38)
39. FlutterNotificationPlugn: <https://pub.dev/documentation/flutter_notification_plugin/latest/flutter_notification_plugin/FlutterNotificationPlugin-class.html> [↑](#endnote-ref-39)
40. Mac: <https://www.apple.com/hu/mac/> [↑](#endnote-ref-40)