



УНИВЕРЗИТЕТ У НИШУ  
ЕЛЕКТРОНСКИ ФАКУЛТЕТ



## УПОТРЕБА ELOQUENT ОБЈЕКТНО РЕЛАЦИОНОГ МАПЕРА ЗА ПРИСТУП ПОДАЦИМА

Дипломски рад

Студијски програм: Електротехника и рачунарство

Модул: Рачунарство и информатика

Студент:

Никола Митић, бр. инд. 16228

Ментор:

Проф. Др Александар Станимировић

Ниш, октобар 2024. год.



Универзитет у Нишу  
Електронски факултет

## УПОТРЕБА ELOQUENT ОБЈЕКТНО РЕЛАЦИОНОГ МАПЕРА ЗА ПРИСТУП ПОДАЦИМА

## USE OF ELOQUENT OBJECT RELATIONAL MAPPER FOR DATA ACCESS

Дипломски рад

Студијски програм: Електротехника и рачунарство

Модул: Рачунарство и информатика

Студент: Никола Митић, бр. инд. 16228

Ментор: Проф. Др Александар Станимировић

Задатак: *Изучавање приступа подацима релационе базе података, њихово управљање Eloquent ORM-ом и коначно имплементација овакве базе кроз систем веб апликације израђене у технологијама Laravel и Angular; додатно повезаних Reverb WebSocket-ом. Радом објаснити технологију приступа, обраде и употребе подацима употребом PHP—а, Laravel—а и Eloquent ORM-а. Затим, направити апликацију тако да омогући модерно управљање такси удружењем, односно омогућити приказ пословања из више угла. Омогућити приказ из угла муштерије, возача и менаџера. Свима омогућити пријаву на систем и преглед профила, а у зависности од типа корисника наручивање возње, прихватање возње и праћене статистике.*

Комисија за оцену и одбрану:

Датум пријаве рада: _____	1. _____
Датум предаје рада: _____	2. _____
Датум одбране рада: _____	3. _____

# УПОТРЕБА ELOQUENT ОБЈЕТКНО РЕЛАЦИОНОГ МАПЕРА ЗА ПРИСТУП ПОДАЦИМА

## САЖЕТАК

Савремена комуникација међу корисницима захтева комплексне системе који у реалном времену преносе информације између правих чворова у мрежи. Сам сервис такси службе представља један вид напредне комуникације међу корисницима. Оваква комуникација мора бити обogaћена различитим акцијама које се могу директно или узрочно извршавати. Она корисницима доноси олакшање у свакодневним активностима транспорта, убрзава пословање и пружа економичност не захтевајући комплексне уређаје.

У раду се прво представљају технологије које се користе у позадини оваквог система. Објашњена је употреба *frontend* и *backend* система, њихова функционалност заснована на размени података, обради тих података и адекватном начину чувању истих у релационој бази података. Представљен је принцип превођења податак базе података у објектно релационе моделе. Извођењем овог поступка добијамо структуру података којом знамо како да манипулишемо и вршимо даљу припрему за употребу. Затим се пружа детаљни опис система који приказује имплементацију жељеног решења и процес креирања истог. Финално долази се и до приказа рада саме апликације, односно демонстрације жељеног решења на примеру у реалности.

Главни циљ рада је имплементација модерних технологија које омогућују лак приступ и обраду комплексних података. Коришћењем *Laravel*-а на *backend*-у који омогућава једноставну обраду захтева и повезивање са *MySQL* базом података добијамо систем који лако и брзо чита и обрађује све податке коришћењем *Eloquent ORM*-а. Имплементацијом *frontend*-а у *Angular*-у добијамо систем који се покреће у сваком веб претраживачу са прилагодљивим приказом. Овај систем остварује директну интеракцију *frontend*-а и *backend*-а лако вршећи модификацију података. Пренос података додатно се може обогатити имплементацијом *WebSocket*-а који омогућују пренос података у реалном времену. На основу пренесених података, коришћењем система догађаја, и имплементацијом *Google Maps API*-а омогућује се приказ кретања корисника на мапи.

Употребом *Eloquent* релационог мапера, подаци су повезани у комплексну структуру, обрађени у *Laravel*-у и приказани прилагодљивим дизајном коришћењем *Angular*-а. Постигнута је аутоматизација управљања системом и пружена једноставна употреба кроз веб претраживач.

**Кључне речи:** комуникација, аутоматизација, навигација, приступ подацима, мапирање података, пренос података у реалном времену, повезивање.

# USE OF ELOQUENT OBJECT RELATIONAL MAPPER FOR DATA ACCESS

## ABSTRACT

Modern communication between users requires complex systems that transfer information in real time between the right nodes in the network. The taxi service itself represents a form of advanced communication between users. Such communication must be enriched by various actions that can be performed directly or causally. It brings relief to users in their daily transport activities, speeds up business and ensures economy without the need for complex devices.

This work first presents the technologies that are used in the background of such a system. The use of frontend and backend systems, their functionality based on data exchange, data processing and adequate storage in a relational database are explained. The principle of translating database data into a relational model is presented. By performing this procedure, we get a well structured data with which we know how to manipulate and do further adjustments. Then a detailed description of the system is given, showing the implementation of the desired solution and the process of its creation. Finally, there is a presentation of the application itself, i.e. a demonstration of the desired solution on an example.

The main aim of this work is the implementation of modern technologies that enable easy access and process of complex data. By using Laravel on the backend, which enables simple processing of requests and connection to the MySQL database, we get a system that easily and quickly processes all data using Eloquent ORM. By implementing the frontend in Angular, we get a system that works in every web browser with a responsive design. This system achieves direct frontend and backend interaction by easily exchanging data. Data transfer can be further enriched by implementing WebSocket, which enable real-time data transfer. Based on the transmitted data, using the event system and implementing the Google Maps API, it is possible to display the user's movement on the map.

With use of Eloquent relational mapper, data is linked into a complex structure, processed in Laravel and displayed in a responsive design using Angular. Automation of system management was achieved and simple use through a web browser was provided.

**Keywords:** communication, automation, navigation, data access, data mapping, real-time data transfer, connectivity.

## САДРЖАЈ

1. УВОД .....	7
2. <i>PHP</i> И РАД СА БАЗАМА ПОДАТАКА .....	8
2.2. <i>PHP</i> .....	8
2.3. Релационе базе података .....	9
3. LARAVEL .....	11
3.2. Опште .....	11
3.3. Рутирање .....	11
3.4. Валидација .....	12
3.5. MVC архитектура .....	13
3.6. Redis .....	14
3.7. Homestead .....	15
4. ELOQUENT ORM .....	16
4.2. Опште .....	16
4.3. Мапирање модела и његова употреба .....	17
5. ИМПЛЕМЕНТАЦИЈА АПЛИКАЦИЈЕ .....	19
5.2. Архитектура апликације .....	19
5.3. MySQL .....	20
5.4. Angular .....	21
5.5. Web Socket .....	23
5.6. WebStorm .....	24
5.7. PhpStorm .....	24
5.8. Опис имплементације базе података .....	25
5.9. Имплементација серверског дела .....	28
5.9.1. Eloquent ORM .....	32
5.10. Имплементација клијентског дела .....	35
5.10.1. Приказ мапе .....	38
6. РАД АПЛИКАЦИЈЕ .....	41
6.2. Опис апликације .....	41
6.3. Профил корисника апликације .....	42
6.4. Случајеви коришћења .....	42
7. ЗАКЉУЧАК .....	54
8. ЛИТЕРАТУРА .....	55

## 1. УВОД

Ходајући у корак са временом и развојем технологије, човек сваког тренутка увећава свој дигитални отисак. Константно се долази до креирања све веће количине података, са често веома комплексном повезаношћу. Како се човек окреће технологији ради олакшања живота свакодневице, потребно је то испратити и са програмерске стране и развити потребне *web* апликације.

Ново креиране *web* апликације у многоне прате и олакшавају функционисање чиме се директно утиче на комплексност имплементације. Да би се ово постигло, апликације и подаци које оне чувају морају имати добру структуру. За израду овакве апликације данас се често окрећемо добро познатим програмским језицима. Један од можда најпознатијих и најраспрострањенијих је свакако *PHP* програмски језик. Он пружа једноставну имплементацију и добру документацију. Олакшање и унапређење имплементације оваквих апликација долази и са развојем многих *freamwork*<sup>1</sup>-ова, а можда највише од *Laravel*-а.

Комуникација апликације са базом података један је од главних осврта технологија *Laravel*-а. Повезивање апликације са релационим базама податка и разумевање садржаја истих постиже се објектно релационим маперима. *Laravel* са *PHP*-ом нам пружа *Eloquent ORM*. Његовим коришћењем се омогућује једноставан приступ подацима, специјално обликованих и здружених у смислене објекте. Креирањем објеката података од уписаних података у табелу релационе базе података постиже се спремност ових података за даљу обраду.

Овај рад бави се складиштењем, анализом приступа и начином обраде података. Како се овим подацима манипулише можемо видети употребом програмских језика и специјално дефинисаних објектно релационих мапера. Такође, овај рад пружа осврт и на имплементацију идејног решења проблема комуникације и транспорта у виду *web* апликације.

Сви подаци и примери коришћени у оквиру веб апликације су фиктивни и као такви користе се искључиво у сврси презентовања рада.

Овај рад подељен је у седам поглавља. Прво, сама увертира овог дипломског рада већ је приведена крају. Затим, следи објашњење *PHP* језика и рада са релационим базама података које се користе као срж овог система. У трећем поглављу детаљно је описана *backend* технологија *Laravel*-а. У четвртном поглављу описана је *Eloquent* релациони мапер којим се обрађује и управља подацима. У петом поглављу објашњене су додатно употребљене технологије у изради апликације дипломског рада, њихова имплементација и детаљна структура ове веб апликације са пропратним примерима. У шестом поглављу налази се приказ употребе веб апликације. У седмом поглављу рад добија свију целину давањем закључног мишљења. А потом, на крају, долази се до литературе попуњене свим референцама којима сам приступао у току израде овог рада са датумима приступа истим.

---

<sup>1</sup> Freamwork – радно окружење садржано од унапред припремљених основних функционалности упакованих у библиотеке

## 2. PHP И РАД СА БАЗАМА ПОДАТАКА

### 2.2. PHP

*PHP* је почео као мали *open source*<sup>2</sup> пројекат који је временом порастао до светски најпознатијег програмског језика за израду веб апликација. Иницијална идеја именовања овог програмског језика била је скраћеница за *Personal Home Page*, међутим, развитком и унапређењем могућности овог програмског језика долази се и до усложњавања самог назива те данас *PHP* представља рекурзивну скраћеницу за *PHP: Hypertext Preprocessor*. Развој програмског језика испраћен је дефинисањем 8 главних верзија, са сваком верзијом језик постаје све комплекснији и моћнији. Развој *PHP*-а додатно подстиче *open source* карактеристика.

Овај програмски језик је скриптни програмски језик, односно сва логика написаног кода налази се у текстуалним фајловима који покретањем извршавају ред по ред скрипте. У зависности од потреба употребе, *PHP* се може користити за командно извршавање наредби или се може интерпретирати и извршавати на серверу. За потребе израде веб апликација, користи се други приступ, односно серверско извршавање кода. Сервери, односно компјутери на мрежи који обрађују наше захтеве, а на којима је смештена веб апликација, могу бити било које софтверске платформе јер *PHP* подржава извршење на свим најпознатијим оперативним системима. [1]

Временом, како долази до развоја програмског језика, долази и до развоја захтева за израду веб сајтова. Како писање оваквих кодова изискује константу употребу истих основних функционалности, долази се до развоја и употребе разних *framework*-а. Они пружају могућност употребе већ написаних функционалности смештених у бројне библиотеке. На овај начин олакшава се развој апликација, брзина израде и лакоћа одржавања. Нека од најпознатијих развојних окружења су:

- *Laravel* – најпознатије окружење за развој *PHP* апликација са највећом базом корисника. Пружа одличну документацију у виду текстуалних објашњења и примера али и добро документованих видео материјала. Користи *MVC*<sup>3</sup> архитектуру;
- *Symfony* – најстарије окружење које опстаје и развија се са сваком *PHP* верзијом. Користи се за апликације израђене за потребе великих организација попут корпорација. Користи *MVC* архитектуру. Логика имплементирања *PHP* функционалности дефинисана је *brick-by-brick* системом, овај систем представља поделу кода у *bundle*-е, односно пакете. Самим тим, у зависности од потребе система, користе се само неопходни пакети, а то даље узрокује израду великих апликације које заузимају мање меморије од других сличних апликација развијених у осталим окружењима; [2]

---

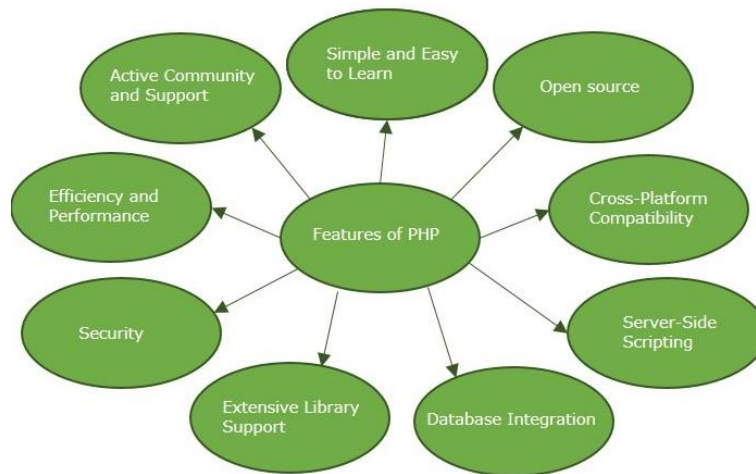
<sup>2</sup> *Open source* – ознака за код који је доступан свима чиме се постиже приступ прегледу начина имплементације и могућност унапређења од стране свих који то желе

<sup>3</sup> *MVC* – Model View Controller



- *CodeIgniter* – најмање развојно окружење *PHP* програмског језика. Генерално се користи за мање и средње апликације, али у случају комплекснијих апликација и потребе за додатним комплекснијим библиотекама, оне се могу динамично додати и проширити радно окружење. Користи *MVC* архитектуру. Веома лако се повезује са базом података; [3]

Најчешће се користи за израду веб апликација из разлога лаке интеграције са *HTML* кодом и добре подршке за лако повезивање са базама података. *PHP* се својом скриптном ознаком `<?php ... ?>` лако додаје у било коју *HTML* конструкцију, додајући додатне могућности обраде и приказа података. Употребом *framework*-а резултати обраде *PHP* кода осим генерисања *HTML* страница, могу бити и генерисање текстуалних и *JSON*<sup>4</sup> форматираних података, генерисање *PDF* докумената, извршавање акција попут слања *e-mail*-ова, и друго. [4]

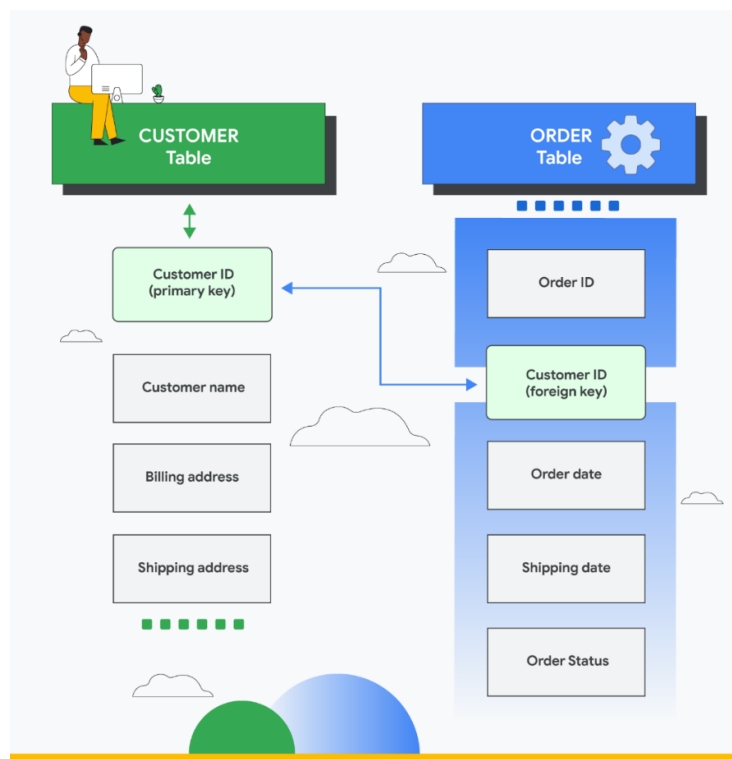


Слика 1 – *PHP* функционалности [5]

### 2.3. Релационе базе података

Релационе базе података представљају начин организовања података у табеле. Свака табела представља засебну групу ентитета из реалног света, редови у табели представљају опис једног ентитета, док се колонама дефинишу особине којима се описују ти ентитети. Како многе ентитете због своје комплексности није једноставно објаснити једном изјавом и како многа објашњења укључују и додатне информације о другим ентитетима, тако и њихово представљање у бази података захтева креирање додатних описних или повезаних табела. Уникатно идентификовање сваког реда података постиже се јединственим идентификационим бројем у тој табели који се назива примарни кључ. Узимањем овог јединственог идентификационог броја и смештањем у другу табелу, у ред са којим се доводи у везу, сада под именом страни кључ, ствара се један од више типова различитих веза у релационој бази података. Овакав систем омогућује детаљно и структурирано чување комплексних података из реалног света.

<sup>4</sup> JSON – JavaScript Object Notation



Слика 2 – Пример релационе базе података [6]

Овакав пример представља саму срж идеје која стоји иза креирања релационих база података, међутим релационе базе као такве се користе у много комплекснијим системима са доста више атрибута и повезаности међу табелама. Пружа се могућност вишеструког начина повезивања и прибављања груписаних и филтрираних података по жељи корисника. За рад са оваквим подацима користе се управљачки системи релационих база података (*RDBMS*<sup>5</sup>) као што су *MySQL*, *PostgreSQL*, *MariaDB*, *Microsoft SQL Server* и *Oracle Database*.

Модел релационе базе података настао је 1970их година са циљем замене хијерархијске структуре организације података. Како би се избегла редундантност података, грешка приликом вишеструких уноса и комплексност прибављања истих креиран је нови систем. Овакав модел не захтева вишеструки унос истих података већ креира једну табелу која се може повезати са више других, а не само са хијерархијском табелом изнад ње. Овим принципом рада избегнута је потреба за константном реорганизацијом табела. [6]

Овако креиране базе података и њихови подаци даље се мапирају како би се употребили у апликацијама направљеним објектно оријентисаним језицима. Објектно релационо мапирање преводи табеле у објекте, а атрибуте табела у атрибуте објекта. На овај начин генерише се објектни преглед података, штеди се време и олакшава рад креирањем предефинисаних упита ка бази и повећава сигурност спречавањем *SQL injection* напада<sup>6</sup> на базу података. [7]

<sup>5</sup> RDBMS – relational database management system

<sup>6</sup> SQL injection напад – напад на базу података који се извршава слањем упита ка бази који има за циљ недозвољену измену или прибављање података

## 3. LARAVEL

### 3.2. Опште

*Laravel* је *framework* који користи *PHP* програмски језик. Направљен је са циљем олакшања развоја веб апликација, без потребе да се све увек развија од самог почетка. Омогућава програмерима да започну креирање својих апликација са мноштвом већ имплементираних ствари и да се фокусирају на имплементацију нових проблема који су специфично потребни за њихову апликацију. Неке од ствари које ово развојно окружење пружа су аутентификација, *middleware*<sup>7</sup>, рутирање, рад са базом података, итд. [8]

Такође, једна од великих предности употребе *Laravel*-а је и то што он долази са *Artisan* командним интерфејсом. Овај *CLI*<sup>8</sup> доноси моћан сет команди које олакшавају процес развоја апликација, аутоматизују разне акције и побољшавају продуктивност програмера. Најпознатије и најкоришћеније команде су *make* и *migrate*. Команда *make* се углавном користи за креирање скелета модела, контролера, миграција и других компоненти, чиме се убрзава процес развоја апликације и избегава процес писања истог основног кода сваке компоненте. Додатно, ова команда пружа могућност развоја самог *Artisan*-а креирањем нових, ручно направљених, команди. Овакве команде представљају специјално развијену логику потребну за конкретне системе и оне се могу покретати на исти начин као и све остале команде. Команда *migrate* се користи за олакшани рад са базом података омогућујући извршење, поништавање и проверу миграционих докумената који служе за дефинисање изгледа базе података. Поред употребе ових команди, *Artisan* доноси додатне могућности управљања базом података, покретања тестова, управљања кешом и *HTML* ресурсима, као и многе друге погодности. [9]

Како ниједна веб или мобилна апликације не може бити израђена само употребом *PHP*-а, овај *framework*, иако је углавном окренут развоју *backend* дела апликације, пружа могућност целокупног развоја веб апликација. Ово се углавном постиже коришћењем *Laravel Blade* компоненти у самом *framework*-у. Предност њихове употребе је то што је већина основних шаблона за израду *frontend* дела већ имплементирана у *framework*-у са додатком *CSS*-а и *JS*-а. Додатно, у случају потребе за модификацијом ових компоненти, програмер их може *publish*-овати и прилагодити својим потребама. Међутим, из мог искуства, у раду са *SPA*<sup>9</sup> препоручујем његову употребу само као моћног *API* сервиса.

### 3.3. Рутирање

Рутирање је веома битан део овог развојног окружења. Представља могућност комуникације корисника са системом путем *HTTP* захтева. У зависности од тога да ли је

---

<sup>7</sup> Middleware – посреднички слој који на основу дефинисаних правила проверава право приступа подацима које чува

<sup>8</sup> CLI – Command-line interface

<sup>9</sup> SPA – Single page application

результат који се рутом враћа приказ веб странице или обрађени податак, руте делимо у два документа унутар *routes* директоријума и то на *web.php* и *api.php*. Креирање рута је почетни задатак приликом имплементирања нових функционалности сваког *Laravel* система и то се постиже употребом *Route* фасаде која омогућава употребу *Router* класе са предефинисаним методама. Ове методе су заправо HTTP методе:

- GET – користи се за прибављање података,
- POST – користи се за креирање и упис нових података у базу података,
- PUT – користи се за измену постојећих података у бази података,
- DELETE – користи се за брисање података. [10]

Креирање структуре руте са овим методама веома је битан процес. У случају прибављања података неког модела, паметно је пратити конвенцију креирања руте. Коришћењем имена модела, руту креирати тако да име модела прати идентификациони број, док је за резултат обраде потребно упутити на функцију унутар контролера који припада том моделу. Правилном конструкцијом руте, само прослеђивањем идентификационог броја ентитета тог модела, *Laravel* ће сам прибавити цео објекат и проследити га функцији на обраду. Овај процес познат је под именом *Route Model Binding* и веома је користан јер скраћује и олакшава процес имплементације жељене функционалности. [11]

### 3.4. Валидација

Осим прослеђивања идентификационог броја или другог уникатног податка ради проналажења инстанце ентитета у бази података, руте малтене увек са собом носе додатне податке. Ови подаци представљају *request data* односно податке захтева. Овакве податке систем не би требао тек тако обрађивати јер се руте система на тај начин могу злоупотребити или неправилно искористити. Због тога се уводи систем валидације.

*Laravel*-ова *Validator* класа састоји се од бројних валидационих функционалности којима се обрађују прослеђени подаци који могу потицати од неке попуњене форме или бити било који тип података који се шаље на *backend*. Валидатором се дефинишу правила којим се, на пример, провера да ли су сви неопходни подаци прослеђени, да ли су одговарајућег типа, да ли није случајно прослеђен неки забрањени податак или недозвољена комбинација података и слично. Ова правила представљају кључ вредност парове раздвојене „|“ карактером. Сваки кључ представља име податка, а вредност скуп правила које тај податак мора да поштује. Скуп правила, вредност, може се дефинисати навођењем правила које се раздвајају „|“ карактером или као низ правила раздвојених запетом унутар угластих заграда.

У случају да подаци не прођу валидацију, вратиће се грешка са *error* атрибутом који ће садржати све информације о невалидности прослеђених података. Оваква информација може се приказати кориснику, а у случају валидације форме често ће се форма поново попуњити прослеђеним подацима, али ће овог пута уз те податке стојати и информација о невалидности истих. Док се у супротном наставља са даљим извршењем. [12]

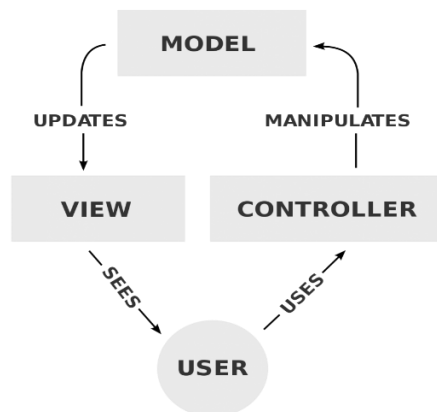
Оваква валидација може се имплементирати директно у контролеру, али ради боље прегледности *Laravel* омогућава креирање валидационих класа. Ове класе проширују

*FormRequest* класу, те додатно, на свој начин дефинишу *authorize* и *rules* функције. Овим функцијама постиже се специјализована провера да ли корисник који шаље захтев заправо има право приступа тој рути и дефинише се низ валидационих правила. Додатно, ова класа пружа могућност предефинисања и *messages* функције. *Messages* функцијом одређују се валидационе поруке које се враћају кориснику као информација о невалидности у случају да прослеђени подаци не пролазе валидациона правила. [13]

### 3.5. MVC архитектура

Користи *MVC* архитектуру која омогућава бржи развој апликације, лако кориговање система и повећање скалабилности истог. Овом архитектуром је развојни процес подељен у три главна дела:

- У овој архитектури *M* стоји за *model*, што заправо представља руковођење подацима. Начин употребе, повезаност и обада података дефинишу се у оваквим класама. Модели представљају спону између контролера и базе података. У њима се може дефинисати које податке можемо читати, уписивати, преображавати, како ће се везе међу подацима тј. моделима у систему посматрати и прибављати.
- Даље, *V* стоји за *view*, односно репрезентацију података на корисничком интерфејсу. Овај део постиже се коришћењем *HTML*-а, *CSS*-а и *JavaScript*-а. Поглед добија све податке обрађене од стране контролера и врши њихов приказ у прегледном формату који одговара људском оку.
- И на крају *C* стоји за *controller*, најкомплекснији део ове архитектуре. Сви захтеви за обрадом или прибављањем података преко рутера долазе до контролера и ту се догађа најбитнији део *backend* логике. Пратећи добру праксу, сав код који представља обраду или директну комуникацију са моделом издваја се у сервисе који се позивају из контролера. Контролер може обрадити податке, форматирати их на жељени начин и вратити у *JSON* формату или тако припремљене проследити погледу и вратити целу компоненту за приказивање. [14]



Слика 3 – *MVC* архитектура у примени [8]

### 3.6. Redis

Осим рада са структурираним релационим базама података *Laravel* омогућује и употребу нерелационих база података. Како само име каже, овакве базе података омогућују упис неструктурираних података по принципу кључ вредност односа. Веома су лаке за подешавање и често се могу користити за кеширање привремених вредности или у потпуности заменити употребу релационих база података уколико је то потребно за одређени тип пословања система. У случају употребе са подацима које је потребно брзо прибавити и не дуго у систему задржати, додатно у случају рада са подацима коју су везани за пословање система али не по мапираној структури објеката, овај систем чувања података је одличан. [15] Генерално, *Redis* који је можда и најкоришћенији тип ове врсте базе података, доста се користи у хибридном раду са стандардним базама података.

*Redis* представља добро развијени систем чувања различитих врста података у кључ вредност односу. Један кључ може имати вредност у виду текста, листе, сета података и хеш табеле<sup>10</sup>. Чување различитих типова вредности имплементирано је различитим специјално генерисаним функцијама које могу да раде са тим типом података.

```
use Illuminate\Support\Facades\Redis;

Redis::set('demo_type', 'string');
Redis::get('demo_type');

Redis::sadd('set', json_encode($demoObject1));
Redis::sadd('set', json_encode($demoObject2));
Redis::smembers('set');
```

*Пример додавања и читања текстуалног података и сета података*

Овако сачувани подаци најчешће се користе за:

- кеширање вредности чије ће прибављање бити веома брзо потребно. Овакав начин употребе се често користи са *Cache* фасадом *Laravel*-а подешавањем *CACHE\_DRIVER* атрибута конфигурационог документа на *redis*,
- чување распореда *Job*-ова за извршење у случају синхроног или одгођеног извршења, подешавањем *QUEUE\_CONNECTION* атрибута конфигурационог документа на *redis*,
- и генерално за пренос података у реалном времену између клијентског и серверског дела система, углавном приликом обавештавања клијентске стране о променама које се дешавају на серверској страни. [16]

---

<sup>10</sup> Hash table – или на српском хеш табеле су подаци организовани по принципу кључ вредност који омогућује веома брз и лак приступ и модификацију података

### 3.7. Homestead

Како би се избегло локално подешавање система програмера за развој у *Laravel*-у развијен је *Homestead*. Он представља употребу виртуелне машине на којој се налази окружење потпуно опремљено свим сервисима за развој *Laravel* апликације, односно пружа кориснику на употребу *PHP*, *MySQL*, *Redis*, *Node* и друге сервисе. Може се покренути на *Windows*, *Mac* или *Linux* системима. Да би се покренуо, потребно је на рачунару подесити *VirtualBox* 6.1.x и *Vagrant*. Пре самог покретања, потребно је дефинисати конфигурациони фајл којим ће *Homestead* знати како да преслика податке са локалног система на виртуелну машину, које портове за комуникацију да отвори и на којим веб адресама да покрене апликацију. [17]

```
---
ip: "192.168.56.56"
memory: 2048
cpus: 2
provider: virtualbox

authorize: ~/.ssh/id_rsa.pub

keys:
  - ~/.ssh/id_rsa

folders:
  - map: ~/code
    to: /home/vagrant/code

sites:
  - map: homestead.test
    to: /home/vagrant/code/public

databases:
  - homestead

features:
  - mariadb: false
  - postgresql: false
  - ohmyzsh: false
  - webdriver: false
  - influxdb: false
```

```
services:
  - enabled:
      - "mysql"
  # - disabled:
  #     - "postgresql@11-main"

# ports:
#   - send: 33060 # MySQL/MariaDB
#     to: 3306
#   - send: 4040
#     to: 4040
#   - send: 54320 # PostgreSQL
#     to: 5432
#   - send: 8025 # Mailpit
#     to: 8025
#   - send: 9600
#     to: 9600
#   - send: 27017
#     to: 27017
```

*Изглед Homestead.example.yaml конфигурационог фајла*



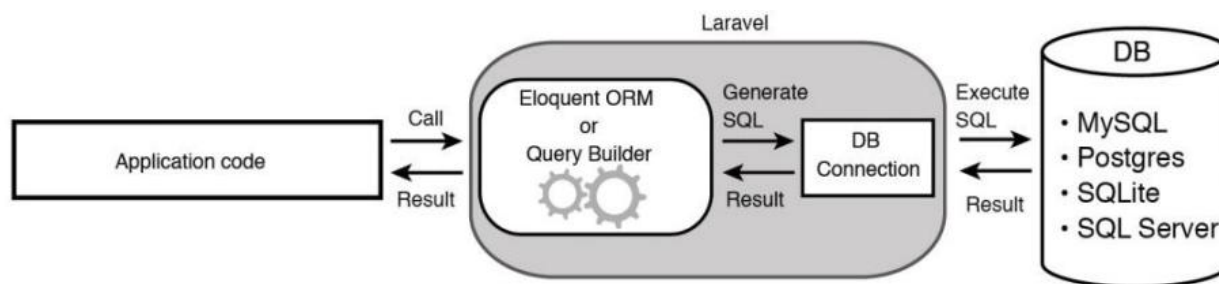
## 4. ELOQUENT ORM

### 4.2. Опште

Приликом прибављања и самом раду са подацима *Laravel* користи *Eloquent ORM* или ти објектно-релациони мапер. Он служи за повезивање података апликације са подацима из базе података. Свака табела базе података посматра се као класа модела од које се креира објекат. Све колоне табела посматрају се као атрибуту објекта који се такође дефинишу у моделу. Како се мапирање ових података врши у моделу, ово представља праву примену *MVC* архитектуре. [18]

Без употребе ове технологије, свака комуникација са базом података би била доста комплекснији и дуготрајнији задатак. Како је сама идеја постојања *framework*-а олакшање развоја апликација, тако је исто и код употребе објектно-релационог мапера. Долазећи са сетом дефинисаних инструкција, остварује се веома брз приступ подацима, али и подацима који су у релацији са тренутно прибављеним моделом. Приступ подацима постиже се на најједноставнији начин, коришћењем метода попут *find(\$id)*, *all()*, *where('column', 'value')*, *first()*. Ове методе позивају се преко дефинисаних *Eloquent* модела, тако што се испише име модела праћено двоструком двотачком и конкретном методом. Осим приступа, на исти начин могу се вршити креирања, измене и брисања података коришћењем метода редом *create(['column1' => 'value', 'column2' => 'value'])*, *update(['column' => 'new\_value'])* и *delete()*. [19]

Коришћење ових предефинисаних инструкција заправо представља *Laravel*-ово превођење *PHP* наредби у *SQL* наредбе и мапирање прибављених података у објекте. Када се креирани објекат сачува, *Laravel* ће у позадини ову акцију извршити уписом података у нови ред табеле у бази података. Док се у случају прибављања података, ред података из базе података у апликацији представља као објекат, а свака измена на овај ред у бази података биће директно пресликана на прибављени објекат. [20] Додатно, свака од ових измена над подацима у бази података биће пропраћена *Laravel*-овим аутоматским акцијама записивања временског тренутака у којем се десила измена. Овај процес аутоматског бележења другачије се назива *automatic timestamps* чиме се постиже најједноставнији вид праћења измена над подацима у систему. [19]



Слика 4 – *Eloquent ORM* принцип рада [20]



### 4.3. Мапирање модела и његова употреба

Како је сваки ентитет базе података у *Laravel*-у представљен као класа, преко Eloquent модела морају се вршити дефинисања начина употребе и повезивања ових података. Нека од најчешће употребљених функционалности модела су вршења дефинисања:

- *fillable* – поља која се могу уносити и мењати у бази података. Овим се упис података у базу података ограничава само на наведене атрибуте, односно колоне. Ова поља ће се подразумевано користити приликом масовног додељивања вредности у случају чувања из контролера или у случају приступа истим,
- *hidden* – поља која је могуће уписати, мењати али не и читати. Често се користе за аутентификациона поља односно лозинку и токене,
- *casts* – поља чију вредност у бази података чувамо у једном типу података, а онда у овој секцији дефинишемо преображавање у други тип података,
- *appends* – поља чије се вредности не чувају у бази података већ се могу израчунати на основу вредности других података. Углавном представљају комплексни, изведени, података, а такви се у бази података не чувају,
- *table* – име табеле са којом се врши мапирање тог модела, мада често у пракси ово можемо и изоставити јер ће *framework* сам препознати о којој табели се ради на основу самог именовања модела,
- дефинисање повезаности модела са његовом фабриком за креирање нових инстанци – начин креирања оваквих инстанци и могућност аутоматске модификације креираних података, [21]
- итд.

Eloquent модели пружају могућност дефинисање релационих односа ка другим моделима, односно табелама. Везе које се у овим моделима могу дефинисати су:

- *one to one* – основни тип везе међу подацима у коме је очигледна припадност једног модела другом. У власничком моделу веза се дефинише методом *hasOne(ModelName::class)* док се у припадајућем моделу веза дефинише методом *belongsTo(ModelName::class)*. Ове методе позиваће се у функцијама које носе име јединине модела са којим се повезују,
- *one to many* – веома сличан тип везе првом типу са проширењем повезаности једног модела ка више других, односно један модел поседује више њих. У овом случају за дефинисање веза користе се методе *hasMany(ModelName::class)* и *belongsToMany(ModelName::class)*. Ове методе позиваће се у функцијама које носе име редом множине и јединине модела са којим се повезују,
- *many to many* – тип везе којим се постиже приказивање вишеструке обостране припадности међу моделима. У оба модела користе се методе *belongsToMany(ModelName::class)* док се у бази података додатно креира посредничка табела за праћење релације познатија као пивот табела. Ове

методе позиваће се у функцијама које носе име множине модела са којим се повезују,

- *polymorphic relation* – тип везе који омогућује припадност једног модела ка више типова различитих модела. Па ће тако припадајући модел користити методу *morphTo()* без наглашавања специфичног модела у оквиру функције која носи име свог модела уз суфикс *able*, док ће власнички модели имати методе *morphOne/Many(ModelName::class, 'modelable')* унутар функције која носи име једине/множине модела са којим се повезује.,
- додатно могу се дефинисати и комплексне посредничке везе након дефинисања основних типова веза коришћењем унакрсног везивања чиме се постиже *has one through* и *has many through*. [22]

Креирање веза међу моделима омогућава олакшани приступ широком спектру података. *Laravel*-ове уграђене *Eloquent* методе на једноставни начин уз прибављање једне инстанце модела, могу прибавити и све жељене повезане моделе. На овај начин постиже се рано прибављање података или ти *eager loading*. Једноставном употребом *with('related\_model\_name')* добија се жељени објекат проширен повезаним моделом. Овакве методе повећавају квалитет написаног кода и спречавају настајање бројних грешака у писању комплексних *SQL* упита. [19]

Након имплементације мапирања *Eloquent*-а у класама модела, треба разумети да се ово мапирање и његови модели користе широм целог *backend*-а и налазе примену и у другим деловима имплементације. Говорећи о овим имплементацијама имамо:

- фабрике – класе којима се наслеђивањем апстрактне класе *Factory* додељује могућност креирања лажних података објекта пратећи мапирану структуру објекта креирану моделом,
- *seeder*-е – класе којима се наслеђивањем апстрактне класе *Seeder* додељује могућност аутоматског уписивања прослеђених података у базу података. Прослеђени подаци углавном се генеришу употребом већ дефинисаних фабрика,
- миграције – класе којима се наслеђивањем апстрактне класе *Migration* и на основу идеје о структури података додељује могућност повезивања са базом података и дефинисања имена и типова поља у бази података, њихових веза са другим објектима, начин њиховог креирања, мењања или брисања,
- *api* руте – скуп рута које заправо представљају инстанце класа, позивом функција класа, у случају правилног именовања рута, користи се *model binding* механизам којим се инстанце модела повезују са идентификаторима прослеђеним рутама и функцијама на обраду се дају тачно обликовани објекти као што је у моделу дефинисано [23]
- контролере – класе којим се од рута примају објекти модела и прослеђују на даљу обраду испуњавајући *CRUD*<sup>11</sup> и друге захтевније захтеве,
- *event*-е – класе којима се прате промене над мапираним објектима података и даље се примењују друге зависне акције, често праћене *listener* класама

---

<sup>11</sup> *CRUD* – акроним за функције које врше наредбе *create*, *read*, *update* и *delete*

- *middleware* – правило приступа одређеним ресурсима провером поређења модела корисника који захтева приступ и модела корисника коме је приступ одобрен,
- итд.

## 5. ИМПЛЕМЕНТАЦИЈА АПЛИКАЦИЈЕ

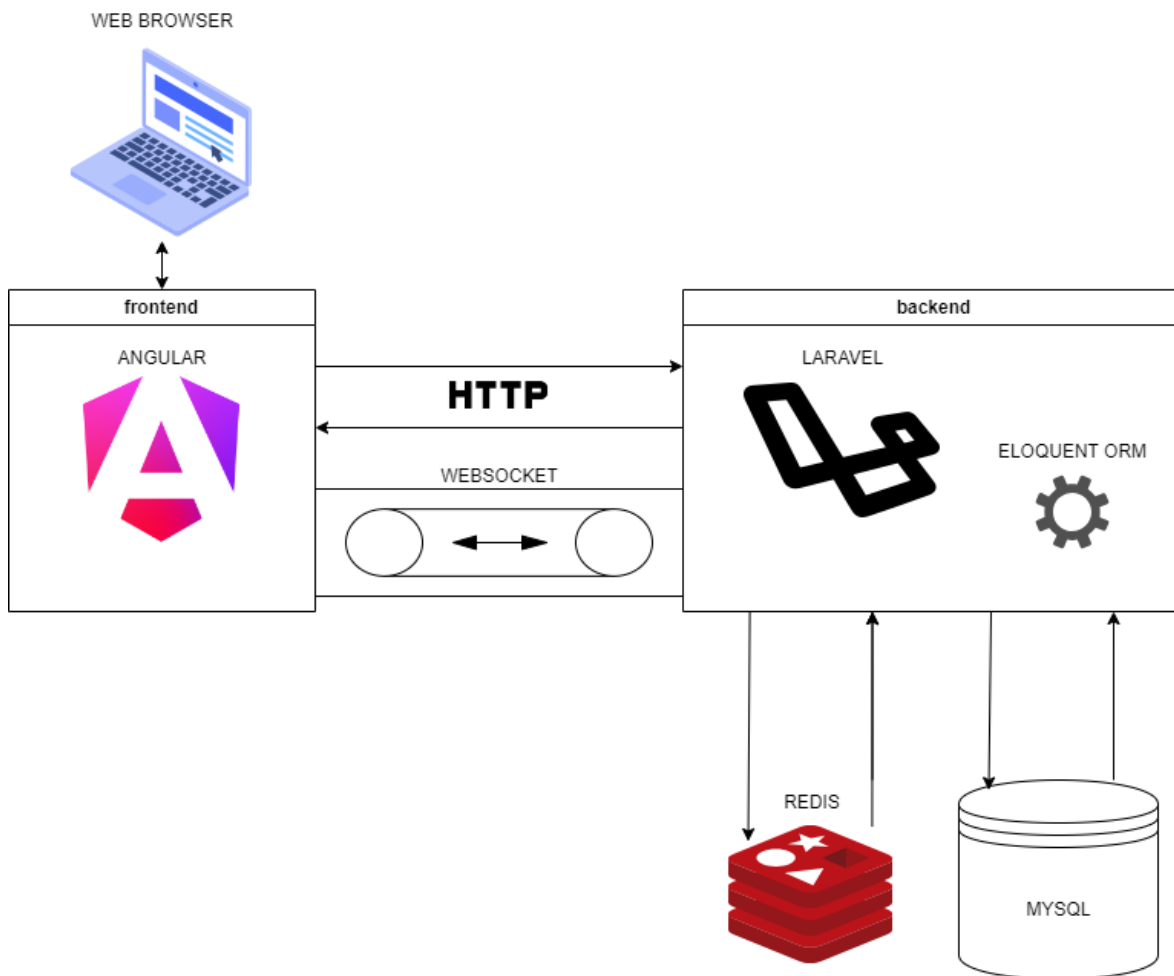
У овом поглављу биће детаљно описана архитектуре и имплементација проучених технологија са циљем доказивања предности употребе ових технологија на реалном проблему. Поглавље се технички ослања на већ објашњену терминологију те се неће наводити поновна објашњења.

Код веб апликације налази се на два *GitHub* репозиторијума. Код је по имплементацији подељен на *web* и *api* део:

- <https://github.com/MiticBNikola/taxi-api/>
- <https://github.com/MiticBNikola/taxi-web/>

### 5.2. Архитектура апликације

За потребе објашњења проучених технологија, односно практичног дела дипломског рада, креирана је веб апликација ЕлФак такси. Апликација представља класичан пример клијент сервер апликације. Апликација омогућава наручивање, прихватање и праћење вожње, као и управљање корисничким профилем. Апликација је намењена свим људима који желе да добију услуге брзог транспорта, као и људима који би лако и брзо зарадили возећи такси док постоји надређени менаџер који прати њихово пословање. Самим тим апликација је морала бити дизајнирана тако да њена употреба буде што једноставнија, али и да приступ истој буде брз. Из тих разлога креирана је веб апликација са прилагодљивим дизајном која се може приказати у сваком веб претраживачу. Креирана је у *Angular*-у и *Laravel*-у са коришћењем *MySQL* и *Redis* база података.



Слика 5 – Приказ архитектуре веб апликације

### 5.3. MySQL

MySQL је најпознатији open source<sup>12</sup> управљачки систем релационих база података. Како сам опис каже, користи се за креирање и управљање релационим базама података. Због своје карактеристике open source платформе, сами корисници по наилажењу још увек не решених проблема имају могућност имплементације свог решења и дељења истог са осталим корисницима, што представља начин раста и развоја овог система. Осим ове карактеристике, систем је веома лак за употребу што му додатно иде на руку. Омогућава рад разноврсних система због варијације чувања количине података, од минималних података неког ресторана до великих комплексних записа података бизнис компанија. [24]

Најбитније својство MySQL базе података је употреба ACID трансакција. Овај акроним представља атомичност, конзистентност, изолованост и издржљивост. Оваквим

<sup>12</sup> open source – код отвореног приступа, значи да имплементацији система свако може да приступи и предложи нове функционалности

атрибутима описује се процес извршења једне радње над подацима у бази података, односно другачије названо трансакције података. Трансакције су атомичне јер се њихово извршење дозвољава само у целости, уколико неку од радњи трансакције није било могуће извршити, трансакција ће се поништити, односно никаква измена неће бити сачувана у бази података, макар до грешке дошло и на крају извршења трансакције. Оне су конзистентне јер њихово извршавање омогућава упис једино валидних података, пратећи дефинисану структуру типа података и међусобног односа истих. Трансакције су и изоловане, односно не дозвољавају конкурентност модификације података у истом тренутку, превodeћи их у синхроне процесе, спречавајући да се подаци система доведу у невалидно стање. Коначно, оне су и издржљиве односно гарантују очување уписаних података неvezано за стање система у случају пада истог. [25]

## 5.4. Angular

*Angular* је *frontend framework* који користи *TypeScript* програмски језик. Креиран је са истом идејом као и *Laravel*, односно тежи да олакша израду комплексних веб апликација без потребе да се све развија од почетка. Апликације овако развијене имплементиране су на *MVC* архитектури, о којој је већ писано.

Дефинисање и понашање елемената апликација одређује се декораторима. Декоратори пружају начин за додавање додатних функционалности и мета податке у елементе апликације. Најчешће се користе за дефинисање компоненти, сервиса, директива, *pipe*-ова и другог. Најбитнији декоратори су:

- класни декоратори, они дефинишу понашање компоненти апликације. Компонента је основни елемент сваке апликације, данас се апликације генеришу од самосталних компоненти без потребе за родитељски модулom.

```
@Component({
  selector: "app-demo",
  template: "./demo.component.html",
  styleUrls: ["./demo.component.sxss"],
  standalone: true,
})
class DemoComponent {...}
```

### *Пример самосталне компоненте*

- декоратори својства, они дефинишу понашање атрибута компоненти апликације. На пример на овај начин могу се дефинисати улазни атрибути и пратити њихове измене које се дешавају у родитељској компоненти, али и дефинисати емитере промена ка родитељској компоненти.

```
@Input() title = '';
@Output() signalChange: EventEmitter<{ title: string}> = new EventEmitter();
```

### *Пример примања вредности и враћања акције*

- декоратори параметра, они омогућују вршење  $DI^{13}$ -а односно убацивања зависности. На тај начин постиже се извлачење комплексних логика у класе сервиса које се коришћењем оваквих декоратора лако укључују у све компоненте система. [26]

```
@Injectable({ providedIn: 'root' })
export class HelperService { ... }

@Component({
  selector: "app-demo",
  template: "./demo.component.html",
  styleUrls: ["./demo.component.sxss",
  standalone: true,
})
class DemoComponent {
  private helperService = inject(HelperService);
  ...
}
```

### *Пример дефинисања и убацивања помоћног сервиса*

Посматрање података дефинисано је двосмерним повезивањем. Ажурност података на приказу постиже се посматрањем везе између приказаног податка на прегледу и атрибута у класи компоненте. Ова веза омогућује приказ промена у реалном времену без потребе за освежавањем странице. [27] Додатно унапређивање у преношењу ажурности података постигнуто је сигнаlima. Сигнали представљају нов концепт у дефинисању података. Понашају се као омотач око вредности податка који у случају измене вредности податка сигнализира остатку апликације нову вредност. Користе се за обавештавање међу компонентама, израчунавање нових вредности изведених атрибута након промена основних вредности и за извршење додатних функционалности након измена неких података. Самим тим уочавамо сигнале за упис, израчунате сигнале и ефекте. [28]

Апликације развијене у *Angular*-у су заправо *SPA* односно апликације са једном страницом. Овакав приступ омогућује иницијално приказивање главне компоненте која представља корен система. На овој компоненти ређају се остале компоненте. Како се сви системи заправо састоје од више различитих страница, најбитније је заправо употребити *router-outlet* у овој компоненти. На овај начин *Angular* ће сам смењивати приказ жељених

---

<sup>13</sup> DI – Dependency Injection

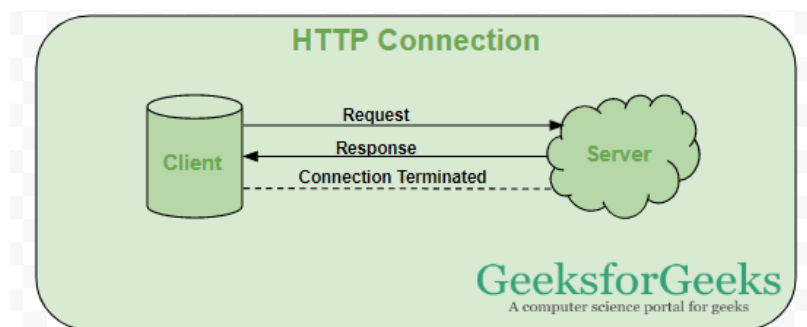
компоненти у главној компоненти на основу руте веб претраживача без регенерисања главе компоненте. Овакав приступ у многоне побољшава перформансе система. [29]

```
<div>
  <app-header></app-header>
  <router-outlet></router-outlet>
  <app-footer></app-footer>
</div>
```

*Пример главне компоненте SPA*

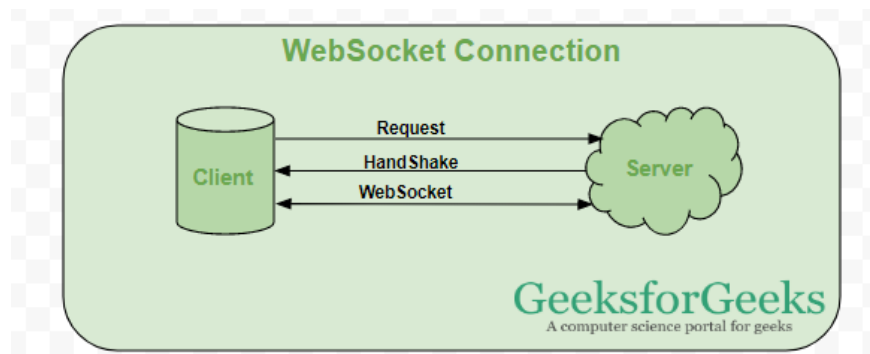
## 5.5. Web Socket

Са развојем апликација долази се до ситуација када слање *HTTP* захтева за прибављање података није довољна функционалност. Често, нови и захтевнији системи очекују неки вид обавештења приликом измена података у позадини или у току процеса у ком сами нису учествовали. Иницијална идеја константног позивања *backend*-а није добро решење, зато су осмишљени *WebSocket*-и. Уместо слања једносмерних захтева са *http://* односно *https://* чији одговор клијент очекује након позива, у овим случајевима користи се двосмерни канал са *ws://* односно *wss://* ознаком.



*Слика 6 – Приказ процеса слања HTTP захтева [30]*

*Web Socket*-и пружају двосмерну комуникацију у систему, омогућавајући интеракцију *backend*-а са *frontend*-ом без постојања константних упитних захтева. Како би се комуникација остварила, користи се *handshake* метода којом клијент и сервер потврђују жељену комуникацију и креирају се комуникациони канали. Овако креиране канале обавезно је затворити када више није потребно ослушкивати податке или пре прекидања рада апликације. Сваки канал носи своје идентификационо име како би апликације могле да ослушкују поруке на том каналу. Поруке на каналу дефинисане су различитим типом догађаја, и преносе жељене податке смештене у *JSON*-у.



Слика 7 – Приказ процеса слања *WebSocket* захтева [30]

Зависно од потреба и технолошке позадине апликације, треба разумети да коришћење *WebSocket*-а не може и не треба заменити *HTTP* конекцију. Пожељно је користити их приликом прибављања података у реалном времену, израде апликације за комуникацију, израде апликације за праћење кретања или слично. У осталим случајевима пожељно је придржавати се традиционалног начина прибављања података јер креирање и одржавање канала комуникације није корисно за апликације које имају само захтеве за повремено прибављање или ажурирање података без интеракције са другим корисницима. [30]

## 5.6. WebStorm

WebStorm је развојно окружење погодно за развој *frontend* дела веб и мобилних апликација. Омогућава једноставни развој оваквих апликација коришћењем технологија као што су JavaScript, React, TypeScript, Angular, Vue, HTML, CSS.

Омогућава кориснику да одмах започне са кодирањем без потребе да инсталира и подешава додатке. WebStorm укључује се што је иницијално потребно за развој апликација у JavaScript-у и TypeScript-у. Оставља могућност за каснију персонализацију са мноштвом различитих додатака и подешавања. [31]

Од додатака са верзијом 2024.2.3 користио сам ESLint и Prettier ради лакшег и бржег декорисања и форматирања кода како би се постигла униформност при развоју на било ком рачунару.

## 5.7. PhpStorm

PhpStorm је развојно окружење погодно за развој *backend* дела веб и мобилних апликација. Омогућава једноставни развој оваквих апликација коришћењем технологија као што су PHP, Laravel, Blade, Symfony, JavaScript, SQL.

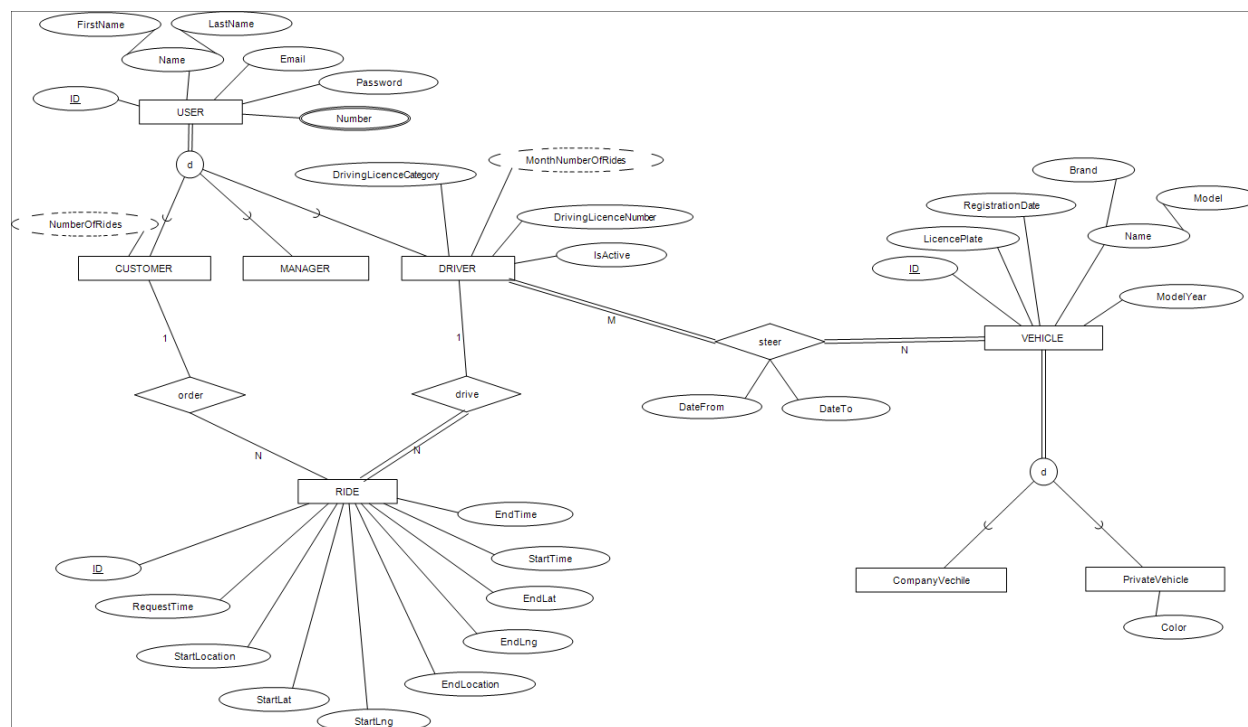


У случају развоја апликације на Windows оперативном систему користи се са виртуелном машином. Кориснички интерфејс додатно је обogaћен лаком и брзом претрагом докумената у пројекту као и њиховог садржаја са могућношћу дефинисања опсега претраге.

У имплементацији коришћена је верзија 2024.2.3. Такође, велика предност овог окружења је и додатак Database који ради по истом принципу као и засебни систем DataGrip. Коришћење овог додатка омогућило ми је брз и лак увид у тренутно стање базе података, као и доста опција за манипулисање истом.

## 5.8. Опис имплементације базе података

Најбитнији део израде практичног дела дипломског рада представљао је процес дефинисања изгледа жељене базе података. Анализом задатака, долази се до издвајања свих могућих ентитета и њихових атрибута. Затим приступа се анализи ентитета, проналажењу њихових сличности и дефинисању евентуалних надкласа издвајањем заједничких атрибута. Следећи корак представљало је разумевање односа ентитета у задатку, односно успостављање веза међу њима. Битно је водити о обавезности учествовања ентитета у вези као и квантитативном учинку. Тако је, ради разумевања изгледа будуће базе података, прво је креиран *EER* дијаграм.



Слика 8 – *EER* дијаграм ЕлФак такси базе података

Посматрањем дијаграма, а затим и применом правила превођења *EER* модела у релациони модел добијамо базу података са табелама:

- CUSTOMER – садржи све податке о кориснику типа муштерија
- DRIVER – садржи све податке о кориснику типа возач
- MANAGER – садржи све податке о кориснику типа менаџер

У превођењу надкласе *User* и подкласа *Customer*, *Driver* и *Manager*, коришћен је други начин превођења овог типа ентитета, односно креиране су табеле за подкласе са свим атрибутима надкласе. Ово је касније довела до комплексније логике регистрација и пријављивања на систем, односно потребе за ручном имплементацијом ових функционалности, без могућности употребе већ постојећих, имплементираних од стране *framework*-а. А самим тим и до интегрисања специјалне логике за проверу аутентификационог статуса корисника употребом нових *middleware*-а у провери.

```
public function authCheck(Request $request): JsonResponse
{
    $type = $request->query('type', 'web');
    if (auth()->guard($type)->check()) {
        $user = auth()->guard($type)->user();

        return response()->json(["user" => $user, "type" => $type]);
    }

    return response()->json(NULL, 401);
}
```

#### *Пример имплементације провере аутентификационог статуса корисника*

- NUMBERS – садржи податке о бројевима телефона корисника апликације, као и њихов тип и идентификациони број. Нема јасно дефинисану релацију, јер применом *morphs* типа везе, омогућујемо креирање припадности броја телефона било ком ентитету базе података.
- RIDE – садржи све податке о вожњи, од креирања захтева, преко управљања истим, до завршетка вожње. Представља најупотребљивију табелу система јер константо долази до ажурирања података исте.
- VEHICLE – садржи податке о свим возилима у систему. Коришћењем трећег начина превођења ентитета у односу надкласа подкласа, добијамо једну табелу са свим подацима надкласе, типом унесеног возила и свим подацима подкласа где су са *NULL* означена поља која тренутном уносу не одговарају.
- STEERS – садржи историју односа управљања возилом од стране возача. Због везе типа више према више, креирана је нова засебна табела у којој се налазе страни кључеви возача и возила, како би се одредила њихова повезаност. Додатно памте се атрибути везе, односно период управљања возилом од стране возача, како би било могуће пратити историју управљања.



Слика 9 – MySQL шема релационе базе података

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=taxidb
DB_USERNAME=homestead
DB_PASSWORD=secret
```

Део конфигурације .env документа везан за комуникацију са базом података

## 5.9. Имплементација серверског дела

Серверски део апликације имплементиран је у *Laravel*-у 11 са *PHP*-ом 8.2. Пре самог почетка имплементирања апликације, добро је осмишљена архитектура исте, затим је на основу задатка креиран дијаграм базе података и преведен у релациони модел, а онда пре самог почетка имплементације подешен и рачунар уз помоћ виртуелне машине и *Homestead*-а.

```
APP_NAME=ElfakTaxi
APP_ENV=local
APP_KEY=base64:po2SuFHsnVC8Gxyso2caUs33H5hs8b05IJy13i0jNQs=
APP_DEBUG=true
APP_URL=http://localhost
```

*Део имплементације .env документа који је основни за сваку Laravel апликацију*

*Laravel*-у је за потребе тестирања приликом развоја додат пакет *laravel/tinker*. Овај пакет се покреће у конзоли и омогућује тестирање имплементиране апликације. Креирањем нових објеката користећи имплементиране модуле, можемо испробати имплементиране функционалности. Сва креирања, измене и брисања која се дешавају унутар *tinker*-а су тренутни тестни подаци и ништа се неће одразити на податке система.

За потребе пријављивања и регистрација на систем додат је пакет *laravel/sanctum*. Овај пакет који користи технологију провере валидности приступа корисника провером његових токена. Токени се у *HTTP* захтеву прослеђују у *headers* секцији. Након инсталације пакета потребно је прецизно подесити конфигурациони *config/sanctum.php* документ овог пакета како не би долазило до потенцијалног неразумевања у комуникацији између *backend*-а и *frontend*-а где би систем захтеве корисника препознавао као неауторизоване и конфигурациони *config/cors.php* документ како систем не би захтеве клијента посматрао као потенцијалне нападе.

```
<?php

return [
    'paths' => ['api/*', 'sanctum/csrf-cookie', 'login', 'logout', 'register',
    'password/reset'],
    ...
    'supports_credentials' => true,
];
```

*Део имплементација config/cors.php*

```

<?php

use Laravel\Sanctum\Sanctum;

return [
    'stateful' => explode(',', env('SANCTUM_STATEFUL_DOMAINS', sprintf(
        '%s%s',
        'localhost,localhost:3000,127.0.0.1,127.0.0.1:8000,::1',
        Sanctum::currentApplicationUrlWithPort()
    ))),
    'guard' => ['web'],
    ...
],
];

```

*Део имплементација config/sanctum.php*

```

SESSION_DRIVER=cookie
SESSION_LIFETIME=120
SESSION_DOMAIN=localhost
SANCTUM_STATEFUL_DOMAINS=http://localhost:4200
COOKIE_SAME_SITE_POLICY=strict

```

*Део имплементације .env документа везан за sanctum*

За потребе чувања краткорочно потребних података у Laravel-у додат је Redis. Обзиром да се ради о малој количини једноставнијих података који се никада не нагомилавају, без потребе подешавања Redis сервера искоришћен је пакет *redis/redis* који представља php библиотеку и лако се покреће са сваком php апликацијом. Додатно, како је за обављање неких обрада података потребно сачекати одређени временски период додата је употреба Laravel Job-ова чије се распоређивање за извршење такође врши коришћењем Redis-а. За употребу Redis-а било је потребно креирати конфигурацију у главном конфигурационом документу.

```

QUEUE_CONNECTION=redis
REDIS_CLIENT=predis
REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

```

*Део конфигурације .env документа везан за Job и config/database.php документ*

```
...
$redisKey = "ride:$rideId:driver-location";
Redis::sadd($redisKey, json_encode(['lat' => $request->get('lat'), 'lng' =>
$request->get('lng'), 'driver_id' => $driver->id]));
...
```

### *Пример додавања података у Redis низ*

За потребе додатне комуникације клијената у систему, односно преношења података у реалном времену тј. додељивања возње најближем кориснику и приказа кретања такси возила на клијентској мапи, у *Laravel*-у је додат и пакет *laravel/reverb*. Овим пакетом омогућено је слање података специјално креираним каналима. Након конфигурације слања података у *config/broadcasting.php* и *config/reverb.php* датотекама, потребно је креирати класе *Laravel event*-а који се проширују *ShouldBroadcast* интерфејсом. Свака од ових класа дефинише које податке добија приликом слања поруке на канал, дефинише како ће проследити те податке, на ком каналу ће их послати и под којим именом.

```
<?php
return [
    'default' => env('BROADCAST_CONNECTION', 'null'),
    'connections' => [
        'reverb' => [
            'driver' => 'reverb',
            'key' => env('REVERB_APP_KEY'),
            'secret' => env('REVERB_APP_SECRET'),
            'app_id' => env('REVERB_APP_ID'),
            'options' => [
                'host' => env('REVERB_HOST'),
                'port' => env('REVERB_PORT', 443),
                'scheme' => env('REVERB_SCHEME', 'https'),
                'useTLS' => env('REVERB_SCHEME', 'https') === 'https',
            ],
            'client_options' => [],
        ], ...
    ],
];
```

### *Део имплементације config/broadcasting.php*

```
BROADCAST_CONNECTION=reverb
REVERB_APP_ID=583497
REVERB_APP_KEY=ifplsmodcxg4mk1efqa5
REVERB_APP_SECRET=8s6rxweoxjlk2v3znate
REVERB_HOST=localhost
REVERB_PORT=8080
REVERB_SCHEME=http
```

*Део конфигурације .env документа везан за config/reverb.php документ*

```
<?php

namespace App\Events;

use ...

class RideRequested implements ShouldBroadcast
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    public Ride $ride;
    public function __construct(Ride $ride)
    {
        $this->ride = $ride;
    }

    public function broadcastOn(): Channel
    {
        return new Channel('drivers');
    }

    public function broadcastAs(): string
    {
        return 'ride-requested';
    }

    public function broadcastWith(): array
    {
        return ['ride' => $this->ride];
    }
}
```

*Пример имплементације једног Laravel event-а са reverb-ом*

Генерално, апликација је дефинисана тако да у *routes/api.php* документу можемо уочити дефинисање свих *API* рута. Руте су подељене по смисленим целинама, а додатно у зависности од логике приступа подацима посебно заштићене *middleware*-ом који ће све *HTTP* захтеве проверити пре приступа контролеру. Након приступа контролеру, пре самог почетка извршавања кода, долази до валидације прослеђених података. Конкретно у задатку генерисане су засебне валидационе класе за сваки од *API* захтева. Затим, следе позиви помоћних сервисних функција које врше обраду података и комуникацију са базом података. Помоћне сервисне функције налазе се у сервисним класама које имплементирају жељене сервисне интерфејсе. Ови интерфејси се у контролеру прослеђују убацивањем зависности (преко DI), док је њихова инстанца регистрована у систему по *singleton*<sup>14</sup> обрасцу.

### 5.9.1. Eloquent ORM

Коришћењем *Eloquent*-а извршено је мапирање табела релационе базе података на објекте који се користе у систему. Сваки од ентитета базе података, односно свака табела базе података, у систему је представљен као класа модела. *Eloquent*-ом се постигло дефинисање поља за поуну, изведених поља и поља за преображавање типа вредности. Дефинисан је назив табеле са којом је потребно извршити мапирање, као и дефинисање релација и начина учествовања модела у истој.

```
<?php

namespace App\Models;

use ...

class Ride extends Model
{
    use HasFactory;

    protected $fillable = [
        'request_time',
        'start_location',
        'end_location',
        'start_time',
        'end_time',
        'customer_id',
        'driver_id',
    ];
}
```

---

<sup>14</sup> Singleton – образац који се брине о томе да класа има само једну инстанцу, док пружа глобални приступ истој



```

protected $casts = [
    'request_time' => 'datetime',
    'start_time' => 'datetime',
    'end_time' => 'datetime',
];

public function customer(): BelongsTo
{
    return $this->belongsTo(Customer::class);
}

public function driver(): BelongsTo
{
    return $this->belongsTo(Driver::class);
}
}

```

### *Пример имплементације Ride модела*

Поред дефинисања изгледа модела, објектно релациони мапер омогућио је дефинисање миграционих датотека. Миграционим датотекама програмски дефинишемо изглед базе података. Миграционе датотеке омогућују дефинисање типова вредности атрибута, подразумеваних вредности у случају када њихово присуство није обавезно, веза ка другим табелама и понашање у случају брисања или измене вредности у повезаним табелама.

```

<?php

use ...

return new class extends Migration {
    public function up(): void
    {
        Schema::create('rides', function (Blueprint $table) {
            $table->id();
            $table->timestamp('request_time');
            $table->string('start_location');
            $table->decimal('start_lat', 8, 6);
            $table->decimal('start_lng', 9, 6);
            $table->string('end_location')->nullable();
            $table->decimal('end_lat', 8, 6)->nullable();
            $table->decimal('end_lng', 9, 6)->nullable();
            $table->timestamp('start_time')->nullable();
            $table->timestamp('end_time')->nullable();
            $table->foreignId('customer_id')->nullable()
                ->constrained()->cascadeOnUpdate()
                ->nullOnDelete();
        });
    }
};

```

```

        $table->foreignId('driver_id')->nullable()
            ->constrained()->cascadeOnUpdate()
            ->nullOnDelete();
        $table->timestamps();
    });
}

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

};

```

*Пример имплементације миграције Ride модела односно rides табеле*

Овако дефинисаним мапером, лако смо добили могућност креирања класних фабрика које прате правила дефинисања модела. У фабрици се додатно дефинише које врсте вредности желимо на местима атрибута. Затим се позивањем инстанце фабрике у класи *DatabaseSeeder* омогућује аутоматско креирање тестних података. Ови подаци, уписују се у креирану релациону базу података. Овај приступ додатно олакшава тестирање имплементираних релационих база података и објектно релационог мапера пружајући јединствене тестне податке без потребе ручног уноса.

```

<?php

namespace Database\Factories;

use ...

class SteerFactory extends Factory
{
    protected $model = Steer::class;

    public function definition(): array
    {
        return [
            'driver_id' => Driver::factory(),
            'vehicle_id' => Vehicle::factory(),
            'date_from' => $this->faker->date(),
            'date_to' => null,
        ];
    }
}

```

*Пример имплементације фабрике за креирање тестних података*

## 5.10. Имплементација клијентског дела

Клијентски део апликације имплементиран је у *Angular*-у 18. На самом почетку имплементирања апликације, добро је осмишљена навигациона структура исте. Потом је праћењем објектно релационог мапера извешено креирање модела и на самом *frontend*-у како би апликација знала које податке може очекивати приликом читања одговора *backend*-а. Како би се та комуникација преноса података и остварила, дефинисан је конфигурациони документ.

```
export const environment = {  
  production: false,  
  API_URL: 'http://localhost:8000/api',  
  API_DOMAIN: 'http://localhost:8000',  
  REVERB_APP_KEY: 'ifplsmodcxg4mk1efqa5',  
  REVERB_APP_SECRET: '8s6rxweoxjlk2v3znate',  
  REVERB_WS_HOST: 'localhost',  
  REVERB_WS_PORT: '8080',  
  GOOGLE_API_KEY: 'AIzaSyDbkojycfNgFWqv_Zw8ectCFTlY-WtpgoQ',  
};
```

*Глобална конфигурација која дефинише параметре комуникације*

*Angular*-у су за потребе израде прилагодљивог дизајна за приказ апликације на паметном телефону и таблету додати пакети *@ng-bootstrap/ng-bootstrap* и *bootstrap*. Ови пакети омогућили су употребу предефинисаних стилски класа којима се апликација различито приказује у зависности од презентационог уређаја. Овакав дизајн постигнут је дефинисањем *div* елемента са класом *row* који у себи има друге градивне елементе дефинисане класама *col-1*, *col-2*, па све до *col-12* и *col-auto*. Овај принцип ради по систему да један ред може имати 12 јединица.

За потребе чувања и употребе података, коришћен је *Angular*-ов нови приступ коришћења података сигнаlima. Након прибављања кључних података о пријављеном кориснику, исти се чувају у *signalStore*, прибављају се дефинисаним функцијама прибављања и ажурирају дефинисаним функцијама промене вредности. *Store* је дефинисан на глобалном нивоу и у свим независним компонентама коришћен је убацивањем зависности. Како се меморија овог система губи са сваким освежавањем странице веб претраживача додатно је употребљена меморија веб претраживача у коју се уписују све кључне вредности тренутног система.



```

withHooks((store) => ({
  onInit: () => {
    const localStorageToken = localStorage.getItem('token');
    const localStorageSession = localStorage.getItem('session');
    const localStorageType = localStorage.getItem('type');
    patchState(store, {
      ...(localStorageToken && localStorageToken !== '' &&
        { token: localStorageToken }
      ),
      ...(localStorageSession && localStorageSession !== '' &&
        { session: localStorageSession }
      ),
      ...(localStorageType && localStorageType !== '' &&
        { type: localStorageType }
      ),
    });
  },
}));
});
);

```

### Пример имплементације store-a

За потребе додатне комуникације клијената у систему, односно ослушкивање порука послатих са *backend-a*, у *Angular-u* су додати и пакети *laravel-echo* и *pusher-js*. Овим пакетима омогућено је ослушкивање канала са информацијом о новим подацима без потребе за константним слањем *HTTP* захтева. Потребно је имплементирати помоћну сервисну класу која ће инстанцирати *Pusher* и *Echo* и омогућити функционалност отварања и затварања комуникационих канала. Овај сервис је убачен у независне компоненте које имају потребу за оваквим типом комуникације и на основу креираних услова отварају односно затварају ове канале.

```

...
private initEcho() {
  inject(NgZone).runOutsideAngular(() => {
    if (isPlatformBrowser(this.platformId)) {
      (Window as any).Pusher = Pusher;
      this.echo = new Echo({
        broadcaster: 'reverb',
        key: environment.REVERB_APP_KEY,
        wsHost: environment.REVERB_WS_HOST,
        wsPort: environment.REVERB_WS_PORT,
        forceTLS: false,
        enabledTransports: ['ws'],
      });
    }
  });
}

```

```

listen(channelName: string, eventName: string, callback: (res: any) => void) {
  if (!this.echo) {
    return false;
  }
  return this.echo.listen(channelName, eventName, callback);
}

leave(channelName: string) {
  if (!this.echo) {
    return;
  }
  this.echo.leave(channelName);
}

disconnect() {
  if (!this.echo) {
    return;
  }
  this.echo.disconnect();
}

```

### *Пример имплементације echo сервиса*

```

private echoService = inject(EchoService);
...
listenToRideAccepted(rideId: number) {
  this.echoService.listen(`rides.${rideId}`, '\\ride-accepted', (res: { ride: Ride
}) => {
    this.toastService.success('Vozač je krenuo ka Vama!');
    this.ride.set(res.ride);
  });
}

```

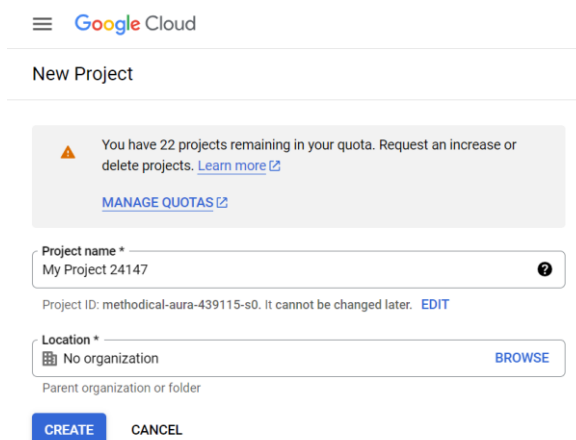
### *Пример употребе echo сервиса за комуникацију WebSocket-има*

#### 5.10.1. Приказ мапе

Иако је главна тема овог рада рад са релационим базама података, сама имплементација практичног дела дипломског рада базира се око приказа мапе на којој је могуће видети своју локацију, унесене почетне и одредишне тачке, локацију возача и навигацију кретања. Оваква функционалност омогућена је коришћењем *Google Maps JavaScript API* сервиса.

Процес прибављања овог сервиса је персонализован и мора га сваки власник апликације радити. Потребно је генерисати *Google Cloud* пројекат на *Cloud Console*-и, односно попунити основне податке о пројекту и додати могућност наплате. *Google*

омогућује одређени број тестних захтева, односно у случају прекорачења долази до наплате по сваком захтеву за лоцирање, навигирање, итд. Омогућено је додавање различитих *API*-а који би прошири могућност употребе апликације. По завршетку конфигурације, битно је прибавити *Maps API Key* јер је њега потребно укључити у захтев за учитавање сервиса приликом учитавања апликације, како би се омогућила употреба *Google* сервиса.



Google Cloud

New Project

You have 22 projects remaining in your quota. Request an increase or delete projects. [Learn more](#)

[MANAGE QUOTAS](#)

Project name \*  
My Project 24147

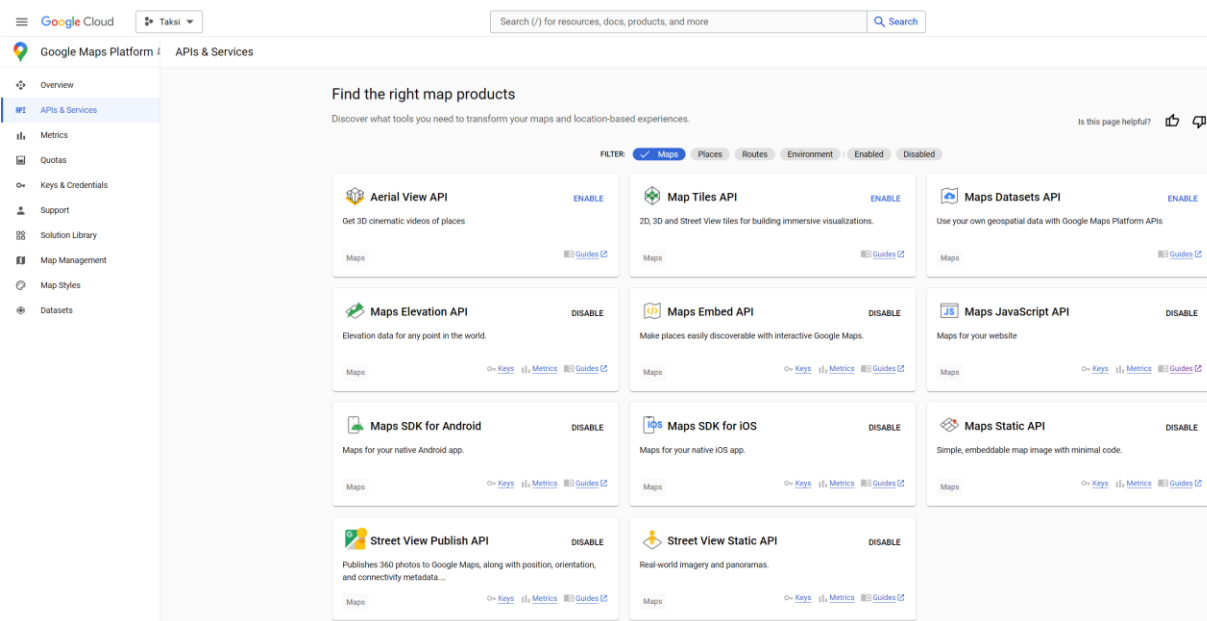
Project ID: methodical-aura-439115-s0. It cannot be changed later. [EDIT](#)

Location \*  
No organization [BROWSE](#)

Parent organization or folder

[CREATE](#) [CANCEL](#)

Слика 10 – креирање Google Cloud пројекта [32]



Google Cloud

Taksi

Search (/) for resources, docs, products, and more

Google Maps Platform APIs & Services

Overview

APIs & Services

Metrics

Quotas

Keys & Credentials

Support

Solution Library

Map Management

Map Styles

Datasets

Find the right map products

Discover what tools you need to transform your maps and location-based experiences.

Is this page helpful?

FILTER: Maps Places Routes Environment Enabled Disabled

**Aerial View API** ENABLE  
Get 3D cinematic videos of places  
Maps [Guides](#)

**Map Tiles API** ENABLE  
2D, 3D and Street View tiles for building immersive visualizations.  
Maps [Guides](#)

**Maps Datasets API** ENABLE  
Use your own geospatial data with Google Maps Platform APIs  
Maps [Guides](#)

**Maps Elevation API** DISABLE  
Elevation data for any point in the world.  
Maps [Keys](#) [Metrics](#) [Guides](#)

**Maps Embed API** DISABLE  
Make places easily discoverable with interactive Google Maps.  
Maps [Keys](#) [Metrics](#) [Guides](#)

**Maps JavaScript API** DISABLE  
Maps for your website  
Maps [Keys](#) [Metrics](#) [Guides](#)

**Maps SDK for Android** DISABLE  
Maps for your native Android app.  
Maps [Keys](#) [Metrics](#) [Guides](#)

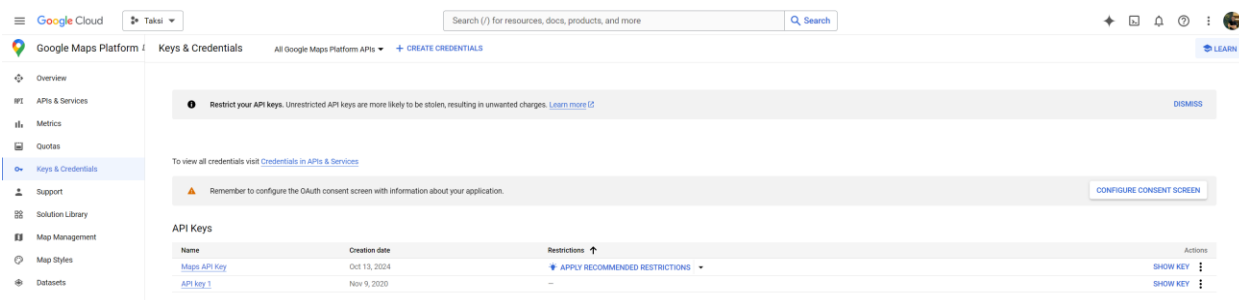
**Maps SDK for iOS** DISABLE  
Maps for your native iOS app.  
Maps [Keys](#) [Metrics](#) [Guides](#)

**Maps Static API** DISABLE  
Simple, embeddable map image with minimal code.  
Maps [Keys](#) [Metrics](#) [Guides](#)

**Street View Publish API** DISABLE  
Publishes 360 photos to Google Maps, along with position, orientation, and connectivity metadata...  
Maps [Keys](#) [Metrics](#) [Guides](#)

**Street View Static API** DISABLE  
Real-world imagery and panoramas.  
Maps [Keys](#) [Metrics](#) [Guides](#)

Слика 11 – Додавање API-а [32]



Слика 12 – Прибављање конекционих кључева [32]

Након успешног подешавања, приступа се употреби *Google* сервиса. У овом раду, извршен је приказ мапе са свим жељеним ознакама и коришћене су методе:

- проналажења назива адресе на основу пружених координата,
- налажења координата на основу пружене адресе,
- налажења путање кретања на основу прослеђених координата почетка, путних стајалишта и одредишта.

```
<google-map height="65vh" width="100%" [options]="{
  mapId: 'e547d2444e8aaf6f',
  mapTypeId: 'roadmap',
  zoomControl: true,
  scrollwheel: true,
  gestureHandling: 'cooperative',
  zoom: 15,
  maxZoom: 30,
  minZoom: 12,
  clickableIcons: false,
  streetViewControl: false,
}" (mapClick)="handlePickingLocation($event)"
>
  @if (myLocation && !directionsResult) {
    <map-advanced-marker
      [position]="{ lat: myLocation.lat, lng: myLocation.lng }"
      [content]="icons()[0] || null"
    />
  }
  @if (driverLocation()) {
    <map-advanced-marker
      [position]="{ lat: driverLocation().lat, lng: driverLocation().lng }"
      [content]="icons()[3] || null"
    />
  } ... // implementation of other markers
  @if (directionsResult) {
    <map-directions-renderer [directions]="directionsResult" />
  }
</google-map>
```

Пример имплементације приказа мапе



```

findLatLng(position: string, address: string): void {
    const bounds = new google.maps.LatLngBounds(
        new google.maps.LatLng(this.cityCenter().lat - 0.1, this.cityCenter().lng -
0.1),
        new google.maps.LatLng(this.cityCenter().lat + 0.1, this.cityCenter().lng +
0.1));
    this.geocoder
        .geocode({ address, bounds })
        .then((result) => {
            this.updatePosition(
                result.results[0].geometry.location.lat(),
                result.results[0].geometry.location.lng(),
                result.results[0].formatted_address,
                position
            );
        })
        .catch((error) => {
            console.error(error);
            this.toastService.error('Nismo pronašli željeno mesto na Google Mapi');
        });
}

```

*Пример употребе google сервиса за одређивање координата адресе*

## 6. РАД АПЛИКАЦИЈЕ

У овом поглављу биће приказан опис и рад веб апликације „ЕлФак такси“, која је осмишљена са идејом унапређења и олакшања скоро свакодневних активности.

### 6.2. Опис апликације

Ова апликација омогућава муштеријама једноставно и брзо наручивање такси возила, самостално одређивање одредишта путовања и праћење локације возила у реалном времену. Такође, апликације је окренута и самом такси удружењу, односно возачима и менаџерима. Она возачима омогућава преглед нових захтева вожњи, одабир, односно прихватање следеће вожње и навигацију како до муштерије, тако и до одредишта. Док менаџерима пружа увид у пословање такси удружења, односно увид у највредније возаче актуелног месеца, преглед историје вожњи са претрагом и свих возача такође са претрагом.

Све групе корисника, након пријаве у систем, имају могућност прегледа својих профила и историје вожњи. Додатно, муштеријама које не желе да се региструју на систем, и даље је омогућен велики број функционалности система али без историје њихових активности.

Апликација је развијена тако да има прилагодљив веб дизајн како би се идеално приказала на паметном телефону и таблет уређају корисника. Иако је фокус употребе система базиран на идеји да сами паметни телефони поседују GPS, те им се на тај начин може одредити почетна адреса вожње, муштерији се оставља могућност измене почетне адресе у случају GPS грешке или ако жели да вожњу наручи за другу особу.

### 6.3. Профил корисника апликације

„ЕлФак такси“ је веб апликација осмишљена са идејом да обједини више такси сервиса и да кориснику олакша саму употребу система. Како су се данас сва такси удружења окренула мобилним апликацијама, овај приступ сам по себи не оптерећује корисника да било шта инсталира на свој паметни уређај. Овој веб апликацији сваки корисник ће лако приступити преко претраживача свог паметног телефона.

Профили корисника које имамо у веб апликацији:

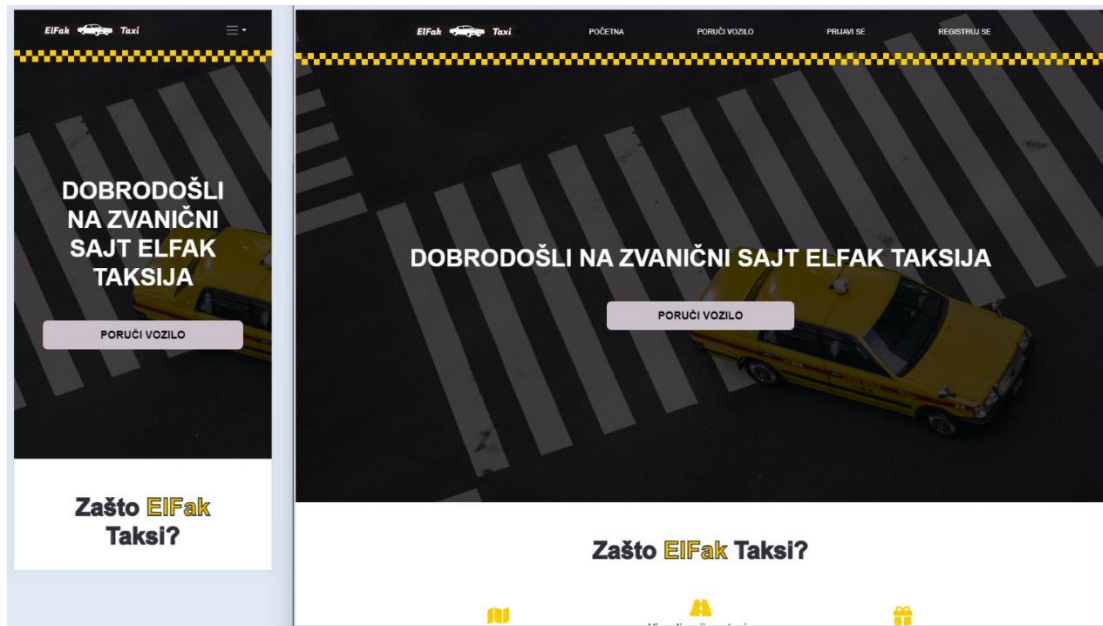
- Муштерија - Свака особа која има приступ интернету и потребу да се брзо транспортује до другог дела града. Овакви корисници имају могућност прегледа почетне стране, поручивање вожње, пријаву односно регистрацију на систем, могућност прегледа, измене и брисање профила, као и праћења историје вожњи.
- Возачи - Сами такси возачи, односно такси удружења, идеални су корисници ове веб апликације јер ће им било који паметни уређај који већ поседују бити довољан за пословање. Пријавом на систем добијају специјални преглед мапе на којој могу прегледати будуће вожње и управљати одредиштем, почетком и крајем исте.
- Менаџери – Запослени у такси удружењу, задужени за преглед пословања, лако могу остварити овакав увид прегледом веб апликације на било ком уређају јер је развијен приказ за паметне телефоне, таблете и рачунаре. Пријавом на систем добијају специјални табеларни преглед пословања такси удружења.

### 6.4. Случајеви коришћења

Имплементирани случајеви коришћења веб апликације:

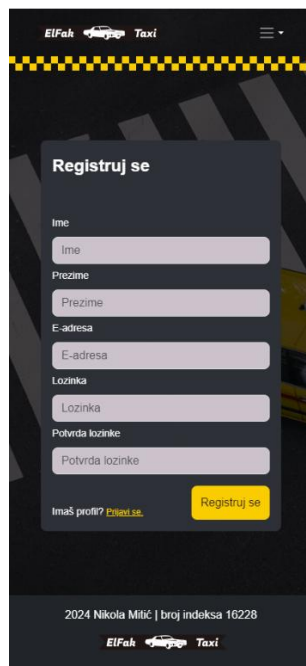
- Приказ информација о веб апликацији  
Опис: Приказ информација о веб апликацији којим се кориснику у кратким цртама јасно објашњавају све предности овог система  
Актери: Било који посетилац веб апликације  
Предуслов: Улаз на веб апликацију било којим претраживачем  
Основни ток:
  - Корисник приступа веб апликацији
  - Приказује се почетна страница са основним информацијама

- Корисник види могућност за даљу навигацију
- Изузеци: /  
Последице: /



Слика 13 – Почетна страница

- Креирање корисничког профила муштерије  
 Опис: Приказ регистрационе странице и креирање корисничког профила муштерије  
 Актери: Муштерија  
 Предуслов: Корисник није већ пријављен на систем и нема профил са истом е-адресом  
 Основни ток:
  - Кликом на навигационо дугме „REGISTUJ SE“ врши се навигација до регистрационе форме
  - Унос података
  - Кликом на потврдно дугме „Registruj se“ врши се провера валидности унесених података
  - У случају исправности података, исти се прослеђују на *backend* и креира се нови профил памћењем података у бази податак
  - У случају невалидних података, приказаше се искачући прозор са садржајем обавештења о грешци приликом попуњавања форме
 Изузеци: Невалидан унос неопходних података  
 Последице: Нови корисник, односно муштерија, пријављен је на систем и приказана му је dashboard страница те може поручити такси возило



Слика 14 – Регистрациона страница

- Пријављивање корисника на систем

Опис: Приказ странице за пријављивање на систем

Актери: Муштерија, возач или менаџер

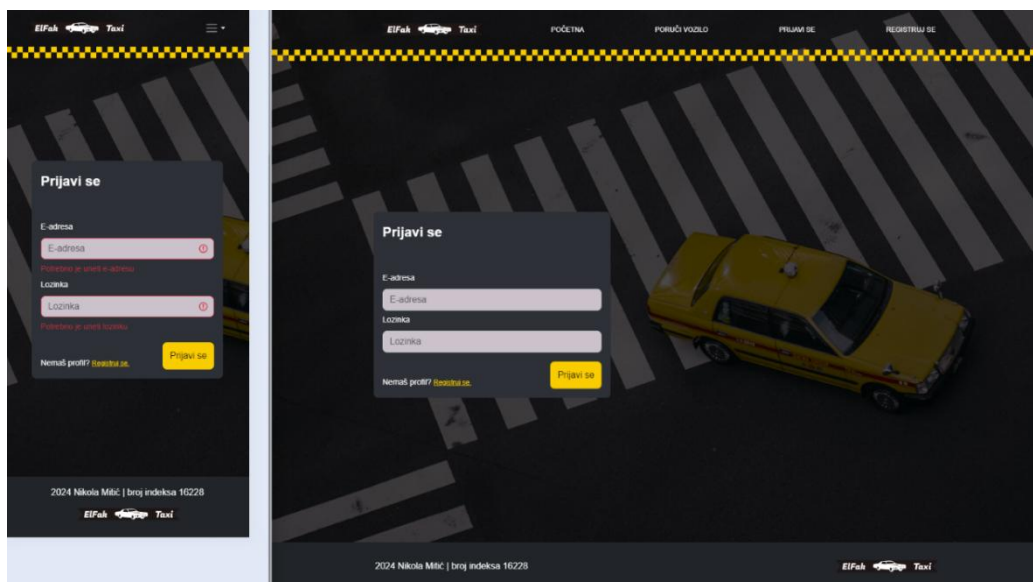
Предуслов: Корисник није већ пријављен на систем и има креиран профил

Основни ток:

- Кликом на навигационо дугме „PRIJAVI SE“ врши се навигација до форме за пријаву на систем
- Унос података
- Кликом на потврдно дугме „Prijavi se“ врши се провера валидности унесених података
- У случају исправности података, исти се прослеђују на *backend* и креира се нови аутентикациони токен
- У случају невалидних података, приказаће се искачући прозор са садржајем обавештења о грешци приликом попуњавања форме

Изузеци: Невалидан унос неопходних података

Последице: Корисник је пријављен на систем и приказана му је dashboard страница, а у зависности од типа корисник може поручити односно прихватити или прегледати вожњу



Слика 15 – Страница пријаве на систем

- **Поручивање возила**

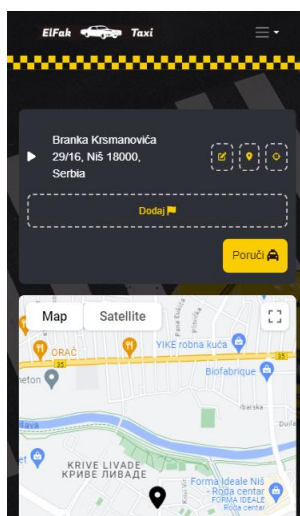
Опис: Уношење почетне и одредишне адресе и поручивање вожње

Актери: Муштерија, са или без корисничког профила

Предуслов: Корисник не сме да буде пријављен као возач или менаџер

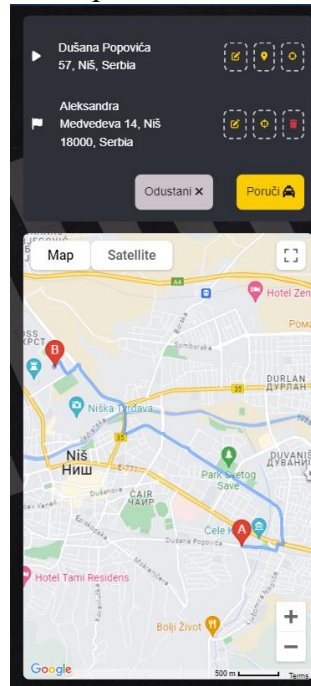
Основни ток:

- Кликом на навигационо дугме „PORUČI VOZILO“ врши се навигација до странице са формом за унос почетне и крајње адресе и приказом мапе града. Муштерија ће иницијално видети своју локацију.
- Унос почетне адресе могућ је уз помоћ GPS лоцирања, отварањем модала за одабир адресе или додиром мапе како би се одабрала тачка
- Унос одредишне адресе могућ је отварањем модала за одабир адресе или додиром мапе како би се одабрала тачка



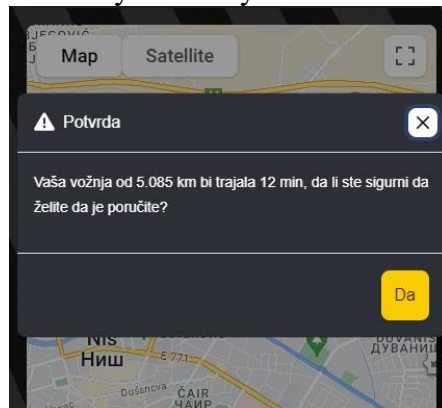
Слика 16 – Страница за поручивање возила

- Након уноса обе адресе, на мапи се илуструје путања



Слика 17 – Приказ илустрације путовања на мапи

- У случају клика на дугме „Odustani“ одустаје се од захтева или у случају клика на дугме „Poruči“ отвара се дијалог са детаљним информацијама о вожњи који је потребно прихватити кликом на дугме „Да“ како би се у систем уписала захтевана вожња



Слика 18 – Приказ потврдног модала

- Систем захтева локацију свих активних возача и одређује најближег те њему првом прослеђује захтев за прихватање вожње
- У случају да најближи возач не прихвати вожњу у року од 60 секунди, захтев се прослеђује свим возачима

Изузеци: /

Последице: Муштерија је креирала захтев за вожњу, исти је забележен у бази података и прослеђен на прихватање

- Отказивање вожње

Опис: У случају да се муштерија предомислила постоји опција отказивања поручене вожње пре него што возач дође по њу

Актери: Муштерија, са или без корисничког профила

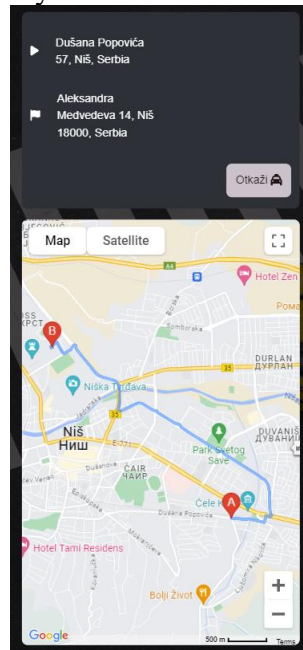
Предуслов: Вожња још увек није започета

Основи ток:

- Кликом на дугме „Откажи“ врши се брисање захтеване вожње у бази података. Уколико нико до тада није прихватио вожњу, овај захтев ће само нестати из листе понуђених возачима. Уколико је неко од возача прихватио вожњу и кренуо ка муштерији, навигација ће се прекинути и добиће информацију о отказивању. Муштерији се приказује почетни подrazумевајући приказ ове странице

Изузеци: Возач је купио корисника и вожња је већ почела

Последице: Вожња се уклања из система и базе података



*Слика 19 – Приказ могућности отказивања вожње*

- Преглед и управљање вожњом

Опис: Возач има могућност прегледа путање вожње, прихватања, мењања одредишта и означавања почетног и крајњег тренутка вожње

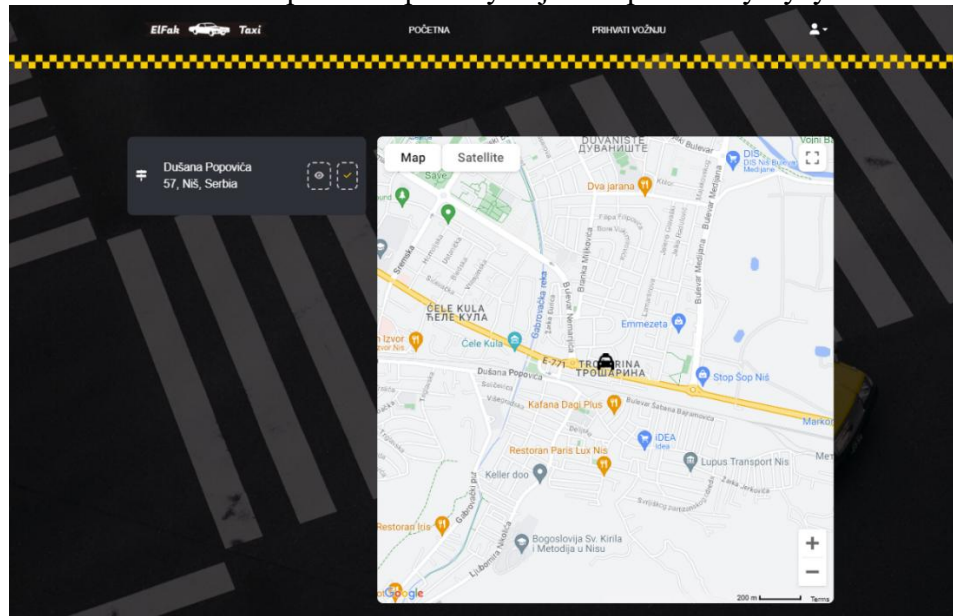
Актери: Возачи

Предуслов: Возач мора да буде пријављен на систем и у систему постоје захтеване вожње

Основни ток:

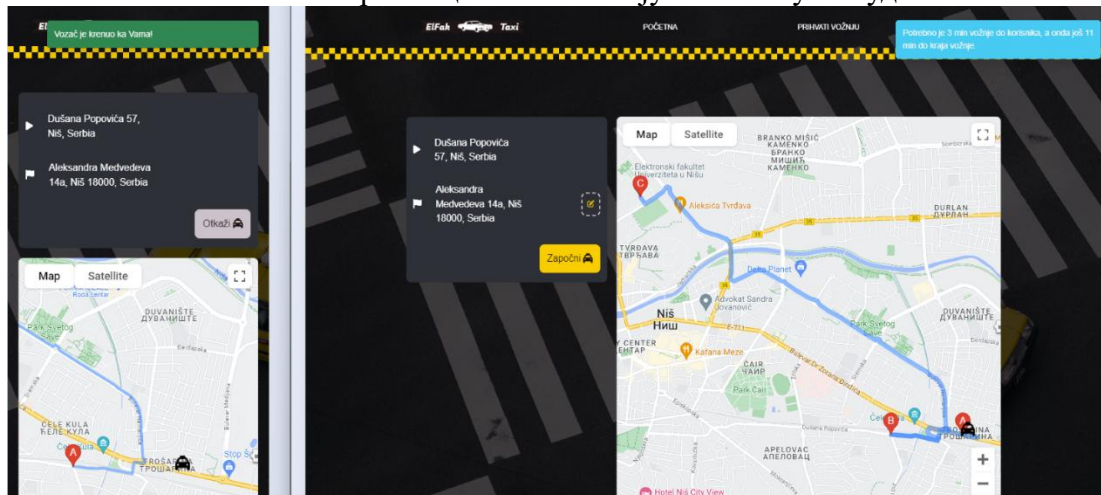


- Корисник у листи доступних вожњи кликом на дугме са иконицом око добија преглед вожње на мапи, а у искачућем прозорчету информације о километражи и времену које ће провести у путу



Слика 20 – Страница за прихватање вожњи

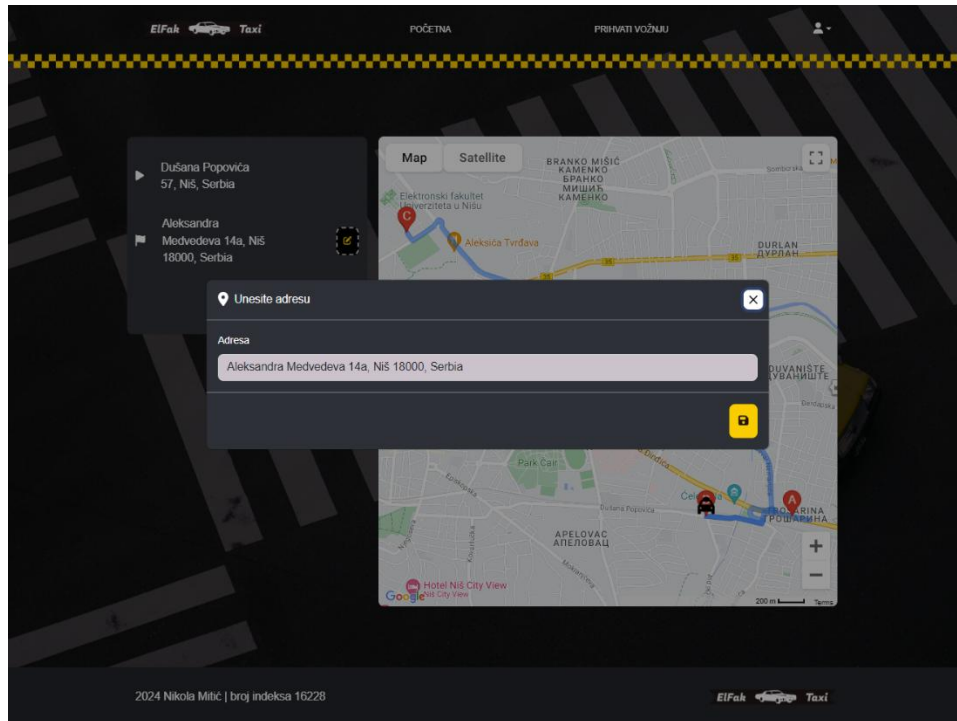
- Корисник у листи доступних вожњи кликом на дугме штикла прихвата вожњу, односно систем бележи идентификациону ознаку возача у бази података. Уколико је возач био најближи полазној тачки, добио је предност од 60 секунди за прихватање вожње, уколико је ово време протекло сви возачи су добили могућност прихватања ове вожње, односно ствара се тркачка ситуација у којој захтевану вожњу преузима возач који је најбрже прихвати кликом на дугме са иконицом штикла. Остали корисници возачи остају без исте у понуди захтеваних вожњи



Слика 21 – Приказ прегледа када возач крене по муштерију

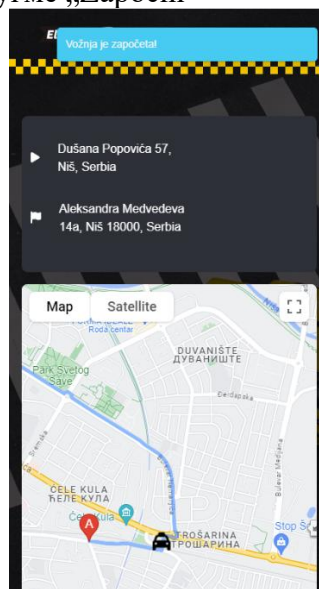


- Након што је прихватио вожњу корисник може да измени унето одредиште или унесе исто у случају да оно не постоји у систему



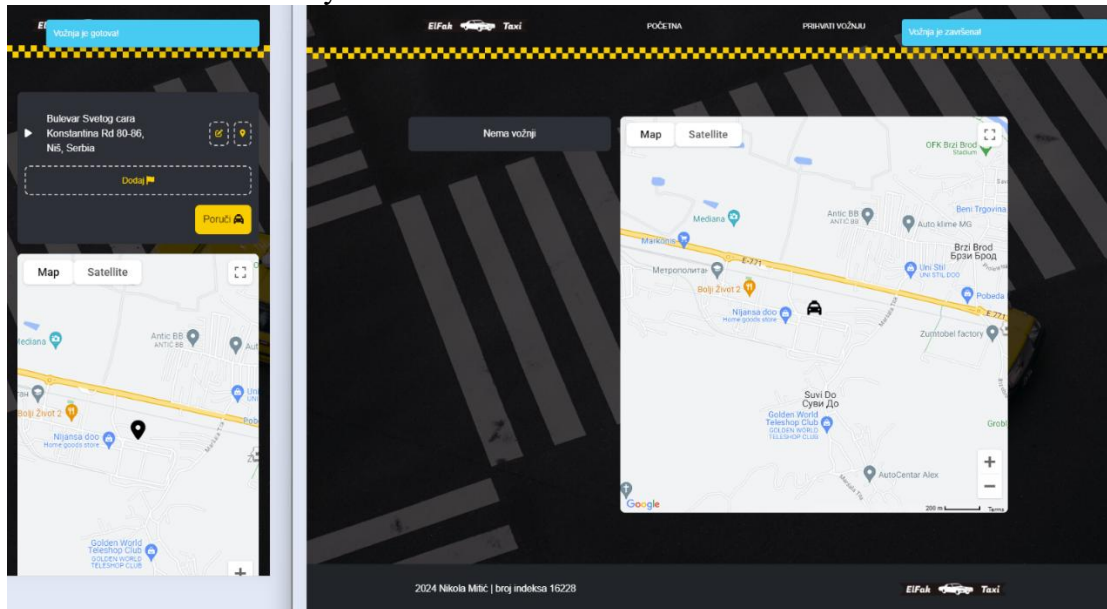
Слика 22 – Приказ корекције завршне адресе вожње

- Након доласка до муштерије возач бележи време почетка вожње кликом на дугме „Зарошпи“



Слика 23 – Приказ прегледа муштерије када је вожња започета

- Након доласка до одредишта или раније, изузетно у случају договора са муштеријом, возач кликом на дугме „Završi“ бележи време краја вожње у систем



Слика 24 – Финални приказ завршетка вожње

Изузеци: Нема доступних вожњи или је муштерија отказала вожњу у току пута до исте

Последице: Вожња у бази података добија све преостале податке

- **Преглед пословања**

Опис: Пријављени корисник има могућност прегледа свих битних података о пословању такси удружења забележених у систем

Актери: Менаџер

Предуслов: Корисник мора бити пријављен на систем

Основни ток:

- Кликком на навигационо дугме „PREGLED“ отвара се приказ прегледа података о пословању
- Корисник прво може видети листу највреднијих возача овог месеца, са бројем остварених вожњи
- Даље, приказује се историја свих вожњи. Табела приказа садржи информације о муштерији, возачу, одредиштима и времену краја вожње. Корисник податке може претраживати уносом жељеног текста за претрагу у поље за претрагу, може одабрати да ли жели да види и вожње на чекању, као и вожње у току
- Приказује се и списак свих возача који постоје у систему. Табела прилаза садржи основне и контакт информације о возачу, укупан број остварених вожњи и број таблица возила које је тренутно у употреби. Корисник податке може претражити уносом жељеног текста за претрагу у поље за претрагу и одабиром приказа само возача у смени или свих возача.

Radnici sa najvećim brojem vožnji ovog meseca:  
• Peter Petrović [8]

**Vožnje**

Pretraga:  Adresa ☐ Zadržavanje ☐ U toku

Vozač	Korisnik	Početna adresa	Završna adresa	Kraj vožnje
Petar Petrović	Nikola Nikolić	Vojvode Mišića 71, Niš 18000, Serbia	Dušanova 7, Niš, Serbia	10/26/24, 1:19 PM
Petar Petrović	N.N.	Višegradska 33, Niš 18000, Serbia	Dimirija Tucovića, Niš, Serbia	10/29/24, 3:01 AM
Petar Petrović	N.N.	Višegradska 33, Niš 18000, Serbia	Dimirija Tucovića, Niš, Serbia	10/29/24, 3:22 AM
Petar Petrović	N.N.	Višegradska 33, Niš 18000, Serbia	Stanoja Bunuševića, Niš, Serbia	10/29/24, 3:30 AM
Petar Petrović	N.N.	Višegradska 33, Niš 18000, Serbia	Dimirija Tucovića, Niš, Serbia	10/29/24, 3:38 AM
Petar Petrović	N.N.	Branka Krstanovića 29/16, Niš 18000, Serbia	Dimirija Tucovića, Niš, Serbia	10/30/24, 4:33 PM
Petar Petrović	N.N.	Branka Krstanovića 29/16, Niš 18000, Serbia	Dimirija Tucovića, Niš, Serbia	10/30/24, 5:17 PM
Milan Milanković	N.N.	Branka Krstanovića 29/16, Niš 18000, Serbia	Dušanova, Niš, Serbia	10/30/24, 2:27 PM

10 vožnji po stranici

**Vozači**

Pretraga:  Samo u sklopu ☒ Sve Proizvode

Ime	Prezime	E-adresa	Telefonski broj	Broj vožnji	Tablice	Kategorija dozvole	Broj dozvole
Milan	Milanković	vozac2@taxi.rs	+381657830027, +381637739288	3	FC520AV	B	111671
Jovana	Jovanović	vozac3@taxi.rs	-	0	MF710NB	B	883529

Слика 25 – Приказ детаља пословања

Изузеци: /

Последице: Корисник је видео податке такси удружења

- Преглед корисничког профила

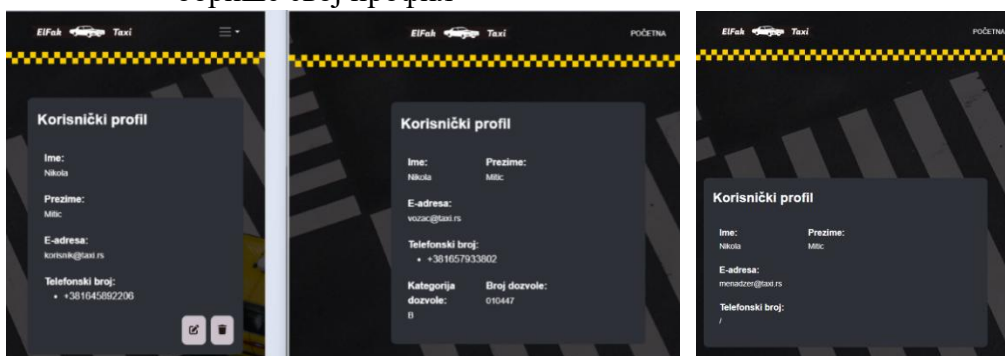
Опис: Пријављени корисник има могућност прегледа својих података забележених у систем

Актери: Муштерија, возач или менаџер

Предуслов: Корисник мора бити пријављен на систем

Основни ток:

- Кликот на навигационо дугме „PPROFIL“ отвара се приказ података о пријављеном кориснику
- У случају муштерије, приказује се и навигационо дугме са иконицом оловчице које води до странице са формом за измену података и дугме са иконицом канта за ђубре којом корисник може да бесповратно обрише свој профил



Слика 26 – Приказ детаља профила муштерије, возача и менаџера

- У случају муштерије и возача, приказује се историја такси вожњи. Гледано из угла корисника муштерије, овај део странице садржи информацију у томе која је ово вожња по реду, јер свака регистрована муштерија има право на сваку десету бесплатну вожњу, те је може од возача и захтевати. У случају да је у вожњи партиципирао непријављени корисник као муштерија, у систему неће постојати подаци о њему.

Korisnik	Početna adresa	Završna adresa	Kraj vožnje
N N	Dušana Popovića 57, Niš, Serbia	Aleksandra Medvedeva 14a, Niš 18000, Serbia	10/12/24, 3:44 AM
Nikola Milić	Jeremije Žvanovića 20, Niš, Serbia	Aleksandra Medvedeva 14a, Niš 18000, Serbia	10/12/24, 3:52 AM

Слика 27 – Приказ историја вожњи муштерије и возача

- У случају возача, приказује се и историју управљања возилима. Ова табела пружа детаљнији опис управљаним возилима, као и период управљања њима.

Tip	Brend	Model	Godina proizvodnje	Registracija	Upravlja od	Upravlja do
Privatno	Peugeot	Golf 5	1986	03-12	6/19/10	6/25/10
Privatno	Peugeot	Golf 5	1986	03-12	6/5/10	6/9/10
Privatno	Peugeot	Golf 5	1986	03-12	7/1/24	U toku

Слика 28 – Приказ историје управљања возилима возача

Изузеци: /

Последице: Корисник је видео своје податке, односно муштерија је обрисала свој профил

- Измена корисничког профила

Опис: Измена података за пријаву на систем или личних података

Актери: Муштерија

Предуслов: Корисник мора бити пријављен на систем

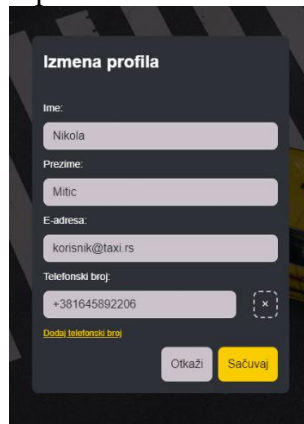
Основни ток:

- Кликом на навигационо дугме са иконицом оловчице отвара се страница са формом попуњеном свим подацима корисника
- Корисник може изменити своје податке и додати нове, изменити или обрисати старе бројеве телефона

- Кликом на потврдно дугме врши се провера валидности унесених података
- У случају исправности података, исти се прослеђују на *backend* и врши се измена података у бази
- У случају невалидних података, приказаће се искачући прозор са садржајем обавештења о грешци приликом попуњавања форме

Изузеци: /

Последице: Подаци корисника измењени су у бази података



Izmena profila

Ime: Nikola

Prezime: Mitic

E-adresa: korisnik@taxi.rs

Telefonski broj: +381645892206

[Dodaj telefonski broj](#)

Otkazi Sačuvaj

Слика 29 – Страница измене профила муштерије

## 7. ЗАКЉУЧАК

Истраживањем PHP програмског језика и технологија за објектну обраду података релационе базе података, као и технологија за креирање веб апликација долазимо до сазнања о ширини и способности ових технологија. Сама имплементација истих била је изазовна, али је на крају довела до креирања система који се може широко примењивати.

Релациони модел базе податак је омогућио стварање комплексне структуре података која се објектно релационим мапером преноси на даљу употребу и обраду Laravel-ом, те се долази до функционалних решења. Употреба ове комбинације технологија омогућава одличну основу за израду разних комплексних система. Једноставност мапирања, прибављања и мутирања података Eloquent ORM-ом омогућује лак даљи развој и унапређење функционалности сваке веб апликације.

Коришћењем Angular-а долази се до употребе прилагодљивог веб дизајна. Па тако, истовременим развојем једне добијамо веб апликацију која се може користити на више различитих уређаја, притом пружајући исти кориснички интерфејс. Овакав приступ доста убрзава пут од идеје до функционалног решења.

У случају имплементираних апликације која је демонстрирала употребу изучених технологија и са жељом развоја система на веће нивое, могуће је ову веб апликацију додатно проширити додавањем нових функционалности попут имплементације система процене цене вожње и додавањем онлајн плаћања, додавањем међу станица у вожњи и праћења броја активних возача. Овакав систем, верујем да би у многоме додатно унапредио јавни превоз.

Описане технологије омогућиле су креирање апликација која је пренела је у реалност идеју о аутоматизацији управљања такси службом без потребе постојања диспечерског центра. Омогућила је једноставну употребу комплексним системом кроз пар клика у веб претраживачу. Решила је проблем навигације у великим гужвама и дала сигурност муштеријама праћењем кретања возила.

## 8. ЛИТЕРАТУРА

- [1] „What is PHP? The PHP Programming Language Meaning Explained,“ [На мрежи]. Available: <https://www.freecodecamp.org/news/what-is-php-the-php-programming-language-meaning-explained/>. [Последњи приступ 15 Децембар 2024].
- [2] „What Is Symfony?,“ [На мрежи]. Available: <https://builtin.com/software-engineering-perspectives/symfony>. [Последњи приступ 15 Децембар 2024].
- [3] „Welcome to CodeIgniter4,“ [На мрежи]. Available: [https://www.codeigniter.com/user\\_guide/intro/index.html](https://www.codeigniter.com/user_guide/intro/index.html). [Последњи приступ 15 Децембар 2024].
- [4] „PHP - Introduction,“ [На мрежи]. Available: [https://www.tutorialspoint.com/php/php\\_introduction.htm](https://www.tutorialspoint.com/php/php_introduction.htm). [Последњи приступ 15 Децембар 2024].
- [5] „PHP - Features,“ [На мрежи]. Available: [https://www.tutorialspoint.com/php/php\\_features.htm](https://www.tutorialspoint.com/php/php_features.htm). [Последњи приступ 15 Децембар 2024].
- [6] „Google Cloud,“ [На мрежи]. Available: <https://cloud.google.com/learn/what-is-a-relational-database?hl=en>. [Последњи приступ 14 Октобар 2024].
- [7] „What is an ORM – The Meaning of Object Relational Mapping Database Tools,“ [На мрежи]. Available: <https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/>. [Последњи приступ 15 Децембар 2024].
- [8] „What is Laravel? Explain it like I’m five,“ [На мрежи]. Available: <https://runcloud.io/blog/what-is-laravel>. [Последњи приступ 15 Октобар 2024].
- [9] „<https://ngonyoku.medium.com/understanding-laravel-artisan-c834aae215fb>,“ [На мрежи]. Available: <https://ngonyoku.medium.com/understanding-laravel-artisan-c834aae215fb>. [Последњи приступ 17 Децембар 2024].
- [10] „Laravel Routing Guide – How Create Route to Call a View,“ [На мрежи]. Available: <https://www.cloudways.com/blog/routing-in-laravel/>. [Последњи приступ 15 Децембар 2024].
- [11] „Route Model Binding,“ [На мрежи]. Available: <https://laravel.com/docs/11.x/routing#route-model-binding>. [Последњи приступ 15 Децембар 2024].
- [12] S. McCool, „Validation,“ у *Laravel Starter*, Birmingham, Packt Publishing Ltd., 2012, pp. 41 - 43.
- [13] „Data Validation in Laravel: Convenient and Powerful,“ [На мрежи]. Available: <https://kinsta.com/blog/laravel-validation/>. [Последњи приступ 15 Децембар 2024].



- [14] „What is Laravel? A Beginner's Introduction,“ [На мрежи]. Available: <https://www.fastfwd.com/what-is-laravel/>. [Последњи приступ 15 Октобар 2024].
- [15] „SQL vs NoSQL,“ [На мрежи]. Available: <https://www.mcloud.rs/blog/sql-vs-nosql/>. [Последњи приступ 1 новембар 2024].
- [16] „How to Use Redis with Laravel: A Comprehensive Guide,“ [На мрежи]. Available: <https://medium.com/@mohammad.roshandelpoor/how-to-use-redis-with-laravel-a-comprehensive-guide-247cfcac0a68>. [Последњи приступ 1 новембар 2024].
- [17] „Laravel Homestead,“ [На мрежи]. Available: <https://laravel.com/docs/11.x/homestead>. [Последњи приступ 15 Октобар 2024].
- [18] „A Practical Introduction to Laravel Eloquent ORM,“ [На мрежи]. Available: <https://www.digitalocean.com/community/tutorial-series/a-practical-introduction-to-laravel-eloquent-orm#how-to-create-a-one-to-many-relationship-in-laravel-eloquent>. [Последњи приступ 15 Октобар 2024].
- [19] M. Shukla, „Eloquent ORM in Laravel: Simplifying Database Interactions,“ [На мрежи]. Available: <https://manoj-shu100.medium.com/eloquent-orm-in-laravel-simplifying-database-interactions-b0269942f190>. [Последњи приступ 11 Јануар 2025].
- [20] I. Jound и H. Halimi, „Comparison of performance between Raw SQL and Eloquent ORM in Laravel,“ Faculty of Computing Blekinge Institute of Technology, Karlskrona.
- [21] „Eloquent ORM,“ [На мрежи]. Available: <https://laravel.com/docs/11.x/eloquent>. [Последњи приступ 15 Октобар 2024].
- [22] „Eloquent: Relationships,“ [На мрежи]. Available: <https://laravel.com/docs/11.x/eloquent-relationships>. [Последњи приступ 11 Јануар 2025].
- [23] „Mastering Route Model Binding in Laravel: A Comprehensive Guide.,“ [На мрежи]. Available: <https://medium.com/@moumenalisawe/mastering-route-model-binding-in-laravel-a-comprehensive-guide-d95d3e0327f2>. [Последњи приступ 31 октобар 2024].
- [24] „MySQL: Understanding What It Is and How It's Used,“ [На мрежи]. Available: <https://www.oracle.com/mysql/what-is-mysql/>. [Последњи приступ 1 новембар 2024].
- [25] „ACID Explained: Atomic, Consistent, Isolated & Durable,“ [На мрежи]. Available: <https://www.bmc.com/blogs/acid-atomic-consistent-isolated-durable/>. [Последњи приступ 1 новембар 2024].
- [26] „What are decorators in Angular?,“ [На мрежи]. Available: <https://www.geeksforgeeks.org/what-are-decorators-in-angular/>. [Последњи приступ 16 Октобар 2024].



- [27] „What is Angular: The story behind the JavaScript framework,“ [На мрежи]. Available: <https://www.altexsoft.com/blog/the-good-and-the-bad-of-angular-development/>. [Последњи приступ 16 Октобар 2024].
- [28] „Angular Signals: A Complete Guide,“ [На мрежи]. Available: <https://kurtwanger40.medium.com/angular-signals-a-complete-guide-04fa33155a46>. [Последњи приступ 16 Октобар 2024].
- [29] „SPA (Single-page application),“ [На мрежи]. Available: <https://developer.mozilla.org/en-US/docs/Glossary/SPA>. [Последњи приступ 16 Октобар 2024].
- [30] „What is web socket and how it is different from the HTTP?,“ [На мрежи]. Available: <https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/>. [Последњи приступ 16 Октобар 2024].
- [31] „JetBrains,“ [На мрежи]. Available: <https://www.jetbrains.com/webstorm/>. [Последњи приступ 12 Октобар 2024].
- [32] „Google Cloud Console,“ [На мрежи]. Available: <https://console.cloud.google.com>. [Последњи приступ 18 Октобар 2024].
- [33] „2.3. Converting ERD to a relational model,“ [На мрежи]. Available: [https://runestone.academy/ns/books/published/practical\\_db/PART2\\_DATA\\_MODELING/03-ERD-to-relational/ERD-to-relational.html](https://runestone.academy/ns/books/published/practical_db/PART2_DATA_MODELING/03-ERD-to-relational/ERD-to-relational.html). [Последњи приступ 14 Октобар 2024].
- [34] „ER diagrams vs. EER diagrams: What’s the difference?,“ [На мрежи]. Available: <https://nulab.com/learn/software-development/er-diagrams-vs-eer-diagrams-whats-the-difference/>. [Последњи приступ 14 Октобар 2024].
- [35] „Extended Entity-Relationship (EE-R) Model,“ [На мрежи]. Available: <https://www.tutorialspoint.com/extended-entity-relationship-ee-r-model>. [Последњи приступ 14 Октобар 2024].
- [36] M. J. U. H. H. J. R. Edy Budiman, „Eloquent Object Relational Mapping Models for Biodiversity Information System,“ Universitas Mulawarman, Samarinda, Indonesia, 2017.