

**UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,
TÎRGU-MUREȘ
SPECIALIZAREA CALCULATOARE**

Aplicatie de căutare însoțitor de călătorie

PROIECT DE DIPLOMĂ

Coordonator științific:
dr. Márton Gyöngyvér

Absolvent:
Benedek Szabolcs

2021

UNIVERSITATEA „SAPIENTIA” din CLUJ-NAPOCA Facultatea de Științe Tehnice și Umaniste din Târgu Mureș Specializarea: Calculatoare		Viza facultății:
LUCRARE DE DIPLOMĂ		
Coordonator științific: dr. Márton Gyöngyvér	Candidat: Benedek Szabolcs Anul absolvirii: 2021	
a) Tema lucrării de licență: Aplicație de căutare însoțitor de călătorie		
b) Problemele principale tratate: <ul style="list-style-type: none"> - Integrări de API-uri externe în propriul proiect - Proiectarea algoritmilor de recomandare a rutelor adecvate - Maximizarea experienței de utilizator pentru site-uri web - Construirea unui site web cu design potrivit și receptiv - Studiul arhitecturii Model View Controller în cadrul Laravel 		
c) Desene obligatorii: <ul style="list-style-type: none"> - Schema bloc al aplicației - Diagrame UML privind software-ul realizat. 		
d) Softuri obligatorii: <ul style="list-style-type: none"> -Aplicație în Laravel 		
e) Bibliografia recomandată: <ul style="list-style-type: none"> [1] Oluwafemi Alofe: “Beginning PHP Laravel: Step to step approach to building an Inventory App”. [2] Linus Torvalds: “Laravel 5.8 Learn A to Z Best PHP Framework”. [3] Bogdan Brinzarea, Cristian Darie, Mihai Bucica: “AJAX and PHP: Building Responsive Web Applications”. [4] Ethan Marcotte: “Responsive Web Design”. [5] Ficsor Lajos, Krizsán Zoltán, Mileff Péter "Szoftverfejlesztés". 		
f) Termene obligatorii de consultații: săptămânal g) Locul și durata practicii: Universitatea „Sapientia” din Cluj-Napoca, Facultatea de Științe Tehnice și Umaniste din Târgu Mureș Primit tema la data de: 12.05.2020 Termen de predare: 28.06.2021		
Semnătura Director Departament		Semnătura coordonatorului
Semnătura responsabilului programului de studiu		Semnătura candidatului

Declarație

Subsemnatul Benedek Szabolcs, absolvent a specializării Calculatoare, promoția 2021 cunoscând prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a Universității Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licență de diplomă se bazează pe activitatea personală, proiectarea este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea,

Data:

Absolvent

Semnătura



**SAPIENTIA ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR
SZÁMÍTÁSTECHNIKA SZAK**

Útitárs kereső alkalmazás

DIPLOMADOLGOZAT

**Témavezető:
dr. Márton Gyöngyvér**

**Végzős hallgató:
Benedek Szabolcs**

2021

**SAPIENTIA UNIVERSITY FACULTY OF
TECHNICAL AND HUMAN SCIENCES
TÎRGU MUREȘ COMPUTER SCIENCE SPECIALIZATION**

Travel companion finder application

DIPLOMA THESIS

Advisor:
dr. Márton Gyöngyvér

Student:
Benedek Szabolcs

2021

Tartalomjegyzék

Extras	1
Kivonat	2
Abstract	3
1. Bevezető	4
2. Célkitűzések	4
3. Elméleti háttér	5
3.1. Laravel	5
3.1.1. Modell-nézet-vezérlő (MVC)	5
3.2. Google Maps API	6
4. Háttérkutatás – Hasonló alkalmazás	7
4.1. BlaBlaCar	7
5. Követelmény specifikáció	7
5.1. Felhasználói követelmények	7
5.2. Funkcionális követelmények	8
5.3. Nem funkcionális követelmények	9
6. Rendszer leírása	9
6.1. Architektúra	10
6.2. Front-end	10
6.3. Back-end	11
6.4. Adatbázis	12
6.5. Többnyelvűsítés	13
6.6. Szekvenciális diagram	14
7. Az alkalmazás funkcionalitásai	15
7.1. Bejelentkezés	15
7.1.1. Bejelentkezés regisztrációval	16
7.1.2. Bejelentkezés Facebook profillal	16
7.2. Fuvarok böngészése	17
7.2.1. Fuvar lefoglalása	18
7.3. Fuvar létrehozása	18
7.4. Fuvar keresése	20
7.5. Aktív fuvarok	20
7.6. Előzmények – útitársak értékelése	21

7.7. Chat	22
7.7.1. Cloud Firestore	24
7.8. Értesítések.....	25
7.9. Jelszó visszaállítása.....	26
7.10. Profil adatok módosítása	28
8. Kutatómunka - Google Maps API	28
8.1. Google Térkép	29
8.2. Használata.....	29
8.3. Directions Service – Útvonal Tervezés	31
8.3.1. Directions-hoz intézett kérés.....	32
8.3.2. Directions-től kapott válasz.....	34
8.3.2.1. geocoded_waypoints	37
8.3.2.2. routes	37
8.4. Google Maps segítségével beépített funkciók	38
9. Jövőbeli terveim	46
10. Összegzés	46
11. Ábrajegyzék.....	47
12. Bibliográfia	48

Extras

Site-ul web creat în cadrul lucrării a primit numele “The Ride”, și își propune să creeze o platformă receptivă și bine concepută pentru persoanele care doresc să călătorească, să facă navetă și să exploreze. Ideea a venit din faptul că există o serie de grupuri pe Facebook care facilitează autostopul online, care oferă căutarea însoțitorilor de călătorie, însă nu există multe aplicații de acest tip, așa că m-am gândit să creez una.

Un aspect important a fost ca website-ul meu să ofere ceva în plus față de aplicațiile similare, cum ar fi Blablacar, care este utilizat pe scară largă și în țara noastră. Ca companie internațională, Blablacar poate fi utilizat în orice țară, ceea ce îl face impersonal. Grupurile Facebook sunt mai răspândite în comparație cu alte aplicații ride sharing, datorită aspectului personal al platformei.

Platforma „The Ride” oferă oportunitatea de a crea o comunitate, încercând totodată să ofere un mediu personal, prietenos, unde căutarea rutelor poate fi implementată între localitățile din România și Ungaria.

În cadrul aplicației mi-am stabilit și obiectivul de a putea complet planifica o rută, vizualizând în detaliu traseul creat. Aplicația oferă așadar, în mod corespunzător rută între destinațiile specificate. Acest funcționalități au fost create folosind Google Maps API, pe care vor prezenta amănunțit în cadrul lucrării.

Cuvinte cheie: voiaj, construirea comunității, receptivitate, Google Maps

Kivonat

A dolgozat keretein belül elkészített weboldal a „The Ride” nevet kapta, és célja az utazni és ismerkedni vágyók számára, egy reszponzív, jól kinéző platform létrehozása volt. Az ötlet abból fakadt, hogy nagyon sok Facebook csoport létezik az online stoppolás, online útitárs keresés megkönnyítésére, viszont alkalmazás nem sok, így az itt látott hiány kitöltésének lehetősége felkeltette az érdeklődésemet.

Fontos szempont volt, hogy az én weboldalam valamivel többet nyújtson a hasonló alkalmazásokhoz képest, például a hazánkban is széles körben által használt Blablacar-nál. A Blablacar, mivel nemzetközi cég, bármelyik országban használható, ezért személytelen, a Facebook csoportok pedig azért elterjedtebbek, mint az útitárskereső alkalmazások, mert személyeseek.

A „The Ride” platformján lehetőséget teremtettem a közösségépítésre, és igyekeztem személyes, barátságos környezetet biztosítani, ahol az útvonalkeresést romániai és magyarországi települések között lehet megvalósítani.

Az alkalmazással azt a célt is kitűztem, hogy az útvonalat, egy fuvar létrehozásakor, lehessen megtervezni, hogy az alkalmazás fuvart tudjon ajánlani a felhasználók számára. Fontosnak tartottam azt is, hogy megjelenítsem, egy létező fuvar esetében, településről-településre lebontva az útvonalat. Ezeket és az ezekhez hasonló funkciókat a Google Maps API segítségével oldottam meg.

Kulcsszavak: utazás, közösségépítés, reszponzivitás, Google Maps

Abstract

The web application built for the exam has been named „The Ride” and the purpose behind is to create a well designed and responsive platform for people who want to travel, commute and explore. The inspiration behind the idea was the fact that there are a number of Facebook groups meant to simplify the search for ride sharing, however, there are not that many individual applications built for this purpose. This was the source of inspiration behind my web application.

An important factor for the project was that it must provide an added value compared to similar solutions, for example Blablacar that is currently available in our country. Facebook groups are more widespread compared to other ride sharing applications due to the personal aspect of the social media platform. Blablacar is an international company whose application is used in multiple countries, which is the reason why building a community is more difficult there. „The Ride”, on the other hand, is only available for regions in Romania and Hungary, which meant to build a stronger, more familiar connection with the local users.

My goal for the application was to be able to create the entire route when adding a new ride, as well as to provide a breakdown of locations that the ride is passing through. It’s crucial that the users are provided with good recommendations for rides. This functionality is built with the help of Google Maps APIs which will be discussed further in the exam paper.

Keywords: traveling, social building, responsivity, Google Maps

1. Bevezető

Napjainkban egyre jobban terjednek a különböző útvonaltervező, online stoppoló, navigációs és más olyan szoftverek, amelyeket a felhasználók az utazás során használnak. Általában közös ezekben a programokban, hogy valamilyen térképet használnak, legtöbbször a fejlesztők a Google ingyenes API-ját, a Google Maps API-t építik be a szoftverjeikbe, mivel azt folyamatosan frissítik és csaknem az egész világ le van benne fedve. Pozitívum még a Google térképénél, hogy nagyon felhasználóbarát, rengetek adatot tudunk lekérni a segítségével, amelyeket ötletesen beépíthetünk saját alkalmazásunkba.

Fontos ezekben az alkalmazásokban a kreatív funkcionalitásokon kívül, hogy (weboldal esetében) reszponzívak, felhasználóbarátok legyenek, megkapó dizájnnal rendelkezzenek és legyen megfelelő felhasználófelületük az egyszerű kezelés érdekében. Mindezek felett fontos, hogy legyen valami plusz az alkalmazásban, amitől egyedi lesz, ez lehet akár egy ötletes funkcionalitás is. Ezek mind nélkülözhetetlenek, hogy piacképes legyen a szoftver, és én is ezeket figyelembe véve alkottam meg a kutatás keretein belül elkészült weboldalamat.

A dolgozat keretén belül bemutatjuk az útvonaltervezés kivitelezését, az útvonal ajánló algoritmusokat, a felhasználóbarát és felhasználói élményt fokozó funkcionalitásokat. Kifogok térni más tényezőkre is, amelyek nélkülözhetetlen elemei egy jól felépített modern weboldalnak, mint: keretrendszerek (esetünkben Laravel) használata a fejlesztőbarát szoftver készítéshez, reszponzív oldalépítés, hogy okostelefonokon, tableteken és számítógépeken is használható legyen az alkalmazás, webdizájn, megkapó felhasználói felület tervezése, Back-end és Front-end közötti kommunikáció, adatbázis tervezése és kezelése, többnyelvűség stb.

A weboldal, amelyet készítettem, a The Ride nevet kapta, ezen keresztül fogjuk tanulmányozni a fent említetteket. A projekt Laravel keretrendszerbe készült, és jelenleg is „domain névvel rendelkezik. Itt megtekinthető: <https://theride.info>.

2. Célkitűzések

Céлом egy olyan szoftver készítése volt, amely megkönnyíti a Romániában és Magyarországon élő emberek számára az online útíráskeresést, mindezt úgy, hogy egy barátságos, családi platformon legyen erre lehetőség, így téve közvetlenebbé az utazást. Szerettem volna egy olyan oldalt alkotni, amiben meg van a Facebook csoportok „barátságos” hangulata, hogy ismerősökkel vagy ismerőseink ismerőseivel találkozzunk, akiket tudunk kötni valahova az életünkben. Ezek mellett az is fontos cél volt, hogy meglegyen benne a nagyobb alkalmazások profizmusa, minden az oldal stílusával járó funkcionalitás bekerülése, valamint új kreatív dolgok beépítése, amik máshol nincsenek.

3. Elméleti háttér

3.1. Laravel

A Laravel egy nyílt forráskódú, PHP keretrendszer, amely model-nézet-vezérlő (MVC) szerkezeti mintára épült. A keretrendszert alapvetően a Symfony php keretrendszerből fejlesztették 2011-ben. Legfontosabb előnye az adatokkal és adatbázisokkal történő munka megkönnyítése fejlesztőbarát módon. Nagyon átlátható szerkezetének köszönhetően, napjaink legnépszerűbb PHP keretrendszere.

A sima PHP-ban, ha nem használunk semmilyen keretrendszert, akkor az adatok lekéréséhez az adatbázisból lekérdezéseket kell írunk, például SQL nyelvben, így a fejlesztőnek ezeket is tudnia kell, hogy helyesen megtudja írni a kódot. A Laravelben nem szükséges tudni más adatbáziskezelő nyelv lekérdezéseit, mivel a keretrendszer ezt megoldja nekünk. Lehetőségünk van létrehozni egy aktív rekordot, amit modellnek nevezünk, és ha a modellünk neve és az adattáblánk neve megegyezik (annyi különbséggel, hogy az adattábla neve többes számban van, például a modell neve Car, akkor az adattábla neve cars), akkor automatikusan tudni fogja, hogy ha a Car összes tartalmát szeretnénk lekérdezni, akkor a cars tábla tartalmára vonatkozik a lekérdezésünk. Ugyanígy az adatok feltöltése is hasonlóképpen letisztult és egyszerű megoldásokkal operál.

3.1.1. Modell-nézet-vezérlő (MVC)

Rövidítése az angol Model-View-Controller-ből ered. Célja egy könnyen átlátható, jól elkülöníthető szerkezetet adni a projektnek. Ez egy olyan szerkezeti minta, amelynek fő tulajdonsága az adat és a felhasználói felület különválasztása, ezáltal a felhasználói felület nem befolyásolja az adatkezelést, és az adatok változtatása sem igényli a felhasználói felület változtatását.

Ahogy a neve is mutatja három részre strukturálja a projektet. Első a Modell, ez lényegében maga az adat, ahogy fentebb is említettem. Az adatbázisunk táblái ilyen modelleken alapulnak. Egy modell, az egy szimbolikus példánya az adatbázisunk egy bizonyos táblájának, így ezen a modellen keresztül tudjuk az adatainkra vonatkozó kikötéseket és szabályokat meghozni. Itt tudjuk például még összekapcsolni a modelljeinket (külső kulcs segítségével) egymással. A Nézet (angolul View) az a része a projektnek, ahol ezek az adatok megjelenítésre kerülnek, a legegyszerűbb úgy elmagyarázni, hogy ez az a rész, amit a felhasználó lát, ide a már feldolgozott adatok kerülnek azok, amelyeket megjelenítünk a Front-end-en. A harmadik része a Vezérlő (Controller), ez az a rész, ahol minden fontosabb művelet elvégzésre kerül az adatainkkal. Például,

ha az adatbázisba szeretnénk adatot bevinni, de ezek az adatok előtte valamilyen logika szerint változnak, akkor ezeket a változásokat itt a vezérlőbe írjuk meg, és azután ugyanitt fel is tudjuk tölteni az adatbázisba a már megfelelő formájukat. Ez visszafelé is igaz, ha szeretnénk megjeleníteni az adatokat a nézetbe, viszont nem abba a formába, ahogyan az adatbázisban szerepelnek, hanem kisebb-nagyobb átalakításokkal. Amennyiben több adat közös műveleteit szeretnénk megmutatni, akkor szintén a lekérés után itt végezzük el a feladatokat, majd innen küldjük a megfelelő adatokat tovább a nézetbe.

3.2. Google Maps API

Az API, angolul „Application Programming Interface”, ami magyarul „alkalmazás-programozási felületet” jelent, a fejlesztők munkáját hivatott megkönnyíteni azáltal, hogy hozzáférést biztosít egy adott szoftver, vagy eszköz utasításkészletéhez. Napjainkban nagyon sok szoftverfejlesztő cég van, amelyek kifejezetten API-k gyártására szakosodottak.

A Google-nek is sok ilyen API-ja van, amelyet bárki használhat, és ezek általában ingyenesek. Azután kell csak fizetni, ha az alkalmazásunkat sok felhasználó használja, ezt a Google az API-tól történő kérések számából tudja megállapítani. Jól kitalált és fejlesztőbarát megoldás ez, mivel ha sokan használják az alkalmazásunkat, akkor valószínűleg ez a termék valamilyen módon profitot fog termel, ebből a profitból pedig már könnyedén kifizethető az API használatának a díja, ami általában jól be van árazva.

A Google Maps API, amelyet én használtam a projektem elkészítésében, arra szolgál, hogy valamilyen kérésre (Request) adjon egy bizonyos választ (Response), és ebből a válaszból a fejlesztő az adatok megfelelő felhasználásával be tudjon építeni valamilyen térképpel kapcsolatos funkcionalitást a projektébe. Mivel webes projektet készítettem, én a Maps Javascript API-t használtam, ami azt jelenti, hogy Front-endről, a Javascript fájlomból tudom intézni a kérést, és a kapott választ is feldolgozhatom itt, tehát nem kell csinálnom semmit a Back-enden, hogy megjeleníthessek egy térképet bizonyos paraméterek szerint. Ezek a paraméterek természetesen lementhetők az adatbázisba, így később lekérve azokat, a Front-enden újabb kérést intézve megjeleníthető ugyanaz a térkép, ugyanolyan útvonallal és minden egyébbel, amit meg szeretnénk jeleníteni a térképen.

4. Háttérkutatás – Hasonló alkalmazás

Az általam fejlesztett alkalmazás nem egyedülálló, vannak hazánkban is hasonló útitárskereső alkalmazások. Véleményem szerint azonban jóval kevesebb van ezekből, mint amennyi a potenciális felhasználók igényeit kellőképpen ki tudná elégíteni. Ezt az is jól mutatja, hogy bár léteznek ezek az alkalmazások, az útitárskeresést az emberek nagyrésze (legalábbis hazánkban) a Facebook csoportokon keresztül végzi. Ez arra enged következtetni, hogy lenne igény az ilyen és ehhez hasonló platformokra ezután is, feltéve, hogy ez a platform valamivel egyedibb, valamivel másabb lenne és akár többet is képes lenne nyújtani, mint a már meglévők.

4.1. BlaBlaCar

A BlaBlaCar egy multinacionális francia cég online útitárskereső platformja. Van már BlaBlaCar-ból weboldal, illetve IOS és Android applikáció is. A céget 2006-ban alapították, és eredményességüket az is bizonyítja, hogy azóta 600 alkalmazottjuk, illetve 80 millió felhasználójuk lett. Romániában a tapasztalataim szerint ez a legnépszerűbb ilyen platform, és mivel ez a legismertebb, általában az emberek, ha útitársakat keresnek, a Facebook csoportokon kívül, ez az egyetlen alkalmazás, ami még szóba jöhet.

Hogy miért gondolom, hogy egy ekkora multinacionális cég mellett is van létjogosultsága az én alkalmazásomnak? Elsősorban nem a BlaBlaCar felhasználóit célozom meg, hanem kifejezetten a Facebook csoportokon keresztül utazókat akarom meggyőzni. Piackutatásaim alapján ezek az emberek azért nem a BlaBlaCar-t használják, mert a széles felhasználói réteg okán elvesz az utazás családi hangulata, melyet őriznek a Facebook csoportok (ahol ugyebár a felhasználók ismerősökkel találkoznak), és ezt a hangulatot szerettem volna én is megteremteni az alkalmazásomban. Ugyanakkor azért is fontos, hogy egy ilyen jellegű platformon inkább ismerős felhasználókkal találkozzunk, mivel sokakat eltántoríthat az idegenekkel való együtt utazás.

5. Követelmény specifikáció

A rendszer különböző követelményeknek kell megfeleljen, mind felhasználói, mind rendszer (funkcionális, nem funkcionális) szinten, annak érdekében, hogy a lehető legoptimálisabb módon működjön.

5.1. Felhasználói követelmények

Az 1. ábrán látható a weboldal Használati Eset Diagrammja (Use Case Diagramm). Ezen az ábrán a felhasználó szemszögéből láthatjuk az oldalt, azaz milyen lehetőségei vannak, miket tud csinálni. Kezdetben, belépés nélkül a felhasználónak lehetősége van böngészni a fuvarok között,

megnyitni azokat, nyelvet váltani, illetve regisztrálni, majd ezt követően bejelentkezni. Bejelentkezés után elérhetővé válik az oldal többi funkciója. A fuvar megnyitása után azonnal lehet foglalni, vagy chaten keresztül kommunikálni a sofőrrel. Lehetőség van saját fuvar létrehozni, illetve a meglévő fuvarjainkat böngészni, törölni. Megnyithatjuk a Fuvart Keresek menüpontot, ami lényegében egy részletesebb szűrővel felszerelt fuvarkeresés. A felhasználónak lehetősége van még ezeken kívül a régi fuvarjait megnézni, értékelést leadni útitársairól, módosítani bizonyos adatait, megtekinteni az értékeléseit és az értesítéseit.



1. ábra – Használati eset diagram

5.2. Funkcionális követelmények

Egy rendszer funkcionális követelményei leírják, hogy a rendszernek milyen funkciókkal kell rendelkezni, hogyan kellene működnie (Például aktualizálások, lekérdezések, jelentések, kimenetek, adatok, más rendszerekkel való kapcsolat). Ezek a követelmények a fejlesztett szoftver típusától, a szoftver leendő felhasználóitól függenek, esetünkben a következők:

- Regisztráció

- Bejelentkezés
- Fuvarok keresése
- Fuvar lefoglalása
- Fuvar létrehozása
- Fuvar törlése
- Értesítés rendszer
- Kapcsolat felvétele a többi felhasználóval
- Felhasználók értékelése
- Profil adatok módosítása
- Előzmények böngészése

5.3. Nem funkcionális követelmények

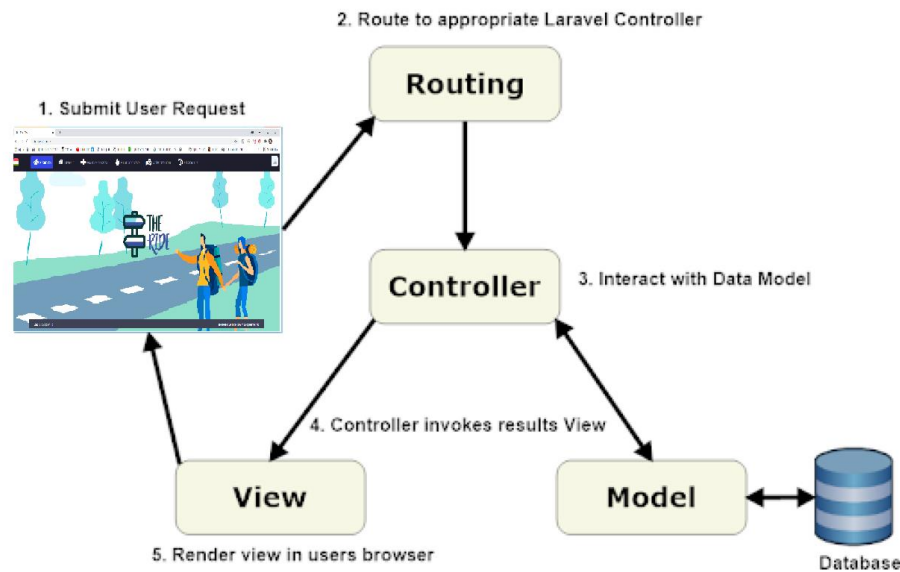
A nemfunkcionális követelmények a funkcionális követelményekkel ellentétben nem közvetlenül a rendszer által biztosított specifikus funkciókkal foglalkoznak, hanem inkább a rendszer egészére vonatkozó tulajdonságokra koncentrálnak.

- Modern böngésző
- Reszponzivitás
- Skálázhatóság
- Biztonság
- Könnyen használhatóság

6. Rendszer leírása

A rendszer leírása során áttekintjük a rendszer architektúráját, illetve a szoftver megírása során használt technológiákat is megfogjuk vizsgálni.

6.1. Architektúra



2. ábra – A rendszer architektúrája

A 2. ábrán látható a rendszer architektúrája, amely lényegében egy megszokott Laravel architektúra. Az adatbázissal a modelljeink kommunikálnak, amik reprezentálnak egy elemet az adott táblából, s ezekkel a modellekkel végzünk műveleteket a vezérlőkben. A feldolgozott adatokat majd átadjuk a nézetnek, ahol meg van írva az oldal HTML, CSS illetve Javascript kódja. Itt az adatokat megjeleníthetjük olyan formában, amilyenben szeretnénk. Visszafele (ha adatot szeretnénk bevinni az adatbázisba) a dolog úgy működik, hogy a nézetből útvonal megadásával (routingolás) tudunk a Front-endről adatokat küldeni a vezérlőknek. Ezt úgy tudjuk megoldani, hogy létrehozzuk előre az útvonalat a Laravel keretrendszer routes mappájában található web.php fájlban. Ennek az útvonalnak megadjuk, hogy melyik vezérlő, melyik függvényét kell meghívni, illetve, hogy milyen és mennyi paramétert fogunk átadni. A Front-enden a bevinni kívánt adatok beviteli mezőit berakjuk egy űrlapba (form), majd ezt az űrlapot oda tudjuk adni paraméterként a már megírt útvonalunknak, így az űrlapunk küldés gombjával (az a gomb, amelyik az űrlapon belül helyezkedik el és a típusa submit) máris a vezérlőbe került minden adat az űrlapról. A vezérlőben ezután ugyanúgy feltudjuk dolgozni a megkapott adatokat, majd a kívánt formában eltároljuk az adatbázisban.

6.2. Front-end

A weboldal Front-endjét HTML, CSS és JavaScript segítségével hoztam létre. Működésének lényege, hogy a nézet fájlokba (amelyekről az előző részben említést tettem) HTML kód írható. Ez a HTML látja azokat az adatokat, amelyeket a vezérlő fájlból küldtünk a nézetbe, így azokat meglehet jeleníteni a weblapon. Ezekbe a nézet fájlokba importálhatunk CSS és Javascript

fájlokat, amelyek segítségével elkészíthetjük a nézetek dizájnját, reszponzivitását, illetve megírhatjuk a felhasználói felület interakcióját.

Az alkalmazás reszponzivitásának megírásához a sima CSS-en kívül használtam egy „Bootstrap” nevű könyvtárat. A „Bootstrap” a világ legnépszerűbb, nyílt forráskódú Front-end eszköztára, melynek felhasználása ingyenes. Ebben az eszköztárban használhatunk változókat és függvényeket a CSS fájljainkban, illetve nagyon sok osztály CSS kódja előre megvan írva, így a fejlesztő sok időt megspórolhat az általános CSS kódok megírásának mellőzésével.

A dizájn elkészítéséhez, hogy profibb legyen az oldalak kinézete, vásároltam egy admin kezelőfelület (Dashboard) témát. A megvásárolt téma a „Metronic”, s a dokumentációja megtalálható a referenciáknál. A Metronic-ot általában az admin felületek elkészítéséhez szokták használni, de felbontva komponensekként is lehet használni, én is így építettem be a projektembe.

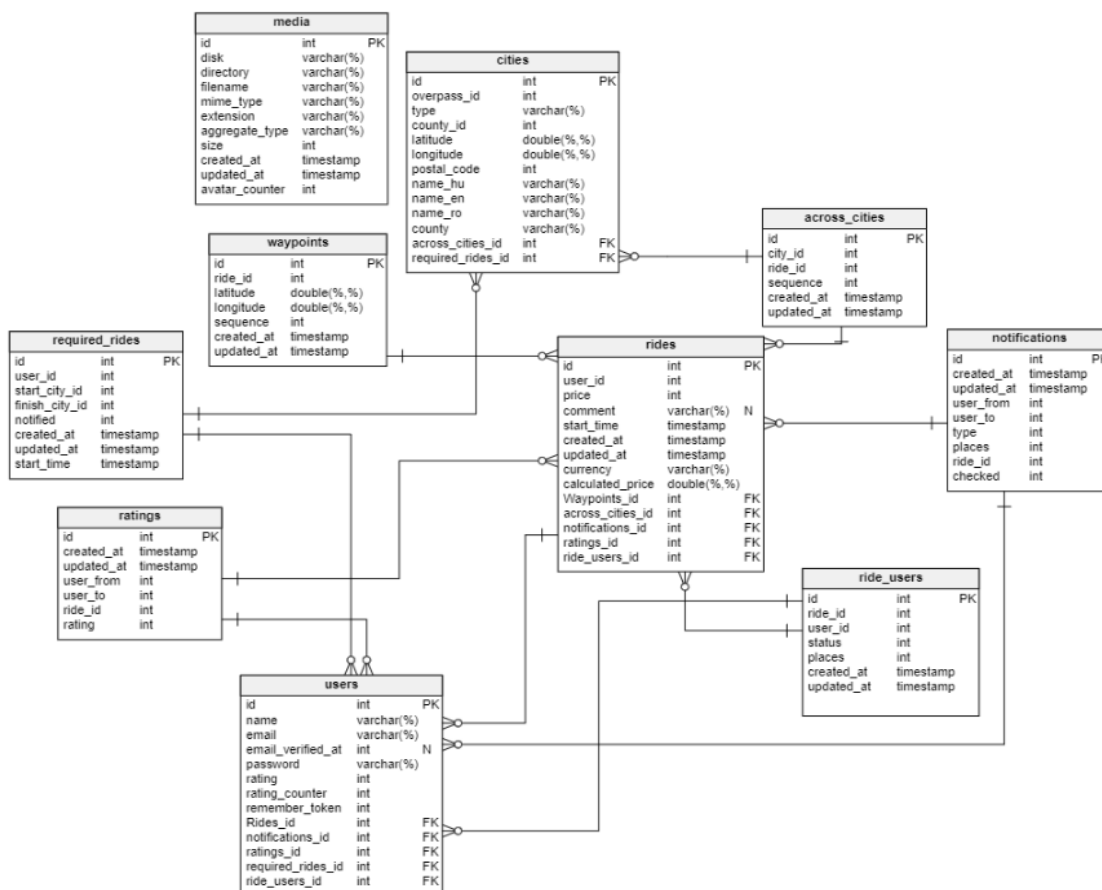
6.3. Back-end

A weboldal Back-endje teljes mértékben Laravelben van írva (aminek működési elvét már magyaráztam a rendszer architektúrájának leírásakor). A keretrendszer szolgáltatásaiból a következőket használtam a projektemben:

- Migration: magyarul vándorlást jelent. Arra való, hogy megkönnyítse a fejlesztők dolgát az adattáblákkal. Itt egyszerűen hozhatók létre, illetve utólag módosíthatók a táblák anélkül, hogy az adatbázisban levő adatok megsérülnének.
- Routing: útvonalak írása, hogy a Front-endről bármikor tudjunk lekérni, illetve bevinni adatokat az adatbázisba.
- Middleware: „köztes szoftver”, segítségével több útvonalat egy csoportba szervezhetünk, ezzel küldve általános paramétereket az URL-be. Az oldal többnyelvűségének a megvalósítására használtam.
- Socialite: segítséget nyújt az alkalmazásunknak, hogy kommunikálhasson különböző cégek API-jával, így megoldható a regisztráció nélküli bejelentkezés egy másik, már létező fiókunkkal. A Facebook felhasználóval való bejelentkezés megírásához használtam.
- Testing: ezzel a szolgáltatással teszteket írhatunk külső könyvtár nélkül. Ezeket a teszteket Egység (Unit) teszteknek nevezik, és főleg a modellek tesztelésére, illetve a vezérlő fájlok funkcióinak tesztelésére használjuk.
- Redirect: a vezérlőből történő gyors oldalváltáshoz.
- Auth: az egyszerű ki- és bejelentkeztetésért felelős.

6.4. Adatbázis

A fejlesztés során MySQL adatbázist használtam, mivel könnyen használható, jól működik a Laravellel, illetve az egyetemen is ezzel dolgoztunk a legtöbbet.



3. ábra – Adatkapcsolati táblák

Nagy segítséget nyújtott az adatbázis felépítésében a keretrendszer migration szolgáltatása. Ennek a lényege, hogy az adattábláinkat migration fájlokban hozom létre, itt adom meg a táblák nevét, illetve oszlopaik nevét és típusát. Miután megírtuk az adattábláink felépítését, a fejlesztő a `php artisan migrate` paranccsal tudja lefuttatni ezeket a migration fájlokat, ez az utasítás felfogja építeni az adatbázist. Mindamelllett, hogy segíti a fejlesztést, ez a megoldás arra is kiváló, hogy a későbbiekben könnyedén tudjuk fejleszteni vagy bővíteni az adatbázist. Ha pedig ki szeretnénk üríteni, és újra létrehozni a táblát, akkor a `„php artisan migrate:fresh”` parancs futtatásával tehetjük meg. Ez a módszer azért is nagyon fontos, mert a fejlesztés során, ha már az alkalmazásunk fut egy szerveren, és szeretnénk módosítani az adatbázison, akkor a deploy folyamatnak beállíthatjuk, hogy automatikusan futtassa a parancsot, így nem kell foglalkoznunk a tábla létrehozásával vagy megváltoztatásával a szerveren.

A következő ábrán egy példa kódot mutatok, a fent említett migration fájlok egyikéből.

```

class CreateCitiesTable extends Migration
{
    public function up()
    {
        Schema::create('cities', function (Blueprint $table) {
            $table->id();
            $table->string('overpass_id');
            $table->string('type');
            $table->integer('county_id');
            $table->double('latitude');
            $table->double('longitude');
            $table->string('postal_code');
            $table->string('name_hu');
            $table->string('name_en');
            $table->string('name_ro');
            $table->string('county');
        });
    }

    public function down()
    {
        Schema::dropIfExists('cities');
    }
}

```

4. ábra – Migration fájl

Az itt látható kód a Cities táblát hozza létre. Megjegyzendő, hogy minden ilyen fájlba kell szerepeljen egy up és egy down függvény. Az up függvényben van megírva az adattábla neve és típusa, míg a down függvényben a tábla törlése kell szerepeljen.

6.5. Többnyelvűsítés

Nagyon fontos szempont volt, hogy a weboldal elérhető legyen román, magyar és angol nyelven, mivel a romániai és a magyarországi felhasználók képezik a célcsoportot. Az alkalmazás a program fő konfigurációs fájljából kapja a nyelveket, ezért, ha kisseretnénk bővíteni az oldal támogatott nyelveit, csupán a konfigurációs fájlt kell módosítanunk. A Nézetekben eleve fordítható formában jelenítjük meg a szöveget úgy, hogy a HTML kódban a következő módon adjuk meg a szöveget: `{{__(‘szöveg’)}}`. Ezután az itt megadott szöveget a keretrendszer lefogja fordítani az éppen aktuális nyelvre aszerint, hogy az adott nyelvhez milyen fordítást adtunk meg, egy direkt ebből a célból létrehozott php fájlban.

```

<?php

return [

    'sure' => 'Biztos vagy benne?',
    'yes' => 'Igen',
    'reserved-places' => 'Lefoglalt helyek: ',
    'passangers' => 'Utasok: ',
    'resignation' => 'Lemondás',
    'driver' => 'Sofőr',
    'passanger' => 'Utas',
    'rate' => 'Értékelés',
    'rated' => 'Értékelve',

];

?>

```

5. ábra –Fordító fájl

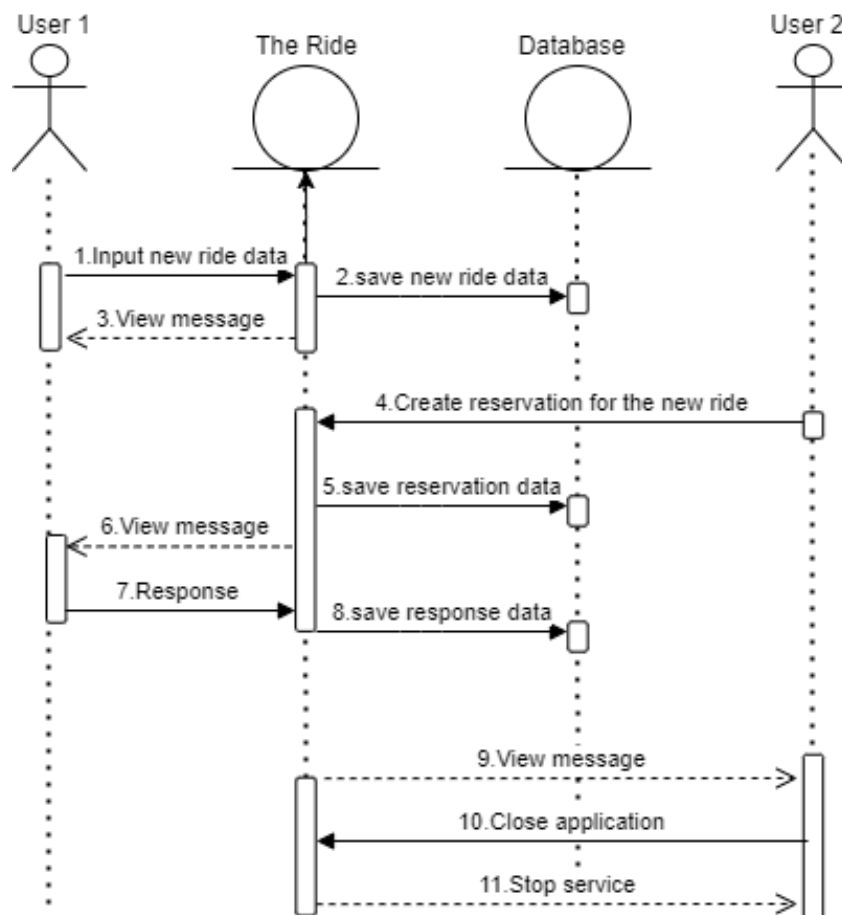
Hogy éppen milyen nyelven jelenjen meg az oldal, onnan tudjuk, hogy az URL-ben paraméterként megvan adva egy lang nevű változó, amelynek három értéke lehet: hu, ro és en. Ezt az értéket a felhasználó bármikor megváltoztathatja. Ha a bal felső sarokban levő zászlóra klikkelve megváltoztatja a lang változót, akkor a weboldalon megjelenő összes szöveg lefordítódik. Az útvonalak megadásakor, hogy ne kelljen minden egyes alkalommal elküldjem a lang változót, amikor paramétereket küldök az URL-ben, azt találtam ki, hogy azokat az útvonalakat, amelyeknek tartalmazniuk kell ezt a változót, berakom egy middleware csoportba. Ennek a csoportnak pedig beállítható, hogy tegyen minden útvonalhoz a megadott paramétereken kívül egy „prefix” változót, ami mindig át fog adódni. Ezt a „prefix” változót állítottam be a lang változónak.

6.6. Szekvenciális diagram

A 6. ábrán a rendszer egy szeletének a szekvenciális diagramja látható. A diagramban annak a szekvenciáját szemléltetem, amikor egy felhasználó létrehoz egy fuvart, ezután egy másik felhasználó foglalást tesz arra a fuvarra, végül pedig a fuvart létrehozó felhasználó dönt, hogy elfogadja vagy esetleg elutasítja a foglalási kérvényt.

Ennek a diagrammnak a lényege, hogy ábrázolja az objektumok üzenetváltását egy időtengely mentén. Jelen esetben 4 objektum van feltüntetve: két felhasználó, az adatbázis, illetve maga az alkalmazás, mint rendszer.

Megjegyzendő, hogy a felhasználó nincs direkt összeköttetésben az adatbázissal, a bevinni kívánt adat először validálódik. A téglalapok a moduloknál azt jelképezik, hogy az adott modul éppen aktív. A bevitt adatokról a rendszer mindig küld valamilyen megerősítő üzenetet a felhasználóknak, ehhez nem kell választ kapjon az adatbázisból, ugyanis pontosan tudja milyen adatok kerültek be.



6. ábra – Szekvenciális diagram

7. Az alkalmazás funkcionálisai

Ebben a fejezetben befogom mutatni a weboldal összes funkcionálisát és azoknak a kivitelezését.

7.1. Bejelentkezés

Az alkalmazásba történő bejelentkezéshez a felhasználónak két lehetősége is van. Egyrészt, hagyományos módon a regisztráció (alapvető adatok megadása, fiók létrehozása) után jelentkezhet be, másrészt a Facebook profiljával a Laravel socialite szolgáltatás segítségével. Erre azért volt nagy szükség, mivel nagy a felhasználói igény napjainkban arra, hogy ne feltétlenül kelljen regisztrálni, lehessen gyorsan belépni az alkalmazásokba úgy, hogy közben megőrizzük a profil adatainkat. Valamint az alkalmazásom fő célcsoportja a Facebook csoportokban útítársat kereső emberek, ezeknek az embereknek pedig értelemszerűen van Facebook profilja, amit használhatnak.

7.1.1. Bejelentkezés regisztrációval

A hagyományos módon történő bejelentkezés első lépése a regisztráció. Regisztráció során a felhasználónak meg kell adni a nevét, e-mail címét, jelszavát, illetve választani kell magának egy profilképet. A sikeres regisztrációhoz el kell fogadni a felhasználási feltételeket és az adatvédelmi nyilatkozatot (ide kerülnek majd különböző jogi információk).

```
protected function validator(array $data)
{
    return Validator::make($data, [
        'name' => ['required', 'string', 'max:40', 'min:3'],
        'email' => ['required', 'string', 'email', 'max:80', 'unique:users'],
        'password' => ['required', 'string', 'min:8', 'max:80', 'confirmed'],
        'Checkbox1' => ['accepted'],
    ]);
}
```

7. ábra – Validáció

Az adatok a Back-enden vannak levalidálva, de a hiba üzenetet megjelenítem a Front-enden, ezt a Laravel segítségével JavaScript kód írása nélkül meg lehet oldani. A vezérlő fájlok között van egy úgynevezett RegisterController.php fájl, ebben a fájlban egy validator nevű függvényben megadhatjuk, hogy a regisztrációs űrlapon melyik beviteli mezőnek, milyen értékeket fogadunk el. Ezáltal amikor a Front-endről elküldjük a regisztrációs kérést, először ez a validáció fog megtörténni, ennek függvényében vagy sikerül a regisztráció vagy hibát küldünk vissza. Ebben az esetben tudjuk már, hogy melyik beviteli mező alatt kell megjeleníteni a hibaüzenetet, és ezt a nézet fájlunkban (ahol van a regisztrációs űrlap) automatikusan meg is jeleníti a keretrendszer. A hibaüzenet stílusát és a szöveg tartalmát tetszés szerint módosíthatjuk.

A sikeres regisztráció után automatikusan beléptetem a felhasználót és a kezdőképernyőre navigálok. A későbbiekben bármikor ki- és bejelentkezhetsz. Sütiket (Cookies) használva az alkalmazás megjegyzi a felhasználó adatait és bejelentkezve tartja, azután is, ha bezárja a böngészőt, ezzel is növelve a felhasználói élményt.

7.1.2. Bejelentkezés Facebook profillal

A Facebook profillal történő bejelentkezéshez a Laravel Socialite szolgáltatását használtam. A Socialite támogatja a Facebook, Twitter, LinkedIn, Google, GitHub, GitLab és Bitbucket profillal történő bejelentkezéseket. Minden ilyen másodlagos szolgáltatást először telepíteni kell a projektünkbe, ebben az esetben a következő terminál paranccsal: „composer require laravel/socialite”.

Ahhoz, hogy bejelentkezhessünk a Facebook felhasználókkal egy másik alkalmazásba, azt a Facebooknak is jóvá kell hagynia, ezért el kell látogatnunk a „Facebook for Developers” platformra. Ezen a platformon regisztrálnunk kell az alkalmazásunkat, meg kell adjuk a domain nevét, majd különböző algoritmusok megvizsgálják elég biztonságos-e ahhoz, hogy a Facebook adatbázisából lekérjük a felhasználók bizonyos adatait. Ha minden megfelelőnek bizonyult, meg is kapjuk a Client Secret nevű kódot, és ezzel a kóddal fog hozzáférni az applikáció (a Facebooknak megadott domain névvel rendelkező applikáció) a Facebook adatbázisának egy szeletéhez.

Következő lépésben a keretrendszer „config/services.php” fájljában közöljük az alkalmazással, hogy a Facebook benne van az általunk használt OAuth szolgáltatók listájában. Az OAuth egy nyílt szabvány engedélyezési folyamatokra, ami lehetővé teszi a felhasználók számára, hogy megosszák saját fájljaikat és adataikat egy harmadik féllel.

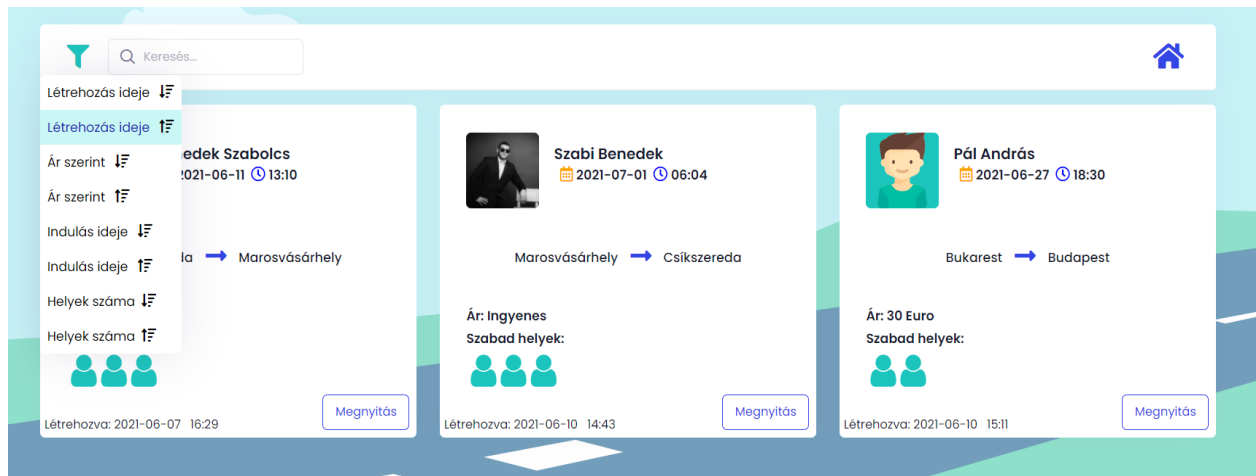
Legvégül elhelyezünk egy gombot a weboldalon, amelynek megnyomására a LoginController nevű vezérlő fájlunkban meghívom a függvényt, amelyik a beléptetésért felelős. Ebben a függvényben a „Socialite::driver(‘facebook’)->user()” paranccsal lekérünk minden nyilvános adatot, amelyet a Facebook biztosít a felhasználóiról. Innentől már csak kiválogatom a számomra szükséges adatokat (e-mail cím, teljes név, profil kép URL-je). Ezekkel az adatokkal regisztrálom a felhasználót a saját adatbázisomba, majd mikor újra Facebookkal óhajt belépni a profil tulajdonosa, akkor már a saját adatbázisunkból fogom beléptetni a rendszerbe.

7.2. Fuvarok böngészése

A Fuvarok menüre navigálva megjelenik az összes fuvar, itt különböző lényeges információkat lehet megtekinteni: ki rakta fel a fuvart, az utazás dátuma, kiinduló pont és végcél, fuvar ára, szabad helyek száma és hogy mikor volt létrehozva a fuvar. A fuvarok között keresni lehet, úgy hogy a fent megjelenő kereső bemeneti mezőbe elkezdjük beírni a város nevét, ilyenkor élőben frissülnek a fuvarok és azok, amelyek nevében nincs benne a keresett szöveg, eltűnnek a listából. Ebben az esetben Front-end keresést csináltam, amit úgy kell elképzelni, hogy nem kérem le újra a fuvarokat a keresés alatt, hanem a meglévő fuvarokból eltüntettem vagy újra megjelenítem az adott fuvart, így a fuvarok tömbben továbbra is benne marad az összes fuvar, de a felhasználó számára csak azokat teszem láthatóvá, amelyek megfelelnek a keresett szónak.

Egy másik lehetőség a fuvarok menüben történő pontosabb keresésre a rendezés. A felhasználónak lehetősége van újra rendezni a fuvarok listát bizonyos attribútumok szerint, ennek beállítása a 6. ábrán látható. Ilyenkor már nem a Front-enden rendezem újra az adatokat, hanem frissül az oldal, és az URL-ben paraméterként beállítódik a rendezésért felelős paraméter. A vezérlőfájlban eszerint a paraméter szerint rendezzük a fuvarokat rögtön azután, hogy lekértük az

adatbázisból, így mikor újra megnyílik a nézet fájlunk, akkor a felhasználó rendezve fogja megkapni a fuvarok listáját.



8. ábra – Rendezés

7.2.1. Fuvar lefoglalása

Miután megtaláltuk a megfelelő fuvar, és megnyitottuk, két lehetőségünk van: chat a sofőrrel (ezt a funkcionalitást később külön fogom megmagyarázni), illetve a fuvar lefoglalása. Ezt egyszerűen úgy tudja megoldani a felhasználó, hogy ráklikkel a Foglалás gombra, ilyenkor leellenőrizzük, hogy be van-e jelentkezve (nem bejelentkezett felhasználók is böngészhetnek a fuvarokat, de nem foglalhatnak), illetve, hogy biztosan nem ő hozta létre a fuvar. Ezt fontos leellenőrizni, mivel a fuvar létrehozó felhasználó is megnyithatja a saját fuvarját. A leellenőrzések után, ha mindent rendben találunk, megjelenik egy felugró ablak, amelyben látható, hogy hány hely van szabadon még az autóban, kiválasztjuk, hány helyet szeretnénk ebből lefoglalni, majd azonnal el is küldődik a kérés a sofőrnek. A sofőr erről azonnal élő értesítést kap az Értesítések menüpontban, ahol elfogadhatja, illetve elutasíthatja ezt a kérést.

Egy fuvar megnyitva elének táruл minden olyan információ, amit az adott fuvarról nyilván van tartva. Ugyanitt megjelenik a térkép az útvonallal, illetve fel lesznek sorolva jelmagyarázattal az útvonal mentén lévő városok (falu, kis város, város). Ezeket a Google Maps API segítségével kiviteleztem, megvalósításukat a későbbieknek, a kutató munka leírásánál fogom taglалni.

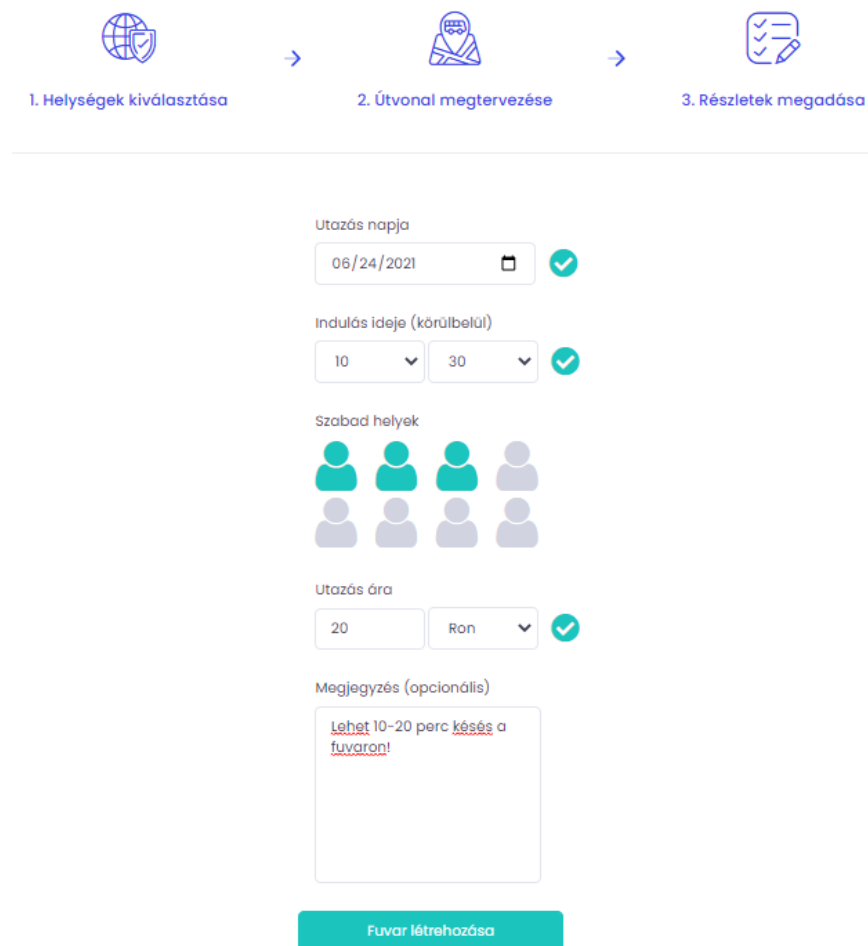
7.3. Fuvar létrehozása

A fuvar létrehozása az oldal legfontosabb funkciója. Ez az a funkcionalitás, amiben a legtöbbet kihoztam a Google Maps API-ból. Az itt írt kódot, és hogy mi történik a háttérben a kutató munka

keretein belül részletesen is elemezni fogom, itt most csak a felhasználó szemszögéből fogom elmagyarázni a funkcionalitás működését.

Három lépésen kell végig haladni, hogy a fuvar létrehozzuk. Először meg kell adni a kezdő és a cél településeit az útvonalnak. Ezeket a településeket az adatbázisból kell kiválasztania a felhasználónak, miután a nevük megjelenik a kereső mező alatt. Az adatbázisban amelyik településnek van románra, magyarra, illetve angolra fordítható neve, azokat is el tároljuk, így bármilyen nyelven keresi a felhasználó, a megfelelő települést fogja az oldal megjeleníteni.

Második lépésben meg kell terveznünk az útvonalat. Először az alkalmazás a legrövidebb útvonalat ajánlja, de a felhasználó más útvonalat is tervezhet, megadva köztes településeket. Itt a felhasználói élmény fokozása végett megmutatjuk azt is, hogy hány kilométer hosszú az útszakasz, valamint, hogy körülbelül mennyi ideig fog tartani az utazás.



1. Helységek kiválasztása

2. Útvonal megtervezése

3. Részletek megadása

Utazás napja

06/24/2021

Indulás ideje (körülbelül)

10 30

Szabad helyek

Utazás ára

20 Ron

Megjegyzés (opcionális)

Lehet 10-20 perc késés a fuvaron!

Fuvar létrehozása

9. ábra – Fuvar részletei

Utolsó lépésben egy űrlap jelenik meg, ahol meg kell adni a következőket: utazás dátuma (nap, óra), szabad helyek száma, utazás ára, megjegyzés (a megjegyzés írása opcionális). Az adatokat itt is validáljuk a Back-enden, hasonlóan, mint a regisztrációnál. Ha minden sikeresnek bizonyult, létrejön a fuvar, erről a felhasználót egy felugró ablak segítségével informáljuk, majd elnavigáljuk a Fuvarok menübe, ahol máris láthatja a többi fuvar mellett, azt amelyiket most hozott létre.

7.4. Fuvar keresése

Ez a funkció arra szolgál, ha a felhasználó egy konkrét útvonalon és egy konkrét dátumon szeretne utazni, akkor könnyedén tudjon ilyen fuvart keresni. Ekkor elnavigál a Fuvar keresése menüre, ahol három bemeneti mezőt talál. Itt meg kell adnia az induló pontot, az érkezési pontot és a dátumot, majd rá kell klikkeljen a Keresés gombra. Ezután az alkalmazás kilistázza azokat a fuvarokat, amelyek ezen az útvonalon vannak, vagy érintik ezt az útvonalat. Fontos megjegyezni, hogy nem kell konkrétan érintse az adott helységet az útvonal. Ha egy-két kilométerrel a megadott célállomás mellett halad el az útvonal, az mégis kilistázódik, mert érdekelheti a felhasználót a fuvar. Megeshet, hogy megbeszéli a fuvart kereső a sofőrrel (akár chaten keresztül), hogy tegye őt ki egy-két kilométerrel arrébb, vagy az is megeshet, hogy kiszáll a célállomásához legközelebbi ponton és tovább valamilyen más módon, akár taxival jut el a céljához. A fuvarok listázásánál azt is figyeljük, hogy ne csak a felhasználó által megadott dátumon levő fuvarokat, hanem két-három nappal korábban (de ugyanazon az útvonalon) történőket is listázzuk. Erre szintén azért van szükség, mert a felhasználót érdekelhetik ezek a fuvarok.

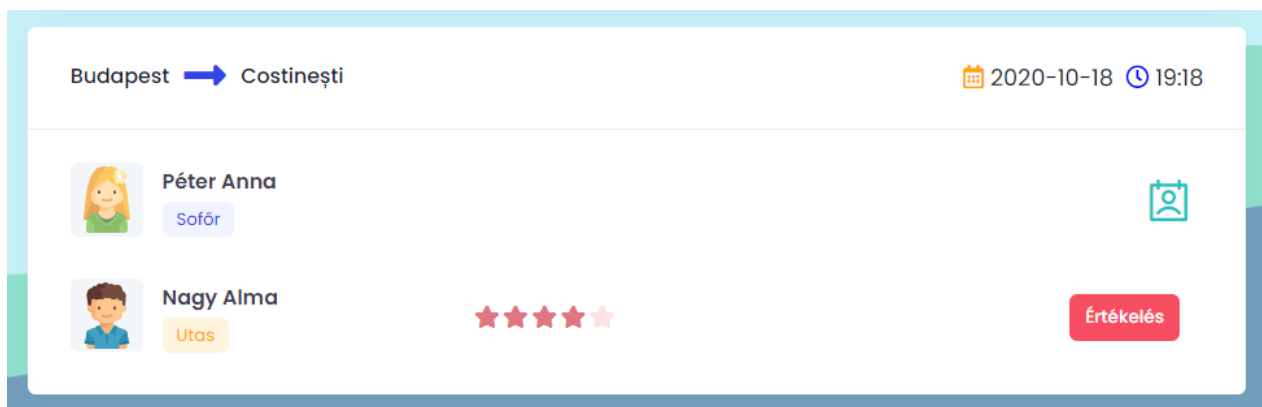
Megtörténhet, hogy a fuvart kereső személy által megadott paraméterekre nincsen találat, viszont ekkor sem engedjük el a felhasználó kezét. Abban az esetben, ha nincs számára megfelelő fuvar, megjelenik egy szöveg, amely informálja a felhasználót, hogy nem találtunk ilyen fuvart, ugyanakkor lehetősége van, hogy az „Utazás figyelőt” bekapcsolja. Ezt az informáló szöveg alatt található gomb lenyomásával lehet kapcsolni. Bekapcsolás után elmentjük, hogy milyen útvonalon szeretne utazni a felhasználó és természetesen, hogy milyen dátumon. Később, ha létrejön egy olyan fuvar, ami az adatok szerint érdekelheti a felhasználót, értesítést küld neki az alkalmazás. Az értesítésekkel később bővebben is foglalkozok. Az értesítések küldése egészen addig történik, amíg lejár az a dátum, amit a fuvar keresés kereteiben megadott a kliens.

7.5. Aktív fuvarok

Ez a menüpont arra szolgál, hogy a felhasználó megtudja nézni azokat a fuvarokat, amelyeknek ő a sofőrje, vagy amelyeket lefoglalt és még mindig aktívak (az utazás dátuma még nem járt le). Itt lehetőség van megtekinteni az adott fuvart, illetve az útítársakat, legyen az sofőr vagy utas. Abban az esetben, ha a felhasználó a későbbiekben le szeretné törölni a fuvart, amit létrehozott, arra is ebben a menüben van erre lehetősége. Ekkor amennyiben vannak a fuvarnak utasai, azok értesítést fognak kapni arról, hogy töröltődött a fuvar. Hogyha utasként van számon tartva a felhasználó, de le szeretné mondani az utazást, azt szintén itt teheti meg, ekkor a sofőr kap arról értesítést, hogy visszamondták a foglalást.

7.6. Előzmények – útitársak értékelése

Az útitársak értékelése funkció az előzmények menüben található. Ebbe a menübe automatikusan bekerülnek a felhasználó azon fuvarjai, amelyeknek az indulási időpontja lejárt. A fuvar továbbra is meg lehet nyitni, megnézni a részleteit, illetve az útvonalat, viszont az is látszik, hogy ez a fuvar már nem aktív, foglalni sem lehet rá. Az itt lévő fuvaroknál látszik az is, kikkel utaztunk, a felhasználók profilképe és neve, akik foglaltak a fuvarra és ezt a foglalást a sofőr elfogadta. Ezeknek a felhasználóknak az értékelése nagyon egyszerűen lehetséges, ahogy a 10-es ábra is mutatja, a felhasználó neve mellett megjelenő 5 csillagból kiválasztjuk hányat szeretnénk adni, majd az Értékelés gombra klikkelve azonnal mentjük ezt az értéket.



10. ábra – Útitárs értékelése

Azért tartottam fontosnak ennek a funkciónak a beépítését, mivel így mikor egy felhasználó létrehoz egy fuvar, akkor azt a fuvar megnyitva más felhasználók látni fogják az értékelését. Ebből az értékből lehet tudni, hogy mennyire jó potenciális útitárs az a felhasználó, akivel utazni fogunk, ez megkönnyítheti a döntést a foglalással kapcsolatban.

Értékelést kapni és adni más felhasználóktól amúgy is hasznos, mivel ez befolyásolhatja a viselkedésüket mind a platformon, mind pedig az utazás alatt személyesen. Ezzel próbáltam pozitívabbá, barátságosabbá tenni a weboldalt és annak közösségét, illetve lényegében az értékelési rendszer ösztönzi a felhasználókat a pozitív viselkedésre.

Ahhoz hogy megtekinthessük más felhasználók rólunk leadott értékeléseit, a profil fülben található Értékelésem menüt kell megnyitni. Már a profil fülben látszik az értékelés átlaga, és az Értékelésem menüben az is látható ki adta le az értékelést, és hogy melyik fuvaron.

A kivitelezés során ahhoz, hogy elküldjük a Front-endről az értékelést a Back-endre, és ehhez ne keljen újra tölteni az oldalt, Ajax-ot használtam. Amikor az eddigi adat küldéséről beszéltem a Front-endről, az az űrlapok kitöltése után történt a Submit gomb megnyomásával, ilyenkor pedig az oldal mindig újra töltődött, vagy elnavigált máshova. Az értékelések menünél fontos volt, hogy

minden értékelés leadása után, ne kelljen újra tölteni az oldalt, hogy folyamatos maradjon a funkcionalitás használata a felhasználók szemében. Ezt lehet Ajax kéréssel kivitelezni, amire egy példakód látható a 11-es ábrán.

```
$.ajax({
  url: "/giveRating",
  type: 'POST',
  dataType: 'json',
  data: { user_from: myId, user_to: userId, ride_id: rideId, rating: clickedRatingNumber, _token: '{{csrf_token()}}' },
  success: function (data) {
  },
  error: function () {
  }
});
```

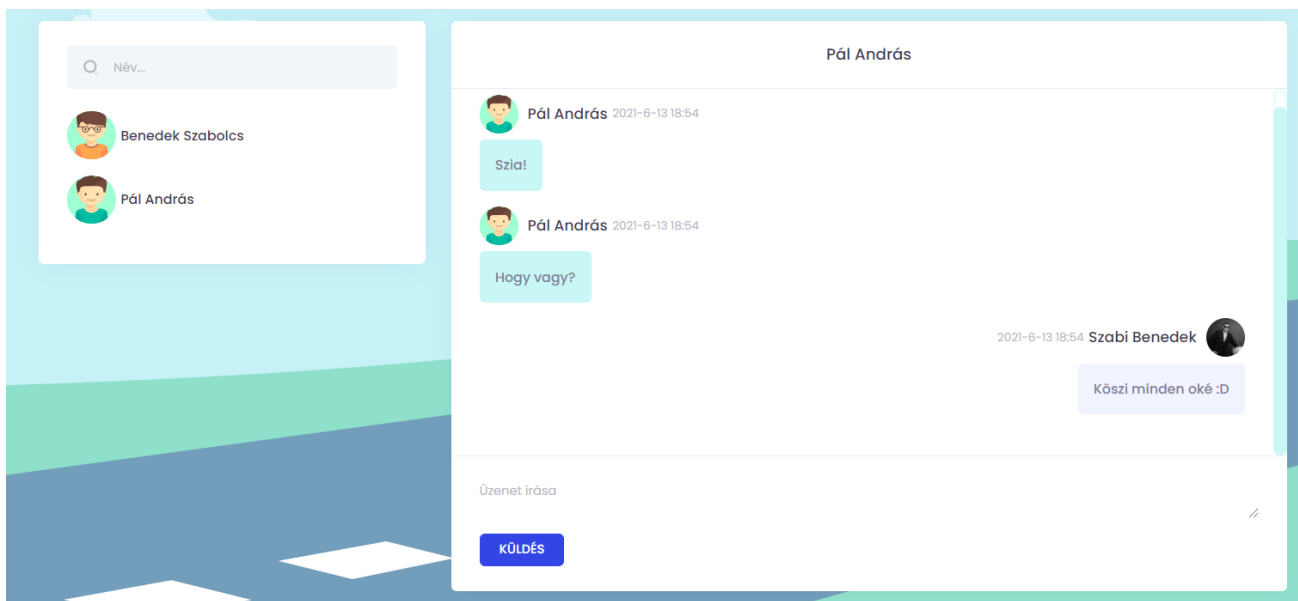
11. ábra – Ajax kérés

Ahogyan a kódból is látszik egy Ajax kérést tudunk Front-Endről intézni. Meg kell adnunk az URL-t, a kérés típusát (legtöbb esetben POST vagy GET), az adat típusát, illetve a paraméterlistát. Ezt az útvonalat előre megírjuk a web.php fájlban, így a kérés elküldése után tudjuk, hogy melyik vezérlő fájlunk melyik függvényét fogja meghívni a kérésünk. Ebben a függvényben feldolgozzuk a kapott adatokat, és ha szükséges visszaküldhetünk egy üzenetet vagy adatokat Json formátumban. Ezzel a módszerrel miután a felhasználó ráklikelt az Értékelés gombra, úgy töltődik fel a leadott értékelés az adatbázisba, hogy azt a felhasználó nem is érzékeli.

7.7. Chat

Nagyon fontos, hogy a chat funkcionalitás megjelenjen egy ilyen típusú szoftverben. A The Ride-on az emberek útitársakat keresnek, akikkel sok részletet meg kell beszéljenek a fuvar előtt, például, hogy pontosan hol találkozzanak, mivel a weboldalon csak a város van meghatározva. Meglehetett volna oldani ezt a problémát úgy is, hogy egy linket rakok ki a Facebook profiljához a felhasználónak, így a Facebookot megnyitva, a Messengert használva is megbeszélhették volna a részleteket, de jobbnak tartottam, hogy az én alkalmazásomban legyen erre lehetőség.

A chat Front-endjének elkészítéséhez a Metronic CSS sablon egy előre elkészített chatjét használtam, amit majd később testre szabtam, hogy illeszkedjen az oldal dizájnjához.



12. ábra – Chat

Fontos, hogy ez az oldal is, mint a legtöbb, reszponzív legyen. Telefonról megnyitva a 12-es ábrán bal oldalon látható mező jelenik meg, ahol láthatók azok a felhasználók, akikkel már chateltünk. Egy gomb segítségével előhozható majd újra eltüntethető ezeknek a felhasználóknak a listája. Ilyen és ehhez hasonló megoldásoknál sokat segített a már említett Metronic CSS sablon.

A chatet meg tudjuk nyitni úgy is, ha a profil fölénél a Chat gombra klikkelünk, ilyenkor a főmenübe kerülünk, ahol ki kell választani a listából, hogy melyik felhasználóval szeretnénk chatelni. A másik módja annak, hogy idekerüljünk, ha fuvar keresés közben egy megnyitott fuvarnál ráklikkelünk a „Chat a sofőrrel” gombra. Ilyenkor megnyílik a chat, a fuvar sofőrje pedig azonnal bekerül a felhasználók listájába, és ha írunk neki, akkor ezt a rendszer megjegyzi, és később is ott marad. Miután egy felhasználó üzenetet küldött egy másik felhasználónak, és ennek a másik felhasználónak épp nincsen megnyitva a beszélgetés, akkor azonnal, élőben értesítést kap. A rendszer azt is megszámlolja, hogy hány olvasatlan üzenete van, és az értesítésnél meg is jeleníti ezt a számot, ezzel is növelve a felhasználói élményt.

Ahhoz, hogy meg tudjam oldani mindezt, úgy gondoltam jobb, ha nem MySQL adatbázist használok, hanem egy olyan helyen tárolom az elküldött üzeneteket, amihez a Front-Endről közvetlenül hozzáférék. Ez azért fontos, hogy ne kelljen bizonyos időközönként vizsgálni az adattábláimat, figyelve, hogy jött-e üzenet, hanem egy olyan szolgáltatást használok, ami azonnal Front-enden értesít, ha egy felhasználó üzenetet ír (ha egy új adat kerül a táblába). Ezesetben szintén a Google szolgáltatásaihoz fordultam segítségül, pontosabban a Firebase - Cloud Firestore adatbázisához.

7.7.1. Cloud Firestore

A Cloud Firestore egy rugalmas, skálázható adatbázis a Firebase családjából. A Firebase a Google szolgáltatása, segítséget nyújt mobil-, web- és szerverfejlesztésben. Valós idejű Figyelők (Listeners) segítségével szinkronban tartja az adatokat az ügyfélalkalmazások között. Offline támogatást is kínál, így olyan gyorsan reagáló alkalmazásokat készíthetünk, amelyek a hálózati késéstől vagy az internetkapcsolattól függetlenül működnek. A Cloud Firestore zökkenőmentesen integrálódik más Firebase és Google Cloud termékekkel is.

Ahhoz, hogy értesítést küldhessünk a felhasználónak arról, hogy üzenete érkezett, és ezt az üzenetet azonnal meg is tudjuk jeleníteni neki, ha épp meg van nyitva a beszélgetés, meg kell írunk az adattábla figyelőjét Front-enden. A 13-as ábrán ennek a figyelőnek a kódjából látható egy részlet.

```
//Go through the database and get the messages, always listen for the new messages
async function ListenerForNewMessage(){
  await firebase.firestore().collection('messages').onSnapshot(function(snapshot) {
    snapshot.docChanges().forEach(function(change) {
      if (change.type === "added") {
        messages.push(change.doc.data());
      }
    });
  });
}
```

13. ábra – Firestore figyelő

Ahogy az ábrán is látható, megjelenik az utasítás előtt az „await” szócska, ami arra szolgál, hogy várja meg a függvény az adatok lekérését az adatbázisból, és csak azután hajtsa végre a függvényben megírtakat, miután a lekérdezésre megkaptuk a választ. Ahhoz, hogy használhassuk ezeket a Firebase függvényeket az alkalmazásunkban, inicializálni kell a Firebase-t és összekell kapcsoljuk a projektünket a Firestore adatbázissal. Ezek elvégzéséhez részletes útmutatót kapunk a Google-től. Meg kell adni a kollekciók nevét, ami lényegében az adattáblánk neve, esetünkben ez a messages (üzenetek), ilyenkor a Figyelőnk ebben a táblában vizsgálja az adatokat.

A függvényen belül a Change nevű objektumba kapjuk meg a lekérdezett adatokat, és a type attribútumból azt is megtudhatjuk, hogy az adatot most újonnan adták-e hozzá a táblához. Ezután bármit csinálhatunk a lekérdezett adattal, esetünkben én hozzáadtam a messages tömb változóba, majd később megjelenítettem a beszélgetésben így létrehozva az élő chatet. Annak érdekében, hogy mindig értesítést kapjon róla a felhasználó, ha valaki üzenetet írt neki, inicializálnom kellett a Firebase-t az egész projektben, nem csak a chat oldalon. A user-blade.js fájlban, amelyik mindegyik oldalon használva van, szintén készítettem egy ehhez hasonló Figyelőt, így ha új üzenet érkezik a Firestore adatbázisban, akkor azt mindig tudatjuk a felhasználóval, értesítés hangeffekttel, illetve a profil fülben látványosan jelzem pirossal, hogy hány olvasatlan üzenet vár még válaszra.

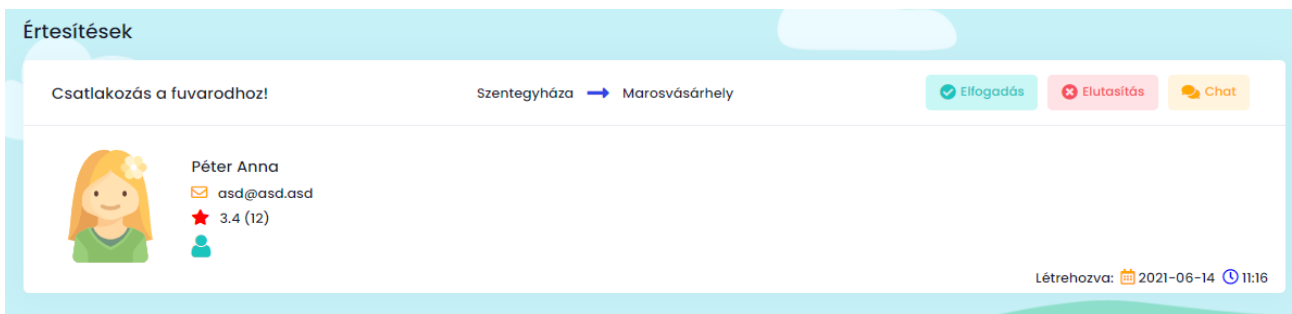
7.8. Értesítések

A The Ride-ba összetett értesítési rendszert készítettem. Fontosnak tartottam, hogy akár a chat esetében, itt se kelljen egy másik alkalmazást használni, ahhoz, hogy megfelelően tudjunk útítársat keresni. Ez alatt azt értem, hogy sok oldalnál úgy van megoldva az értesítési rendszer, hogy egy e-maillt kap a felhasználó, ha az oldal valamilyen információt akar vele közölni, és hogy ezt elolvassa, értelemszerűen be kell lépjen a levelező fiókjába.

Az én alkalmazásom 7 féle értesítést tud küldeni:

- Valaki csatlakozni akar a Fuvarodhoz
- A csatlakozási kérelmedet elfogadta a sofőr
- A csatlakozási kérelmedet elutasította a sofőr
- A rendszer talált egy fuvart, ami érdekelhet (az utasítás figyelő bekapcsolásakor)
- Valaki visszamondott egy foglalást a fuvarodon
- A sofőr törölte a fuvart, amelyiken foglalásod volt
- Kaptál egy értékelést valakitől

Mindegyik értesítésnél, ha új érkezik, akkor akárcsak a chat esetében, azonnal tudatjuk a felhasználóval, hangeffektet, illetve vizuális jelzést is használva. Abban az esetben, ha nem jött új értesítés, viszont a régiék közül van olyan, amit még nem nyitott meg a felhasználó, akkor miután bejelentkezett, ugyanezzel a módszerrel értesítjük, hogy ideje megnyitni az Értesítések menüt. A Notifications táblába vannak eltárolva az értesítések, és van egy Checked oszlop, ami egy boolean érték, ez jelzi, hogy megnyitotta a felhasználó az adott értesítést, vagy sem. Ez az érték mindaddig hamis, amíg biztosan tudjuk, hogy a felhasználó egyszer sem látta a közölni kívánt információt. Mivel az értesítéseket nem a chatnél használt Firestore adatbázisban tárolom, hanem mint a többi adatot, MySQL-be, ezért nehezebb dolgom volt az élő értesítés küldés beépítésére. A módszer, amit választottam nem a legprofibb, a kelleténél jobban terheli a rendszert, de kevés felhasználóval rendelkező alkalmazások esetében nyugodtan használható. Egy időzítőt építettem be, ami folyamatosan (minden négy másodpercen) figyel, hogy lett-e új adat beírva a Notifications táblába. Abban az esetben, ha új adat bekerülését észleltük, és a felhasználó azonosító (userId), amelyikre küldték, megegyezik a bejelentkezve lévő felhasználó azonosítójával, akkor máris tudjuk, hogy figyelmeztetésre van szükség, mivel a kliens kapott egy új értesítést.



14. ábra – Értesítés

Ahogy a 14-es ábrán is látható, megpróbáltam az értesítéshez fűződő összes lényeges információt megmutatni. Ez a példa egy értesítés arról, hogy ha valaki csatlakozni szeretne egy általunk létrehozott fuvarhoz. Ebben az esetben látható a fuvar kezdő és végállomása, de megnyitható részletesen is innen az Értesítések menüből azzal, hogy ráklikkelünk a kezdő vagy végállomásra. Láthatók a felhasználó adatai, aki velünk szeretne utazni: neve, e-mail címe, értékelése (a zárójelben látható, hogy hány leadott értékelésnek az átlaga az adott összeg) illetve, hogy hány helyet szeretne lefoglalni a fuvar során.

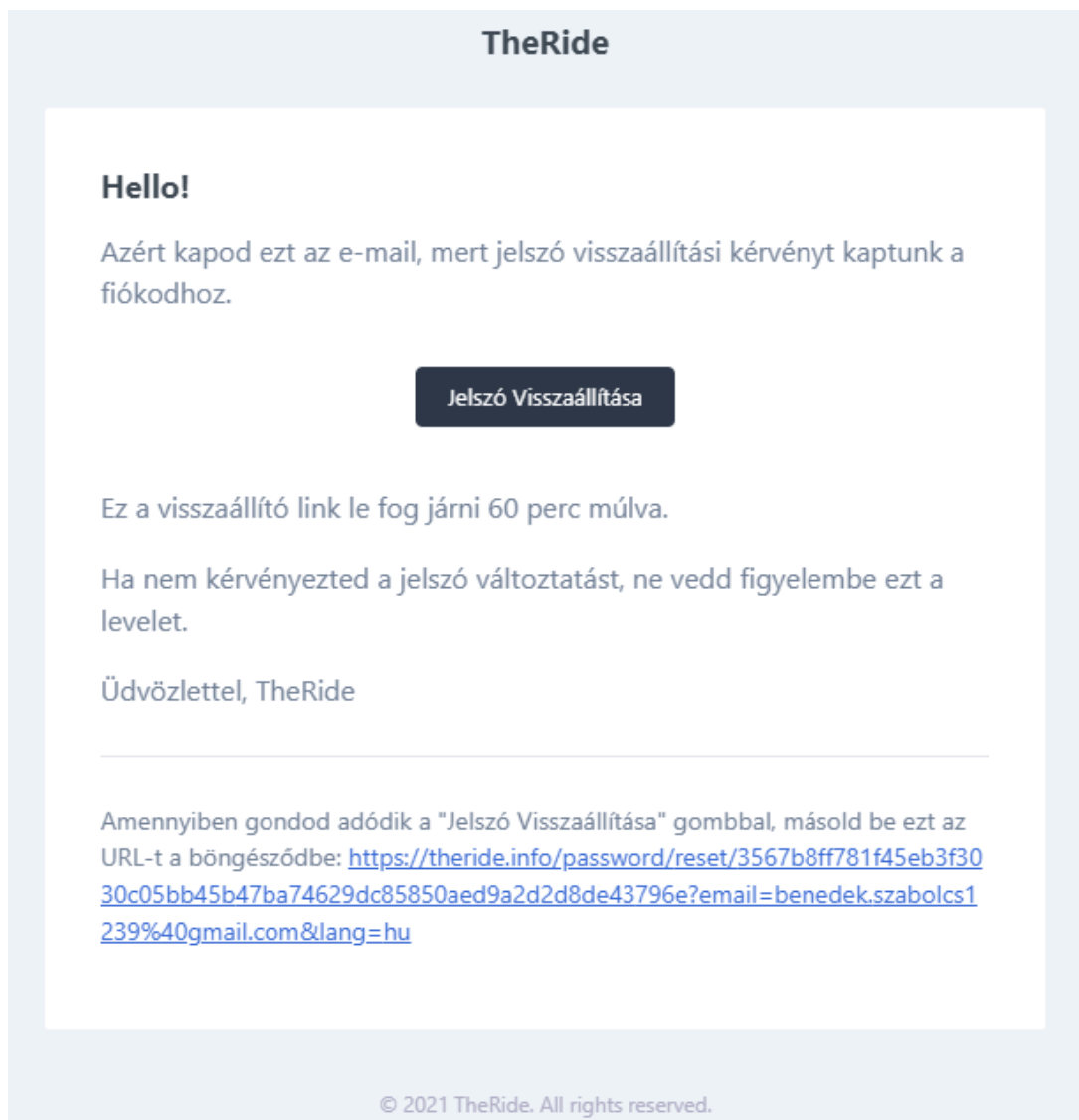
Lehetőségünk van ezesetben elfogadni vagy elutasítani a kérelmét, mindkét esetben a másik fél erről szintén értesítést fog kapni. Ezek mellett pedig lehetőség van Chatelni az értesítést küldővel, hogy a fuvar részleteit egymást közt megtudják beszélni a felhasználók.

7.9. Jelszó visszaállítása

Nagyon fontos az ilyen jellegű alkalmazásoknál, hogy legyen lehetőség az elfelejtett jelszót visszaállítani. Ez a funkcionalitás nagyon elterjedt, és minden valamire való alkalmazásban benne is kell legyen. Általában mindig meg kell adni regisztrációkor egy e-mail címet, így abban az esetben, ha a felhasználó elfelejti a regisztrációkor megadott jelszavát, viszont az e-mail címe benne van az adatbázisban, akkor ezzel az e-mail címmel azonosíthatja magát úgy, hogy a rendszer levelet küld az e-mail fiókjára, és a levelet megnyitva hozzáfér a profiljához. Ilyenkor általában az első lépés egy új jelszó megadása szokott lenni.

A Laravel keretrendszer megalkotásakor gondoltak rá, hogy gyakran fog szerepelni ez a funkcionalitás weboldalaknál, ezért nagyon egyszerűen használható módon bele tudjuk építeni ezt az alaphoz megírt funkciót a projektünkbe. Én magam is ezt a megoldást választottam. Amikor a felhasználó elfelejti a jelszavát, a bejelentkező panelnél van lehetősége kiválasztani a jelszó visszaállítását az „Elfelejtette a jelszavát?” szövegre klikkelve. Ilyenkor megjelenik egy beviteli szövegmező, ahová be kell írnia a regisztrációkor megadott e-mail címét, majd a „Link küldése a jelszó visszaállításához” gombra klikkelve kérhető a visszaállító linket. A rendszer ilyenkor ellenőrzi, hogy az adatbázisban valóban szerepel-e az e-mail cím valamelyik felhasználó fiókjában, és ha igen, akkor ki is küldi az e-mailt. Mindkét esetben (ha valós az e-mail cím, ha

nem) tudatjuk a felhasználóval, hogy elküldtük a visszaállításhoz szükséges e-mailt, vagy pedig közöljük vele, hogy a megadott e-mail cím nem szerepel az adatbázisban.

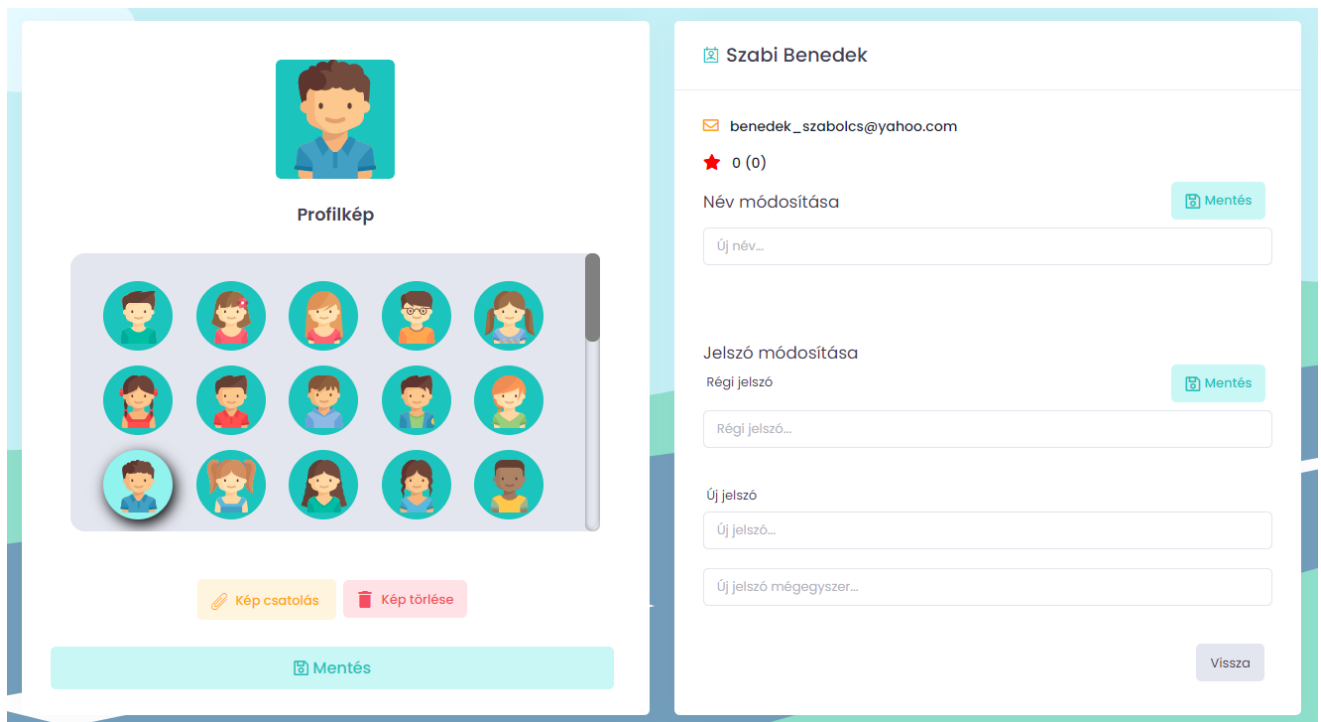


15. ábra – E-mail a jelszó visszaállításához

A fent látható e-mailhez van egy előre elkészített sablon a Laravel fájlljai között, amit tetszés szerint módosíthatunk. Én pluszba azt is beállítottam, hogy az e-mail olyan nyelven érkezzon meg a felhasználóhoz, amilyen nyelven használta az oldalt, akkor amikor a jelszó visszaállítását kérvényezte. Ennek kivitelezéséhez elküldtem az oldal URL-jében levő lang változót, és az e-mail felépítésénél a változó értékének függvényében töltöttem fel szöveggel a sablont. Az elküldött e-maillal küldünk egy kódot is, ami 60 másodpercig aktív, ennyi ideje van a felhasználónak, hogy a linket használva visszatérjen a The Ride-ra, ahol egy űrlap várja. Ebben az űrlapban meg kell adnia az új jelszavát, majd az új jelszót még egyszer, a biztonság kedvéért. Ezeket mentve máris a főoldalon találja magát, és a profiljába ezentúl már az új jelszóval léphet be.

7.10. Profil adatok módosítása

A weboldal utolsó funkcionalitása a profil adatok módosítása. Ahogy az elnevezésből is kitalálhatjuk, egy egyszerű profil fület kell elképzelni, ahol meg van jelenítve minden a profillal kapcsolatos adat. Ilyenek a felhasználónév, e-mail cím, értékelés, profilkép. A felhasználónak pedig lehetősége van megváltoztatni a jelszavát, felhasználónevét vagy profilképét.



16. ábra – Profil adatok

A név és jelszó módosításnál űrlapokba tettem be a bemeneti mezőket, és a Mentés gombra klikkelve küldöm el az adatokat egy GET metódussal a Back-Endre, ahol a szokásos módon validálom, majd vissza küldöm a hibüzenetet, vagy a sikeres módosításról szóló értesítést.

A felhasználónak már regisztráláskor van lehetősége profilképet választani az alap avatárokból vagy feltölteni sajátot. Abban az esetben, ha Facebookkal regisztrált, akkor a profilképe ugyanaz lesz, mint a Facebookon a jelenlegi profilképe. Itt bármikor lecserélheti a profilképét, egyszerűen kiválasztja, vagy feltölti az új képet, majd a Mentés gomb megnyomásával már le is cserélte.

8. Kutatómunka - Google Maps API

A dolgozatom utolsó nagy fejezetében a kutató munkámat fogom kifejteni, amit a Google maps API-on végeztem. Megnézzük mi is az a Google Maps, milyen API-t nyújt használatához a Google, hogyan kell a saját projektünkben inicializálni, hogyan intézzünk kérést hozzá, mi a tőle kapott válasz, és végül, hogy hogyan használtam fel az alkalmazásomban.

8.1. Google Térkép

A Google Maps vagy Google Térkép a Google által fejlesztett ingyenes térkép és navigációs szolgáltatás. Az elérhető térképek és műholdfelvételek lefedik az egész Földet. A cég 2005-ben jelentette be a fejlesztést, ekkor csak az Internet Explorer és a Mozilla támogatta, később még természetesen hozzájött az Opera és Safari is. Hat hónapos bétateszt után elindult, „Google Local” névvel.

A többi Google alkalmazáshoz hasonlóan szinte teljesen JavaScriptre épül. A térkép mozgatásakor a kis térkép szeletek folyamatosan töltődnek le a szerverről. Ahhoz, hogy élőben tudja a Térkép lekérni az adatokat, a Google fejlesztői Ajax-ot használtak, amit én is szemléltettem az Előzmények funkcionalitás bemutatásakor.

A térképen számos nagyobb várost már 3D-ben is bejárhatunk, ebben az esetben az épületek és a többi tereptárgy is textúrázott. A térképhez az adatbázist a Google különböző helyekről szerzi be, a Navteq nevű cég volt először a fő partnerük, későbbiekben (ma már ez a megszokott) a térképeket a közösség frissíti a Google Térképkészítő segítségével. A Google mára már számos alkalommal végigjárta a Föld útjainak nagy részét, és saját autóival 360 fokos kamera segítségével fényképezte azokat. Így mára akár „gyalogosan” is végig sétálhatunk a városokon, és valódi formájukban megtekinthetjük azokat.

A könnyen kezelhető felhasználófelület, valamint a keresésre alkalmas térképek és műholdképek miatt a Google Maps szolgáltatás hamar népszerűvé vált. 2005 júniusában megjelent a kutatásom fő alapeleme, a Google Maps API, melynek programozásához szükséges fejlesztői kulcs bárki számára ingyenes. A megjelenet API letisztultsága, gyorsasága, interaktivitása és fejlesztőbarát mivolta miatt gomba módra kezdtek szaporodni az olyan alkalmazások, melyek kisebb vagy nagyobb formában használják a Google Maps-et, s ilyen alkalmazás a The Ride is. Jelenleg (2021 júniusa) több, mint 5.200.000 weboldal használja a Google Maps API-t.

8.2. Használata

Ahhoz, hogy bármilyen Google szolgáltatást használjunk a saját projektünkben, először kell szereznünk egy API kulcsot (API key). Ezt az egyedi kulcsot ingyenesen generálhatjuk a Google for Developers platformon. Ehhez részletes útmutatót kapunk a következő linken:

<https://developers.google.com/maps/documentation/places/web-service/get-api-key>

Miután megvan a kulcs egy JavaScript fájl formájában berakjuk a Google Maps API-hoz tartozó linket a HTML fájlunkba „<script></script>” címkék (tags) közé beleépítve a kigenerált kulcsot is. Ehhez a fájlhoz paraméterben tudunk adni egy language változót, ami arra szolgál, hogy

milyen nyelven jelenjen majd meg a térkép. Az én projektem ugyebár három nyelvű, ezért a saját lang változóm értékének függvényében töltöm be a megfelelő nyelvű JavaScript fájlt, ahogyan a 17-es ábrán is látható.

```
@if(app()->getLocale() == 'ro')
<script src="https://maps.googleapis.com/maps/api/js?language=ro&key=AIzaSyBhsouaDYFdYICiHS4AIS50Elhkr_36IN0&callback=initMap"></script>
@elseif(app()->getLocale() == 'en')
<script src="https://maps.googleapis.com/maps/api/js?language=en&key=AIzaSyBhsouaDYFdYICiHS4AIS50Elhkr_36IN0&callback=initMap"></script>
@else
<script src="https://maps.googleapis.com/maps/api/js?language=hu&key=AIzaSyBhsouaDYFdYICiHS4AIS50Elhkr_36IN0&callback=initMap"></script>
@endif
```

17. ábra – Maps API-hoz szükséges fájl betöltése

A HTML kódba szükséges létrehoznunk egy elemet (mondjuk egy átlagos div-et), aminek beállítunk egy azonosítót, és majd a térképet erre az elemre fogjuk rátenni a JavaScript kódból. Ennek az elemnek a méretét CSS segítségével olyanra módosítjuk amilyenre szeretnénk.

Miután megvan a kulcs, inicializáltuk a Google Maps API fájlt, és létrehoztuk a térkép megjelenítésére szolgáló HTML elemet, készítünk egy JavaScript fájlt (amit szintén inicializálunk a HTML kódba) és ebben a fájlban a 18-as ábrán látható módon megjelenítjük a térképet.

```
map = new google.maps.Map(document.getElementById('map'), {
  center: {lat: -34.397, lng: 150.644},
  zoom: 8
});
```

18. ábra – Térkép létrehozása

A fenti kód lefutása után azonnal megjelenik a térkép a map azonosítóval rendelkező HTML elemben. Később a map változót, amit itt hoztunk létre, használhatjuk különböző funkcionálisok beépítésére. Minden térképnek kell legyen egy zoom és egy center attribútuma. A center arra való, hogy milyen koordináták legyenek a térkép közepén, a zoom feladata pedig, hogy beállítsa, mennyire legyen rá közelítve a térképre. Az elkészült térképnek ezek csak a kezdőértékei, természetesen a Google Térképet a felhasználó interaktívan bejárhatja: görgetéssel közelebbre vagy távolabbra állíthatja a zoom-ot, illetve a bal egérgombot nyomva tartva „húzogathatja” a térképet más irányokba. A zoom és centeren kívül sok más attribútumot is beállíthatunk ezek csak a legalapvetőbbek.

Hogyha az egész Földet egyetlen képként kínálná fel a Google, akkor egy hatalmas térképre lenne szükség. Ennek eredményeként a Google Maps és a Google Maps JavaScript API-n belüli térképek „csempékre” (tiles) és nagyítási szintekre (zoom levels) vannak felosztva. Alacsony nagyítási szinteken a térképcsempék kis készlete széles területet fed le, nagyobb nagyítási

szinteknél a lapok nagyobb felbontásúak és kisebb területet fednek le. Az alábbi lista azt a hozzávetőleges részletességet mutatja, amelyet várhatóan láthatunk az egyes nagyítási szinteken.

- 1. Egész világ
- 5. Kontinens
- 10. Város
- 15. Utcák
- 20. Épületek



19. ábra – Nagyítási szintek szemléltetése

A 19-es ábrán Tokyo látható a Google Térképen különböző nagyítási szintbeállítások után. Az első képen a zoom 0, a másodikon 7 és a harmadikon 18.

Mindezen felül lehetőségünk van saját dizájnt adni a térképnek. A térkép konstruktorába, (ahol megadtuk a zoom és a center attribútumokat) megadhatunk egy styles tömböt, amelyben objektumok vannak, és itt tetszés szerint testre szabhatunk elég sok mindent. Egy másik fontos attribútum, amit én is használtam, egy boolean érték, a neve pedig `disableDefaultUI`. Ezt az értéket, ha nem állítjuk be mi magunk, akkor alapból `false`. Én a saját térképemnél `true`-ra állítottam, mivel nem szerettem volna, ha Felhasználói Felület elemek lettek volna a térképemen. Tehát ezt kikapcsolva csak egy térképet kapunk (ahogyan a 19-es ábrán is van), úgy hagyva, ahogy van, viszont megjelennek gombok, amelyekkel különböző interaktivitásokat lehet csinálni. Az én alkalmazásomban ezekre nem volt szükség, csak zavarók lettek volna.

8.3. Directions Service – Útvonal Tervezés

A `DirectionsService` egy objektum, amely használatával útvonalakat jeleníthetünk meg a Google Térképen. Ez az objektum kommunikál a Google Maps API Directions szolgáltatással, amely fogadja a kéréseket, és hatékony útvonalat ad vissza. Használatához be kell állítanunk a

Google platformján (ahol kulcsot generáltunk az API használatához) a Directions API használatát, s fontos, hogy ugyanarra a projektre állítsuk be, ahol a Maps JavaScript API is szerepel.

Működését úgy kell elképzelni, hogy létrehozunk egy `directionService` változót, szintén Google konstruktorral. Ehhez nem kell újabb JavaScript fájlokat importálnunk, ha hozzáadtuk a projektünkhöz a Directions API-t, akkor a generált kulcsból használhatjuk ezt a konstruktort és a Direction Service függvényeit. Ezután kérést kell intézzünk a Maps API Directionsnak, beállítva különböző mezőket, majd erre a kérésre egy választ ad a Google a beállított mezők függvényében. Ezt a választ úgy ahogy van, meg tudjuk jeleníteni a térképen, de arra is van lehetőség, hogy szétszedjük darabokra, és különböző adatokat szedjük ki belőle, amiket ilyen vagy olyan módon beépíthetünk az alkalmazásunkba. A kapott objektum json formátumban van, így nagyon könnyű vele dolgozni.

8.3.1. Directions-hoz intézett kérés

Kérés intézéséhez a létrehozott `DirectionService` típusú objektum `„Route()”` függvényét kell meghívunk. Ennek a függvénynek kell tartalmaznia azt az objektumot, amelybe beállítjuk az útvonalunk adatait, majd pedig ennek a `„Route()”` függvénynek a válasz metódusában (callback method) fogjuk megkapni a válasz objektumot, amelyről részletesebben a következő fejezetben írok. A 20-as ábrán arra mutatok egy példát, hogy miként küldtem el a kérést a The Ride-ba.

```
var directionsService = new google.maps.DirectionsService;
//Request the first direction depends on 2 waypoints
function requestDirections(start1, start2, end1, end2) {
    startLatLng = new google.maps.LatLng(start1, start2);
    endLatLng = new google.maps.LatLng(end1, end2);

    directionsService.route({
        origin: startLatLng,
        destination: endLatLng,
        travelMode: google.maps.DirectionsTravelMode.DRIVING
    }, function(result) {
        renderDirections(result);
    });
}
```

20. ábra – Kérés küldése a Google Directions-hoz

Ahogy az ábrán is látható, létrehozom a `DirectionsService` típusú objektumot, majd pedig egy függvényben intézem a kérést. Ezt a függvényt akkor hívom meg, miután a felhasználó kiválasztotta a kezdő és cél állomásokat. Ezeknek az állomásoknak a koordinátái el vannak tárolva az adatbázisban, tehát nem kell más tennem, csak a koordinátákból létre kell hozzak egy-egy

Google Maps LatLng típusú változót, ezeket a változókat ezután felismeri a „Route()” függvényben levő objektum, és el tudja küldeni a kérést, a megfelelő kezdő és célállomásokkal. Ezeknek az attribútumoknak és a „travelMode-nak” kötelező módon mindig szerepelniük kell egy ilyen kérdésben.

A következő ábrán azt szemléltetem, hogy ezeken kívül milyen beállítási lehetőségek vannak még egy ilyen kérdésben, aztán mindegyiket el fogom magyarázni.

```
{
  origin: LatLng | String | google.maps.Place,
  destination: LatLng | String | google.maps.Place,
  travelMode: TravelMode,
  transitOptions: TransitOptions,
  drivingOptions: DrivingOptions,
  unitSystem: UnitSystem,
  waypoints[]: DirectionsWaypoint,
  optimizeWaypoints: Boolean,
  provideRouteAlternatives: Boolean,
  avoidFerries: Boolean,
  avoidHighways: Boolean,
  avoidTolls: Boolean,
  region: String
}
```

21. ábra – Directions kérés beállítási mezői

- **origin, destination:** Google koordináták a kezdő és végállomáshoz.
- **travelMode:** Milyen járművel utazunk, lehet: Driving, Walking, Bicycling, Transit
- **transitOptions:** Olyan értékeket határoz, amelyek csak azokra a kérésekre vonatkoznak, ahol a travelMode Transit. Olyanokra kell gondolni, hogy metróval, busszal akarunk-e utazni stb.
- **drivingOptions:** Olyan értékeket határoz meg, amelyek csak azokra a kérésekre vonatkoznak, ahol a travelMode Driving.
- **unitSystem:** Meghatározza, hogy milyen egységrendszert használjon az eredmények megjelenítéséhez.
- **waypoints[]:** Egy tömb, melyet akkor használunk ha az útvonal mentén köztes célállomások vannak. A tömbben ezek a célállomások vannak benne, szintén Google koordináta típusúak. A projektben én is használtam ilyen köztes állomásokat, erről később részletesebben is beszámolok.
- **optimizeWaypoints:** Boolean érték. Akkor állítjuk át true-ra, ha használunk köztes állomásokat és szeretnénk, hogy az útvonalunk optimalizálva legyen ezeknek a köztes céloknak a figyelembevételével a leghatékonyabb rendezéssel. Lényegébe az időtakarékosság függvényében újra rendezi a köztes célokat.

- **provideRouteAlternatives:** Ha igazra van állítva, akkor a Directions szolgáltatás egyenél több útvonal alternatívát is biztosít. Ne feledjük, hogy az útvonal alternatívák biztosítása megnövelheti a kiszolgáló válasz idejét.
- **avoidFerries:** Ha igazra állítjuk, akkor a kiszámított útvonal kerüli a kompokat.
- **avoidHighways:** Ha igazra állítjuk, akkor a kiszámított útvonal kerüli az országutakat.
- **avoidTolls:** Ha igazra állítjuk, akkor a kiszámított útvonal kerüli a fizetős utakat.
- **region:** Specifikálhatjuk vele a régió kódját, ez egy két karakteres szöveg.

Lehetőségünk van ezeket a beállításokat interaktívan változtatgatni, ezzel készítve egyedi útvonaltervező funkcionalitást a weboldalunkban. Nincs más dolgunk, mint egy Felhasználói Felületet biztosítani a térképünk mellé, majd a felhasználó beállításai szerint újabb és újabb kéréseket intézünk a Directions-nek, az ezekre kapott választ pedig folyamatosan megjelenítjük a térképünkön.

8.3.2. Directions-tól kapott válasz

A 20-as ábrán levő kódrészletből a `directionService „Route()”` függvényének válasz metódusában levő `result` változóba kerül bele a Google-től kapott válasz, egy json objektum formájába. Ezt az objektumot, ahogy a kódban is látszik, átadom egy `renderDirections` nevű függvénynek, és abban a függvényben jelenítem meg a térképen, ahogy a 22-es ábrán is látható.

```
//Render the direction depending on the google response
function renderDirections(result) {
    var directionsRenderer = new google.maps.DirectionsRenderer;
    directionsRenderer.setMap(map);
    directionsRenderer.setDirections(result);
}
```

22. ábra – Útvonal megjelenítése

Egy útvonal megjelenítéséhez a Google térképünkön nem kell más, mint létrehozni egy `DirectionRenderer` típusú változót (szintén a Google Maps konstruktorral), majd ennek a változónak a `„setMap()”` függvényével beállítjuk, hogy melyik térképet használja, végül pedig a `„setDirections()”` függvénynek átadjuk a Google Directions-tól kapott válaszbjektumot. A kód helyes lefutása után, ilyenkor megjelenik az útvonal a térképen a kérésben küldött beállítások szerint.

Ahogy azt fentebb is taglaltam, megéri megfigyelni a result objektumot, és másra is felhasználhatjuk azon kívül, hogy megjelenítjük a térképen. Önmagában az is egy jó funkcionalitás, hogy útvonalat jelenítünk meg egy weboldalon, de így, hogy kezünkben van egy ilyen összetett objektum, amelyben nagyon sok lényeges információ van a megjelenített útvonalról, a térképen kívül a weblap más részein is használhatjuk ezeket az információkat, minél interaktívabb módokon. A The Ride-nál én több módon is felhasználtam ezeket. Például, ahogyan a Fuvar Létrehozása funkció leírásánál is látható a 9-es ábrán megjelenítem, a térkép feletti HTML részben a tervezett útvonal hosszát kilométerben és az útvonal megtételéhez szükséges körülbelüli időt. Ezek és az ehhez hasonló megoldások véleményem szerint nagyban növelik a felhasználói élményt.

A következőkben a 23-as ábrán megmutatok egy ilyen válaszobjektumot, majd pedig elmagyarázom mire szolgálnak a lényegesebb elemek. Ezt a választ a 20-as ábrán látható kérésre kaptam válaszul a result változóba.

- **request:** A választömbben helyet kapott a kérdésben elküldött tömb is.
- **routes:** Egy tömb, amely tartalmazza a konkrét útvonalakat. Minimum egy eleme van abban az esetben, ha az útvonalnak csak egy kezdő és egy végállomása van.
- **status:** Ahogyan a nevében is benne van a válasz státuszát jelzi, ha minden rendben van, akkor „OK” -nak kell itt lennie.

A gyökérelemekből kiemelem és részletesebben bemutatom a geocoded_waypoints és routes tömböket, mivel ezekben vannak a lényegesebb információk az útvonalról.

8.3.2.1. geocoded_waypoints

Ennek a tömbnek az elemei 0 alaphelyzettől kezdve a kezdőpont, a köztes megállók és a célállomás. Mindegyik elem a tömbben megfelel egy-egy Google koordináta típusnak, és mindegyik egy-egy geokódolás eredménye, amit a Google Directions függvényei végeznek. Ezeket az elemeket magyarul útpontoknak nevezzük, és a következő részleteket tartalmazzák:

- **geocoder_status:** A geokódolási művelet eredményeként kapott állapotkódot jelöli.
- **place_id:** Egyedi azonosítója az útpontnak. Ezek az azonosítók a Google adatbázisában vannak eltárolva, innen tudja a Google Directions, hogy konkrétan hol van az adott útpont. Rengetek ilyent tárol a Google, mivel csaknem méter pontosan tudunk útvonalat tervezni, például ha egy házszámmal messzebb levő célállomást adtam volna meg, akkor egy teljesen más azonosító lenne most itt.
- **types:** Az útpont típusát jelzi. Értékei lehetnek: street_address (utca név), route (egy elnevezett út), intersection (útkereszteződés) stb.

8.3.2.2. routes

A válaszban kapott elemek közül ez a tömb tartalmazza a legfontosabb információkat. Ebben a tömbben van eltárolva a konkrét útvonal. Az útvonalban a következő információk vannak:

- **summary:** Tartalmaz egy rövid szöveget az útvonalról, amely alkalmas az útvonal elnevezésére és megkülönböztetésére az alternatíváktól.
- **overview_polyline:** Egyetlen szöveget tartalmaz, amely az útvonal kódolt vonalláncának (polyline) ábrázolását tartalmazza. Ez a vonallánc a kapott útvonalak összefésült útja.
- **bounds:** Tartalmazza az overview_polyline nézetablak (viewport) mezőjének a méretét.

- **warnings:** Egy lista a figyelmeztetésekről, amelyeket magunknak kell megjelenítenünk és lekezelnünk.
- **waypoint_order:** Egy tömböt tartalmaz, amelyben az útpontok sorrendje van az útvonalon. Ezeket az útpontokat átrendezhetjük, ha a kérésnél beállítjuk az ehhez szükséges `optimize true` beállítást.
- **copyrights:** Jogi okokból küldött adat, tartalmazza az évszámot, a Google cég nevét, és lényege, hogy biztosítja a szerzői jogvédelmet a kapott válasz objektumon.
- **overview_path:** Egy nagyon sok elemből álló tömb, amiben koordináták vannak. Mindegyik ilyen koordinátának van egy `latitude` (szélességi kör) és egy `longitude` (hosszúsági kör) adattagja. Ezek a koordináták az útvonal mentén helyezkednek el a kiindulási ponttól a célállomásig különböző távolságokra. Amint látható a 23-as ábrán, Csíkszeredától Marosvásárhelyig 294 elem van ebben a tömbben.
- **legs:** Az útvonal mentén levő kisebb útvonalak. Legalább egy elem van benne akkor, ha az útvonalban csak egy kezdő és egy végpont van. Abban az esetben, ha az útvonalunkon van egy köztes pont, akkor ebben a tömbben két elem lesz: egy a kezdőponttól a köztes pontig, és egy a köztes ponttól a végpontig. Minden ilyen „lábban” a következő fontosabb információk vannak: `distance` (a két pont közti távolság mééterre pontosan), `duration` (utazás időtartama, másodpercre pontosan), `end_address` (a célpont címe: Ország, város, utca és szám), `end_location` (`latitude` és `longitude` koordinátái a célpontnak), `start_address` (a kezdőpont címe: Ország, város, utca és szám), `start_location` (`latitude` és `longitude` koordinátái a kezdő pontnak), `steps` (egy tömb, amibe a lépések vannak leírva, amit a GPS navigáció során követni kell, ahhoz hogy autóval megtegyük az utat, részletesen levannak írva a pontról-pontig tartó útmutatások, ezért sok navigációs alkalmazásban ez a tömb az egyik legfontosabb eleme a Directions válasz objektumának).

8.4. Google Maps segítségével beépített funkcionalitások

Ahogy az Alkalmazás Funkcionalitásai fejezetben is említettem, a Fuvar Létrehozása menüben hoztam ki a legtöbbet a Google Maps és Google Directions szolgáltatásokból. Itt olyan funkcionalitást építettem be az alkalmazásomba, ami hatással van az oldal más funkcióira is és felhasználóbarát módon segíti az útvonaltervezést, az útitárskeresést, valamint a fuvarkeresést. Be fogom mutatni a funkcionalitást részletesen és elmagyarázom, hogyan kiviteleztem a fontosabb lépéseket, ezután pedig megnézzük, hogyan hat ez a funkcionalitás az alkalmazás többi részére.

1. Helységek kiválasztása

2. Útvonal megtervezése

3. Részletek megadása

Kiindulási hely

Marosvásárhely

Érkezési hely

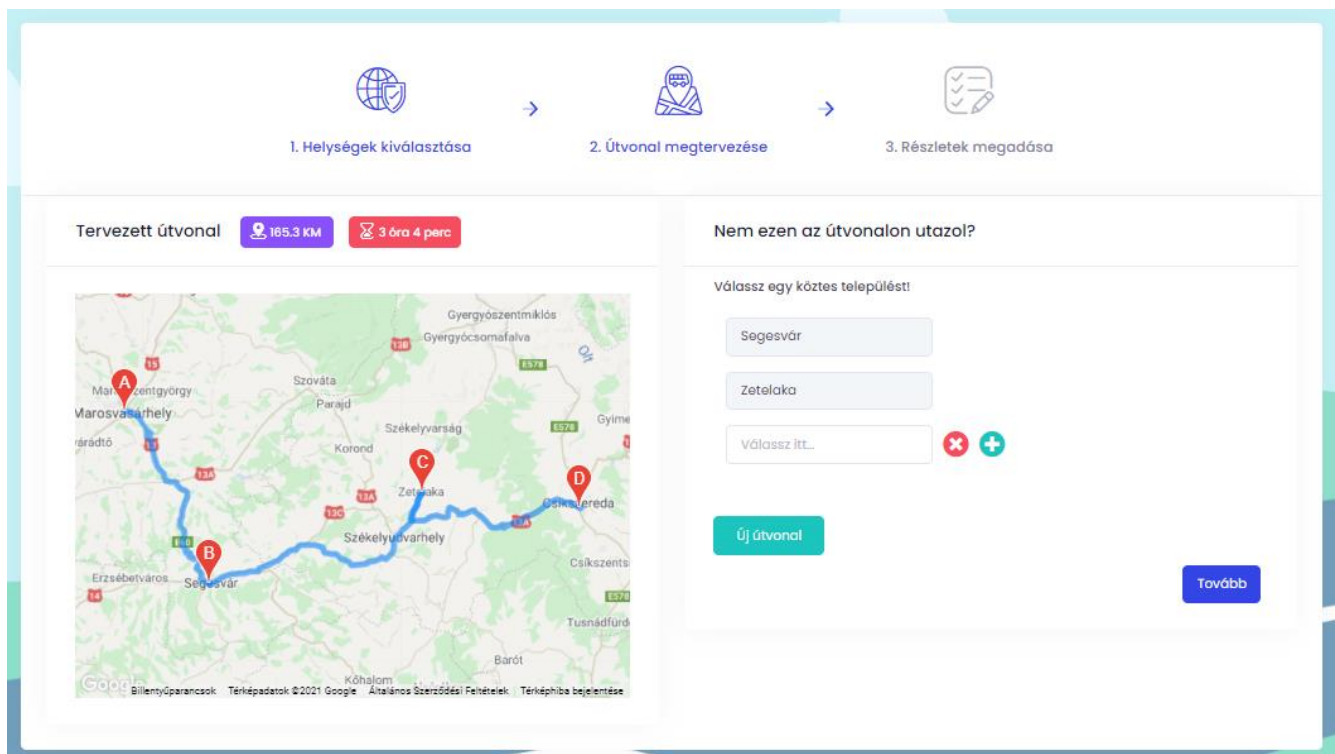
Csíkszereda

Tovább

24. ábra – Helységek kiválasztása

Egy fuvar létrehozásának első lépése a helységek (kiindulási hely és érkezési hely) kiválasztása. Az itt levő bemeneti mezők valójában keresők. A felhasználó, ahogy elkezd a helység nevét begépelni, úgy töltődnek be a helységnevek, amelyek közül, így választhat, hogy ráklikkel az adott helységre. A helységekhez rendelkezésemre állt egy adatbázis, amit a TheMarketive cég biztosított számomra nyári gyakorlat során. Ebben az adatbázisban a romániai és magyarországi községek, falvak és városok szerepelnek, a saját adatbázisomban a cities táblában tárolom. Összesen 15.134 helység kapott itt helyet, lényegében csaknem az összes helység a két országban. A helységekről a következő információk vannak eltárolva: `overpass_id`, `county_id` (megyéhez kapcsolható másodlagos kulcs), `type` (falu, város, nagy város), `latitude`, `longitude` (a koordinátái), `postal_code` (posta kód), `name_hu`, `name_ro`, `name_en`. A helységnevek le vannak mentve román, magyar és angol nyelven is. Van olyan helység, hogy le van fordítva mindhárom nyelvre, de egyes helységeknak van, hogy csak román, vagy csak magyar nevük van. Lényegében ezt a nevet jelenítem meg a kereső mező alatt, az teljesen mindegy, hogy a felhasználó milyen nyelven írja be a keresendő helyiséget, mivel ebben a keresőbe nem függ a keresett eredmény az oldal beállított nyelvétől. Hogyha mondjuk Bukarestre szeretne rákérdezni a kliens, akkor úgyis megtalálja, hogy “Bucharest”, vagy úgy is, hogy “Bucuresti”. Azért volt fontos ezt így megoldani, hogy biztosan megtalálja a felhasználó, a keresett helyiséget.

Miután a felhasználó kiválasztotta a helységeket, és ráklikkel a Tovább gombra, a háttérben elmentem az adatbázisból kiszedett `latitude` és `longitude` értékeit a kezdő és végállomásoknak. Ezután az alkalmazás tovább lép a fuvar létrehozásának következő lépésére, az útvonal megtervezésére.



25. ábra – Útvonal megtervezése

Az útvonal megtervezésénél, ahogyan a Direction Service-hez küldött kérésnél már elmagyaráztam, létrehozok egy-egy geokódolt Google koordinátát, a kiválasztott helységek latitude és longitude értékeiből, aztán ezekre a koordinátákra intézek egy kérést (20-as ábra), amit megjelenítek a térképen.

A 25-ös ábrán látható jobb oldali panel arra való, hogy a felhasználónak lehetősége legyen részletesen megtervezni az útvonalat, amin utazni fog. Ezt úgy teheti meg, hogy megad köztes településeket az útvonal mentén, amelyeket érinteni szeretne. Összesen 5 köztes települést adhat meg, úgy jelennek meg az újabb bemeneti mezők, hogy miután kiválasztott egy köztes települést, ráklikkel a bemeneti mező mellett levő + ikonra. Ha meggondolja magát, vagy rossz települést adott meg, lehetősége van az X ikonnal eltüntetni a megadott helységet. Miután beállította a köztes településeket az Új útvonal gombbal tudja frissíteni az útvonalat.

A kivitelezés során a Google Directions kérés waypoints beállítását használtam. Lényegében, ahogy fentebb is elmagyaráztam ez egy tömb, amiben megadom a köztes pontokat egy útvonal mentén, olyan sorrendben, ahogyan azt a felhasználó beállította. A waypoints tömbbe szintén geokódolt Google koordinátatípusok kell szerepeljenek, ezért itt is minden beviteli mező egy-egy kereső, melyek hasonlóan működnek, mint a kezdő és végállomásokat kiválasztó keresőmezők. Miután a felhasználó kiválasztott egy helységet, eltárolom annak latitude és longitude értékeit először a bemeneti mező id értékeként, majd később az Új útvonalra klikkelés után geokódolt koordináta típussá alakítom, és összefűzöm őket egy tömbbe. Végül újabb kérést intézek a

Directions Service-hez, immár átadva a waypoints tömböt is. A következő ábrán ennek a kódját lehet megtekinteni.

```
//Go through all the waypoint inputs and get the coordinates
for(var i=0; i<waypointCounter; ++i){
    if(document.getElementById(`middle${i}`).value != 'nothing'){
        splittedMiddle = document.getElementById(`middle${i}`).value.split(" ");
        stopp = new google.maps.LatLng(splittedMiddle[0], splittedMiddle[1]);
        waypts.push({
            location: stopp,
            stopover: true
        });
    }
}
requestDirections2(splittedStart[0], splittedStart[1], waypts, splittedEnd[0], splittedEnd[1]);

//Request direction, with waypoints on the route
function requestDirections2(start1, start2, waypts, end1, end2) {
    startLatLng = new google.maps.LatLng(start1, start2);
    endLatLng = new google.maps.LatLng(end1, end2);

    directionsService.route({
        origin: startLatLng,
        destination: endLatLng,
        waypoints: waypts,
        travelMode: google.maps.DirectionsTravelMode.DRIVING
    }, function(result) {
        renderDirections(result);
    });
}
```

26. ábra – Köztes állomásokkal intézett kérés

A `renderDirections` függvényben megjelenítem az új útvonalat a térképen, valamint újraszámolom az útvonal hosszát és idejét. Megjegyzendő, hogy ezeket az értékeket nem csak egyszerűen kiveszem a `result` objektumból, hanem a `result.routes.legs` tömböt kell végigjárnom és azoknak összeadni az útvonalaik hosszát és idejét, mivel ugyebár ha vannak az útvonalon köztes pontok, akkor az útvonalunk több kisebb útvonalból tevődik össze, és ezért csak úgy tudom megkapni a fő útvonalam idejét és hosszát ha minden alútvonal idejét és hosszát összeadom. Ezután az összeadott értékeket átalakítom olyan formában, hogy a felhasználó számára érthető értékek legyenek, majd végül megjelenítem a térképem felett.

Ezzel el is készült az alkalmazásom interaktív útvonaltervező funkcionalitása.

Fontos megnézni, hogy konkrétan milyen adatokat mentek el egy fuvarról a létrehozásakor, mivel ezek hatással vannak az oldal más funkcionalitására is. Minden új fuvarkor elmentem az adatbázisba a következő információkat:

- A konkrét fuvar.
- A fuvar köztes állomásait (ha vannak): fuvar azonosítója, latitude, longitude és hogy hányadik állomás.
- A fuvar létrehozó felhasználót sofőrként: fuvar azonosítója, felhasználó azonosítója, státusz (utas vagy sofőr).
- Értesítést, abban az esetben, ha egy felhasználó bekapcsolta a “fuvar figyelőt”, és ez a fuvar megfelel az általa beállított akadtoknak (erről később részletesebben is írok)
- A fuvar mentén levő városok: fuvar azonosítója, város azonosítója, hányadik a sorrendben.

A fuvar mentén levő városok eltárolásához szintén a saját adatbázisomhoz fordultam segítségül, összekombinálva a Google Directions-tól kapott válasszal. Ezt úgy oldottam meg, hogy miután a felhasználó befejezte az útvonal tervezését, a háttérben a program végig járja ezt az útvonalat, majd minden lépésben ellenőrzi, hogy az adatbázisban levő városok közül bele esik-e valamelyik az útvonal körülbelül 5 kilométeres sugarába. Abban az esetben, ha beleesik, akkor a város azonosítóját a megfelelő sorrendben eltárolom egy tömbben, majd ha végig jártam az útvonalat, ezt a tömböt mentem. A fuvar létrehozásakor, a tömböt a többi információval együtt elküldöm a Back-Endre, ahol feltöltöm az adatbázisba, és így megkapom a fuvar útvonala mentén levő városokat.

Az útvonal végig járása és a közelében levő városok koordinátájának vizsgálata a 27-es ábrán látható.

```
//Create latlng object type from all path over the route
for (var j=0; j<result.routes[0].overview_path.length; ++j){
    var latlng = { lat: result.routes[0].overview_path[j].lat().toFixed(2), lng: result.routes[0].overview_path[j].lng().toFixed(2) }
    path.push(latlng);
}

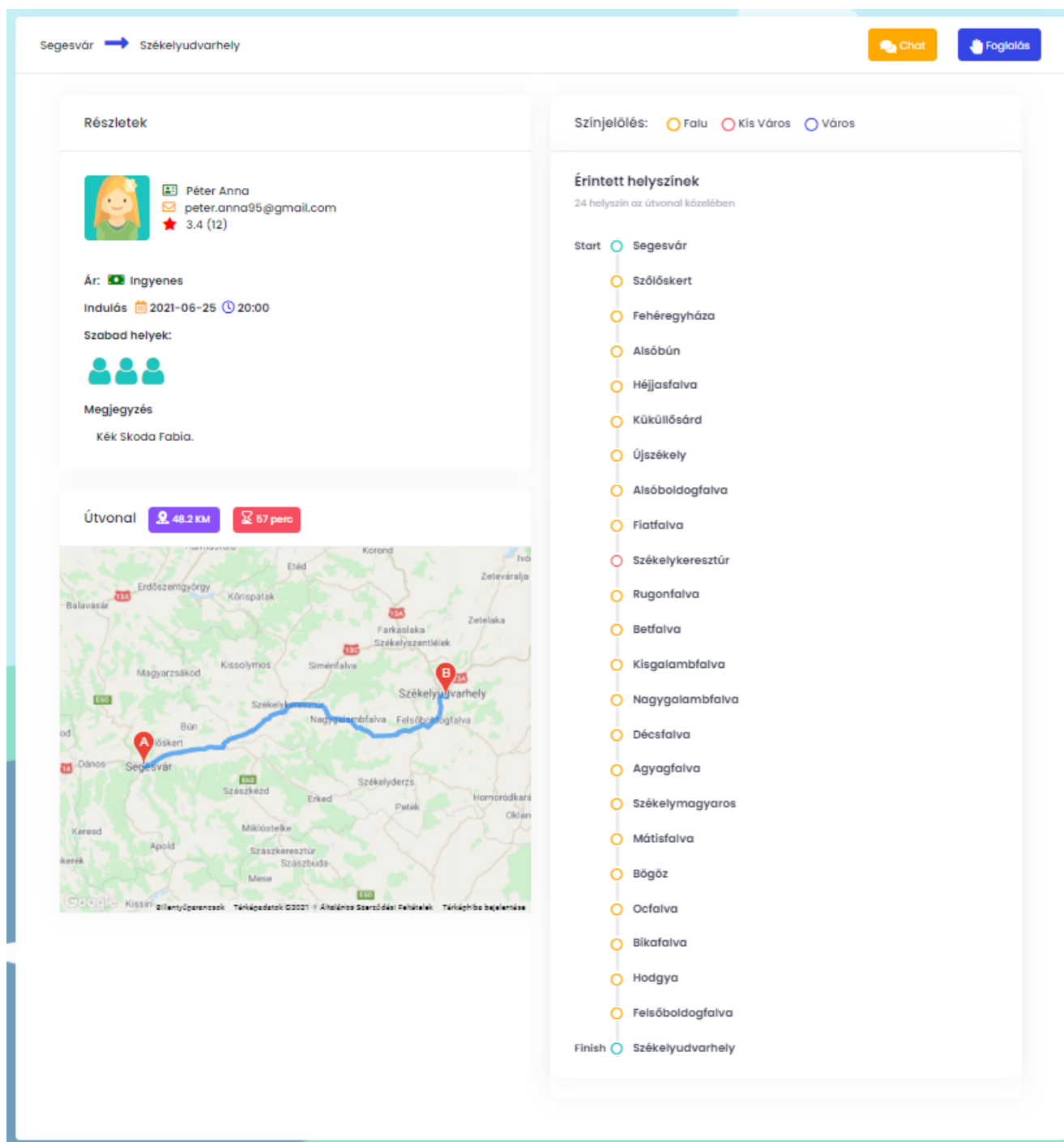
//Get all cities near a route, store them in the right sequence
for(var i=0; i<path.length; ++i){
    for(var j=0; j<citiesDB.length; ++j){
        if( (Math.abs(path[i].lat - citiesDB[j].latitude) < 0.015) && (Math.abs(path[i].lng - citiesDB[j].longitude) < 0.015) ){
            if(!citiesBetween.includes(citiesDB[j].id)){
                citiesBetween.push(citiesDB[j].id)
            }
        }
    }
}
}
```

27. ábra – Útvonal mentén levő városok mentése

Az útvonal bejárását a válaszból kapott routes tömb overview_path tömbjének bejárásával tudom elvégezni. Ebben az overview_path nevű tömbben koordináták vannak az útvonal kezdő pontjától a végéig, ezeket a koordinátákat átalakítom egy latlng objektummá, és egy path tömbbe rendezem őket. A path tömböt bejárva mindegyik lépésben vizsgálom, hogy a koordináták adott közelébe esik-e valamelyik adatbázisban tárolt város. A városokat itt már előre lekértem egy tömbbe, hogy ne kelljen minden lépésben lekérdezést intézni az adatbázisból. Látható, hogy a hosszúsági és szélességi körökből kivonjuk az adott város hosszúsági és szélességi körök mentén levő fekvésüket. Ezután ennek az értéknek az abszolút értéke kevesebb kell legyen, mint 0.015 ahhoz, hogy a várost az útvonal mentén fekvőnek tekinthessük. Ezt az értéket növelve vagy csökkentve változtathatjuk a méretét annak a sugárnak, ami az útvonal mentén haladva összegyűjti a városokat. A 0.015-ös érték egy körülbelül 5 kilométeres sugarat takar. Azért választottam ekkorát, mert nem akartam, hogy egy túl távoli város bekerüljön, de azt sem akartam, hogy csak a konkrét útmenti városok kerülhessenek be. Az alkalmazás ugyanis az útvonal mentén levő városok segítségével tud útvonalat ajánlani és értesítést küldeni ezzel kapcsolatba, én pedig azt szerettem volna elérni, hogy akkor is ajánlja be a felhasználónak az adott fuvart, ha az útvonal nem érinti konkrétan azt a várost ahová szeretne utazni, viszont ott van valahol az 5 kilométeres körzetében. Elvégre az alkalmazás lényege az útitárskeresés és a közösségépítés. Megeshet, hogy a fuvart kereső megbeszéli a sofőrrel, hogy tegyen egy pár kilométeres kitérőt a kedvéért, vagy ha nem is tesz kitérőt, akkor is megeshet, hogy a megtenni kívánt útvonaluk nagy részben egyezik, és ezért érdekelheti a felhasználót. Véleményem szerint fontos, hogy előre gondoljunk az ilyen valós esetekben felmerülő eshetőségekre szoftverfejlesztés során.

Említettem már, hogy a fuvar létrehozása hatással van az oldal más funkcionalitására, az egyik ilyen a fuvar megnyitása. Szerettem volna úgy elkészíteni a fuvar megtekintését (részletes adatok egy fuvarról), hogy a felhasználó láthassa Google Térképen az útvonalat, emellett pedig megjelenjenek a fuvar mentén levő városok. Így, hogy a városok le vannak mentve az adatbázisba, nincs más dolgom, mint a fuvar megnyitásakor lekérnem a fuvarhoz kapcsolódó városok azonosítóját, és felhasználva a lekérdezett sequence adattagot, a megfelelő sorrendbe rendezem őket. Az azonosítókat használva ezután lekérem a városok nevét (attól függően milyen nyelvre van állítva az oldal) és a típusát, majd megjelenítem ezeket.

Ahhoz, hogy a térkép is megjelenjen, lekérem a fuvarhoz kapcsolódó köztes állomásokat, amiket ugyebár szintén elmentettem. Ezekből az állomásokból majd, ahogyan az útvonal tervezésnél is, geokódolt koordináta típusokat készítek és kérést intézve a Google Directions-höz, megjelenítem lényegébe ugyanazt az útvonalat, amit a felhasználó a fuvar létrehozásakor megtervezett. A 28-as ábrán látható egy fuvar részletes megtekintése.



28. ábra – Fuvar adatai

Megjegyzendő, hogy ezen az oldalon is megjelenítem a fuvar hosszát és idejét, hasonló módszerrel, mint a fuvar létrehozásánál. Megjelennek még a felhasználó adatok, valamint minden a fuvar létrehozásánál megadott adat. Látható, hogy ez a funkcionalitás is a Google API-nak köszönhetően lehet ilyen részletes. Véleményem szerint ez a részletesség növeli az oldal szerethetőségét és a felhasználói élményt.

Az utolsó funkcionalitás, amelynél használtam a Google API-kat, az a Fuvar keresése menü, és azon belül a Fuvar figyelő bekapcsolása után küldendő értesítések. Fuvar keresésénél, ahogyan a funkció leírásánál, már részletesen leírtam, a felhasználó megadja a kezdő és végpontokat, valamint a dátumot. Ezután a program elkezd vizsgálni az elmentett fuvarok mentén levő

városokat és ezeket hasonlítja össze a felhasználó által keresett út mentén levő városokkal, majd ha egyezik az út, vagy a felhasználó által megtenni kívánt út, a része a fuvar útvonalának, akkor a fuvar kilistázódik a keresési eredmények közé. Ennek a funkciónak a megírásánál nem használtam a Google Maps függvényeit és kérést sem intéztem egyik Google API-hoz sem, viszont, ha a fuvar létrehozásánál nem tároltam volna el a városokat (amit ugyebár a Google API-al tudtam megtenni), akkor ez a funkció nem lenne működőképes.

A fuvar ajánlása egy alfunkciója a fuvar keresésnek, lényegébe bekapcsolhatja a felhasználó a Fuvar figyelőt, ekkor el lesz mentve a keresett útvonal. A fuvar létrehozásánál, figyeljük, a bekapcsolt Fuvar figyelőket és hasonlítjuk az éppen létrehozott fuvarhoz. Abban az esetben, ha nagy részben vagy teljesen megegyezik a két útvonal, értesítést küldünk a felhasználónak, amelyik bekapcsolta az adott útvonalra a Fuvar figyelőt, hogy „létrejött egy fuvar, ami úgy gondoljuk érdekelhet!”. A 29-es ábrán ennek a vizsgálatnak a kódja látható, a keretrendszer egyik vezérlő fájljából.

```
//Go through all required rides
foreach ($required_rides as $required_ride) {
    //Check if the new ride go through the start and finish city of the required ride
    if(in_array($required_ride->start_city_id, $cities) && in_array($required_ride->finish_city_id, $cities)){
        //Check if the finish city is after the start city in the new ride
        if(array_search($required_ride->start_city_id, $cities) < array_search($required_ride->finish_city_id, $cities)){
            //Check if the date is okay
            if((Ride::find($ride->id)->start_time) >= $required_ride->start_time){
                //Check if the user turned on Ride watcher
                if($required_ride->notified == 1){
                    //Send the notification
                    $notification = Notification::create([
                        'user_from' => $ride->user_id,
                        'user_to' => $required_ride->user_id,
                        'ride_id' => $ride->id,
                        'type' => 4,
                    ]);
                }
            }
        }
    }
}
```

29. ábra – Értesítés küldése az új fuvarról a bekapcsolt Fuvar figyelőre

Ezzel a funkcióval a végére is értünk a Google Maps API segítségével létrehozott funkciók bemutatásának, és vele együtt a kutatómunka leírásának is.

9. Jövőbeli terveim

A jövőben szeretném az alkalmazást tovább fejleszteni és új funkcionalitásokat beépíteni. Szeretném tanulmányozni a más, hasonló alkalmazásokat, hogy azok milyen funkciókat használnak a felhasználói élmény növelése érdekében, és ezeket valamilyen módon, hasonlóan beépíteni a saját weboldalamba. Szeretnék a jövőben dolgozni még a reszponzivitáson is, és mindenképp meg akarom oldani, hogy az alkalmazás használható legyen mobil telefonon, letölthető alkalmazásként. Mostanában ugyanis egyre nagyobb az elvárás, hogy egy weboldal megnyitható legyen alkalmazásként, azaz a felhasználóknak ne kelljen a mobiljukról linkről, vagy böngészőből keresve megnyitni az alkalmazást. Ennek a megoldására különböző segéd szoftverek léteznek, ahol a szoftver egy alkalmazásnak „álcázza” az elkészített weboldalt, amely a felhasználó szemszögéből úgy fog működni, mintha alkalmazás lenne. Telepítéskor, tulajdonképpen egy linken keresztül, az erre a célra előre beállított böngészőt használva, nyitja meg az alkalmazásnak álcázott weboldalt.

Fontosnak tartom azt is, hogy komolyabb piackutatást végezzek, és jobban utána nézzek, hogy hogyan is működik egy weboldal marketingje.

Szeretném, ha az oldalon kialakulna egy jó közösség, nagy felhasználó számmal, elvégre az oldal működésének is a nagy felhasználó bázis a szíve lelke. Az összes tovább fejlesztés megoldása után szeretném az oldalt online és offline is módban is reklámozni.

10. Összegzés

A dolgozat és a weboldal elkészítése alatt nagyon sokat tanultam a webfejlesztésről, API-okról, a Google és Facebook különböző fejlesztőknek nyújtott szolgáltatásairól, verzió követő rendszerekről, valamint a reszponzív dizájn készítésről. Úgy gondolom sikerült mindent bemutatnom a dolgozatba, amit szerettem volna és nagyon sok tapasztalatot szereztem hivatalos dokumentumok összeállításáról is. A dolgozat végeztével született egy alkalmazás, ami a szívemhez nőtt és továbbra is szeretnék dolgozni rajta. Nem utolsó sorban pedig a karrier kezdéshez is hozzá segít, ha van egy ilyen komplex referencia projekt az önéletrajzomba.

11. Ábrajegyzék

- 1. Ábra: Use case diagram - 8. oldal
- 2. Ábra: A Rendszer architektúrája – 9. oldal
- 3. Ábra: Adatkapcsolati táblák – 12. oldal
- 4. Ábra: Migration fájl – 13. oldal
- 5. Ábra Fordító fájl – 14. oldal
- 6. Ábra: Szekvenciális diagram – 15. oldal
- 7. Ábra: Validáció – 16. oldal
- 8. Ábra: Rendezés – 18. oldal
- 9. Ábra: Fuvar részletei – 19. oldal
- 10. Ábra Útitárs értékelése – 21. oldal
- 11. Ábra: Ajax kérés – 22. oldal
- 12. Ábra: Chat – 23. oldal
- 13. Ábra: Firestore figyelő – 24. oldal
- 14. Ábra: Értesítés – 26. oldal
- 15. Ábra: E-mail a jelszó visszaállításához – 27. oldal
- 16. Ábra: Profil adatok – 28. oldal
- 17. Ábra: Maps API-hoz szükséges fájl betöltése – 30. oldal
- 18. Ábra: Térkép létrehozása – 30. oldal
- 19. Ábra: Nagyítási szintek szemléltetése – 31. oldal
- 20. Ábra: Kérés küldése a Google Directions-hoz – 33. oldal
- 21. Ábra: Directions kérés beállítási mezői – 33. oldal
- 22. Ábra: Útvonal megjelenítése – 35. oldal
- 23. Ábra: Directions válaszobjektum – 36. oldal
- 24. Ábra: Helységek kiválasztása – 39. oldal
- 25. Ábra: Útvonal megtervezése – 40. oldal
- 26. Ábra: Köztes állomásokkal intézett kérés – 41. oldal
- 27. Ábra: Útvonal mentén levő városok mentése – 42. oldal
- 28. Ábra: Fuvar adatai – 44. oldal
- 29. Ábra: Értesítés küldése az új fuvarról a bekapcsolt Fuvar figyelőre

12. Bibliográfia

Könyvek:

- [1] Oluwafemi_Alofe: “Beginning PHP Laravel: Step to step approach to building an Inventory App”. Ebook, 2020.
- [2] Linus Torvalds: “Laravel 5.8 Learn A to Z | Best PHP Framework”. Ebook, 2020.
- [3] Bogdan Brinzarea, Cristian Darie, Mihai Bucica: “AJAX and PHP: Building Responsive Web Applications”. Ebook, 2006.
- [4] Ethan Marcotte: “Responsive Web Design”. A Book Apart, 2011.
- [5] Ficsor Lajos, Krizsán Zoltán, Mileff Péter "Szoftverfejlesztés". Miskolci Egyetem, Általános Informatika Tanszék, 2011.

Weboldalak:

- [1] Google Firebase dokumentáció: <https://firebase.google.com>
- [2] Google Maps dokumentáció: <https://developers.google.com>
- [3] Wikipédia oldal a Google térkép elméleti háttéréről:
https://hu.wikipedia.org/wiki/Google_T%C3%A9rk%C3%A9p