Szalkai Krisztián Önálló laboratórium Dokumentáció

YN6AKK

Tartalomjegyzék:

[2 Az alkalmazás specifikációja 1](#_Toc59047467)

[3 A weboldal működésével kapcsolatos elvárások 2](#_Toc59047468)

[4 Tervezés 2](#_Toc59047469)

[4.1 Adatbázis struktúra 2](#_Toc59047470)

[4.2 Felülettervek 4](#_Toc59047471)

[4.3 Ütemezés 9](#_Toc59047472)

[5 Megvalósítás 9](#_Toc59047473)

[5.1 Backend 9](#_Toc59047474)

[5.1.1 Adatbázis elérési réteg (DAL) 10](#_Toc59047475)

[5.1.2 Adatátviteli osztályok (Dto) 11](#_Toc59047476)

[5.1.3 Web API réteg 11](#_Toc59047477)

[5.1.4 Üzleti logikai réteg (BLL) 16](#_Toc59047478)

[5.2 Frontend 17](#_Toc59047479)

[5.2.1 Felépítés 17](#_Toc59047480)

[5.2.2 Regisztráció/ Bejelentkezés 19](#_Toc59047481)

[5.2.3 Foglalás 21](#_Toc59047482)

[5.2.4 Felhasználói adatok kezelése 22](#_Toc59047483)

[5.3 Összefoglalás 24](#_Toc59047484)

# Az alkalmazás specifikációja

Az elkészítendő alklamazás egy taxi szolgálat webolda, melyen a felhasználók tudnak regisztrálni, bejelntekzni, időpontokat foglalni a különböző szolgáltatásokhoz, illetve kezelni tudják a saját adataikat, eddigi foglalásaikat. Az alkalmazás ezen kívül lehetőséget nyújt az adminisztrátoroknak az összes leadott foglalás kezelésére, az adminisztrációs tevékenységek elvégzésére.

A taxiszolgálat által nyújtott szolgáltatások a következők:

* Pontból pontba történő fuvar meghatározott időpontban.
* Adott időtartamra auto bérelése folyamatos sofőrszolgálattal.

Mindkét szolgáltatás esetén meghatározható a foglaláshoz, hogy milyen típusú autóval, milyen extra szolgáltatásokkal (dohányzó/nem dohányzó, légkondi kisállatos, stb) kéri a felhasználó.

A weboldalon a felhasználó a következő funkciókat kell elérje:

* Regisztráció
* Belépés
* Foglalás a két féle szolgáltatásra
* Saját adatok megváltoztatása, törlése
* Előző foglalások megtekintése
* Foglalás lemondása (Adott határidőn belül)
* Kijelentkezés
* Online fizetés

Adminisztrátori belépéssel:

* Foglalások megtekintése
* Szabadnapok felvétele
* Felhasználók és foglalásaik megtekintése
* Sofőrök/ autók felvétele, kezelése

Az alkalmazás backend része .NET CORE technológiával készül, Entity Framework Core felhasználásával az adatbázis eléréséhez. Az adatbázist fel kell készíteni több féle Adatbázis kezelő használatára is (MsSql, PostgreSql). A frontend felület React keretrendszer használatával készül.

# A weboldal működésével kapcsolatos elvárások

A weboldalnak minden személyes adatot biztonságosan kell tárolnia, az adatokhoz csak a megfelelő jogosultságú felhasználók férhetnek hozzá. A személyes adatok illetve a felhasználói profil a felhasználó kérésére bármikor törölhető kell legyen.

Foglalás során a rendszernek automatikusan kell elvégezni a következőket:

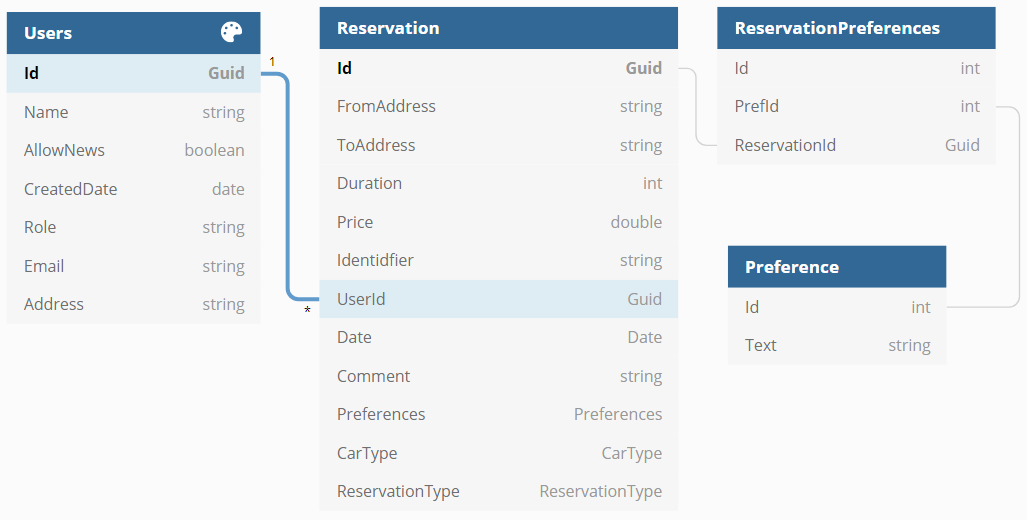
* Ár számítása a preferenciák, extra szolgáltatások, szolgáltatás típusa és az esetleges felárak ( pl: Dugódíj) alapján.
* Átirányítás biztonságos fizetési lehetőségekhez (pl: Stripe, PayPal, Barion). Fizetés esetén az esetleges problémák kezelése (Sikertelen fizetés, helytelen összeg, hálózati prbléma).
* A foglaláshoz egyedi azonosító generálása és megerősítő email küldése a felhasználónak.

# Tervezés

A terveztési folyamatot az adatbázis megtervezésével kezdtem, majd tovább haladtam a frontend alapvető felépítésének megterverzésére, végül megalkottam a félév során követendő ütemtervet.

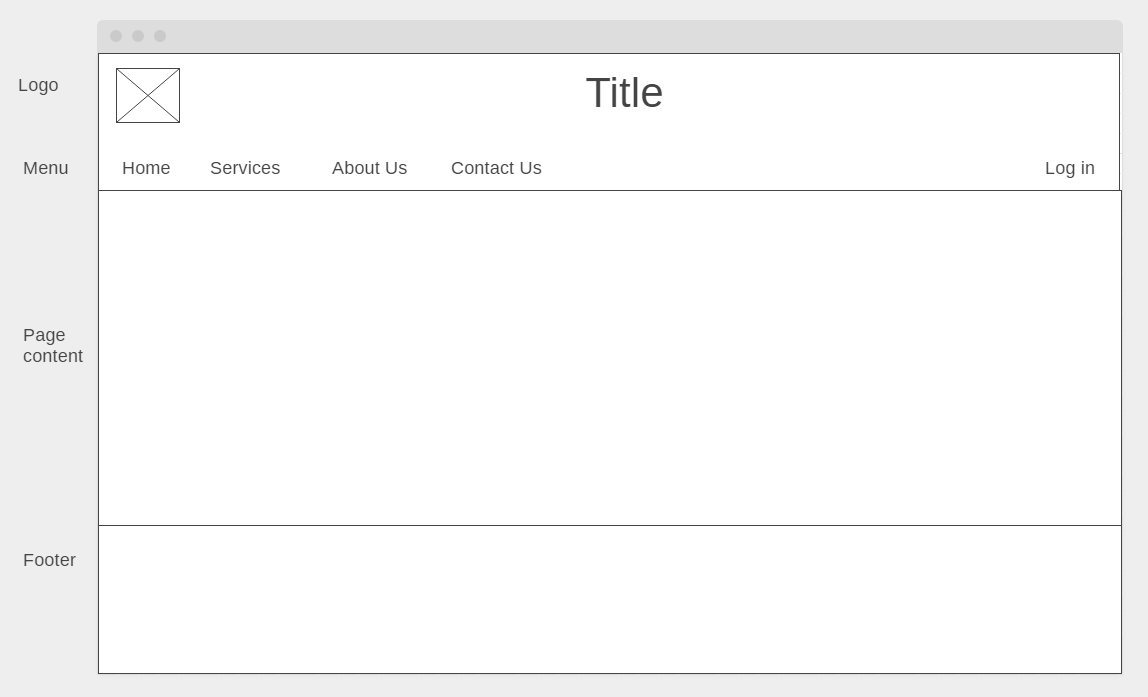
## Adatbázis struktúra

A tervezés során a specifikációban meghatározott funkciókat helyeztem előtérbe. A felhasználók tárolásakor az általános mezőkön kívül felvételre került az “AllowNews” mező, melynek szerepe, hogy tárolja a felhasználóról, hogy kíván-e promóciós anyagokat kapni. A foglalás során a következő adatokat kell tárolni:

A foglalás ideje, honnan hova, vagy időre szóló foglalás esetén honnan mennyi időre, opcionális megjegyzés, a számított ár, a foglalás típusa, a kért autó típus, illetve a foglaláshoz tartozó egyéb preferenciák - mint dohányzó/nem dohányzó, állatos, nagy csomagtartó, stb. A foglalás során szükség van még egy egyedi azonosítóra, mellyel a felhasználó a foglalásra hivatkozhat.

A tervezett adatstruktúra

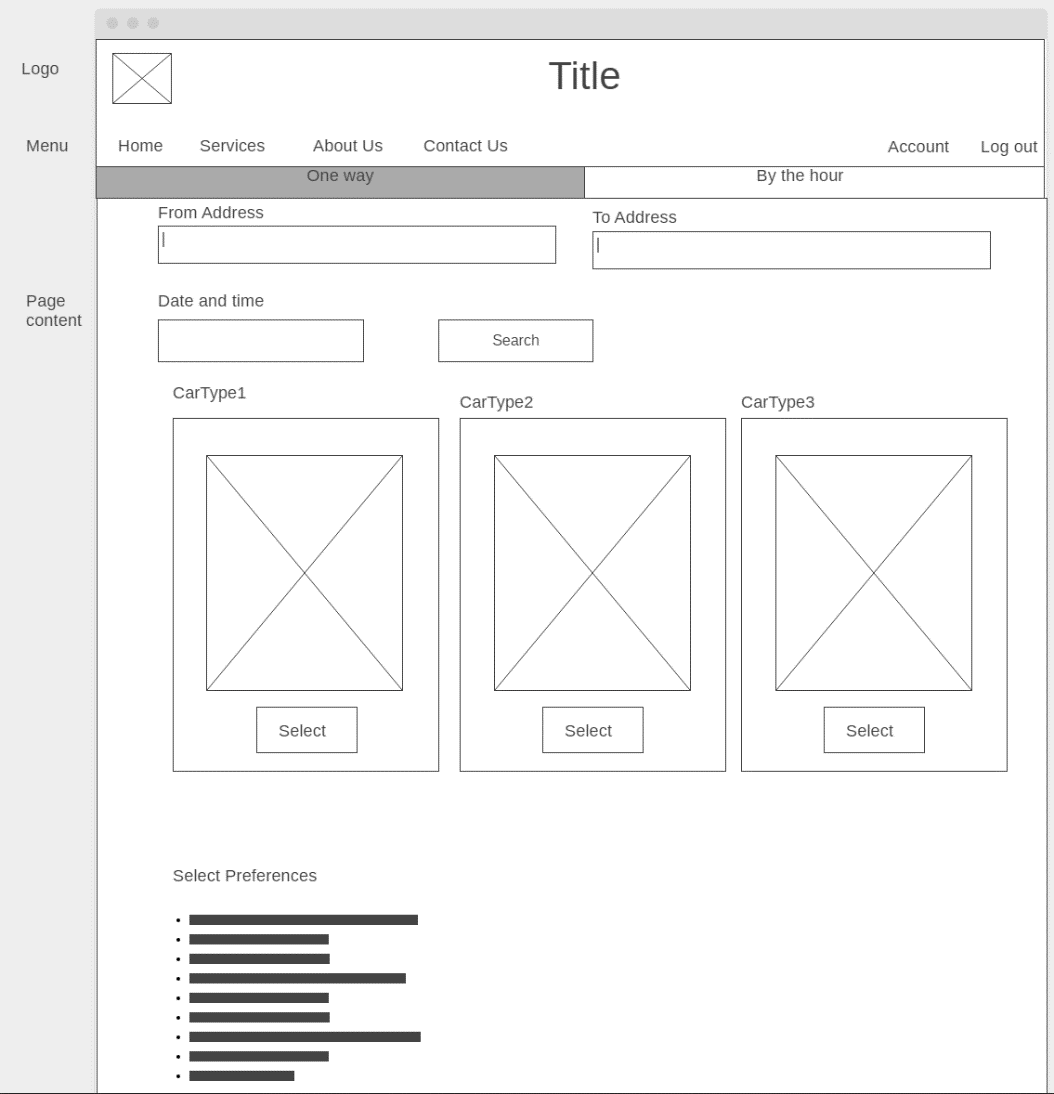
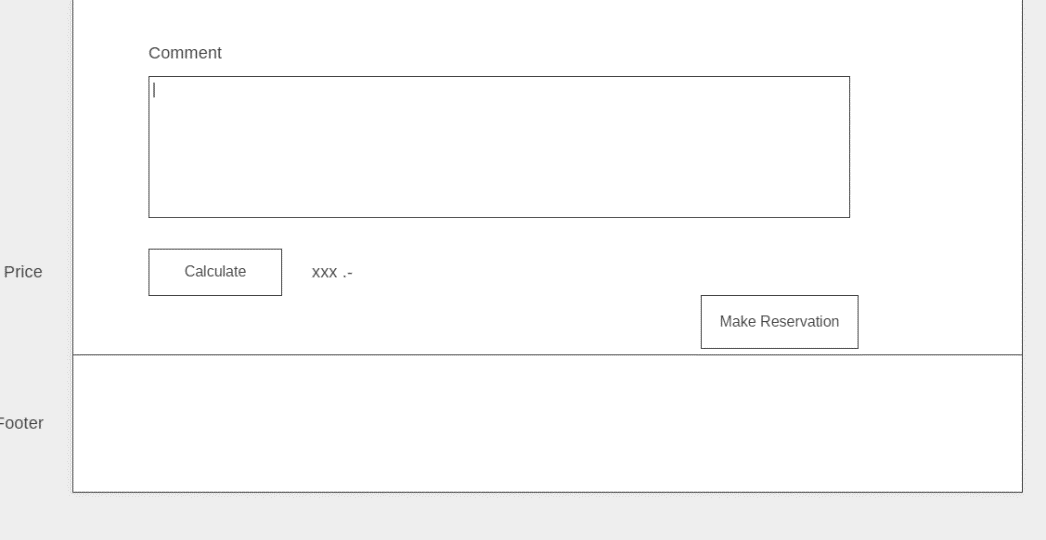
## Felülettervek

A tervezést a szükséges felületek véggigondolásával és megtervezésével folytattam. A felületeket WireFrame segítségével készítettem. A felülettervek készítése nagyban elősegíti a webszerver végpontjainak megalkotását, struktúrálását.

Általános layout

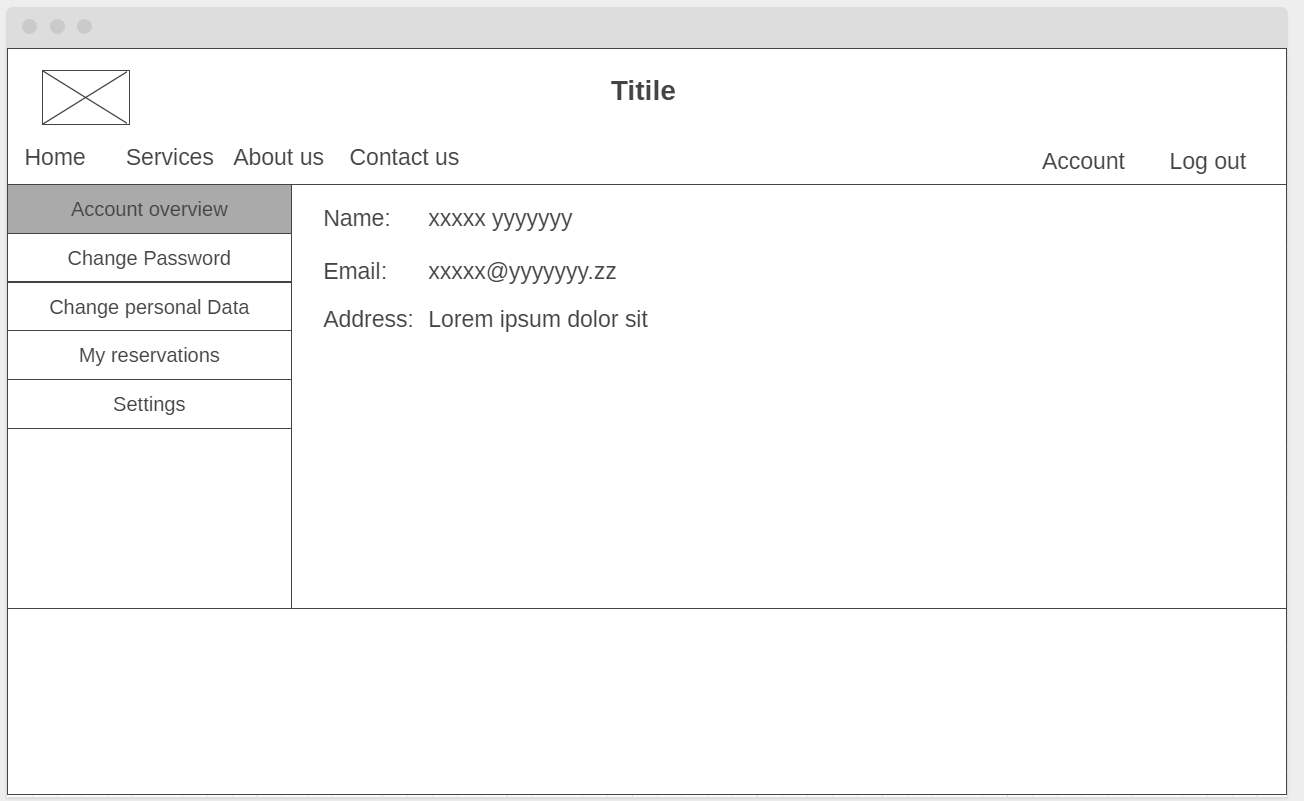
Az alkalmazás alapvető felépítése a fenti ábrán látható. A lap tetején mindig egy Header látszik, melyen megjelnik a logó a szolgáltatás neve és a navigációs menü. A lap alján egy Footer található, melyen helyet kaphatnak a kapcsolatfelvételi információk, felhasználási feltételek, oldaltérkép, stb. A kettő között helyezkednek el a különböző oldalak tartalmai.

1. Foglalás

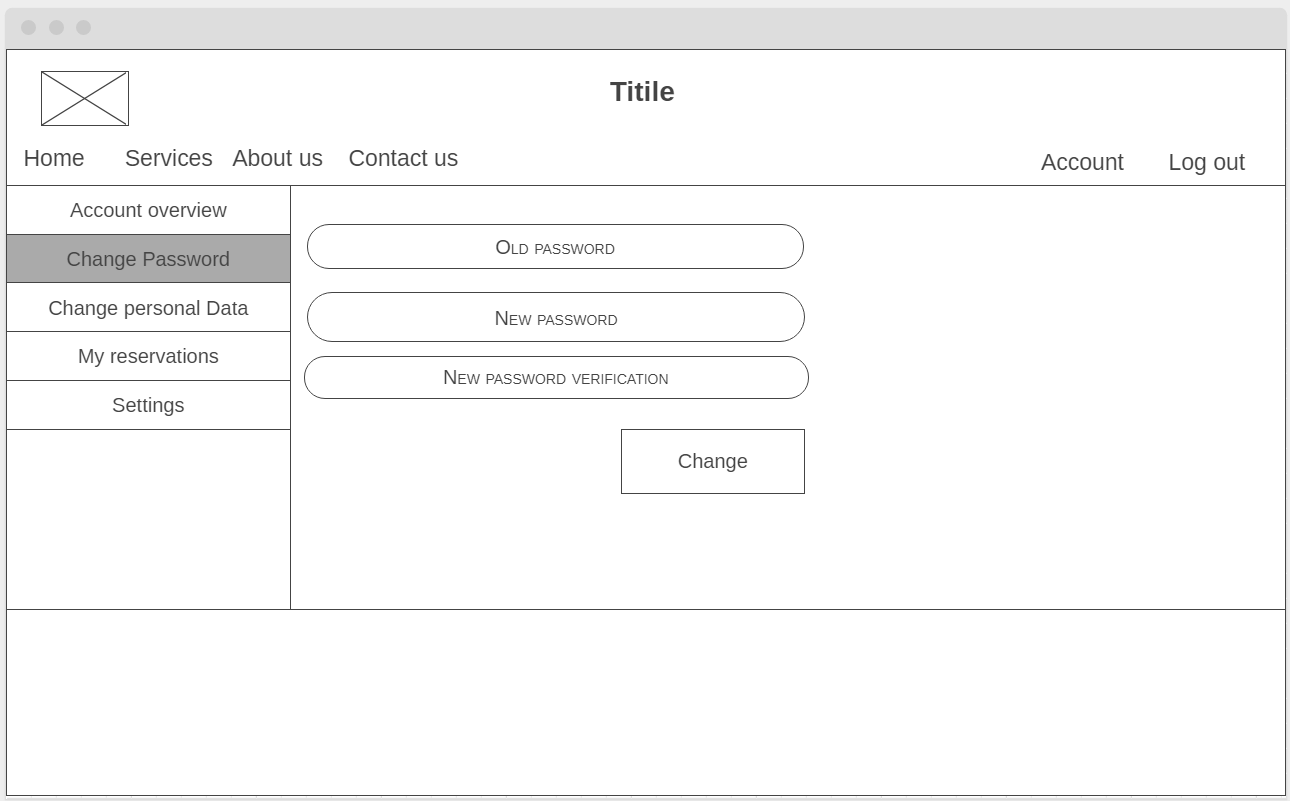


Az 1.-es képen látható a foglalás terve. Az összes adat megadásával a felhasználó lekérdezheti az árat a szervertől, majd foglalhat a megadott paraméterekkel.

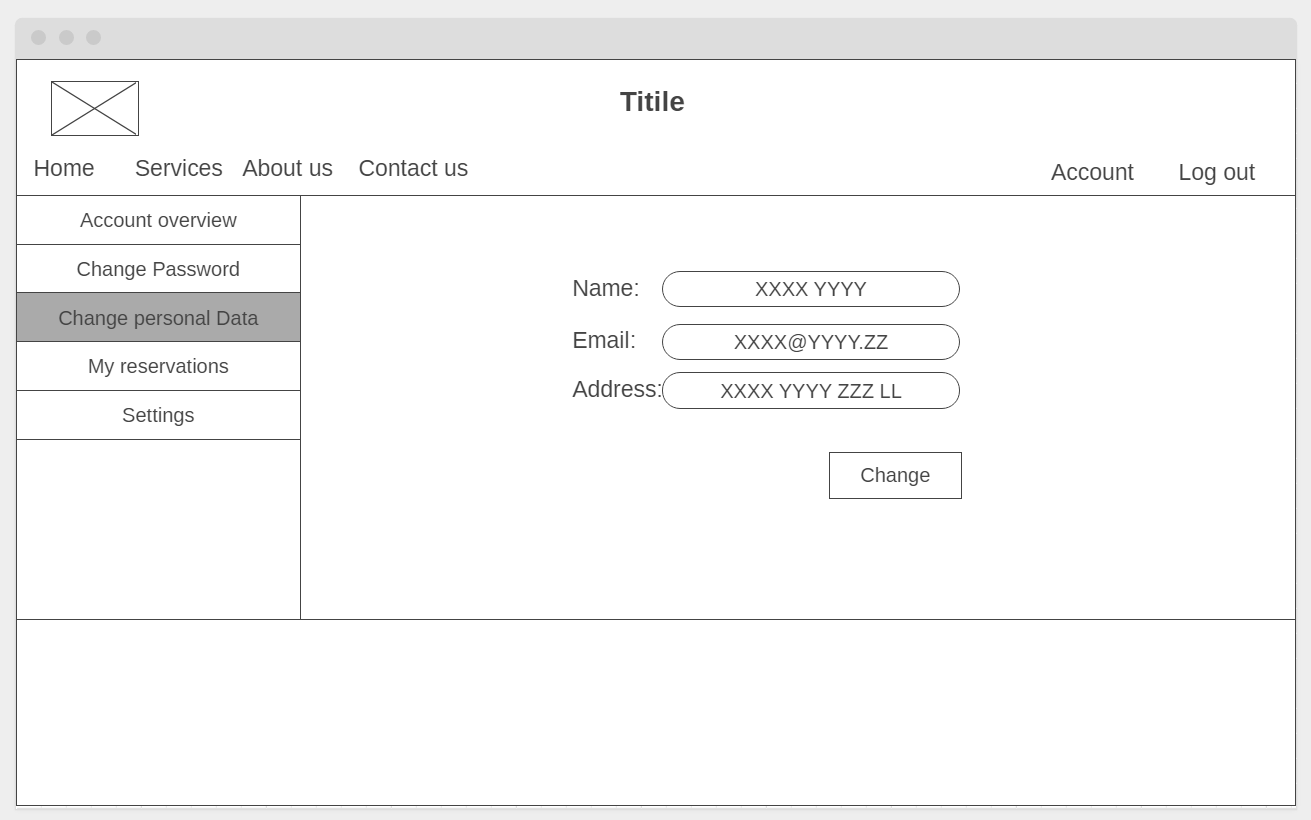
A 2.-5. képeken láthatóak a felhasználókezeléssel kapcsolatos oldalak. A felhasználónak lehetősége van a jelszava megváltoztatására, személyes adatainak kezelésére, saját foglalásainak megtekintésére, illetve beállításainak változtatására.



2. Személyes adatok áttekintése

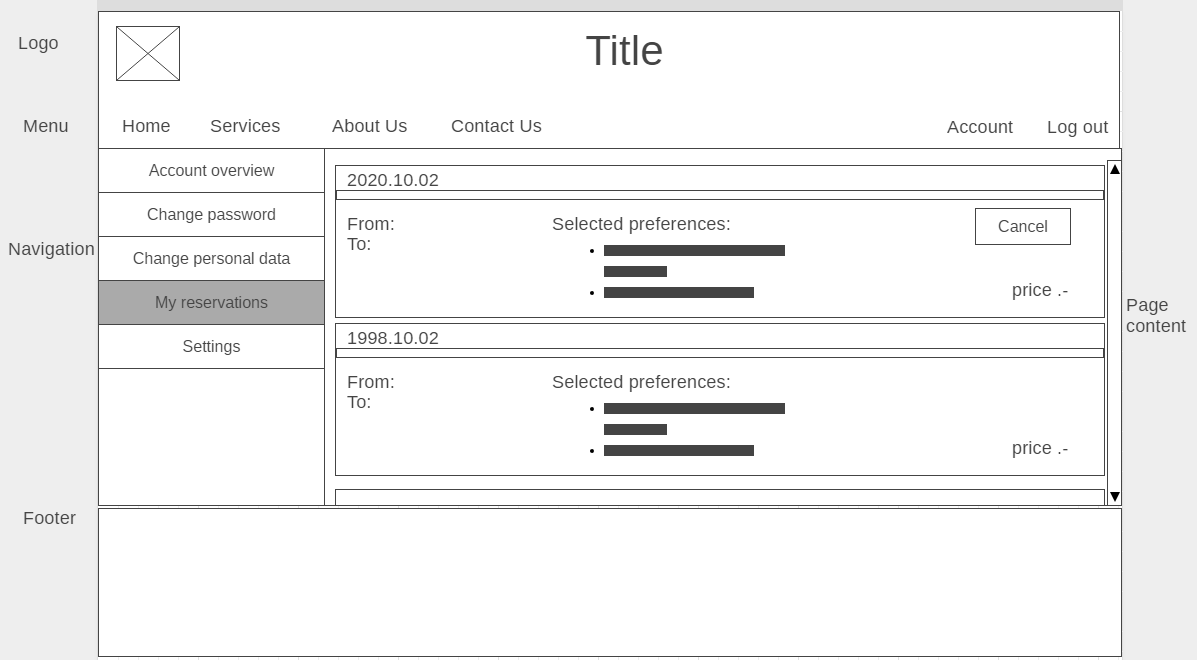
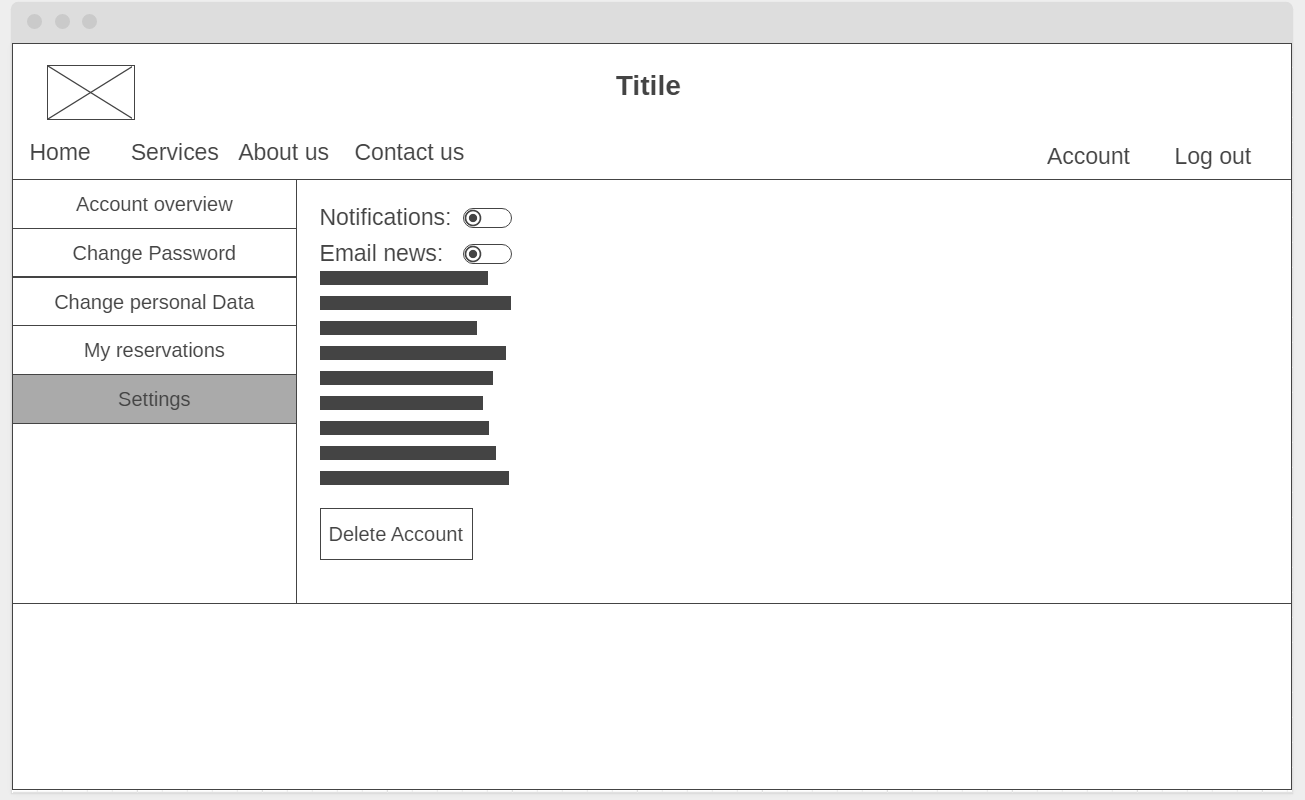


3. Jelszó megváltoztatása



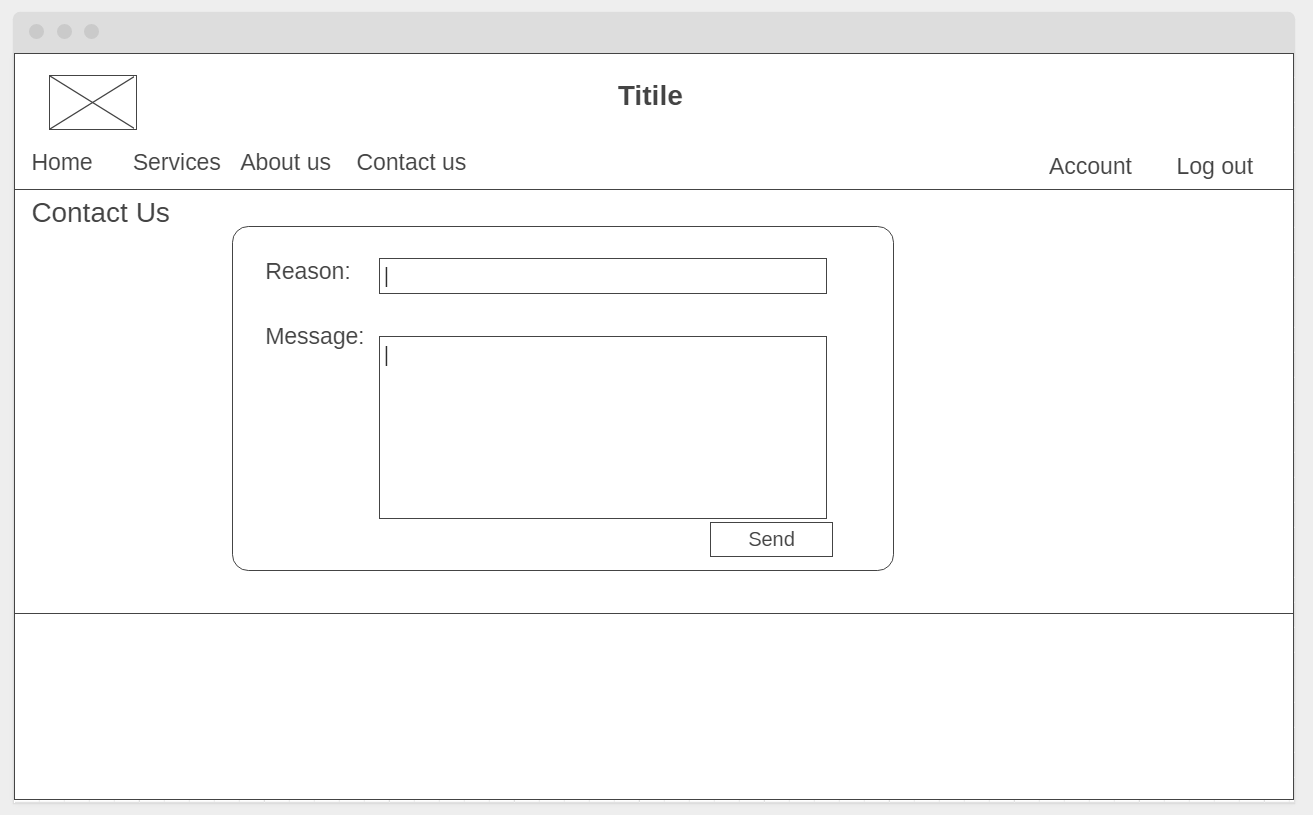
4. Személyes adatok kezelése

A saját foglalások megtekintése lapon lemondható a foglalás ha egy adott határidőn kívül van.

5. Saját foglalások és Beállítások

Az utolsó wireframen a contact formot láthatjuk, melyel automatikus emailt küldhetnek a felhasználók. A többi oldlaról nem készítettem tervet, mivel főleg szöveges tartalmak lesznek backend integráció nélkül.



## Ütemezés

Az ütemezés az alábbi linken található:

<https://docs.google.com/spreadsheets/d/1PN8QogsAB1-gRkF9XY7Tzh9eF9Rkxda7Zg41fPSTu_c/edit?usp=sharing>

A párhuzamos fejlesztés helyett előre vettem a backend fejlesztését a wireframek alapján, majd utána fejlesztettem frontendet.

# Megvalósítás

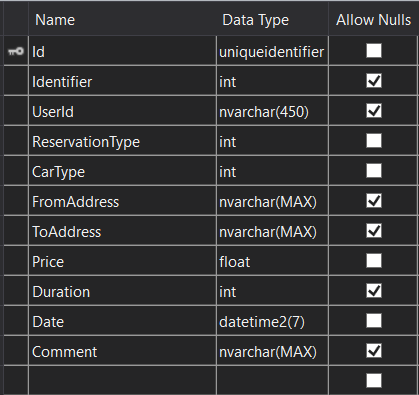
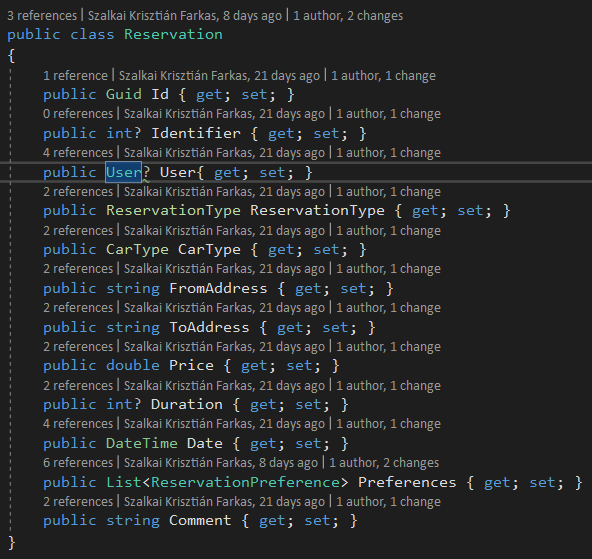
A következőkben kitérek a fejlesztés részleteire időrendben haladva és bemutatom a fejlesztés során felmerült problémákat, illetve megoldásokat.

## Backend

Ahogy az ütemezés résznél is említettem, a fejlesztést a backend elkészítésével kezdtem. A szerver egy .Net Core Web API melyen a szolgáltatásokat HTTPS kérések segítségével lehet elérni bizonyos végpontokról. Ennek a megoildásnak az az előnye, hogy asd péojdfvosdf

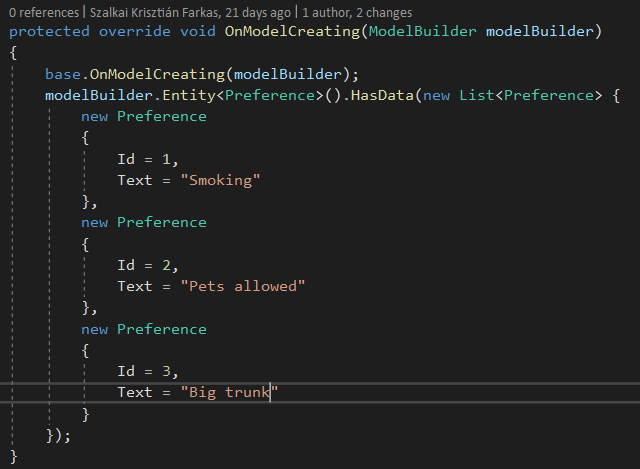
### Adatbázis elérési réteg (DAL)

Első lépésként létrehoztam a projekteket és az adatstruktúrát. Az adatbázist code-first megközelítéssel építettem, ez azt jelenti, hogy az adatbázis táblái és mezői a kódban szereplő osztályokhoz igazodnak. A következő képeken például a foglalások leképezése látható a hozzá tartozó adattáblába.



6. Egy POCO osztály és leképezése

Több-több kapcsolat esetén- mint itt a Preferences – explicit meg kell határozni egy kapcsolótáblát. Az ilyen és ehhez hasonló konfigurációs beállításokat könnyen felvehetjük az Entity Framework Core (EFC) beépített függvényeket tartalmaz a konfigurációra. Az olyan adatokat is itt tudjuk létrehozni amelyeket az adatbázis létrejöttével szeretnénk ha az tartalmazna, ezt a folyamatot seedelésnek nevezzük, így vesszük fel például a szolgáltatás által níújtott preferenciákat is.

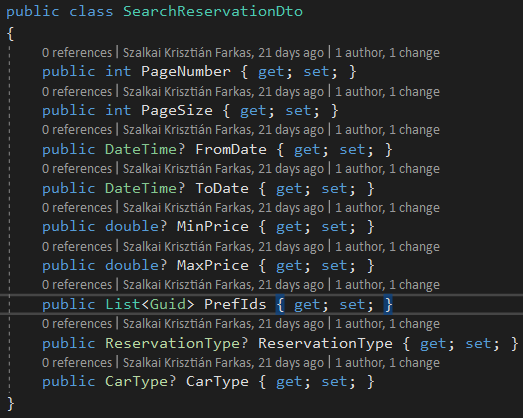
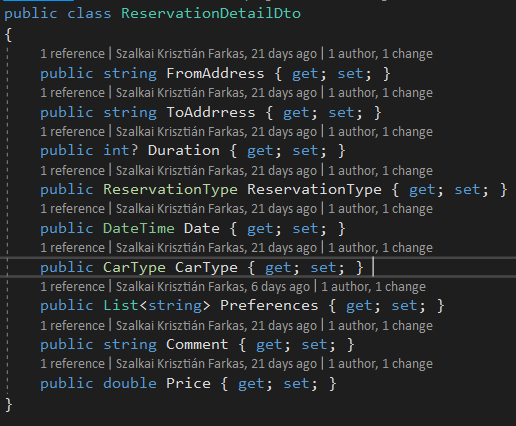
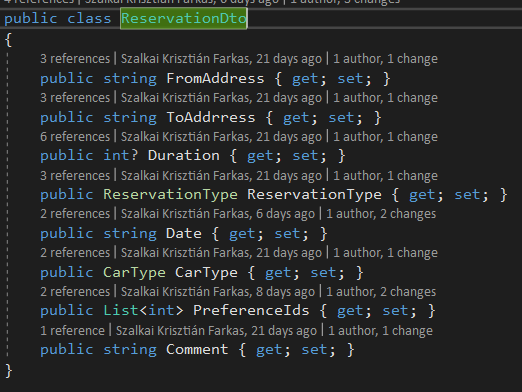


7. Seedelés

Az ilyen módon létrehozott adatbázis előnye, hogy az EFC keretrendszer képes tetszőleges adatbázis típusokat létrehozni, csak a Migrációs Scripteket kell újragenerálnunk melyekkel a rendszer létrehozza a tényleges adatbázist. Az alkalmazás készítése során MSSQL adatbázist használtam, azonban a funkcionális követelményeknek megfelelően[fent] generálható migrációs script a legtöbb relációs adatbázishoz, sőt néhány NoSql tárhoz is.

### Adatátviteli osztályok (Dto)

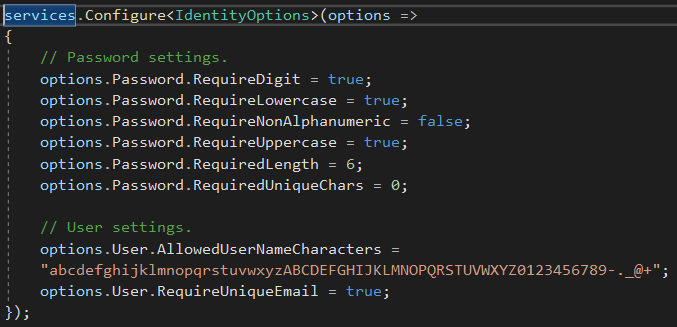
Az adatbázis entitásait több szempontból sem érdemes egy az egyben kiadni a felhasználók felé. Egyrészt szenzitív információt is tárolunk bizonyos osztályokban így érdemes ezeket elrejteni a felhasználók elől, másrészt felesleges is minden lekérdezésnél minden adatot elküldeni, így teljesítmény szempontjából is szerencsésebb csak a lényeget küldeni. A legtöbb esetben egy entitásnak van egy lista „nézete” egy részletes és keresési paramétereket tartalmazó. Az alábbi ábrákon látható például a foglaláshoz tartozó DTO osztályok



8. DTO objektumok foglaláshoz

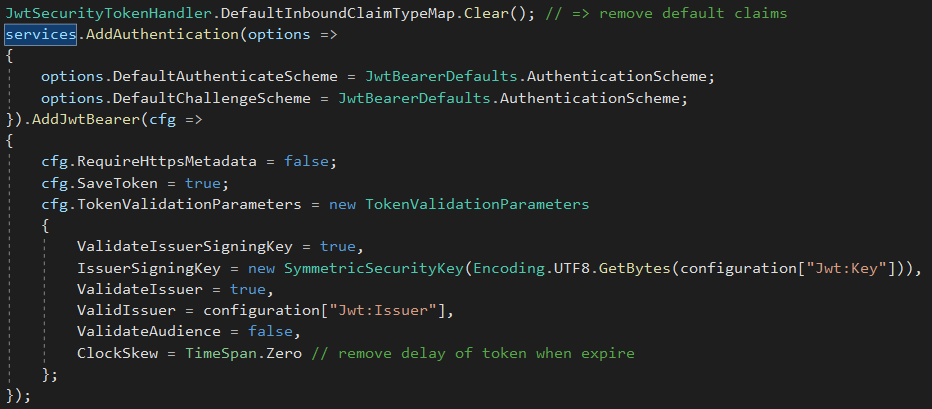
### Web API réteg

A .Net Core keretrendszer az API-hoz beérkező kéréseket egy pipeline architektúra segítségével dolgozza fel, mely a kódból tetszőlegesen konfigurálható. A felhasználók kezelését az Identity Core NuGet csomag segítségével, oldottam meg mely könnyű felhasználókezelést biztosít .Net Core webalkalmazásokhoz, ezen kívül JWT tokent használtam magának a kliensnek/ felhasználónak az azonosítására.



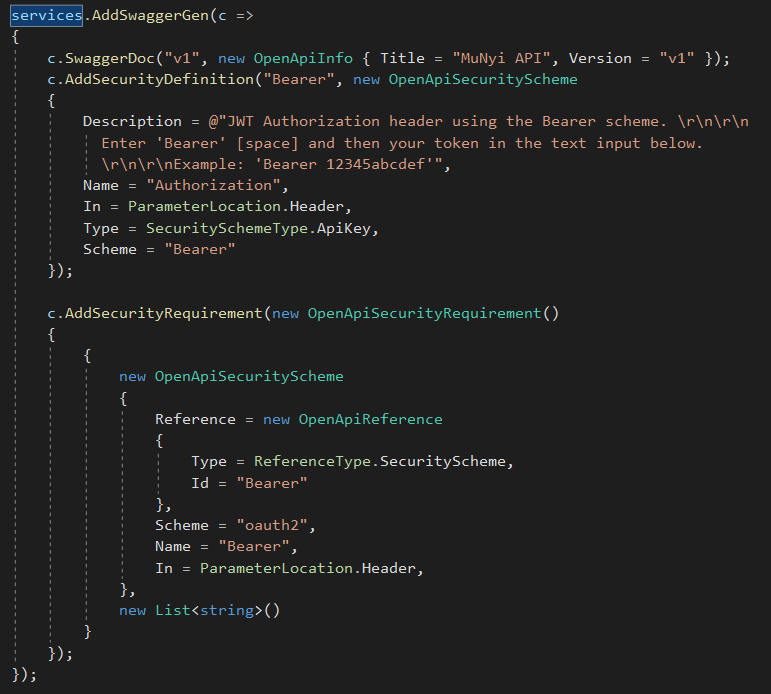
9. Identity Core konfigurálása

A 9-es ábrán látszik, hogy egyszerű tulajdonságok állításával konfigurálható a felhasználókezelő beállításai. Hasonlóképpen, a JWT token konfigurálása is ilyen tulajdonságokkal történik, amit a 10-es ábra szemléltet.



10. JWT token konfigurációja

A fejlesztés során a Swagger csomagot használtam tesztelésre, míg nem implementáltam a frontendet, ez egy OpenApi alapú Middleware melynek feladata, hogy az API végpontjai kipróbálhatóak legyenek a fejlesztés közben. A definiált végpontok alapján készít egy htmlben elérhető alkalmazást melyre legenerálja az összes hívható függvényt és hozzá a hívásokat is, ezen kívül képes dokumentációt is generálni egy jól kommentezett kódbázishoz, ám ezt a funkciót nem használtam. Az alábbi ábrán látható a Swagger konfigurálása, a JWT token használata miatt kicsit bonyolultabb beállítani, de többnyire itt is beépített tulajdonságokat használhatunk.



11. Swagger konfiguráció

A három rétegű architektúra használata miatt itt kell regisztrálnunk a végpontkezelőink által használt szolgáltatásokat is, illetve az alkalmazásomban használok egy saját készítésű Middleware-t is, mely a globális hibakezelést segíti. Hibakezelő middleware használata megakadályozza, hogy egy kezeletlen hiba kijusson a felhasználó felé éles környezetben, ezzel érzékeny adatokat szivárogtatva a rendszerről.



12. Hibakezelő Midldeware

A fenti képen láthatjuk, hogy minden kérés lefutása során elkapjuk a kivételeket és ezek típusától függően egységes hibaüzenetet képez, ezen kívül a szervert futtató környezet konfigurációja alapján fejlesztői módban a hiba üzenetét és a stack trace-t is elküldi, ám éles futás esetéén sosem engedi ezeket a felhasználó felé kezeletlen hiba esetén sem.

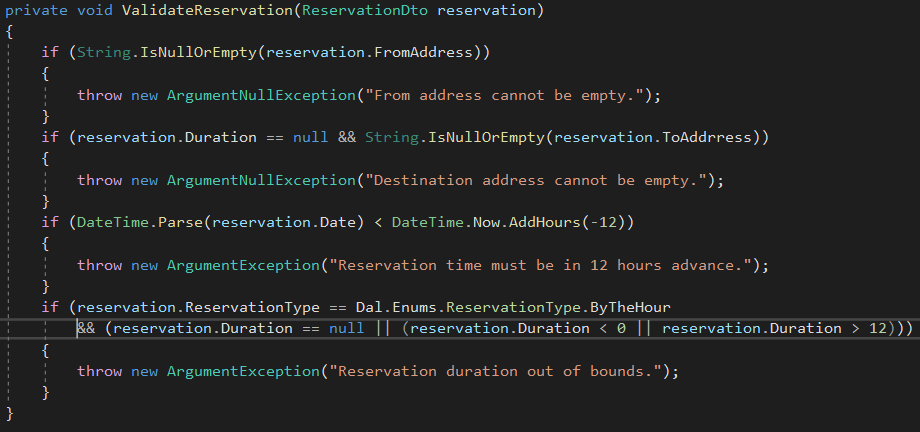
A végpontok felépítését a funkciók mentén választottam el. A Controllerek szétválasztása nem szükséges lépés ám segít a kód átláthatóságán, karbantarthatóságán.

#### Preference controller

Ez a komponens felelős a preferenciák kezeléséért, a „/preference” végponton érhető el. Jelenleg egyetlen funkciója, hogy a kliensnek visszaadja az összes választható preferenciát, ám a jövőben szükség lehet a rekordok módosítására, hozzáadására, törlésére például egy adminisztrációs felületről.

#### Reservation Controller

A foglalásokhoz kapcsolódó végpontok ebbe a komponensbe kerültek, a „/reservation” útvonal alatt érhető el. Itt találhatók a foglalások keresése, a lekérdezése, létrehozása és lemondása végpontok. Mikor egy kérés beérkezik, több validációs lépésen keresztül megy mielőtt kiszolgálásra kerül. Először is, csak olyan címről fogadunk kéréseket, melyek a szerver konfigurációja során megadásra kerültek. Második lépésben, ha a kérést egy bizonyos autorizációs szinthez kötöttük, - a mi rendszerünkben a bejelentkezés és regisztrálás kivételével minden kérés ilyen – a szerver összehasonlítja a kérésben érkező JWT tokent, az elvárttal és ez alapján dönti el, hogy az adott kliens jogosult-e a kérés meghívására. A Controllerekben szereplő végpontokra külön-külön változtathatjuk az elvárt szerepköröket annotációk segítségével. Amennyiben az előző ellenőrzések sikeresek voltak, a szerver megkísérli kiszolgálni a kérést. Mivel a kliens oldalon bármi megváltoztatható az ahhoz értő és a rendszerünkben kárt tenni kívánó egyének által, ezért érdemes saját validációkat is végezni a kérésben szereplő adatokon. Erre egy példát láthatunk a 13. ábrán, melyben a foglalási kérések validációját valósítottam meg.

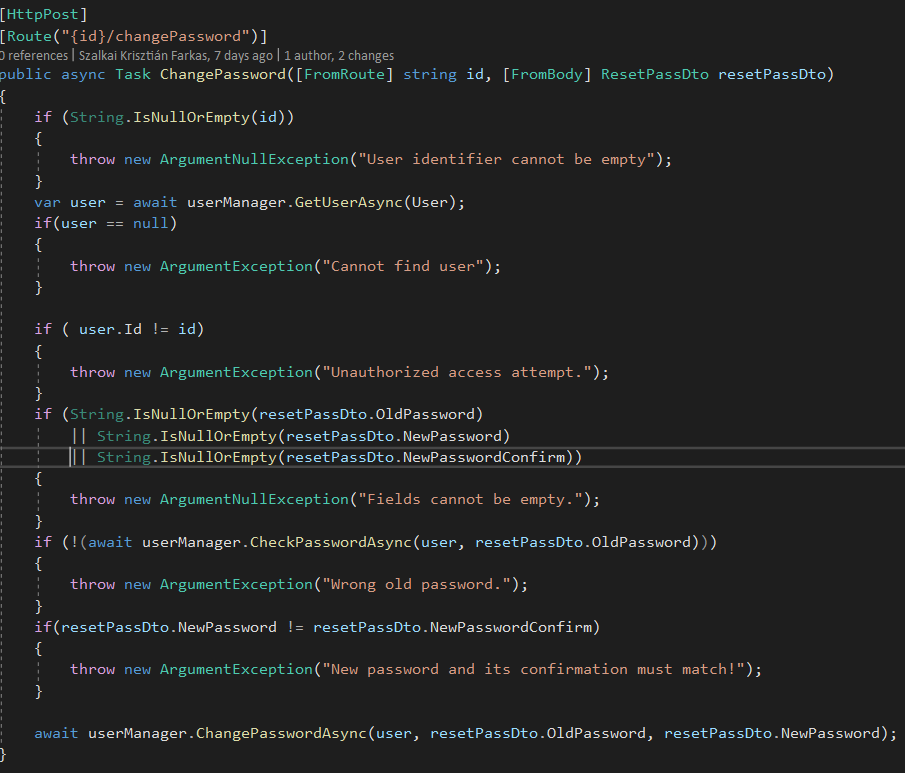


13. Foglalások validációja

Azzal, hogy a hívás ezen pontján validálunk, nem csak a támadóktól védjük a rendszert, hanem a teljesítményt is növeljük hiszen ezen a ponton még nem került példányosításra az üzleti logikai komponens, nem történt adatbázis hívás sem.

#### User controller

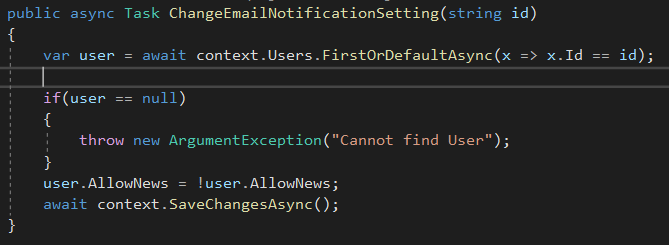
A „/user” útvonal alatt találhatóak a felhasználókkal kapcsolatos végpontok. A felhasználó regisztrál a rendszerbe, itt is fontos, hogy a validációkat elvégezzük a bemeneti mezőkre és a felhasználó felé jelezzük is, hogy ha valami problémát találtunk, például a jelszó nem egyezik a megerősítő jelszóval, vagy hiányzik belőle egy szám, pedig ezt kötelezőnek tekintjük. Regisztráció után a felhasználó bejelentkezhet a rendszerbe, ez után már el tudja érni a többi funkciót is. A bejelentkezés a szerver szempontjából nem más, mint a megadott adatok alapján ellenőrizni, hogy valóban egy regisztrált felhasználó kívánja használni a rendszert, majd ha mindent rendben talált generál egy token-t, mellyel a kliens a továbbiakban minden egyes kérését validálni tudja. A felhasználói adatok megváltoztatásáért is ez a komponens felelős, itt különösen fontos, hogy ne higgyünk semmi másnak ami a kliensről érkezik, csak az általunk generált tokennek, például jelszó változtatás az alábbi ábrán látható módon zajlik.



A végponthoz a kliens a korábban megkapott felhasználó azonosítóját használja, azonban le kell ellenőriznünk, hogy ez valóban az általunk generált tokenben szereplő felhasználó-e. Ha ezt nem tennénk meg, más felhasználó azonosítójának birtokában megváltoztathatnánk a jelszavát!

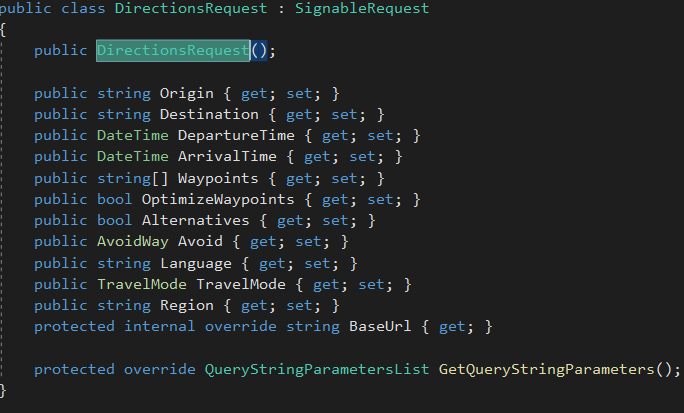
### Üzleti logikai réteg (BLL)

Ebben a rétegben találhatók a hívások azon részei melyek az adatokon végzett tényleges műveletekért felelősek. A komponensek és azok definíciói el vannak választva egymástól interfacek segítségével, így az esetleges változások a működésben transzparensek az azokat felhasználó rendszerek felé. Ilyen műveletre ad példát a 14.ábra melyen egy egyszerű adatelérés látszik, mellyel a felhasználó beállításai között módosítunk egy elemet.



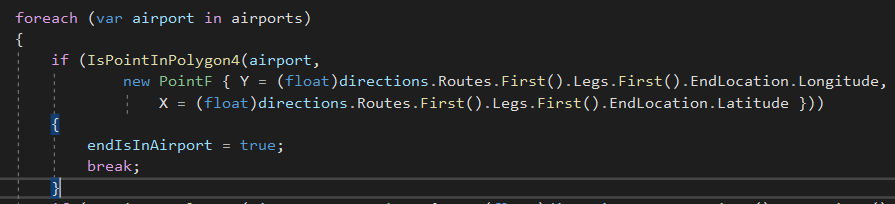
14. Email értesítési beállítás megváltoztatása

A szerver megalkotása során sok műveletet kellett létrehozni, ezért csak az érdekesebbeket emelném ki, ilyen például az ár számítása. Az útvonal kiszámításához a Google Places Api nyilvános csomagot használtam melynek segítségével különböző adatokat kérhetünk le két pont megadásával.



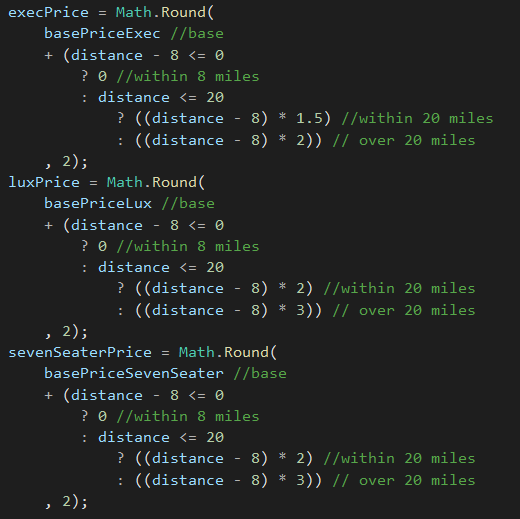
15. Google places api kérés

Mivel az elkészítendő rendszer egy taxi szolgáltatás az útvonal tervezésénél autóra állítottam a Travel mode beállítást és így kérdeztem le a legrövidebb útvonalat a két felhasználó által megadott útvonal között. Az ár számításánál számos tényezőt figyelembe kellett venni. Repülőteret érintő útvonal esetén vagy díjköteles zónába hajtáskor például felárat kell számolni, ezt egy poligon tartalmazási algoritmus segítségével oldottam meg. Az alábbi ábrán szereplő airports tömbben szerepelnek az előre felvett repterek koordinátái, melyekre leellenőrizhetjük, hogy a lekért útvonal tartalmazz-e.



16. Poligon tartalmazás ellenőrzése

A fentiek figyelembevétele után kiszámítható az ár a 17. ábrán látható módon.



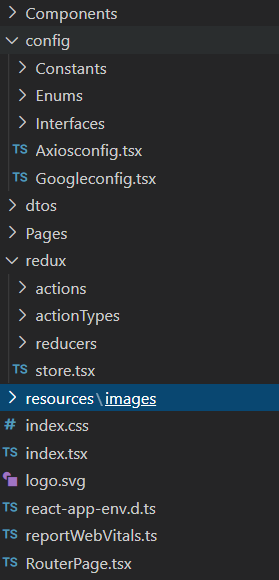
17. Árszámítás különböző autótípusokra

## Frontend

A végpontok létrehozása után a logika implementálásával párhuzamosan elkezdtem fejleszteni a weblapot az alkalmazáshoz. A React alkalmazások készítése során egyáltalán nem elvárás a komponensek szétválasztása, az egész weblap állhat egyetlen html fájlból, ám karbantarthatóság szempontjából sokat segít ha jól strukturáljuk a kódunk. A kódot typescript nyelvben írtam, mivel ennek sok előnye van a javascriptel szemben, többek között, hogy típusos, így az elgépelési és statikus hibák nagyrésze könnyen észrevehető, ellentétben elődjével, ahol órákat lehet tölteni egy elütött karakterből adódó hiba felgöngyölítésével.

### Felépítés

A rendszer struktúráját a bevezetésben említett okok miatt az alábbi módon határoztam meg:



A Components mappába kerültek az olyan felületi elemek kódjai, amik nem önmagukban alkotnak egy lapot, hanem beépített komponensekként működnek. Ilyen például a Header, Footer, vagy egy navigációs menü.

A config mappába helyeztem a használt modulok és technológiák konfigurációjához szükséges osztályokat, illetve a konstansok, interfacek, enumok kódjait.

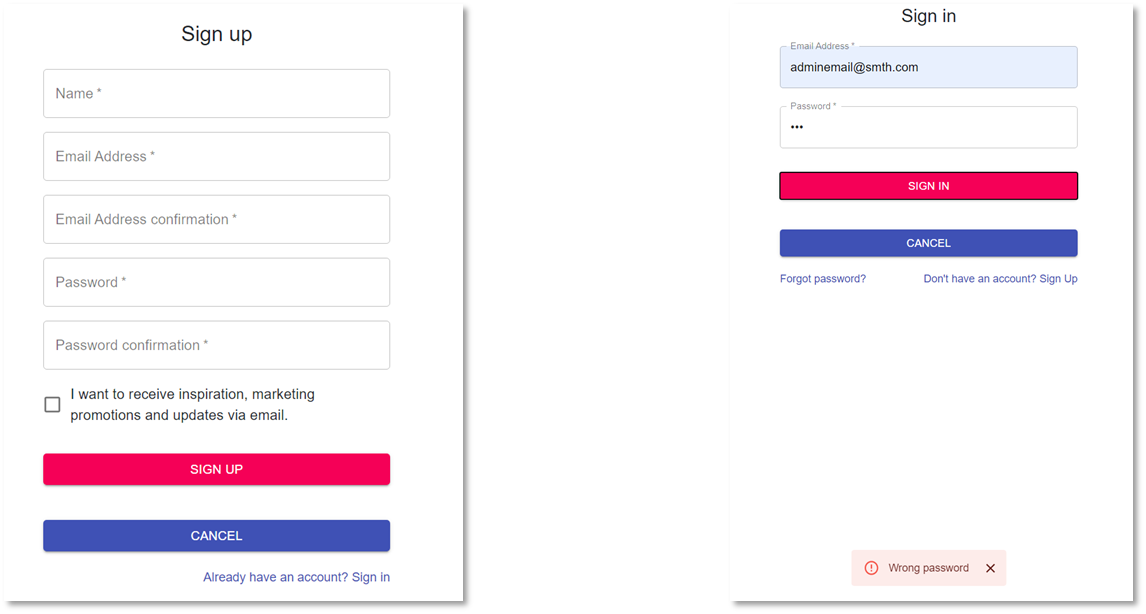
A dto mappa tartalmazza az előző[5.1.2] fejezetben említett hálózati kommunikáció során használt osztályok definícióit.

A Pages mappában találhatóak a teljes értékű lapok kódjai, mint a Login vagy a Reservation vagy a Contact Us, ezek felhasználjak a Components mappában található komponenseket.

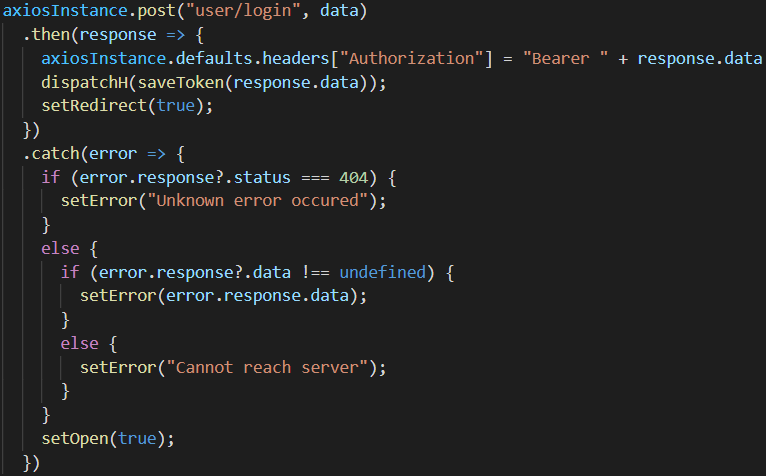
A redux mappában található minden ami a redux keretrendszer működéséhez szükséges. A redux egy olyan könyvtár mely lehetővé teszi az oldalak számára, hogy a teljes oldal újra töltése nélkül jelenítsen meg változásokat, illetve a különböző adatok melyek akár egyserre több helyen is megjelenhetnek egységes helyről kerüljenek kiírásra, változás esetén, megfelő módon frissüljenek. Az actions komponensek felelősek az adatokon végezhető műveletek definiálásáért, az actionTypes mappa a typescript nyelv miatt szükséges, így a komponens működése és definíciója elválasztható, a kód jobban átlátható lesz. A reducers mappában találhatók az adatok tárolásáért és manipulálásáért felelős osztályok.

Végül az index fájl az oldal belépési pontja, illetve a RouterPage a legkülső komponens, mely a különböző oldalak közötti navigálás lebonyolításáért felelős.

### Regisztráció/ Bejelentkezés

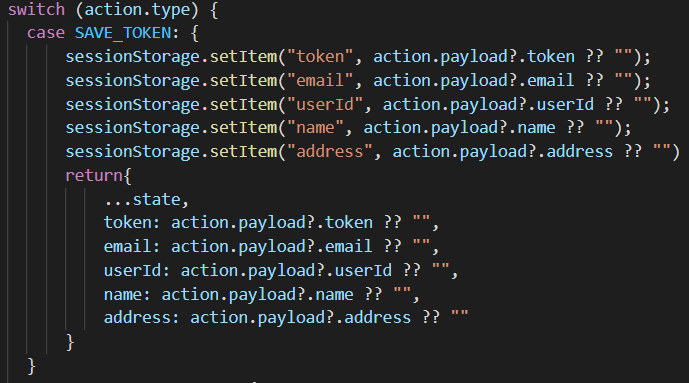


A felületek kialakítása során A Material-UI könyvtárat használtam, mely előre elkészített elemeket tartalmaz, mint gombok, navigációs elemek, beviteli mezők, témák. A bejelentkezés kliens oldali validációira is a beépített funkciókat használtam, mely a fenti ábrán látható módon jelzi a hibát a felhasználónak. Hálózati kommunikációra az Axios könyvtárat használtam mellyel egyszerűen lehet a kódból http/s kéréseket kiadni. Bejelentkezéskor elküldjük a felhasználó által küldött adatokat a szervernek, majd ha sikeres a kérés beállítjuk a továbbiakban a kérésekhez használt tokent, illetve a felhasználó adatait.



18. Bejelentkezés kérés elküldése

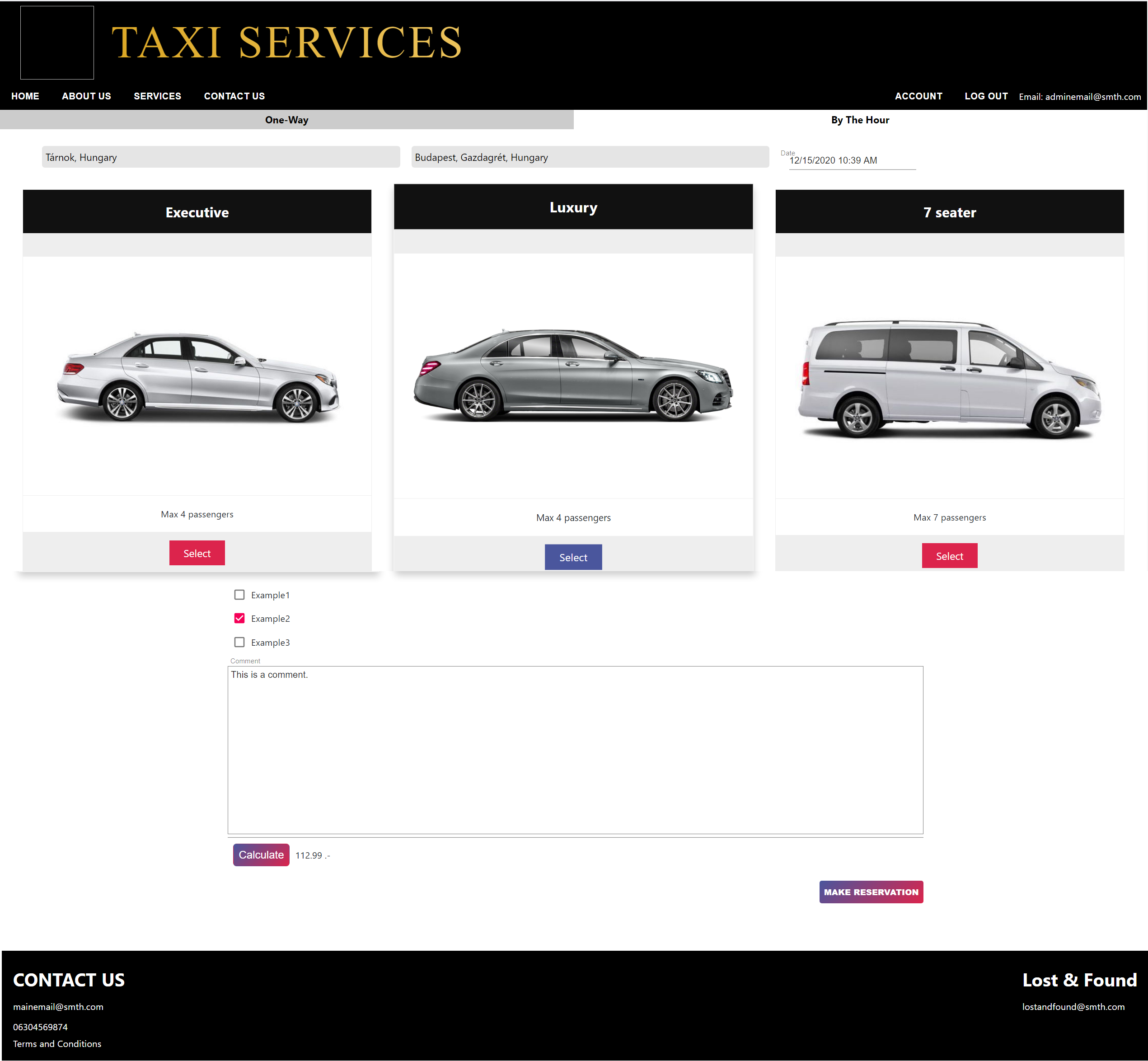
A 18. Ábrán található kérésen látszik, hogy az adatok mentését a redux keretrendszerre bízzuk és nem minden oldalnál külön kezeljük a változásokat. A dispath metódus meghívja az adott adatok kezeléséért felelős kódot, melyet a reducers mappában található osztályokban írtam meg, az alábbi képen látható egy példa a fenti esemény kezelésére.



19. Bejelentkezés során meghívott esemény kezelése

A felhasználó adatait, a böngésző Session Storage tárolójába írjuk, ez azért fontos, hogy az oldal újra töltése esetén ne kelljen újból bejelentkeznie a felhasználónak. Azért döntöttem a Session Storage használata mellett a Local helyett, mivel az előbbi a böngésző bezárásával törlődik, így kisebb az esély, hogy például publikus gépen történő bejelentkezés esetén a felhasználó véletlenül bejelentkezve maradjon.

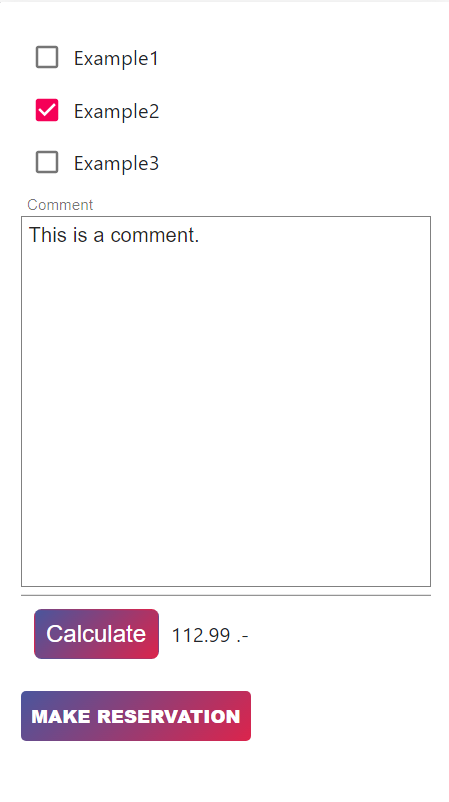
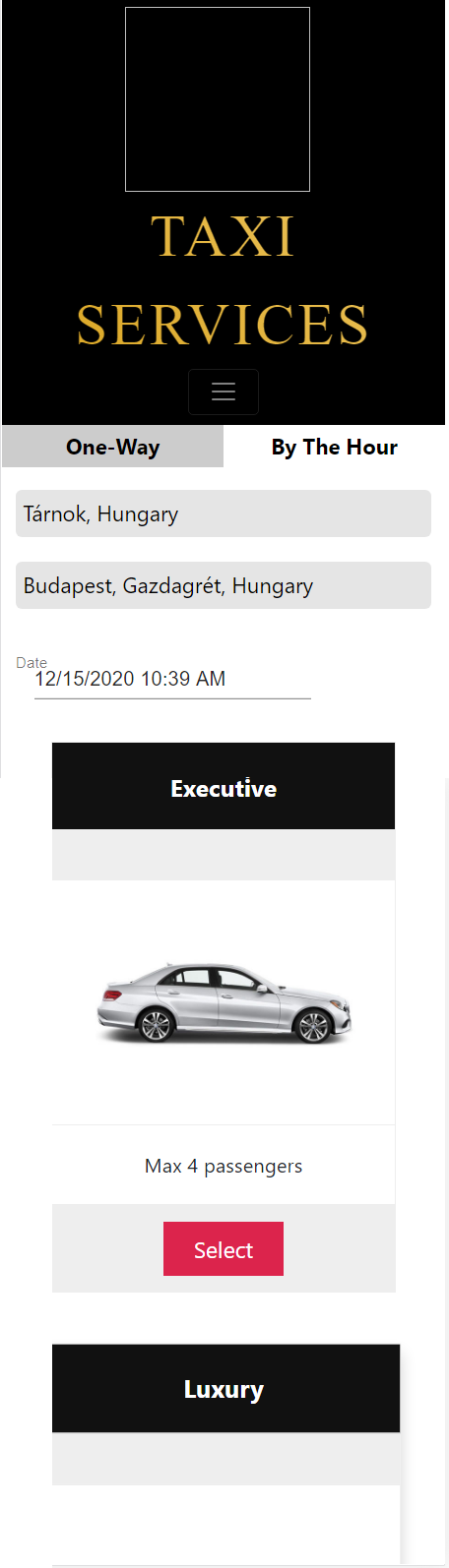
### Foglalás



20. Foglalás képernyő

A foglalás képernyő a fenti ábrán látható. A kereső mezők nem a Material-UI jól megszokott beviteli mezői, hanem a Google Places Api-hoz tartozó javascript vezérlők, melynek személyre szabása meglehetősen nehézkes volt, ám szerencsére nem lehetetlen. A beviteli mezők létrehozásakor meg lehet adni a találatok szűrésére paramétereket, mely segít a releváns találatok megjelenítésében, illetve a kérések árát is csökkenti.

Az alkalmazás készítése során mindenhol törekedtem a reszponzivitás megtartására, különösen mivel később mobilos applikációként is működhet az alkalmazás. A foglalások esetében a mobil eszközökön megjelenített nézet az alábbi képen látható.

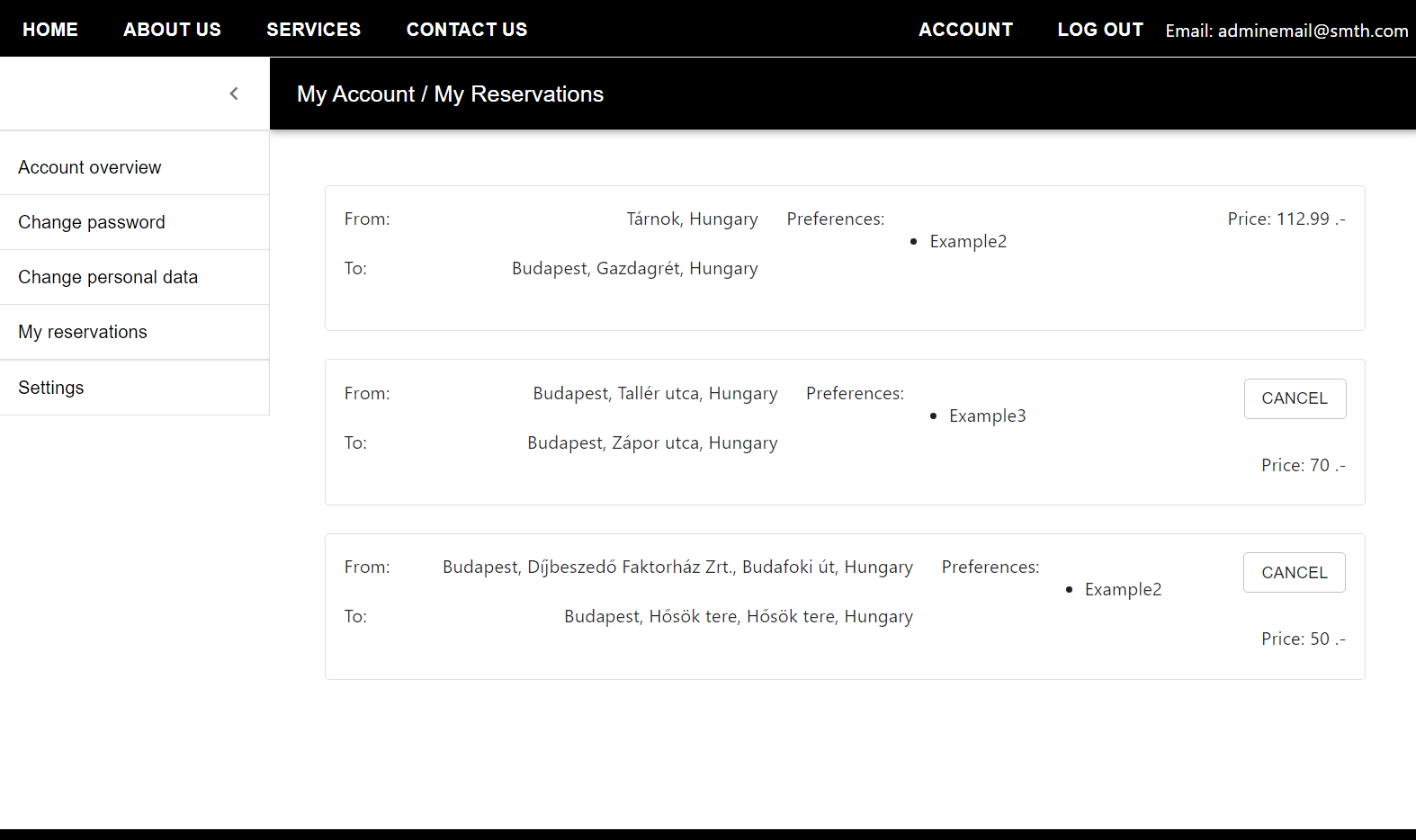


21. Reszponzív foglalás nézet

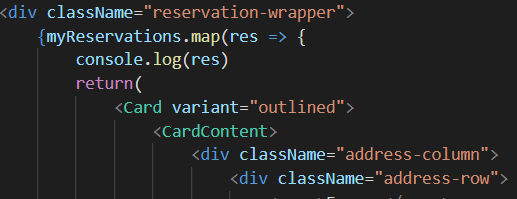
Az egy sorban lévő mezők betörnek egymás alá, a lap szélén lévő margók csökkennek, a gombok mérete megnő, illetve a menük, mint a navigációs menü az alkalmazás fejlécében összecsukódnak.

### Felhasználói adatok kezelése

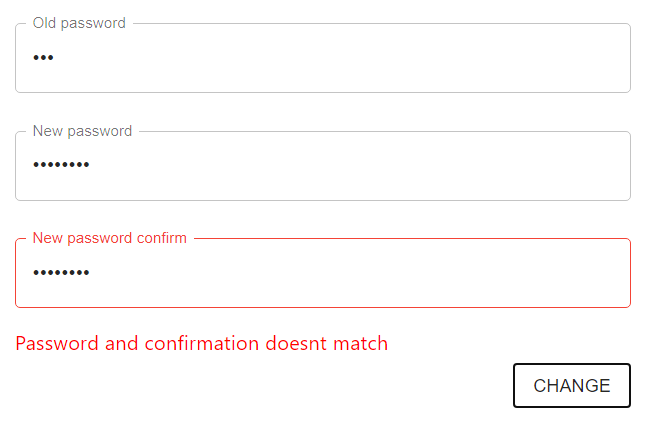
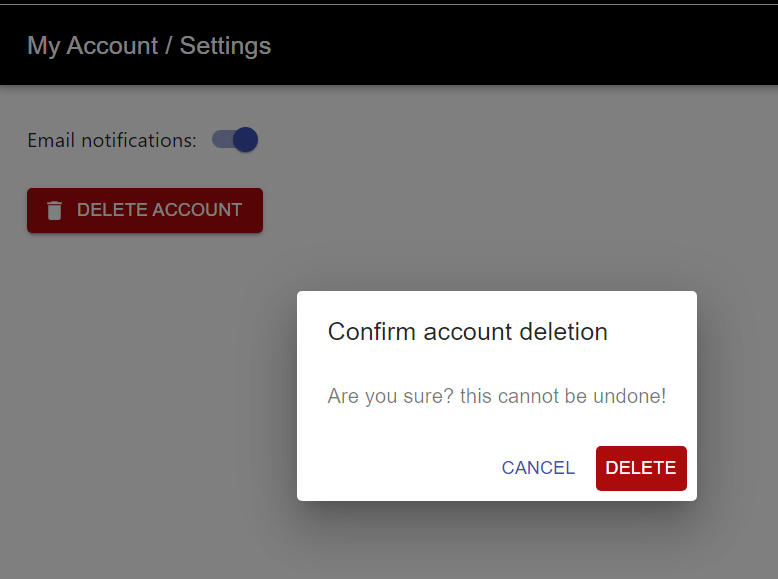
Az Account oldal több komponenst foglal magába, van egy saját navigációs menüje, mely a reszponzivitás érdekében összecsukható.



Az itt található lapok közül kettőt emelnék ki, az egyik a fenti képen látható saját foglalások. A React keretrendszer egyszerű módot ad a különböző komponensek adott feltételek esetén való megjelenítésére. Mivel a kódban szabadon vegyíthető a html és a typescript az alábbi képen látható módon adhatunk feltételeket a komponensek megjelenítésére, illetve a bal oldalon látható módon a lista nézetek megjelenítésére.



Bármilyen alkalmazás készítése során rendkívül fontos, hogy a felhasználót megfelelően értesítjük arról, mi is történik éppen, illetve, hogy ne engedjünk semmi visszafordíthatatlan cselekvést többszöri megerősítés nélkül végrehajtani. Erre az elvre mutat jó példát a felhasználó törlése a beállítások képernyőn. A 22. ábra bal oldalán látható, hogy a felhasználó törlése gomb jól elválik a környezetétől, illetve megerősítés szükséges a kérés végrehajtásához.



22. Felhasználó visszajelzések

A fenti ábra jobb oldala pedig arra mutat példát, hogy egyrészt jelszó változtatásakor – vagy email cím megadásakor, stb. – nem szerencsés ellenőrizetlen mezőt megadni, mert a felhasználó könnyen kizárhatja magát a saját fiókjából egy egyszerű elgépelés eredményeként, másrészt nem elég a hibát detektálni, azt meg is kell jeleníteni. Visszajelzés nélküli események használata helytelen fejlesztői gyakorlatra utalnak (Bad practice).

## Összefoglalás

A félév során megvalósított rendszer összességében megfelel a funkcionális követelményekben megfogalmazottaknak, azonban rengeteg továbbfejlesztési lehetőségnek lehet még helye. A következő félévekben szeretnék hozzáadni a rendszerhez több funkciót is.

A rendszernek szüksége van a fizetések támogatására valamilyen online formában, például paypal vagy Barion integrálásával. Ezen kívül jelenleg a szerver nem törődik azzal, hogy van e szab autó a foglalni kívánt időszakban, így a jövőben szükség volna egy naptárkezelő algoritmusra is mely nyomon követi a rendszerbe regisztrált erőforrásokat.

Ahhoz, hogy a rendszer a gyakorlatban is jól használható legyen a fentieken kívül be kellene vezetni adminisztrátori és dolgozói felületeket is, ahol az ügyintézők a felhasználók foglalásait tudnák kezelni, illetve a paramétereket konfigurálni, a dolgozók pedig a hozzájuk rendelt foglalásaikat tudnák kezelni, szabadnapokat vehetnének fel. A kényelmes használathoz előnyös lenne legalább a dolgozói és felhasználói felületeket mobilos alkalmazásban is elérhetővé tenni.