



УНИВЕРЗИТЕТ У НИШУ  
ЕЛЕКТРОНСКИ ФАКУЛТЕТ



Веб апликација за поделу трошкова заснована на  
React-у и Node.js-у

Дипломски рад

Студијски програм: Електротехника и рачунарство

Модул: Рачунарство и информатика

Ментор:  
Доц. др Александар Миленковић

Студент:  
Вукашин Бранковић  
бр. индекса: 16011

Ниш, фебруар 2024. године

УНИВЕРЗИТЕТ У НИШУ  
ЕЛЕКТРОНСКИ ФАКУЛТЕТ

## Веб апликација за поделу трошкова заснована на React-у и Node.js-у

### A Web Application for Splitting Expenses Based on React and Node.js

Дипломски рад

Студијски програм: Електротехника и рачунарство

Модул: Рачунарство и информатика

Студент: Вукашин Бранковић, број индекса: 16011

Ментор: Доц. др Александар Миленковић

**Задатак:** Проучити технологије за развој веб апликација као што су React.js, Node.js и MongoDB. На основу изучених технологија развити апликацију за поделу трошкова која ће омогућити корисницима да међусобно деле заједничке трошкове. Такође имплементирати јаку ауторизацију и аутентификацију. Користити Google сервис за пријављивање корисника на апликацију.

Датум пријаве рада: \_\_\_\_\_

Датум предаје рада: \_\_\_\_\_

Датум одбране рада: \_\_\_\_\_

Комисија за оцену и одбрану:

---

---

---

Ниш, фебруар 2024. године

# САДРЖАЈ

1.Увод .....	7
2.MERN stack .....	8
2.1 Предња и задња страна апликације .....	9
2.2 Клијент/Сервер модел.....	10
3. React.js .....	12
3.1 Основни концепти .....	13
3.1.1 Компоненте .....	13
3.1.2 Пропс .....	13
3.1.3 JSX (JavaScript XML) .....	13
3.1.4 ESLint.....	14
3.1.5 Prettier .....	14
3.1.6 Vite .....	15
3.2 React hooks .....	15
3.2.1 useState.....	15
3.2.2 useReducer .....	16
3.2.3 useEffect.....	17
3.2.4 Custom Hooks .....	18
3.3 React Router DOM.....	18
3.3.1 useNavigate .....	19
3.4 Context API.....	20
3.4.1 useContext.....	21
3.5 React icons .....	21
3.6 Tailwind CSS .....	22
4. Node.js.....	23
4.1 Express.js.....	24
4.2 Giphy API .....	24
4.3 RESTful API .....	25
4.4 Google OAuth 2 .....	26
5. MongoDB.....	27
5.1 Mongoose .....	28
6. Имплементација апликације FairShare .....	29
6.1 Опис апликације .....	29
6.2 Случајеви коришћења.....	29
6.3 Имплементација предње стране апликације .....	34
6.3.1 Имплементација main.jsx фајла.....	37
6.3.2 Имплементација главне компоненте апликације .....	37
6.3.3 Имплементација компоненте за ауторизацију.....	38

6.3.4 Имплементација глобалног контекста.....	39
6.3.5 Имплементација компоненте за пријављивање.....	41
6.4 Имплементација задње стране апликације .....	43
6.4.1 Имплементација веб сервера.....	44
6.4.2 Имплементација рута .....	45
6.4.3 Дефинисане шеме и модели .....	47
6.4.4 Имплементација метода контролера .....	49
7. Приказ рада апликације .....	51
7.1 Почетна страница .....	51
7.2 Главна страница апликације и администрација .....	52
7.3 Опција за приказ пријатеља .....	54
7.4 Опција додај пријатеља .....	55
7.5 Опција за приказ група .....	55
7.6 Опција направи групу .....	56
7.7 Опција за приказ трошкова .....	57
7.8 Опција за додавање трошка.....	58
7.9 Опција за аналитику.....	60
7.10 Опција за приказ мапе трошкова .....	62
7.11. Интерфејс апликације на малим екранима .....	64
8. Закључак.....	71
9. Референце.....	72

# Веб апликација за поделу трошкова заснована на React-у и Node.js-у

## САЖЕТАК

У оквиру овог дипломског рада имплементирана је апликација за поделу трошкова под називом *FairShare*. Дељење заједничких трошкова понекад може бити стресно, нарочито у ситуацијама као што су путовања, концерти и други групни догађаји. Ова апликација помаже корисницима и дели трошкове уместо њих тако да се корисници могу препустити уживању у заједничким активностима са пријатељима уместо да брину о финансијским поделама.

Главни циљ овог рада је анализирати најбоље начине за поделу трошкова и развити апликацију која ће омогућити корисницима да деле заједничке трошкове са својим пријатељима. Ова апликација би значајно олакшала разним групама пријатеља њихово дељење заједничких трошкова јер би апликација аутоматски поделила трошак уместо њих. Ручно дељење заједничких трошкова може бити компликовано и одузима пуно времена и енергије, поготово када је у питању већа група пријатеља са великим бројем трошкова. Апликација врши аутоматизацију поделе заједничких трошкова, омогућујући брзу и прецизну поделу без потребе за папирима и дигитронима. Такође, она нуди поделу трошкова и у страним валутама, јер путем API-а прибавља последњи курс динара за тренутни дан и конвертује износ страних валута у динаре. Уз помоћ интерактивне мапе, корисници могу видети све локације на којима су били, забављали се и делили заједничке трошкове. На овај начин апликација омогућава боље управљање и планирање финансијама. Осим овога, сваки корисник има и увид у своја међусобна дуговања са својим пријатељима.

Апликација је урађена употребом најактуелнијих стандарда и технологија које се користе за изградњу *fullstack* веб апликација. Она се састоји од предње стране (*frontend*) и задње стране (*backend*). Предња страна је имплементирана коришћењем популарне библиотеке *React*, програмског оквира *Tailwind* и додатних библиотека као што су *React Leaflet* и *React Google Chart*. Задња страна имплементирана је коришћењем *Node.js* окружења и *Express.js* програмског оквира. Такође, коришћена је и библиотека *Mongoose* а за базу користи се *MongoDB*.

**Кључне речи:** React.js, Node.js, Express.js, MongoDB, Mongoose, TailwindCSS, React Leaflet, React Google Charts.

# A Web Application for Splitting Expenses Based on React and Node.js

## ABSTRACT

Within the scope of this thesis, an application for splitting expenses named *FairShare* was implemented. Sharing common expenses sometimes can be stressful, especially in situations such as travel, concerts and other group events. This application helps users by splitting the expenses for them, allowing them to enjoy shared activities with friends without worrying about financial divisions.

The main goal of this work is to analyze the best methods for splitting expenses and to develop an application that enables users to share common expenses with their friends. This application would significantly easier the process for many groups of friends to share their common expenses, as it would automatically divide the expenses instead of them. Manually splitting common expenses sometimes can be complicated and time and energy consuming, especially in situations when dealing with a large group of friends with many expenses. The application automates the splitting of shared expenses, enabling quick and accurate splitting without the need of using paper and calculators. Additionally, it offers splitting expenses in foreign currencies, as it gets the latest exchange rate of the dinar for the current day through an API and converts the value from foreign currencies into dinars. Also, with the help of an interactive map, users can access the locations where they have been, had fun, and shared common expenses. In this way, the application allows better financial management and planning. Additionally, each user has insight into their debts with their friends.

The application is developed according to the most current standards and technologies that are used for building fullstack web applications. Application has a frontend and a backend. The frontend is implemented using the popular React library, the Tailwind framework, and additional libraries such as React Leaflet and React Google Chart. The backend of this application is implemented using the Node.js environment and the Express.js framework. Also, the Mongoose library is used, and MongoDB serves as the database.

**Keywords:** React.js, Node.js, Express.js, MongoDB, Mongoose, TailwindCSS, React Leaflet, React Google Charts

## 1. Увод

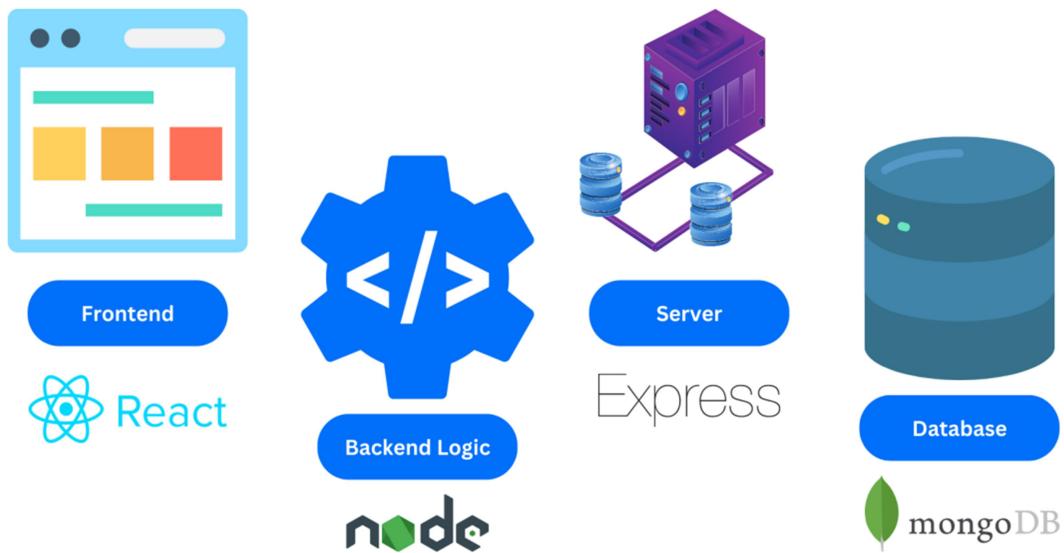
У свакодневном животу, подела заједничких трошкова је чест изазов са којим се различите групе пријатеља свакодневно суочавају. Најчешћи проблеми настају током путовања, концерата или неких других заједничких активности код којих долази до великог броја заједничких трошкова, као што су: смештај, превоз, храна, изласци, гориво, итд. Када је у питању подела заједничких трошкова, често се дешава да је тешко поделити рачун на једнаке делове и да неко увек плати више или мање од осталих. Апликација је имплементирана са намером да помогне разним групама пријатеља тако што ће поделити трошкове уместо њих и омогућити им квалитетан увид у њихове заједничке трошкове.

Овај рад је подељен у осам поглавља. Прво поглавље је уводно. У другом поглављу представљен је скуп технологија који је коришћен приликом развоја пројекта и објашњене су полазне теоријске основе везане за развој веб апликације. У трећем поглављу објашњена је библиотека *React* и њени најбитнији концепти. У четвртом поглављу објашњено је *Node.js* окружење и *Express.js* програмски оквир који су коришћени за развој серверске стране. У петом поглављу објашњена је *MongoDB* база података и *Mongoose* алат. У шестом поглављу приказана је имплементација практичног дела апликације уз разна објашњења и код. У седмом поглављу представљен је целокупан приказ корисничког интерфејса апликације. У осмом поглављу дат је закључак дипломског рада. И на крају, наведене су све референце које су коришћене у изради самог дипломског рада.

## 2.MERN stack

Веб апликације изграђене су коришћењем различитих технологија. Комбинација ових технологија назива се „*stack*“. *MERN stack* (скраћеница од *MongoDb*, *Express*, *React*, *Node*) представља скуп технологија које се користе за развој *full-stack* веб апликација. Свака од ових технологија је *open-source* и потпуно је бесплатна за коришћење [1].

Програмери уз овај скуп технологија могу развити комплетну веб апликацију коришћењем само *JavaScript* језика, што доприноси знатном олакшању развоја апликације јер нема потребе за учењем других језика осим *JavaScript-a*. *MERN* је постао јако популаран због своје брзине развоја апликације, велике скалабилности и једноставности учења [1].



Слика 2.1. Преглед *MERN* технологија [1]

**MongoDB** је бесплатна *open-source NoSQL* база података оријентисана на документе, која за разлику од стандардних *SQL* база које користе табеле, *MongoDB* користи „колекције“ за складиштење података [2].

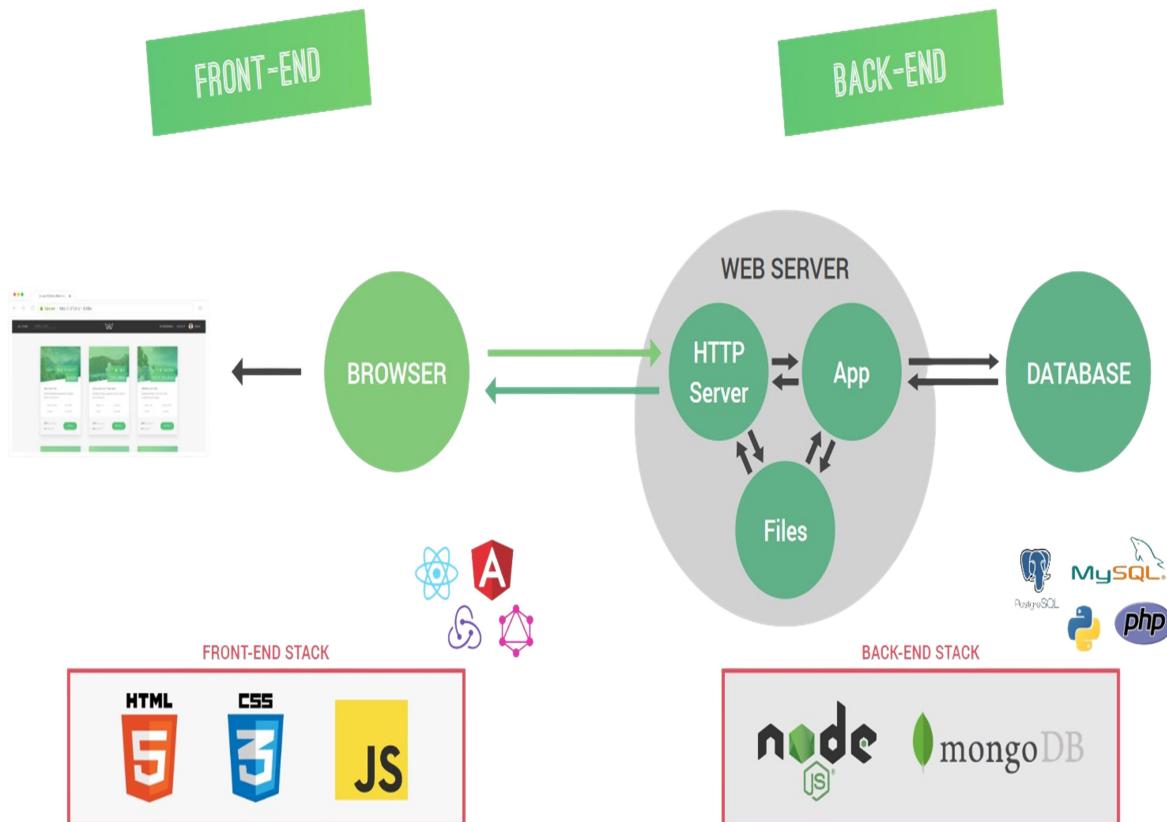
**Express.js** је јако популаран *framework* за *Node* који олакшава писање кода серверске стране и омогућава нам да дефинишемо руте, односно да дефинишемо шта сервер ради када пристигне *HTTP request* који одговара одређеној рути. Уз помоћ њега могуће је креирати *RESTfull API-e* који се касније позивају са клијентске стране [3].

**React.js** је библиотека која се користи за изградњу предње стране (*frontend*). Ова библиотека омогућава креирање динамичних веб страница и *SPA* апликација. *SPA* (скраћеница од *Single Page Application*) су апликације код којих се целокупан приказ врши на само једној *HTML* страни.

**Node.js** покреће серверску страну. То је платформа која коришћењем *Google V8 Engine* омогућава извршење *JavaScript* кода ван интернет прегледача корисника. Због тога могуће је развити серверску страну коришћењем *JavaScript* језика. Међутим креирање веб сервера коришћењем *Vanilla JavaScritip-a* је веома тежак и исцрпљујући посао. Због тога је потребан неки *framework* као што је *Express.js* [3]. У наредним поглављима, детаљно је објашњена свака од *MERN* технологија.

## 2.1 Предња и задња страна апликације

Свака веб апликација састоји се из предње стране (*frontend*) и задње стране (*backend*). Предња страна је део апликације коју корисник види, односно то је графички интерфејс путем којег корисник приступа када користи веб апликацију. За развој предње стране користе се основне технологије као што су *HTML*, *CSS*, *JavaScript* уз које је могуће користити и додатне библиотеке и програмске оквире. Код *MERN* стека, за развој предње стране користи се библиотека *React*. Корисник предњој страни приступа преко свог интернет прегледача.



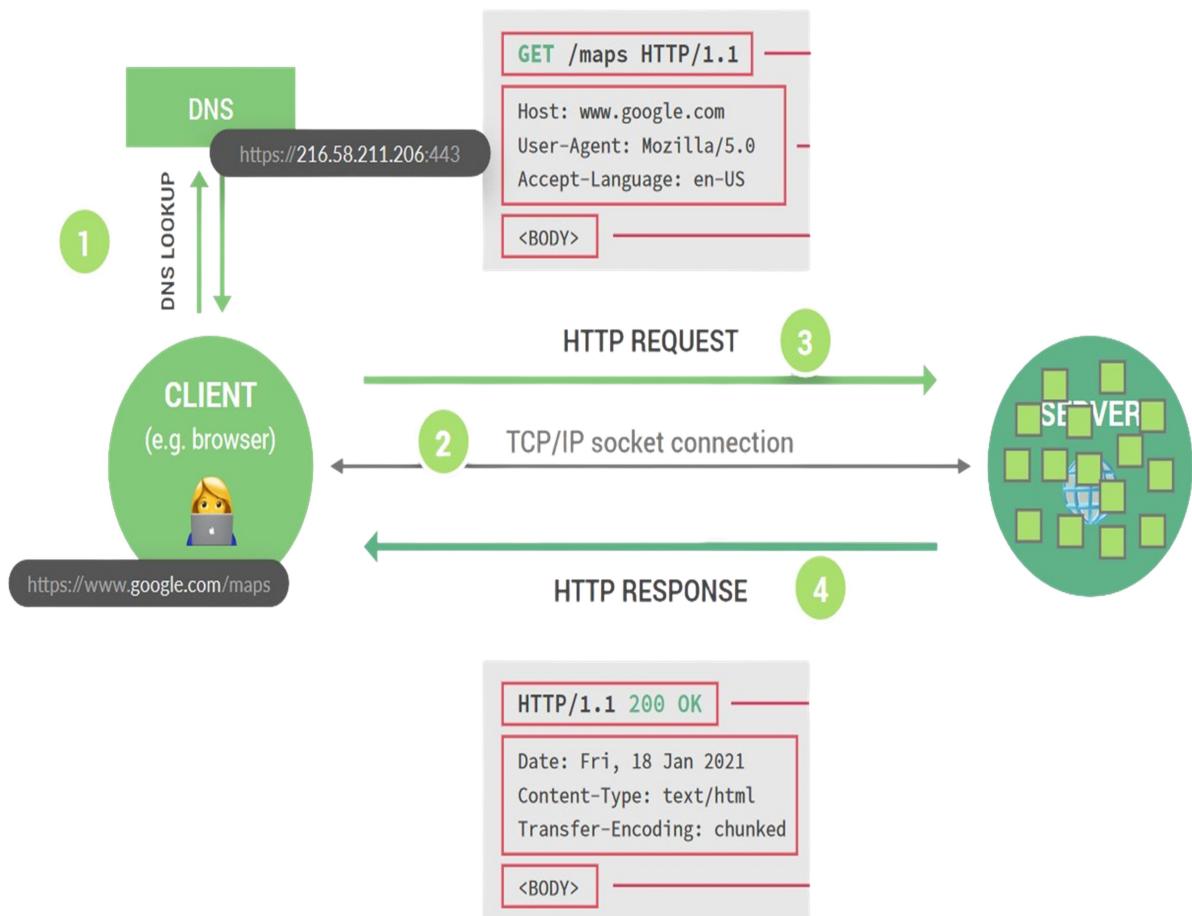
**Слика 2.1.1.** Приказ клијентске и серверске стране [4]

Задња страна (*backend*) овухвата логику која се извршава у позадини и коју корисник не може видети. Састоји се из веб сервера и базе података. Веб сервер је рачунар који је повезан на интернету. На њему памте се фајлови неопходни за рендеровање веб странице и он врши приступ бази података. Састоји се из неколико слојева. Први слој је слој *HTTP* сервера. Преко њега могуће је вршити комуникацију са клијентом путем *HTTP* протокола. Он прима захтеве клијената и на основу врсте захтева генерише одређени одговор који враћа клијенту [4].

Други слој је апликативни слој који у позадини врши све оно што је неопходно за правilan рад веб апликације. Трећи слој представља слој који је задужен за генерисање упита ка бази података [4]. За развој серверске стране могу се користити различити језици и технологије као што су: *Python/Django*, *PHP/Laravel*, *ASP.NET* и друге. За развој серверске стране у *MERN* стеку користи се *Node.js* и *Express.js*, а као база користи се *MongoDB*.

## 2.2 Клијент/Сервер модел

Клијент/Сервер модел представља интеракцију између два програма при којем један шаље захтев (*request*), а други одговара на тај захтев одређеним одговором (*response*). Програм који шаље захтев назива се клијент, а програм који одговара на захтев назива се сервер. Овај модел омогућава добар начин повезивања различитих програма који су дистрибуирани на различитим локацијама широм мреже. Целокупан веб базира се на овом моделу [5].



Слика 2.2.1. Клијент-сервер комуникација [4]

Клијент је особа или организација која преко своје машине, путем одређених захтева, приступа одређеним сервисима које сервер пружа, на пример неком веб сајту. Како би клијент приступио сајту, он уноси *URL* сајта у свој интернет прегледач. Сваки *URL* састоји се из протокола (*http* или *https*), домена и ресурса којем се приступа. Онда интернет прегледач контактира *DNS* сервер како би прибавио праву *IP* адресу сервера на основу *URL*-а којег је корисник унео [4].

Након овога успоставља се *TCP/IP* сокет конекција између клијента и сервера. Веза се успоставља *Three-Way-Handshake* процедуром. *TCP (Transmission Control Protocol)* и *IP (Internet Protocol)* су комуникациони протоколи који дефинишу начин на којем се подаци преносе кроз веб. *TCP* протокол ради тако што сваки захтев или одговор подели на велики број малих пакета. Сваки пакет када пристигне шаље *ACK* одговор пошиљаоцу, ако се не достави онда *TCP* аутоматски поново шаље тај пакет. Када сви пакети пристигују, они се спајају у почетни захтев или одговор који је послат. Овим начином остварује се брз трансфер кроз веб и смањује се настанак могућих загушења. *IP* протокол адресира и рутира ове пакете кроз веб и обезбеђује да сви пакети дођу на одредиште користећи *IP* адресу одредишта [4]. Када се успостави *TCP/IP* конекција, клијент може да пошаље *HTTP* захтев серверу. *HTTP (HyperText Transfer Protocol)* је протокол који се користи за пренос података на веб-у. Постоји и сигурна верзија овог протокола, а то је *HTTPS* протокол који енкриптује податке који се преносе. Сваки захтев садржи почетну линију, заглавље и тело захтева. Заглавље садржи додатне информације, док тело захтева обично садржи одређене податке које се прослеђују серверу и једино уз *POST* захтев тело захтева се користи. Почетна линија садржи тип *HTTP* захтева, одредишни ресурс и верзију *HTTP* протокола [4]. Најбитнији захтеви су:

- 1) ***GET***: Користи се за преузимање података са сервера, пример је захтев за веб страницом.
- 2) ***POST***: Користи се за слање података ка серверу, на пример када се попуњава и шаље одређена форма серверу.
- 3) ***PUT***: Користи се за ажурирање већ постојећих података на серверу.
- 4) ***DELETE***: Користи се за брисање података са сервера.

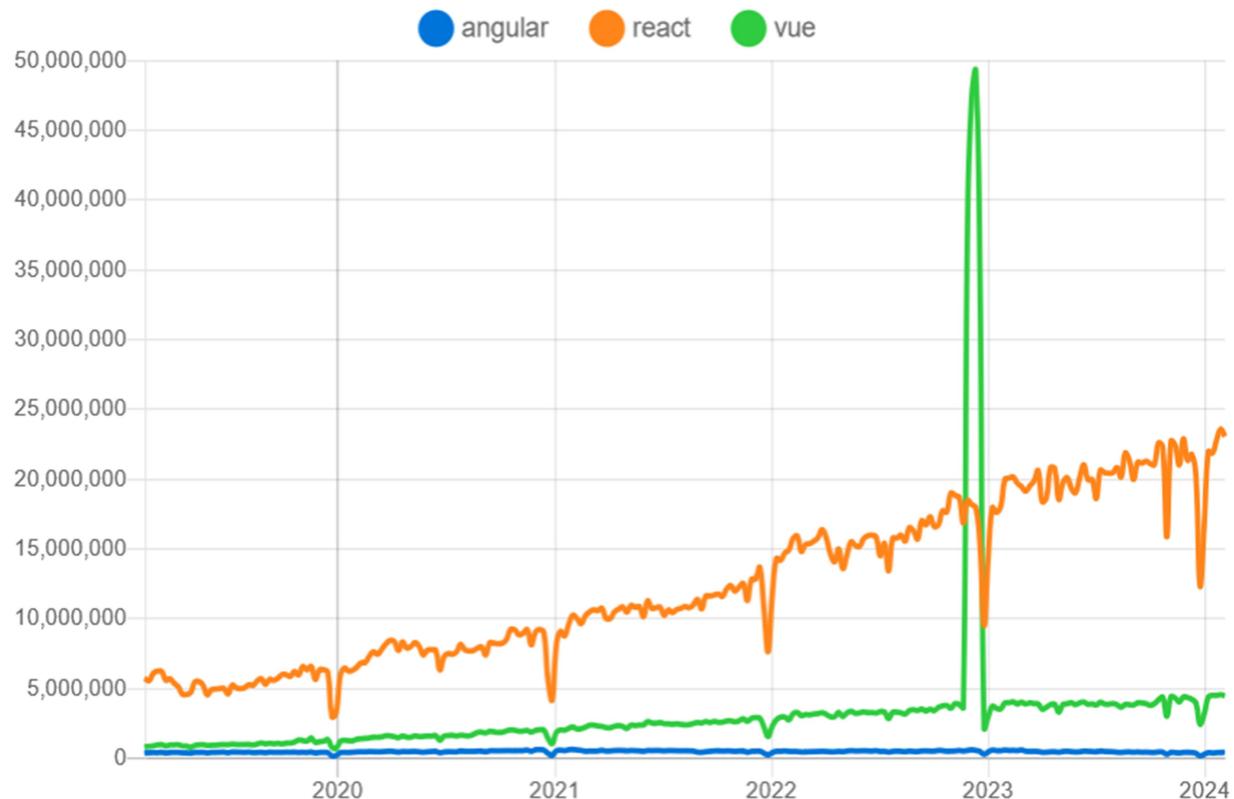
Када сервер прими захтев, он ће га обрадити и генерише *HTTP* одговор. Одговор садржи почетну линију у којем се наводи верзија *HTTP* протокола, статусни код и порука. Постоје различите врсте статусних кодова. Статусни код 200 означава да је захтев био успешан, кодови који почињу четворком означавају грешку која је настала на клијентској страни а ако почињу петицом означавају грешку која је настала на серверској страни. Одговор такође садржи заглавље и тело одговора у којем ће се налазити веб страница коју је корисник захтевао или неки други подаци у *JSON* формату [6].



Слика 2.2.2 HTTP статисни кодови [6]

### 3. React.js

*React.js* је *JavaScript* библиотека која се користи за креирање динамичних и интерактивних корисничких интерфејса. Настао је 2011. од стране Jordan Walkie-а, инжењера који је у том времену радио за *Facebook* [7]. *React* је постао доступан 2013. године и од тада је комплетно променио начин изградње корисничких интерфејса веб апликација. Користи се за изградњу *SPA* апликација. То су апликације које се састоје од само једне *HTML* стране на којој се врши рендеровање целокупног приказа. На тај начин избегава се учитавање осталих страница и тиме се стиче велика брзина приказа, као да се апликација извршава на „native“ машини. *React* је деклеративан и представља апстракцију *DOM*-а. Врши се описивање како кориснички интерфејс треба да изгледа на основу тренутних података коришћењем *JSX*-а. Није потребно директно манипулисање *DOM*-ом, већ се само описује изглед интерфејса [7]. Декларативни начин писања кода је лакши за разумевање и читање. *React* такође користи концепт виртуелног *DOM*-а, где уколико дође до промене вредности неких података, *React* ажурира само оне компоненте код којих су се десиле промене, а не цело *DOM* стабло. Такође, *React* никад не приступа *DOM*-у директно. *React* је библиотека, а не *framework*. Библиотека је само један алат, а *framework* је скуп алата. *Framework* диктира начин како апликација треба бити структурирана, док библиотека само пружа алат за изградњу, а избор је на програмеру како ће тај алат користити [8]. То је велика предност јер програмер има слободу да уз *React* формира апликацију по свом избору и да користи произвољан скуп додатних алата. Са више од 20 милиона преузимања, убедљиво је најкоришћенија и најпопуларнија *JavaScript* библиотека данас. Једна од великих предности *React*-а у односу на остале *frontend JavaScript* програмске оквире јесте постојање огромног броја додатних библиотека које је могуће укључити у пројекат.



Слика 3.1. Број преузимања прт пакета за: Angular, React и Vue током времена [9]

## 3.1 Основни концепти

У овом поглављу говори се о основним концептима у *React*-у и његовим алатима коришћеним приликом пројектовања клијентске стране веб апликације.

### 3.1.1 Компоненте

Свака *React* апликација сачињена је из одређеног броја компоненти. Компонента представља један део корисничког интерфејса. Има своје сопствене податке, логику и изглед. Оне се могу више пута користити кроз апликацију. Компоненте могу бити различитих димензија и сложености. Могу бити мале, као што су дугме, поље за унос података, навигациони бар или могу бити велике и да обухватају већи број неких мањих компоненти, као што је почетна страница веб апликације. Свака компонента представља једну *JavaScript* функцију која обавезно мора да врати *JSX*, такође функција враћа само један елемент. Комплексан кориснички интерфејс добија се комбинацијом већег броја мањих компоненти. Све компоненте заједно формирају такозвано „стабло компоненти“ [10].

### 3.1.2 Пропс

Пропс омогућава пренос података са родитељске компоненте на дете компоненту. То је веома битан алат преко којег се одређени подаци и функционалности могу преносити између компоненти. Било шта се може пренети као пропс: појединачне вредности, низови, објекти, функције, чак и друге компоненте. Пропс су вредности које се као аргумент преносе компоненти [7].

Вредност пропса не сме се мењати. Код *React*-а, пропс је могуће прослеђивати само у једном смеру, за разлику од *Angular*-а код којег може у оба смера. То значи да се подаци могу пренети једино одозго на доле кроз стабло, односно са родитељске компоненте на дете компоненту. Није могуће пренети податке у супротном смеру [8].

Наредбе као што су: *if/else*, *for*, *switch*, *while*, нису дозвољене у *JSX*-у. Такође, компонента се поново рендерује сваки пут када добије нови пропс. Обично се то дешава јер је родитељско стање ажурирано. Такође, постоји и дете пропс. То је посебан пропс који омогућава креирање компоненти које се могу користити већи број пута кроз апликацију.

### 3.1.3 JSX (JavaScript XML)

*JSX* је декларативна синтакса која комбинује *HTML*, *CSS*, и *JavaScript* и користи се за описивање изгледа и начина рада *React* компоненте. *JSX* је сличан као *HTML*, али код њега може се ући у *JavaScript* мод и писати *JavaScript* код коришћењем витичастих заграда {}. *Babel* врши конверзију *JSX*-а и *JavaScript*. Сваки *JSX* елемент конвертује се у *React.createElement* позив функције. Ова конверзија је неопходна јер интернет прегледачи не разумеју *JSX*, они само разумеју *HTML*. Позивом оне функције креирају се *HTML* елементи који се касније могу интерпретирати у интернет прегледачу [8].

*JSX* се користи како би *React*-у рекли шта се треба видети на екрану, али не и како се то остварује. *React* то сам може закључити. *JSX* је првобитно измишљен за *React*, међутим све више стиче популарност и почиње да се користи и у другим модерним библиотекама и алатима за развој веб апликација [11].

```
import PropTypes from 'prop-types';

Button.propTypes = {
  children: PropTypes.node,
  onClick: PropTypes.func,
  style: PropTypes.string,
};

export default function Button({ children, onClick, style }) {
  return (
    <button onClick={onClick} className={style}>
      {children}
    </button>
  );
}
```

*Слика 3.1.3.1.* Пример имплементације једне *React* компоненте

### 3.1.4 ESLint

*Linting* је процес у програмирању где се код анализира у потрази за потенцијалним грешкама, неправилностима и осталим нерегуларностима. То је веома важан корак у развоју веб апликација јер омогућава да се грешке открију рано и тиме повећа квалитет кода и смањи време потребно за развој апликације [11].

*Linting* такође може форсирати неке специфичне стандарде приликом писања које могу знатно побољшати читљивост и разумљивост кода и тиме учинити код лакшим за одржавање. *ESLint* је скоро па обавезан алат у изградњи веб апликација. Главна предност је та што програмер може да конфигурише *ESLint* и да изабере које нерегуларности жели, а које не да *ESLint* пријави током писања кода.

### 3.1.5 Prettier

*Prettier* је опциони сређивач кода који се може користити у развоју веб апликација. Омогућава аутоматско структуирање кода по најновијим стандардима и тиме штеди време програмерима јер не морају више водити рачуна о синтакси. *Prettier* дефинише правила по којима се код форматира и генерише јако прегледан и читљив код [7].

### 3.1.6 Vite

*Vite* (француска реч за брзо) је модерни алат за изградњу веб апликација који користи најновије технике за убрзање развоја пројекта. *Vite* обезбеђује напредне *HMR* (*Hot Module Replacement*) технике које када се деси нека промена у датотекама, брзо ажурирају само оне делове апликације код којих је дошло до промена, без освежавања целокупне странице [12].

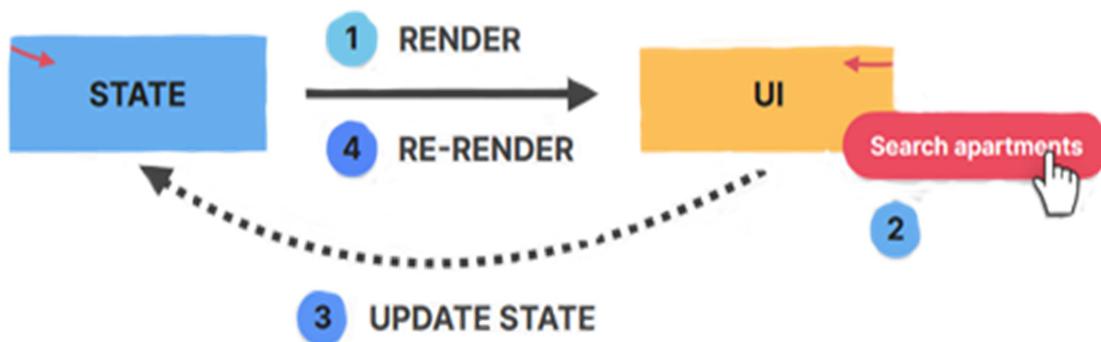
Дизајниран је да буде једноставан за употребу без потребе за компликованом конфигурацијом. *Vite* нуди подршку за разне модерне *JavaScript* програмске оквире као што су *Vue*, *Svelte*, *React*, итд. Такође има подршку и за *TypeScript* језик. Има исто и јако активну заједницу и добро написану документацију [12]. Због свих ових карактеристика постао је веома брзо један од водећих алата за развој веб апликација и њега сам користио за развој апликације за поделу трошкова.

## 3.2 React hooks

*React hooks* (куке) садрже одређену логику коју је могуће више пута користити унутар различитих компоненти. Оне додају одређену функционалност компонентама. У називу обавезно почињу са „use“. *React* долази са одређеним бројем већ имплементираних кука које је могуће користити. Такође, *React* омогућава корисницима дефинисање њихових личних кука (*custom hooks*). У следећим деловима поглавља, говори се о кукама које сам користио у изградњи веб апликације.

### 3.2.1 useState

Главни проблем са којим се *frontend* програмери сусрећу је како да приказ на корисничком интерфејсу буде синхронизован са подацима које је неопходно приказати. Преко *vanilla JavaScript*-а је веома тешко, скоро и немогуће остварити такву синхронизацију, јер је неопходно извршити пуно манипулација над *DOM*-ом. На пример, уколико желимо да променимо неку вредност на приказу, прво уз *querySelector* морамо да селектујемо елемент чију вредност желимо да променимо, па да променимо вредност и онда поново да вратимо тај елемент у *DOM* стабло. То све захтева пуно кода и резултујућа апликација је јако тешка за разумевање и одржавање. Као потреба да се програмерима што више олакша писање кода, и омогући лака синхронизација између података и корисничког интерфејса, настало је концепт под називом „*state*“.



Слика 3.2.1.1. Приказ синхронизације између стања и компоненте [8]

*State* је најбитнији концепт у *React*-у. Свака промена стања резултира поновним рендеровањем компоненте која то стање садржи. На тај начин врши се синхронизација података са корисничким интерфејсом. Тако је *React* и добио назив. Он реагује на сваку промену стања поновним рендеровањем компоненте која то стање садржи. Стање представља податак коју компонента може садржати и која се може мењати током времена. То је меморија компоненте која омогућава креирање динамичних компоненти. Стање може бити локално или глобално. Локално стање користи се од стране само неколико компоненти, обично родитеља и деце, док глобалном стању може приступити било која компонента у апликацији. Стање се може делити између компоненти. Међутим, уколико је дете компонента дубоко угњеждена у стаблу компоненти, преношењем стања преко пропса може доћи до одређеног проблема под називом „*props drilling*“, тако да је у том случају најбоље користити глобално стање. Ова кука враћа два елемента, променљиву и функцију за сетовање променљиве. Вредност променљиве могуће је мењати искључиво функцијом за сетовање. Сваки пут када се вредност стања промени, компонента која садржи то стање ће се поновно рендеровати.

### 3.2.2 useReducer

Ово је још једна кука у *React*-у која се користи за управљање стањем апликације. Представља алтернативу *useState* куке. Пружа бољу организацију кода и логике за ажурирање стања. Веома је корисна када имамо неко сложено стање које се састоји из више вредности или кад имамо стање које се израчунава на основу претходног стања [8].

```
const [state, dispatch] = useReducer(reducer, initialState);
```

*useReducer* има два аргумента. Први је *reducer* функција која на основу текућег стања и акције враћа ново стање. Други аргумент представља почетну вредност стања. *Reducer* функција обично користи *switch/case* структуру и на основу типа акције дефинише начин на којем се ажурира стање.

```
function reducer(state, action) {
  switch (action.type) {
    case 'login':
      return { ...state, user: action.payload, isAuthenticated: true };
    case 'validateJwt':
      return { ...state, user: action.payload, isAuthenticated: true };
    case 'logout':
      return { ...state, user: null, isAuthenticated: false };
    case 'registration':
      return { ...state, user: action.payload, isAuthenticated: true };
    default:
      throw new Error('Unknown action');
  }
}
```

Слика 3.2.2.1. Пример коришћења *reducer* функције

Акције су објекти који обавезно садрже поље „*type*“ и произвољан број других поља који су неопходни за ажурирање стања. Акција се као аргумент прослеђује *dispatch* функцији која даље преноси акцију *reducer* функцији. Када се позове *dispatch* са одређеном акцијом, *React* прослеђује ту акцију *reducer* функцији. *Reducer* функција на основу *type* својства, извршава одређени део кода и враћа ново стање којим се замењује текуће стање. *useReducer* омогућава бољу организацију кода у ситуацијама када имамо јако пуно стања због тога што је целокупна логика за управљање стања централизована и одвојена је од *UI* логике. Уколико је компонента једноставнија са мањим бројем стања, онда је *useState* боља опција.

```
dispatch({ type: 'login', payload: res.data.user });
```

Слика 3.2.2.2 Пример *dispatch* функције

### 3.2.3 useEffect

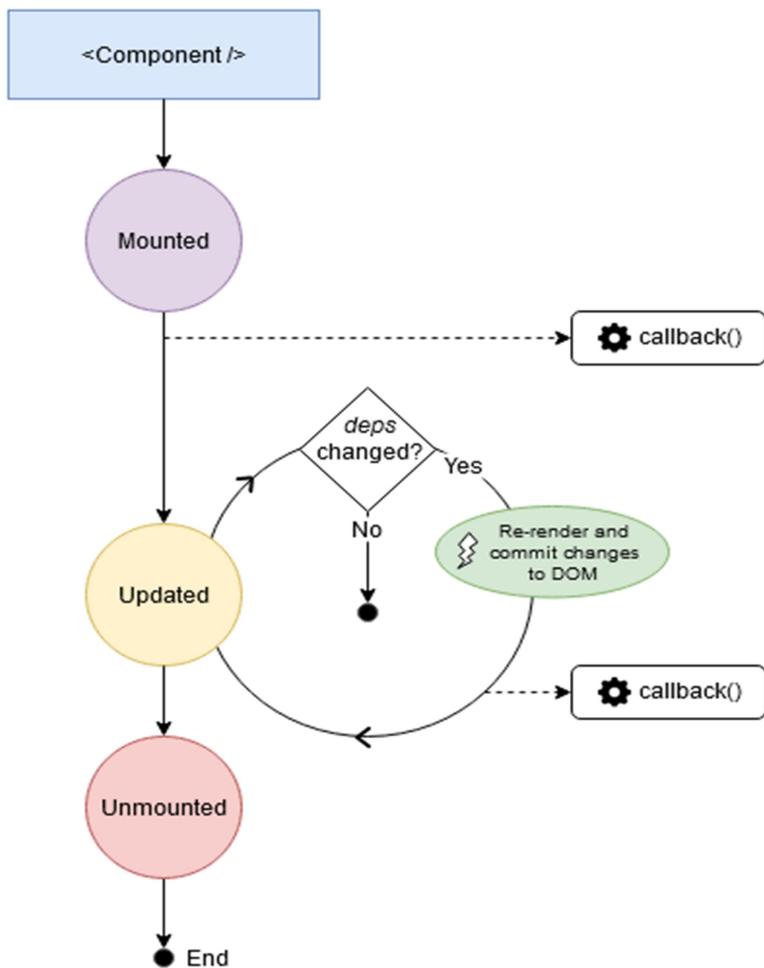
*useEffect* је често коришћена *React* кука који омогућава синхронизацију компоненте са неким екстерним системом или догађајем. *useEffect* нам омогућава да пишемо код који се може извршити у различитим тренуцима током животног циклуса апликације или приликом настанка различитих догађаја. Ова кука је идеална када имамо различита учитавања података из базе или са неког *API*-а [10].

```
const [zavisnost, setZavisnost] = useState('');

useEffect(() => {
  console.log('poziva se');
  //
  //
}, [zavisnost]);
```

Слика 3.2.3.1. Пример коришћења *useEffect* куке

Свака *useEffect* кука има два аргумента. Први је *callback* функција која садржи код који ће се извршити када се активира *useEffect*, а други аргумент је низ зависности. Овај низ одређује када ће се овај ефекат поново извршити. Ако низ није прослеђен, ефекат ће се извршити након сваког рендеровања и ререндеровања компоненте. Ако је низ празан ([]), ефекат ће се извршити само приликом првог (иницијалног) рендеровања компоненте. Уколико су у низу уписане неке променљиве, онда ће се ефекат извршити приликом иницијалног рендеровања компоненте и при свакој промени променљиве из низа зависности [10].



Слика 3.2.3.2. Приказ како `useEffect` кука функционише [13]

### 3.2.4 Custom Hooks

*React* омогућава дефинисање личних кука. Личне куке су функције које се креирају на начин тако да их је могуће користити више пута у пројекту. Оне омогућавају поновну употребу кода, чиме се избегава непотребно дуплирање логике између компоненти. Такође, постиже се и боља организованост кода. По конвенцији именовања, назив сваке личне куке мора почети са „`use`““. Свака лична кука треба да има само једну сврху и потребно је пројектовати на начин како би се могла користити већи број пута кроз пројекат.

## 3.3 React Router DOM

*React Router Dom* је једна од најпопуларнијих и најкоришћенијих библиотека уз *React*. Она омогућава изградњу правих *SPA* (*Single Page Application*) апликација. Уз помоћ рута, различити *URL*-ови повезују се са различитим приказима на корисничком интерфејсу [14].

Код традиционалних веб апликација, клик корисника на неки линк захтевао је од интернет прегледача да пошаље захтев серверу, након чега би преузео неопходне *HTML*, *CSS* и *JS* фајлове и на крају изрендеровао и приказао страницу кориснику. Сваки клик је понављао овај процес испочетка. То је било лоше за перформансе јер је корисник сваки пут чекао страницу да се учита [14].

*React Router* је омогућио рутирање на клијентској страни. Овај начин рутирања омогућава апликацији да ажурира *URL* и прикаже нови интерфејс кориснику без потребе за захтевањем нове *HTML* странице са сервера. Целокупан приказ рендерује се само на једној страници. Апликација би само покупила неопходне податке са сервера и ажурирала кориснички интерфејс са новим подацима без освежавање странице. Ово је довело до знатног побољшања перформанси веб апликација и веома је побољшало корисничко искуство [14].

Уз помоћ *React Router*-а можемо креирати угњеждене руте. То су руте унутар неких других рута. Ово нам омогућава лако креирање комплексног корисничког интерфејса код којих одређене странице садрже заједничке компоненте, као што су *header* и *side-bar*. Такође нуди могућност заштите ruta од неовлашћеног приступа тако да само пријављени корисници могу приступати одређеним деловима апликације. *React Router* је лако проширљив, омогућава програмерима да лако прилагођавају и додају нове руте по потреби. Ово је једна добро документована, редовно ажурирана библиотека која има пуно корисника и јако велику заједницу [7].

### 3.3.1 useNavigate

*useNavigate* је кука која враћа функцију која омогућава програмску навигацију. Основни задатак ове куке је да омогући навигацију до различитих рута унутар апликације. Функција *navigate* подржава различите начине навигације у зависности од тога шта јој се проследи као аргумент. Може се навести апсолутна путања до неке руте, или релативна путања у односу на тренутну руту. Уколико се наведе „-1“ вратиће се на претходну руту, а ако се наведе „1“ прелази се на следећу руту. Такође је могуће пренети и стање приликом навигације између ruta. Може се и заменити тренутна ставка у историји претраживача уместо да се дода нова. Ова кука омогућава програмерима да лако дефинишу путање између различитих делова апликације [10].

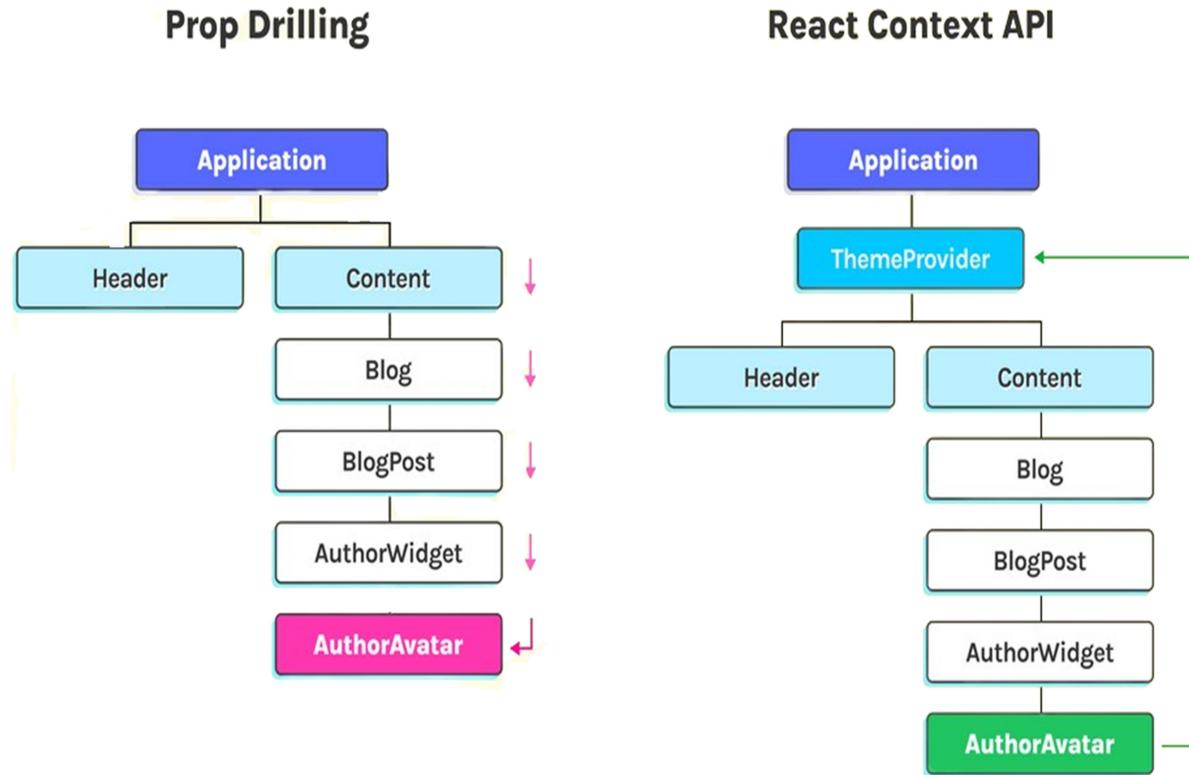
```
let navigate = useNavigate();
navigate('/app', { replace: true });
```

Слика 3.3.1.1. Пример употребе *useNavigate* куке

### 3.4 Context API

*Context API* је јако моћан алат који *React* нуди за управљање глобалним стањем унутар апликације. Омогућава брз и лак начин за пренос стања између различитих компоненти без потребе да се та стања преносе као пропс. Глобалном стању може приступити било која компонента у апликацији. Ово спречава такозвани „*prop drilling*“ проблем који може настати уколико се стање преноси као пропс до компоненте која је дубоко угњеждена у стаблу компоненти [16].

Једини проблем је тај што промена глобалног стања доводи до поновног рендеровања сваке компоненте која то стање користи. Такође, поновно рендеровање родитељске компоненте доводи до поновног рендеровања њених потомака што може довести до великог броја непотребних рендеровања које се могу одразити на драстичан пад перформанси у великим апликацијама. Зато је неопходно бити обазрив приликом коришћења *Context API*-а. *Context API* састоји се од два главна концепта, то су *Provider* и *Consumer*. *Provider* компонента приhvата неку вредност као пропс и ова компонента користи се за постављање контекста. Она обезбеђује дату вредност контекста свим компонентама које се налазе унутар њеног стабла. Док *Consumer* компонента, омогућава потомцима компонентама да приступе вредностима контекста које *Provider* компонента пружа.



Слика 3.4.1. Приказ начина решавања “*prop drilling-a*” коришћењем *Context API*-а [17]

### 3.4.1 useContext

Ова кука је део *React Hooks API*-а и омогућава приступ глобалном стању без потребе за коришћењем *consumer* компоненте. Кука враћа тренутну вредност контекста за најближег провајдера изнад у стаблу компоненти. Користи се на следећи начин.

```
const context = useContext(AuthContext);
```

Слика 3.4.1.1. Пример употребе *useContext* куке

## 3.5 React icons

*React icons* је популарна библиотека која се користи у *React* апликацијама за лако укључивање иконица. Библиотека садржи хиљаде иконица које корисник може укључити у свој пројекат. Иконице је могуће стилизовати преко пропс-а и на тај начин могуће је мењати боју, ротацију, величину или додати још неке стилове. Ово омогућава савршено уклапање иконица у целокупни дизајн апликације.



Слика 3.5.1. Неке од иконица које је могуће користити из *React icons* библиотеке [18]

## 3.6 Tailwind CSS

*Tailwind* је *utility-first CSS framework* који се користи за стилизовање веб апликација. Развијен је са намером омогућавања флексибилнијег и ефикаснијег начина писања стилова без креирања засебних *CSS* фајлова [19].

За разлику од неких других *CSS* програмских оквира, као што је *Bootstrap* који нуди предефинисане компоненте, *Tailwind* користи *utility* класе. То су мале, атомичне класе, које дефинишу једну врсту стила, попут маргине, боје позадине итд. *Tailwind* нуди на стотине оваквих малих класа које су готове и спремне за коришћење. Такође, омогућава програмерима да сами дефинишу и конфигуришу своје класе, што га чини врло флексибилним алатом за стилизовање. Комбинацијом више ових класа добија се жељени изглед компоненте [19].

*Tailwind* има пуно предности у односу на традиционалан начин писања стилова. Прва предност је да програмер не мора више водити рачуна о именовању класа већ *Tailwind* сам то ради уместо њега. Смишљање имена класа неким програмерима је представљао мукотрпан посао јер је било неопходно пратити одређени начин именовања. Следећа предност је та што нема више засебних *CSS* фајлова већ се класе пишу унутар same компоненте. Ово доводи до огромне уштеде времена, нарочито при одржавању већ направљених апликација, јер када програмер жели да промени неки стил неког елемента унутар компоненте, он то одмах може учинити без отварања и претраге екстерних *CSS* фајлова.

Такође, још једна од предности јесте веома лак и интуитиван начин креирања респонзивног дизајна и „*hoover*“ ефеката. Он има и веома добру документацију и објашњења на њиховом званичном сајту преко којег се веома лако може савладати његово коришћење. Међутим, главни недостатак јесте што *JSX* изгледа доста непрегледно коришћењем *Tailwind*-а, али у великом броју случаја, програмери се брзо навикну. Такође, неопходно је обавити инсталацију и одређена подешавања како би овај *framework* за *CSS* могао да се користи.

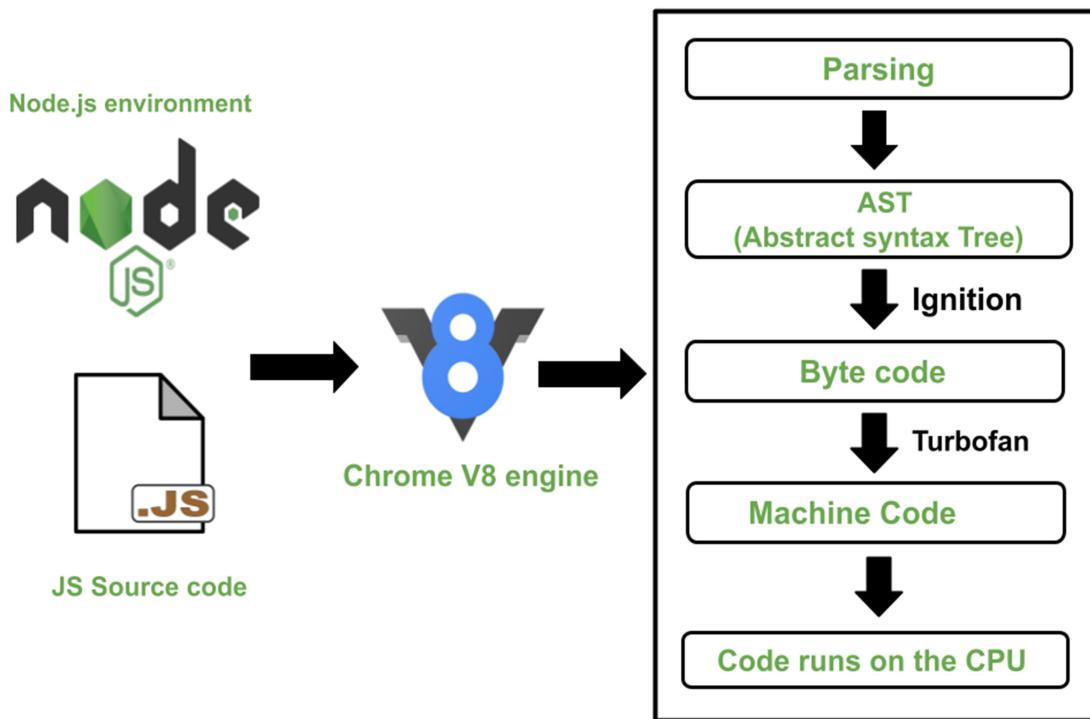
```
<button
  type="submit"
  className="my-5 w-full rounded-lg bg-teal-600 p-2 text-white
  transition-colors duration-300 hover:bg-teal-400">
  Registrujte se
</button>
```

Слика 3.6.1. Пример стилизовања дугмета коришћењем *Tailwind*-а

## 4. Node.js

*Node.js* је веома популарна *open-source* платформа за извршавање *JavaScript* кода ван интернет прегледача коришћењем *V8 JavaScript engine*. Он садржи све компоненте неопходне за извршавање *JavaScript*-а. Ово представља огромну предност *frontend* програмерима који користе *JavaScript* за развој клијентске стране јер сада могу развијати и серверску страну апликације без потребе за учењем новог језика. Такође, једна од предности јесте и постојање *NPM* библиотеке која нуди велики број бесплатних пакета које је могуће укључити у пројекат. *Node* омогућава изградњу брзих и скалабилних *backend* апликација које се могу покретати на различитим оперативним системима као што су *Linux*, *Windows*, *Mac OS X*, итд [20].

Захваљујући *V8 engine* који је развио *Google* за свој *Google Chrome*, *Node* може брзо и ефикасно извршавати сложене апликације. Овај *engine* омогућава компајлирање *JavaScript* кода директно у машински и тиме омогућити брзо извршавање и високе перформансе. *V8* писан је коришћењем језика *C++*. Захваљујући њему, *Node* је постао моћна платформа за развој серверске стране [21].



**Слика 4.1.** Приказ како *V8 engine* извршива *JavaScript* код писан у *Node.js* окружењу [21]

*Node.js* апликација извршава се унутар само једног процеса, без креирања додатних нити за сваки пристигли захтев. Он обезбеђује скуп асинхроних операција које спречавају блокирање ове главне нити извршења. Када се обавља нека улазно/излазна операција, као што је на пример читање података из базе, уместо да се

блокира главни нит и тиме беспотребно троши процесорска моћ на чекање, *Node.js* ће ову операцију извршавати у позадини и наставиће са извршавањем осталих операција. Када се операција из позадине заврши и подаци постану доступни, позваће се *callback* функција и резултат ће се пребацити на главну нит извршења. Ово омогућава да *Node* без проблема обрађује и на хиљаде конкурентних конекција са само једним сервером [20].

## 4.1 Express.js

*Express.js* је флексибилни веб апликациони оквир (*framework*) за *Node.js*, који обезбеђује пуно алата за развој веб апликација и *API*-а. Он олакшава успостављање и управљање веб сервера. Користи „*middleware*“ који омогућава програмерима да додају разне функционалности у своје апликације. Такође, пружа моћан систем рутирања и управља различитим захтевима који стижу од корисника. Рутирање омогућава дефинисање како апликација одговара на одређени захтев клијента [22].

Дизајниран је да буде брз и ефикасан што га чини погодним за изградњу јако брзих веб апликација и *API*-а. Он је један од најпопуларнијих веб оквира за *Node.js* и има велику заједницу корисника и добру документацију. Погодан је за различите врсте пројекта због своје изванредне флексибилности. Веома је једноставан за учење и представља прави избор за нове *backend* програмере. *Express.js* омогућава модуларни приступ приликом пројектовања апликација што омогућава да се различити делови апликације могу развијати независно као *middleware*, што олакшава одржавање и тестирање. Он пружа корисне методе за управљање *HTTP* захтевима и одговорима. Ове методе олакшавају рад са *JSON* подацима, управљају статусним кодовима, постављају заглавља итд. Такође, пружа добре механизме за управљање грешака путем *middleware* функција [23]. Не долази са утрађеном подршком за базе, али се коришћењем одговарајућих *Node.js* библиотека лако може повезати са свим базама, као што су *MongoDB*, *PostgreSQL*, *MySQL*, итд. Он такође пружа разне сигурносне механизме као што су заштита од *XSS* напада, *CSRF* заштита, управљање сесијама итд. Веома је флексибилан и проширљив и има широку примену. *Express* је посебно популаран за изградњу *RESTful API*-а због своје способности да лако управља захтевима и одговорима у *JSON* формату и подржава све *HTTP* методе потребне за *RESTful API* [4].

## 4.2 Giphy API

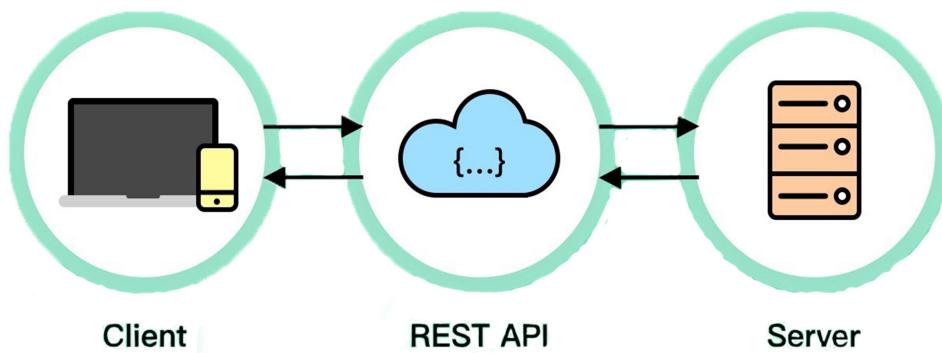
*Giphy* је популарна платформа која омогућава корисницима да претражују, додају и користе различите слике у *GIF* формату. *GIF* (скраћеница од *Graphics Interchange Format*) су кратке анимиране слике које користе 256b палету боја и јако су популарне код млађих генерација за дељење анимираних слика шаљивог карактера. Ова платформа нуди бесплатне веб сервисе које омогућава претрагу и преузимање *GIF* слика из њихових библиотека. Како би се *Giphy API* користио, неопходне је прво регистровати се на *Giphy Developer* порталу и креирати апликацију како би добили *API* кључ. Тада ће потребан за аутентификацију *API* захтева. *Giphy* нуди различите врсте кључева. Постоји јавни *API* кључ намењен за тестирање и приватни *API* кључ за произведене апликације. Преко овог *API*-а добијамо жељене *GIF* слике које унапређују корисничко искуство додавањем шаљивог карактера.

## 4.3 RESTful API

Веб *API* (скраћеница од *Application Programming Interface*) је софтвер који се користи од стране неког другог софтвера како би се омогућила комуникација између апликација. Веб сервис је специфичан тип *API*-а који се извршава на веб-у и користи технологије као што су *HTTP* и *JSON* које омогућавају комуникацију и размену података између апликација без обзира на њихове технологије и платформу. *Rest* (скраћеница од *Representational State Transfer*) је архитектурни образац за развој веб сервиса. Постоје и други, старији обрасци као што су *SOAP* и *XMLRPC*, али *Rest* је тренутно најпопуларнији. Овај архитектурни образац користи се за изградњу *RESTfull* веб сервиса. Он дефинише скуп ограничења која њиховом применом доводе до скалабилних и лако одрживих веб сервиса [4]. Основни принципи *REST* архитектуре су:

- Сваки ресурс мора бити идентификован путем јединственог идентификатора (*URI*), што омогућује директан приступ ресурсу [24].
- Клијент/Сервер архитектура је раздвојена, што омогућује независан развој клијентске и серверске стране и омогућује да се клијентска страна може користити уз више различитих серверских страна [24].
- *Stateless*, односно сваки захтев клијента ка серверу мора садржати све информације које су неопходне како би се захтев успешно обрадио. Сервер не сме памтити информације претходних захтева како би обрадио текући. Сваки захтев третира се као нови без обзира на претходне [24].
- Користи се стандардизован интерфејс за комуникацију између клијента и сервера. Користе се једино *HTTP* методе (*GET*, *POST*, *PUT*, *DELETE*) за *CRUD* операције. [24].
- За пренос података углавном се користи *JSON* [24].
- *Rest* омогућава употребу слојевитог система где захтев клијената пролази кроз више слојева сервера. Ово побољшава сигурност и скалабилност система [24].

Веб сервиси који прате наведене *REST* принципе називају се *RESTfull* веб сервисима. Дизајнирани су тако да буду лаки и интуитивни за коришћење. Омогућавају лаку комуникацију између клијента и сервера на интернету.



Слика 4.3.1. Приказ употребе REST API-а

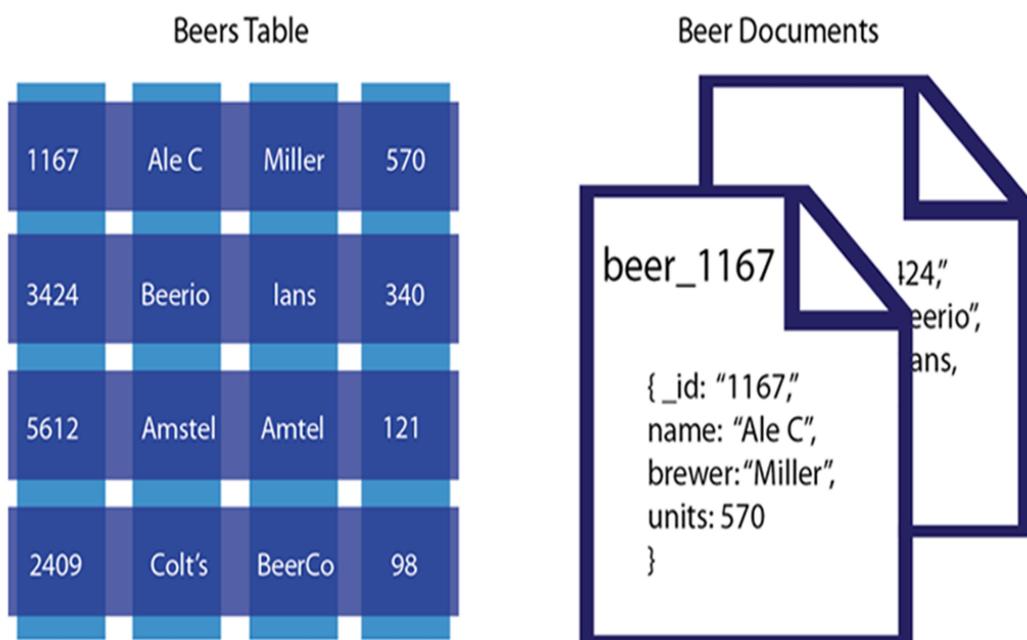
## 4.4 Google OAuth 2

*Google OAuth2* је популарни начин за аутентификацију корисника коришћењем његовог *Google* налога. Приликом пријављивања, апликација преусмерава корисника на *Google* страницу за пријављивање где се он пријављује. Неопходно је да корисник да дозволу апликацији за приступ његовим основним подацима као што је имејл адреса, корисничко име, профилна слика итд. Уколико корисник да дозволу, *Google* враћа апликацији све неопходне податке за успешну аутентификацију корисника. Јако битна ствар је да *Google* не дели шифру корисника. Уз помоћ ових сервиса, приступање апликацији је знатно брже јер корисник више не мора правити налог на апликацији и валидирати имејл адресу већ се може пријављивати својим *Google* налогом на брз и једноставан начин.

## 5. MongoDB

*MongoDB* је бесплатна *open-source NoSQL* база података оријентисана на документима, која за разлику од стандардних *SQL* база које користе табеле, *MongoDB* користи „колекције“ за складиштење података. Свака колекција састоји се од већег броја докумената унутар којих се подаци смештају у базу података. Документи су формата слични *JSON*-у, који се назива  *BSON*. На први поглед изгледа потпуни исто као *JSON* али са битном разликом да свака вредност има свој тип који може бити *string*, *boolean*, *date*, *double*, *object*, итд [25].

Примарни кључ је обавезан и за њега је резервисано поље *\_id*. *MongoDB* аутоматски генерише уникатан кључ за сваки документ који је у бази. За примарни кључ, користи се посебан тип података који се зове  *ObjectId*. Уз помоћ овог кључа могуће је референцирати и повезивати моделе коришћењем *Mongoose* алата.



Слика 5.1. Разлика између документа и табеле [27]

*MongoDB* база је скалабилна и могуће је лако извршити дистрибуцију података на различитим машинама. Она је такође веома флексибилна и није потребно дефинисати никакву шему како би се подаци памтили у бази. Нема никаквих ограничења, сваки документ може имати различиту структуру и она се може мењати током времена. Ово је једна од најкоришћенијих база података уз *Node.js*. Још једна битна ставка код *MongoDB* је концепт угњеждених докумената где се један или више докумената могу убацити унутар неког другог документа како би сви заједно чинили један документ. Ово омогућава бржи приступ подацима и лакше моделе података, међутим ово није увек најбоља опција. Код стандардних релационих база, сви подаци су нормализовани.

## 5.1 Mongoose

*Mongoose* је jako популарна, *Object Data Modeling (ODM)* библиотека која се користи заједно са *Node.js* и *MongoDB* базом. Основна намена јесте да омогући структурни и лакши приступ *MongoDB* бази података коришћењем *JavaScript-a*. Она одржава повезаност између података и обезбеђује валидацију шема [26].

*MongoDB* база не форсира шеме, документи који се памте могу бити различите структуре и да имају различита поља. Међутим у великим броју случаја су ипак потребне шеме како би се избегле грешке које се могу јавити. Због тога се користе разни алати као што је *Mongoose* који нам омогућавају валидацију докумената. *Mongoose* користи шеме за дефинисање структуре података унутар *MongoDB* колекције. Уз помоћ ових шема, могуће је дефинисати типове података и поставити одређена ограничења, валидације и почетне вредности. На основу шеме креирају се модели. Модели нам омогућавају да извршимо *CRUD* операције над базом. *Mongoose* нуди уграђену валидацију и богат скуп интуитивних *API-а* које је могуће користити за претраживање, брисање, додавање и ажурирање докумената из *MongoDB* базе. Могуће је писати и *middleware* функције у *Mongoose*-у које омогућавају извршавање одређеног кода пре или након одређених операција над базом. *Mongoose* нуди симулацију релација као код *SQL* база података путем референцирања. Са базом повезује се једноставно преко конекционог стринга. Он такође врши аутоматско каствовање података. *Mongoose* операције враћају промисе, што значи да је неопходно користити *async/await* операције. Он значајно олакшава рад са *MongoDB* базом [3].

## 6. Имплементација апликације FairShare

У овом поглављу описана је апликација за поделу трошкова *FairShare* и њена комплетна имплементација. Поглавље се заснива на претходно дефинисаним технологијама и теоријским основама. Целокупан код ове апликације налази се на следећем *GitHub* репозиторијуму: <https://github.com/IronWolf333/Zavrski-rad>

### 6.1 Опис апликације

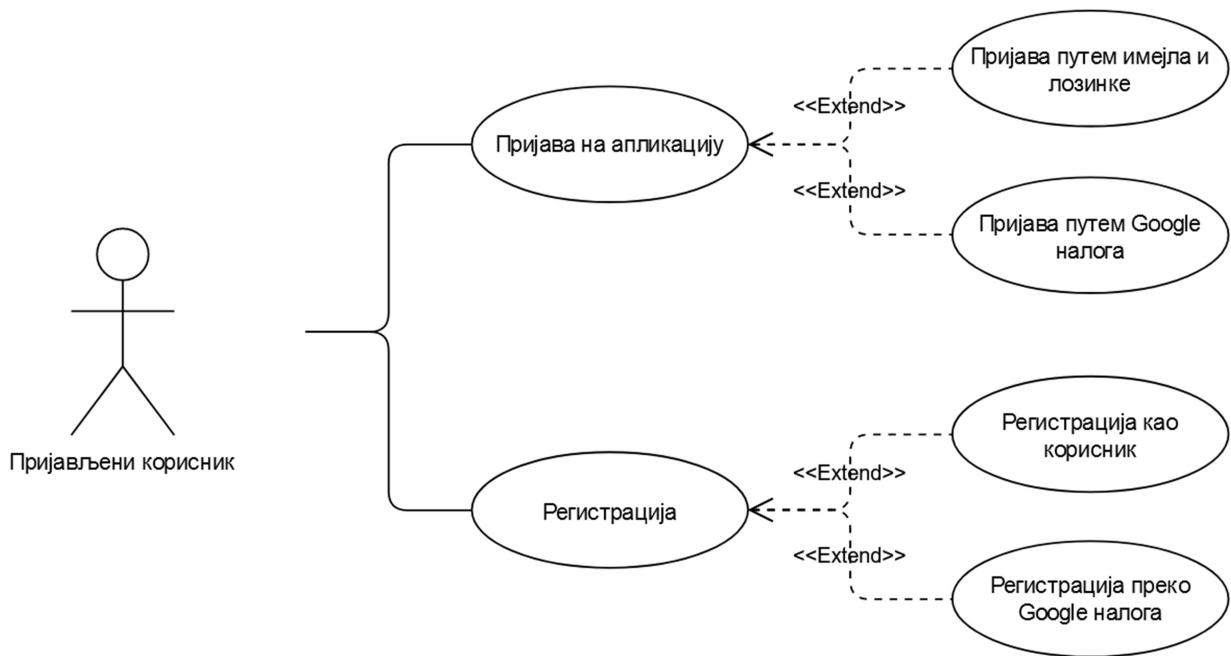
Апликација *FairShare* корисницима омогућава увид и поделу заједничких трошкова које корисници имају са својим пријатељима. Како би корисник користио апликацију, он мора да се пријави својим *Google* налогом или да се региструје и направи нови налог преко којег ће се пријављивати. Сваки корисник има увид у своју листу пријатеља у коју може додати друге кориснике апликације као своје пријатеље. Он такође може бити члан већег броја група и може да креира нове групе. Корисник може креирати трошкове и има увид у све трошкове које је креирао. Трошак може бити у динарима или у некој страној валути. Сваки трошак корисник може поделити са својим пријатељима или да подели са пријатељима који су у одређеној групи. Корисник има увид у своја међусобна дуговања са својим пријатељима и укупним трошковима оствареним у групама. Такође путем различитих графика и мапа, има увид у све информације везане за своје трошкове. Више о апликацији говори се у наредним деловима.

### 6.2 Случајеви коришћења

Апликација има три профила корисника, у зависности од улоге коју корисник поседује биће доступни другачији случајеви коришћења.

- 1.) **Посетилац** (непријављени корисник) може приступити једино почетној страници и страници за регистрацију и пријављивање. Он има приступ основним информацијама о апликацији које се налазе на почетној страници. Уколико жели да користи апликацију, посетилац се мора пријавити или регистровати.
- 2.) **Корисник** је профил који обухвата све посетиоце који су се пријавили или регистровали на апликацију. Он има своју листу пријатеља, група и трошкова. Те листе су само његове и ниједан други корисник им не може приступити.
- 3.) **Администратор** је профил који има највише привилегија. Осим функционалности које садржи корисник, администратор садржи и додатну функционалност. Он има увид у све кориснике који су се регистровали на апликацију. Може видети њихове основне информације, датум регистраовања и има опцију за брисање корисника. Администратор може обрисати било ког корисника осим корисника који је администратор.

Уколико корисник није пријављен на систем, он има могућност пријаве преко *Google* налога или пријаве путем регистрованог налога путем имејла и шифре. Уколико нема налог, корисник се на апликацију може регистровати уношењем основних података. На слици 6.2.1 налази се приказ случајева коришћења за непријављеног корисника.



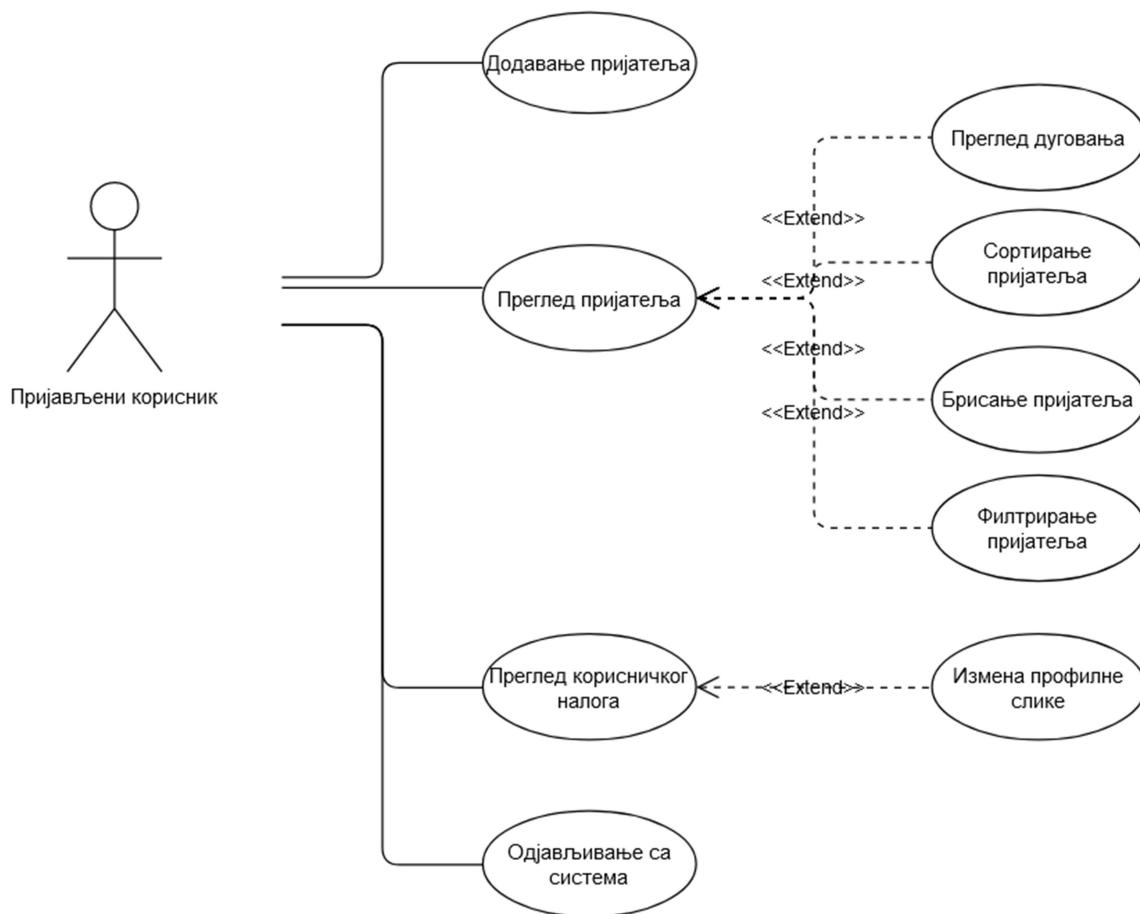
**Слика 6.2.1.** Дијаграм случајева коришћења за непријављеног корисника

Сваки пријављени корисник има следеће могућности:

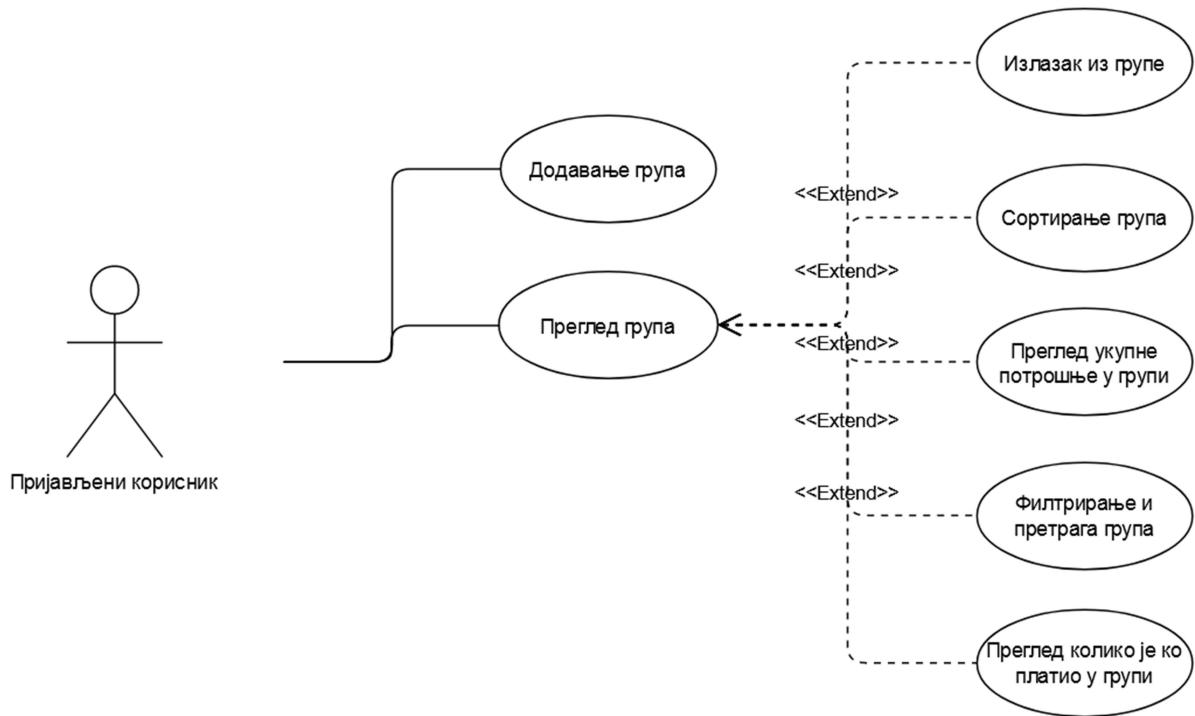
- Преглед корисничког налога и измена профилне слике
- Преглед свих његових пријатеља
- Преглед дуговања која корисник има са пријатељима
- Сортирање листе пријатеља по различитим критеријумима
- Претрага пријатеља
- Додавање новог пријатеља
- Брисање пријатеља
- Преглед свих група у којима је корисник
- Преглед укупне потрошње остварене у свакој од група
- Преглед колико је свако платио унутар групе
- Филтрирање и сортирање група по различитим критеријумима
- Претрага групе
- Напуштање групе
- Додавање нове групе
- Преглед свих трошкова које корисник има
- Филтрирање и сортирање трошкова по различитим критеријумима
- Додавање и подела трошка у различитим валутама
- Претрага трошкова
- Одабир да ли се трошак дели са пријатељима или групом
- Додавање локације настанка трошка
- Брисање трошка
- Графички преглед информација о дуговањима преко графика

- Графички преглед трошкова по категоријама преко графика
- Графички преглед трошкова оствареним по групама преко графика
- Преглед локација и информација о свим трошковима на мапи
- Одјављивање са система

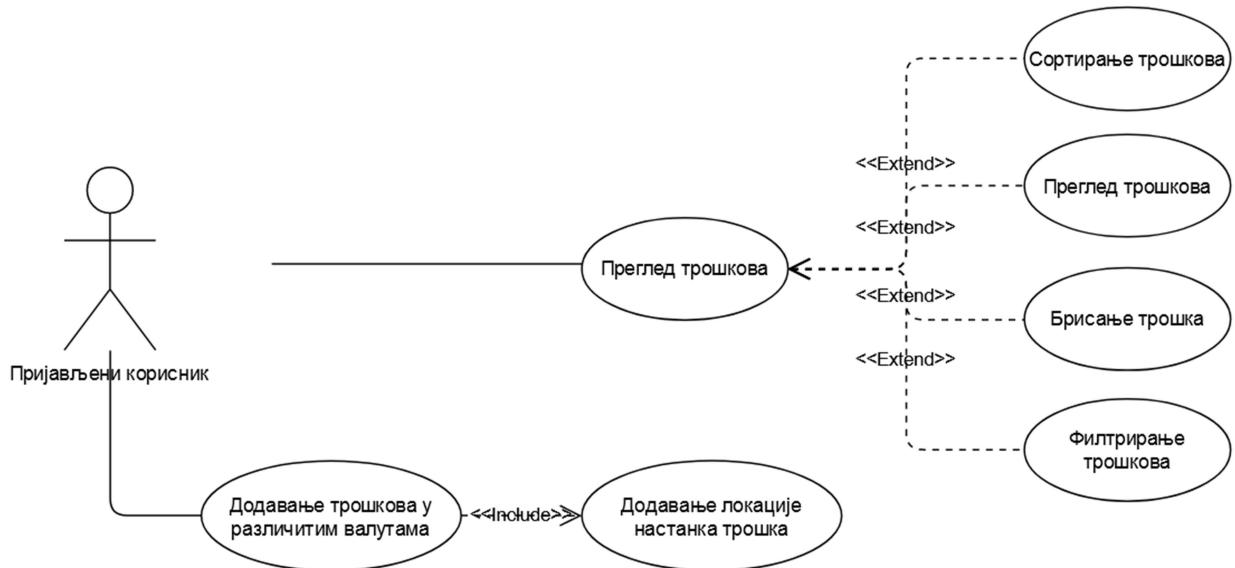
На сликама (6.2.2, 6.2.3, 6.2.4 и 6.2.5) налазе се прикази случајева коришћења за пријављеног корисника.



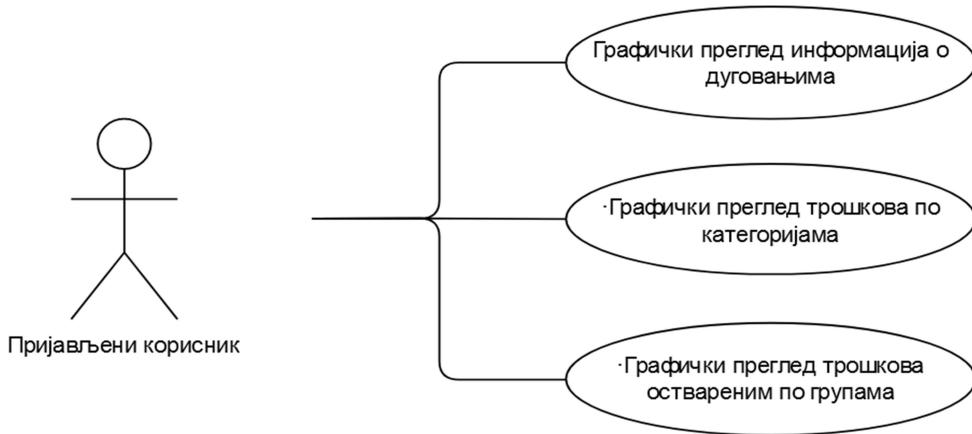
**Слика 6.2.2.** Дијаграм случајева коришћења за пријављеног корисника (први део)



Слика 6.2.3. Дијаграм случајева коришћења за пријављеног корисника (други део)



Слика 6.2.4. Дијаграм случајева коришћења за пријављеног корисника ( трећи део)

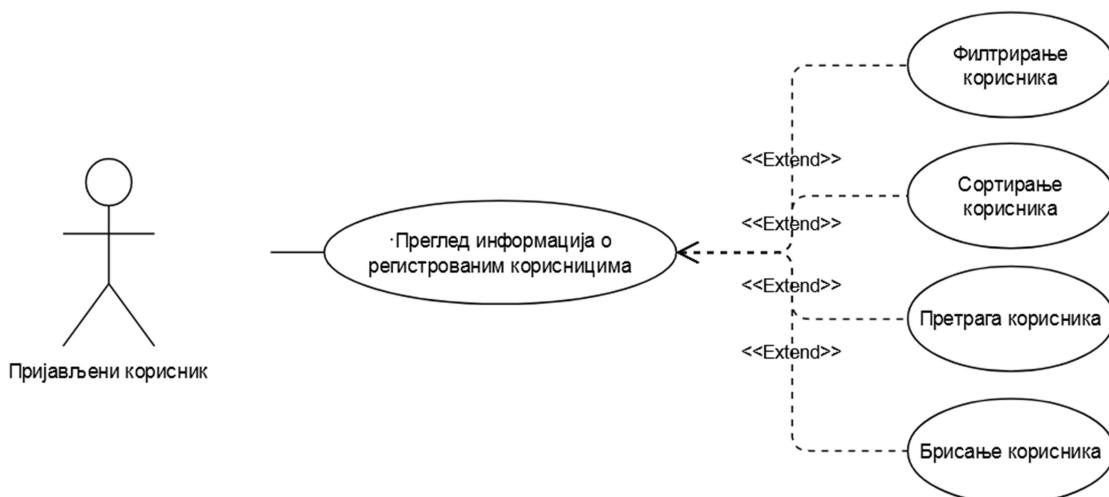


**Слика 6.2.4.** Дијаграм случајева коришћења за пријављеног корисника (четврти део)

Администратор поред могућности које има пријављени корисник, има и следеће могућности:

- Преглед информација о регистрованим корисницима
- Филтрирање и сортирање листе корисника по различитим критеријумима
- Претрага корисника
- Брисање корисника

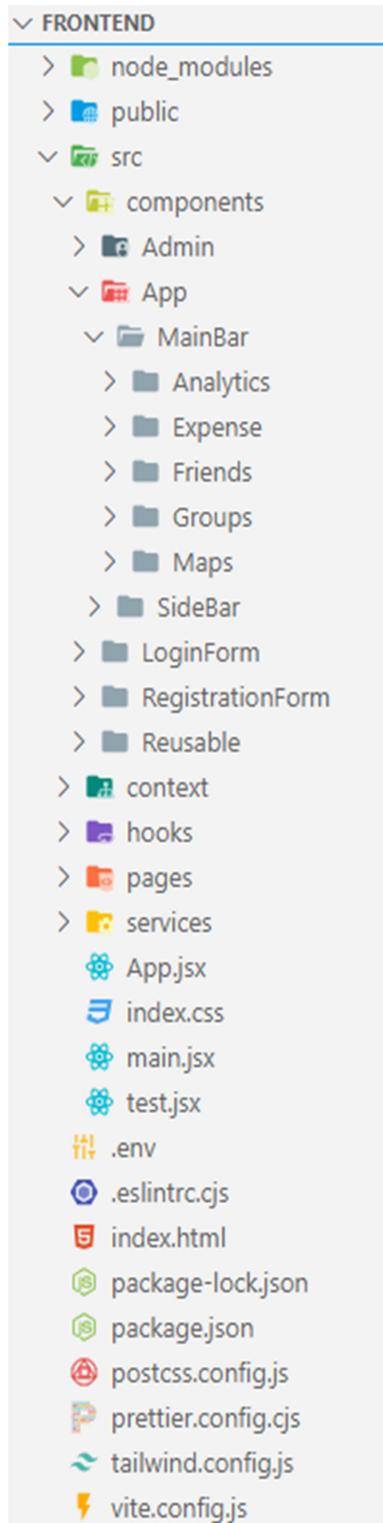
На слици 6.2.5 налази се приказ случајева коришћења за администратора.



**Слика 6.2.5.** Дијаграм случајева коришћења за администратора

## 6.3 Имплементација предње стране апликације

Предња страна апликације (*frontend*) имплементирана је коришћењем библиотеке *React*, алата *Vite* и програмског оквира *Tailwind*. Структура предње стране приказана је на следећој слици. У овом поглављу приказана је имплементација најбитнијих компоненти и делова предње стране.

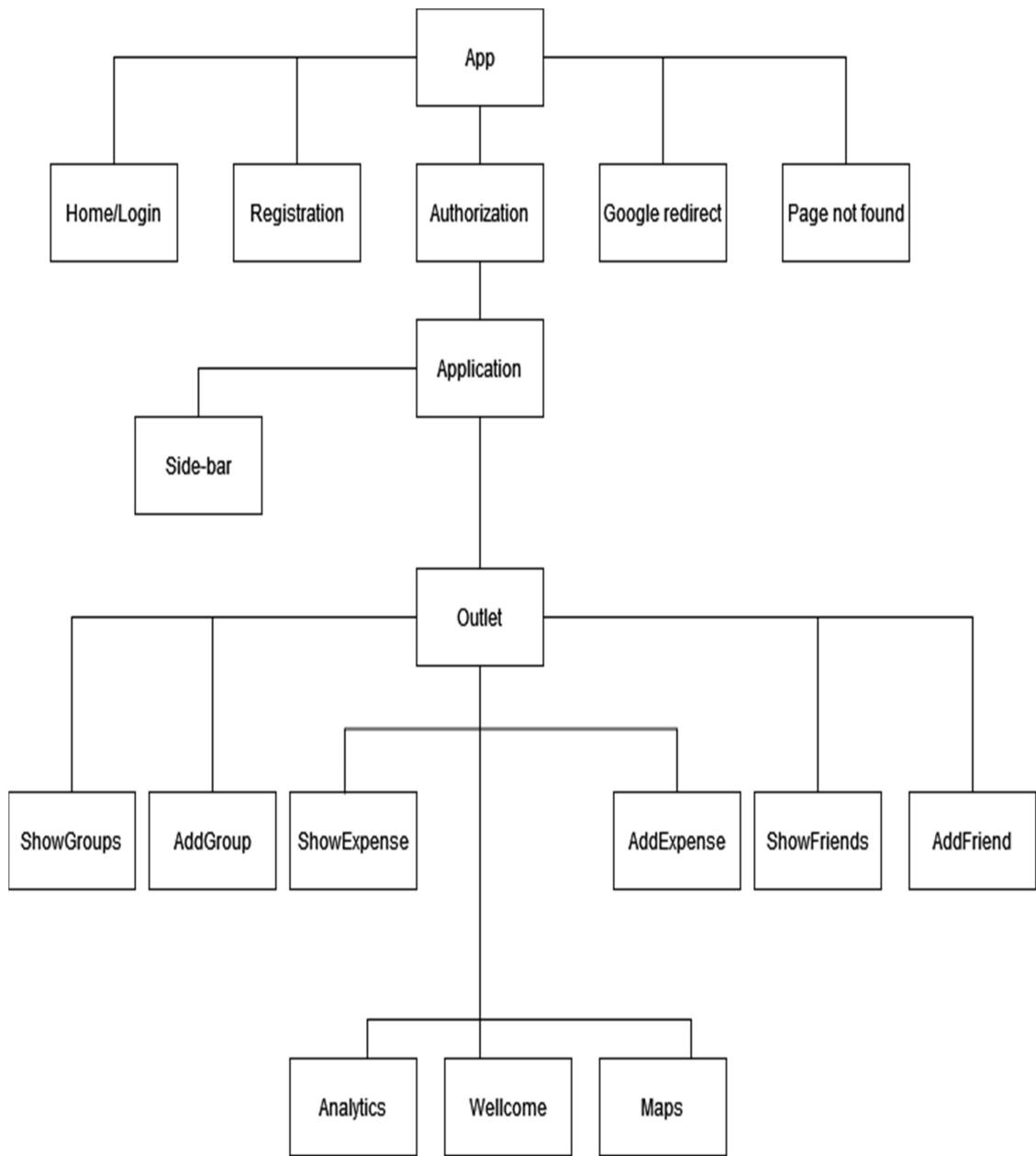


Слика 6.3.1. Приказ садржаја предње стране апликације

Предња страна апликације, састоји се од следећих фајлова и фолдера:

- ***node\_modules*** – Садржи све неопходне пакете који су потребни *React* апликацији за њено извршење.
- ***public*** – Користи се за чување јавно досупних ресурса који ће бити доступни када је апликација покренута у интернет претраживачу.
- ***src*** – Садржи све компоненте, странице, сервисе, личне куке, као и остале фајлове који представљају градивне блокове ове апликације.
- ***components*** – Унутар овог фолдера налазе се све компоненте које се користе у овој апликацији. Због боље организације пројекта, компоненте су подељене по фолдерима.
- ***Admin*** – У њему налазе се компоненте које чине приказ админ панела којем корисници са датим привилегијама могу приступити.
- ***App*** – Садржи компоненте које чине приказ апликације којој корисник приступа након пријављивања.
- ***MainBar*** – Садржи компоненте које омогућују различите приказе кориснику, као што је његова листа пријатеља, група, трошкова, итд.
- ***Analytics*** – Садржи компоненте за приказ података о трошковима, групама и дуговањима путем графика.
- ***Maps*** – Садржи компоненте за рад са мапама и приказ свих трошкова на мапи.
- ***Expense*** – Садржи компоненте за приказ и унос трошкова.
- ***Friends*** – Садржи компоненте за приказ и додавање пријатеља.
- ***Groups*** – Садржи компоненте за приказ и креирање група.
- ***SideBar*** – Садржи компоненте које чине главни навигациони мени који се налази на левој страни апликације.
- ***LoginForm*** – Фолдер садржи компоненте за приказ почетне странице апликације на којој се корисник пријављује.
- ***RegistrationForm*** – Унутар овог фолдера налазе се компоненте које чине приказ странице за регистрацију.
- ***Reusable*** – Садржи компоненте које се више пута користе у апликацији.
- ***services*** – Садржи сервисе који су неопходни за комуникацију са сервером. Они су задужени за слање *HTTP* захтева серверу и враћање пристиглих одговора.
- ***pages*** – Садржи све странице које се јављају у апликацији.
- ***hooks*** – Фолдер садржи личне куке које се користе приликом логовања и регистрације корисника.
- ***context*** – Садржи глобално стање којем свака компонента апликације може приступити. У њему се памти корисник који је пријављен на апликацију.
- ***.gitignore*** – Садржи списак фајлова који се игноришу у *Git-u* при позивању *commit* наредбе
- ***.env*** – Користи се за чување одређених конфигурационих вредности.
- ***.package.json*** – Садржи све информације о инсталираним пакетима, њиховим верзијама и скриптама за покретање апликације.
- ***tailwind.config.js*** – Конфигурациони фајл за *Tailwind* који је неопходан за правilan рад овог програмског оквира.
- ***prettier.config.cjs*** – Конфигурациони фајл за *prettier*.

На слици 6.3.2 приказано је стабло компоненти апликације *FairShare*. Ово стабло није потпуно, постоји пуно мањих компоненти које чине сваку од већих компоненти у стаблу, али због обимности приказа изостављене су. На слици приказане су само главне компоненте ове апликације. У наставку су објашњене и дата је имплементација за неке од њих.



*Слика 6.3.2. Приказ стабла компоненти*

### 6.3.1 Имплементација main.jsx фајла

Ово је улазна тачка *FairShare* апликације. Овде долази до иницијализације и рендеровања главне компоненте апликације. *ReactDOM* пакет омогућује интеракцију са *DOM-ом* и постављање главне компоненте у *DOM* стабло. Компонента се поставља унутар *root* елемента. *<React.StrictMode>* је компонента која доводи до тога да се главна компонента два пута рендурује како би се боље пронашле грешке и недостаци у апликацији.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App.jsx';
import './index.css';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    | <App />
    </React.StrictMode>,
);
)
```

Слика 6.3.1.1. Приказ main.jsx фајла

### 6.3.2 Имплементација главне компоненте апликације

Главна компонента *FairShare* апликације је *<App>*. На почетку неопходно је укључити потребне модуле из *react-router-dom* библиотеке како би се омогућило дефинисање ruta унутар апликације. Све руте обухваћене су *AuthProvider* компонентом која пружа контекст за аутентификацију, омогућавајући деци компонентама да приступају подацима о аутентификацији и пријављеном кориснику апликације. *BrowserRouter* компонента користи се као омотач за све руте унутар апликације. *Routes* компонента обухвата све *Route* компоненте. Свака *Route* компонента повезује одређену путању (назначену атрибутом „*path*“) са одређеном компонентом која ће бити рендерована када апликација буде на тој путањи.

Постоје угњеждене руте унутар *Route* компоненте за путању „*app*“ које омогућавају различит приказ кориснику. На пример, на путањи „*app/friends*“ рендероваће се компонента за приказ корисникove листе пријатеља, док на путањи „*app/groups*“ рендероваће се компонента за приказ група у којима је корисник. Веома важна компонента је *Authorization* која спречава неауторизован приступ апликацији и *Application* компоненти корисницима који нису пријављени. Постоји и *wildcard* рута дефинисана путањом „\*“ која обухвата све руте које нису дефинисане у апликацији и она приказује компоненту *PageNotFound* када корисник приступа рути која није дефинисана.

```

export default function App() {
  return (
    <AuthProvider>
      <BrowserRouter>
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/registration" element={<Registration />} />
          <Route path="/googleAuth" element={<Google />} />
          <Route
            path="/app"
            element={
              <Authorization>
                | <Application />
              </Authorization>
            }
          >
            <Route index element={<Wellcome />} />
            <Route path="friends" element={<ShowFriends />} />
            <Route path="admin" element={<ShowUsers />} />
            <Route path="addfriend" element={<AddFriendForm />} />
            <Route path="groups" element={<ShowGroups />} />
            <Route path="addgroup" element={<AddGroupForm />} />
            <Route path="expenses" element={<ShowExpenses />} />
            <Route path="addexpense" element={<AddExpenseForm />} />
            <Route path="analytics" element={<ShowAnalytics />} />
            <Route path="maps" element={<ShowMaps />} />
          </Route>
          <Route path="*" element={<PageNotFound />} />
        </Routes>
      </BrowserRouter>
    </AuthProvider>
  );
}

```

*Слика 6.3.2.1. Приказ имплементације главне компоненте апликације*

### 6.3.3 Имплементација компоненте за ауторизацију

Компонента која се користи за контролу приступа апликацији је **<Authorization/>**. Омогућава приказ *Application* компоненте једино уколико је корисник аутентификован и успешно пријављен на апликацију. *useAuth* је лична кука која се користи за приступ информацији о статусу аутентификације корисника. Уколико корисник није аутентификован, апликација ће га редиректовати на *Home* страницу и тиме спречити приступ, а ако јесте дозволиће приказ *Application* странице.

```

import { useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import useAuth from '../hooks/useAuth';
import PropTypes from 'prop-types';

Authorization.propTypes = {
  children: PropTypes.node,
};

export default function Authorization({ children }) {
  const { user, isAuthenticated } = useAuth();
  const navigate = useNavigate();

  useEffect(
    function () {
      if (!isAuthenticated) navigate('/');
    },
    [user, isAuthenticated, navigate],
  );

  return isAuthenticated ? children : null;
}

```

*Слика 6.3.3.1. Приказ имплементације Authorization компоненте*

#### 6.3.4 Имплементација глобалног контекста

Унутар AuthContext.jsx фајла имплементиран је глобални контекст аутентификације у апликацији коришћењем *ContextAPI-а* и *useReducer* куке. *AuthContext* креиран је позивом *createContext()* функције коју на почетку морамо укључити. Овај контекст користи се за пружање информација о аутентификацији и пријављеном кориснику свим компонентама у апликацији. Дефинисана је структура почетног стања *initialState* у којој се памте информације о пријављеном кориснику и томе да ли је аутентификован или није. *Reducer* функција управља променама стања у зависности од типа акције. *AuthProvider* компонента обезбеђује вредност стања и *dispatch* функцију свим њеним потомцима компонентама. Такође обезбеђује и глобално стање за *SideBar* како би компоненте знале да ли је навигациони бар отворен или није и тиме ажурирале свој приказ.

```

const AuthContext = createContext();

const initialState = {
  user: null,
  isAuthenticated: false,
};

function reducer(state, action) {
  switch (action.type) {
    case 'login':
      return { ...state, user: action.payload, isAuthenticated: true };
    case 'validateJwt':
      return { ...state, user: action.payload, isAuthenticated: true };
    case 'logout':
      return { ...state, user: null, isAuthenticated: false };
    case 'registration':
      return { ...state, user: action.payload, isAuthenticated: true };
    case 'failed/login':
      return { ...state, user: null, isAuthenticated: false };
    case 'photoChange':
      return { ...state, user: action.payload, isAuthenticated: true };
    default:
      throw new Error('Unknown action');
  }
}

function AuthProvider({ children }) {
  const [{ user, isAuthenticated }, dispatch] = useReducer(
    reducer,
    initialState,
  );

  const [open, setOpen] = useState(true);

  const onSidebar = () => {
    setOpen(!open);
  };

  return (
    <AuthContext.Provider
      value={{ user, isAuthenticated, dispatch, open, onSidebar }}
    >
      {children}
    </AuthContext.Provider>
  );
}

export { AuthProvider, AuthContext };

```

*Слика 6.3.4.1 Приказ имплементације AuthContext.jsx фајла*

### 6.3.5 Имплементација компоненте за пријављивање

Ова компонента користи се за приказ форме на почетној страници преко које се корисник пријављује на апликацију. Имплементација је приказана на слици 6.3.5.2. Компонента има два стања, *email* и *password*, у којима се памте подаци које корисник уноси када се пријављује на апликацију. Компонента враћа *JSX* код који описује структуру форме за пријаву. Постоје два дугмета. Једно обрађује пријаву корисника преко његовог имејла и шифре, а друго дугме обрађује пријаву корисника преко његовог *Google* налога. Уколико се корисник пријављује путем имејла и шифре, кликом на дугме, позваће се *login* функција коју добијамо преко личне куке *useLogin* која ће послати захтев серверу и обрадиће пристигли одговор. Уколико је аутентификација успешно прошла, позваће се *dispatch* функција која ће у глобалном стању да упише податке о кориснику који се пријавио и апликација ће редиректовати корисника и омогућити приступ садржају апликације. Уколико пријава није успешна позваће се *dispatch* за тип „*failed/login*“ и корисник неће моћи приступити главном садржају апликације.

```
export default function useLogin() {
  const { dispatch } = useAuth();
  let navigate = useNavigate();

  async function login(email, password) {
    try {
      const res = await axios.post(
        'http://localhost:3000/api/auth/login',
        { email, password },
        { withCredentials: true },
      );
      if (res.data.status === 'success') {
        dispatch({ type: 'login', payload: res.data.user });

        navigate('/app', { replace: true });
      }
    } catch (err) {
      console.log('Greska prilikom prijavljivanja');
      dispatch({ type: 'failed/login' });
      console.log(err.message);
    }
  }
  return login;
}
```

Слика 6.3.5.1. Приказ имплементације *useLogin* личне куке

```

export default function LoginForm() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');

  const login = useLogin();

  const handleClick = async (e) => {
    e.preventDefault();
    await login(email, password);
  };

  const handleClickGoogle = (e) => {
    e.preventDefault();
    window.location.href = 'http://localhost:3000/api/auth/auth/google';
  };

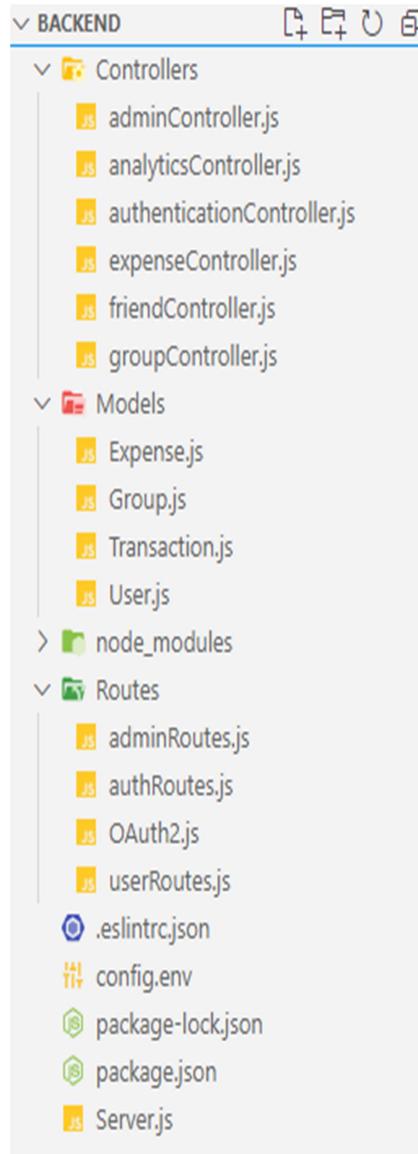
  return (
    <main
      className="m-8 min-w-[50px] max-w-[550px] rounded-2xl
      bg-white shadow-2xl md:flex-row md:space-y-0"
    >
      <form className="flex flex-col justify-center p-8 md:p-14">
        <Header />
        <Input
          email={email}
          password={password}
          setEmail={setEmail}
          setPassword={setPassword}
        />
        <button
          type="button"
          onClick={handleClick}
          className="my-5 w-full rounded-lg bg-teal-600 p-2 text-white
          transition-colors duration-300 hover:bg-teal-400 "
        >
          Prijavite se
        </button>
        <button
          type="button"
          onClick={handleClickGoogle}
          className="text-md mb-4 w-full rounded-lg border border-zinc-300 p-2
          transition-colors duration-300 hover:bg-blue-500 hover:text-white "
        >
          <img src={'/google.png'} className="mr-2 inline h-6 w-6" />
          Prijavite se preko Google naloga
        </button>
        <Footer />
      </form>
    </main>
  );
}

```

*Слика 6.3.5.2. Приказ имплементације LoginForm компоненте*

## 6.4 Имплементација задње стране апликације

Задња страна апликације имплементирана је коришћењем платформе *Node.js*, програмског оквира *Express.js*, *ODM* алата *Mongoose* и базе *MongoDB*. Структура задње стране приказана је на слици 6.4.1.



Слика 6.4.1. Приказ садржая задње стране.

- **node\_modules** – Садржи све неопходне пакете који су потребни апликацији за њено извршавање.
- **Routes** – Фолдер у којем су имплементиране све руте апликације.
- **Models** – Фолдер у којем су дефинисане шеме и модели.
- **Controllers** – Фолдер у којем су имплементиране методе контролера.
- **Server.js** – Улазна тачка апликације.
- **.package.json** – Садржи све информације о инсталirаним пакетима, њиховим верзијама и скрипте за покретање апликације.
- **config.env** – Користи се за чување одређених конфигурационих вредности.

#### 6.4.1 Имплементација веб сервера

У овом делу је објашњена имплементација веб сервера који је повезан са MongoDB базом података и који обрађује корисничке захтеве. На почетку укључене су неопходне библиотеке потребне за рад апликације, а то су: *express* (за успостављање веб сервера), *mongoose* (за слање упита ка *MongoDB* бази), *dotenv* (за управљањем конфигурационих променљивих из *.env* фајла), *cors* (за подешавање *CORS* политике), *cookieParser* (за обраду колачића који се шаљу заједно са захтевом), *session* (за управљање сесије корисника), *passport* (за аутентификацију корисника преко *Google* налога).

```
const express = require("express");
const mongoose = require("mongoose");
const dotenv = require("dotenv");
const cors = require("cors");
const cookieParser = require("cookie-parser");
const session = require("express-session");
const passport = require("passport");
```

Слика 6.4.1.1. Приказ укључених библиотека.

Након овога врши се креирање инстанце *Express* апликације помоћу *express()* функције. Учитавају се конфигурационе променљиве из *.env* фајла. Додају се *middleware* функције користећи *app.use()* наредбу. Ове функције обрађују захтеве пре него што захтеви дођу до рута. Омогућава се обрада *JSON* података у телу захтева, обрада колачића који пристижу уз захтев, дозвољава се приступ ресурсима захтевима који нису истог порекла и врши се иницијализација *passport-a* за аутентификацију преко *Google* налога.

```
const app = express();

dotenv.config({ path: "./config.env" });
app.use(express.json());
app.use(cookieParser());
app.use(cors({ origin: true, credentials: true }));

app.use(
  session({
    secret: "secretKey",
    resave: false,
    saveUninitialized: true,
  })
);
app.use(passport.initialize());
app.use(passport.session());
```

Слика 6.4.1.2. Приказ почетног кода веб сервера

Након овога, врши се укључивање рута које одређују како ће апликација одговорити на одређене *HTTP* захтеве. Помоћу *mongoose.connect()*, апликација се повезује са *MongoDB* базом података користећи конекциони стринг који је запамћен у конфигурационом фајлу. Уколико је конекција успешно остварена, исписаће се порука у конзоли, а ако није исписаће се грешка. На крају апликација ослушкиваје долазне захтеве на порту који је дефинисан у конфигурационом фајлу и у конзоли исписује поруку да је сервер покренут.

```
const authRoute = require("./Routes/authRoutes");
const adminRoute = require("./Routes/adminRoutes");
const userRoute = require("./Routes/userRoutes");

app.use("/api/auth", authRoute);
app.use("/api/admin", adminRoute);
app.use("/api/user", userRoute);

mongoose
  .connect(process.env.DATABASE_LOCAL)
  .then(() => console.log("Successfully connected to MongoDB"))
  .catch((err) => console.error("Error connecting to MongoDB", err));

app.listen(process.env.PORT, () => {
  console.log(`server running on port ${process.env.PORT}`);
});
```

*Слика 6.4.1.3 Приказ дефинисања рута, повезивања са базом и покретање сервера*

## 6.4.2 Имплементација рута

Руте се користе како би проследиле одговарајуће пристигле захтеве клијента ка одговарајућим функцијама контролера који обрађују ове захтеве и генеришу одређене одговоре. Свака ruta дефинисана је путањом и *HTTP* методом као што је (*GET*, *PUT*, *POST*, *DELETE*). Када пристигне захтев на одређеној путањи са одређеним *HTTP* захтевом, ruta ће проследити тај захтев одређеном контролеру који ће га обрадити. Уз помоћ ruta, могуће је организовати апликацију на веома јасан и модуларан начин. Позивом *express.Router()* методе креира се инстанца рутера који се користи за дефинисање различитих ruta у апликацији.

Овај рутер убацује се у главну апликацију коришћењем *app.use()* методе. Свака од метода контролера заштићена је са *protect middleware* функцијом која проверава да ли је уз захтев пристигао *JWT* токен и да ли је токен валидан. Уколико јесте, прелази се на методу контролера, а ако није генерише се одговор у којем се саопштава клијенту да нема валидан *JWT* токен. На тај начин апликација је заштићена од неовлашћеног приступа подацима. На слици 6.6.2.1 приказане су неке од ruta апликације које преусмеравају пристигле захтеве клијената ка одређеним методама контролера.

```

const router = express.Router();

router.get(
  "/expenseLocations",
  authenticationController.protect,
  expenseController.getExpenseLocations
);

router
  .route("/photoChange")
  .put(authenticationController.protect, friendController.updateUserPhoto);

router
  .route("/friends")
  .get(authenticationController.protect, friendController.getFriendsPagination)
  .post(authenticationController.protect, friendController.addFriend)
  .delete(authenticationController.protect, friendController.removeFriend);

router
  .route("/friendsForm")
  .get(authenticationController.protect, friendController.getFriends);

router
  .route("/group")
  .get(authenticationController.protect, groupController.getGroupsPagination)
  .post(authenticationController.protect, groupController.addGroup)
  .delete(
    authenticationController.protect,
    groupController.removeUserFromGroup
  );

router
  .route("/groupForm")
  .get(authenticationController.protect, groupController.getGroups);

router
  .route("/expense")
  .get(authenticationController.protect, expenseController.getExpensePagination)
  .post(authenticationController.protect, expenseController.addExpense)
  .delete(authenticationController.protect, expenseController.deleteExpense);

module.exports = router;

```

**Слика 6.4.2.1.** Приказ неких од дефинисаних рута апликације

### 6.4.3 Дефинисане шеме и модели

Због тога што *MongoDB* база нема никаквих ограничења приликом памћења документа (документи могу бити различите структуре и да садрже различите сетове поља), неопходно је увести одређена ограничења како би се омогућила конзистентност и валидација података. *Mongoose* библиотека омогућава дефинисање стриктних шема које дефинишу структуру документа. Модели су засновани на шемама и они омогућују интеракцију са базом података и слање упита. Уз помоћ модела могуће је прибављати, додавати, ажурирати, или брисати документе из базе.

```
const userSchema = new mongoose.Schema({
  username: {
    type: String,
    required: [true, "Molimo vas unesite vaš username!"],
  },
  googleId: {
    type: String,
  },
  name: {
    type: String,
    required: [true, "Molimo vas unesite vaše ime i prezime"],
  },
  email: {
    type: String,
    required: [true, "Molimo vas unesite vaš imejl"],
    unique: [true, "Zao nam je ali ovaj imejl već postoji"],
    lowercase: true,
    validate: [validator.isEmail, "Molimo vas unesite validan imejl"],
  },
  password: {
    type: String,
    required: [true, "Molimo vas unesite šifru"],
    select: false,
    minlength: 4,
  },
  role: {
    type: String,
    enum: ["user", "admin"],
    default: "user",
  },
  photo: {
    type: String,
    default: "https://api.dicebear.com/7.x/adventurer/svg?seed=Midnight",
  },
  friends: [
    {
      friend: {
        type: mongoose.Schema.Types.ObjectId,
        ref: "User",
      },
      balance: {
        type: Number,
        default: 0,
      },
    },
  ],
  createdAt: {
    type: Date,
    default: Date.now,
  },
});
```

Слика 6.4.3.1. Пример дефинисане шеме

На слици 6.6.3.1 приказана је шема корисника у којој је дефинисана структура документа који ће се памтити у бази у колекцији users. Памте се основни подаци о кориснику као што су: корисничко име, име, имејл, шифра, улога, слика, *googleId* (уколико је корисник пријављен преко google налога) и низ пријатеља. Овај низ садржи референце ка корисницима који су пријатељи и садржи поље *balance* у којем се памти дуговање према том пријатељу. Такође памти се и датум регистрације корисника на апликацију.

*Mongoose* библиотека омогућава дефинисање виртуалних референци између докумената. Уместо да се унутар документа памте референце ка групама у којима је корисник, помоћу *virtual* методе могуће је дефинисати виртуелну везу између корисника и групе, и онда када су потребне информације о групама у којима је корисник помоћу поља *groups* и методе *populate* можемо прибавити све неопходне податке о групама из базе. На крају *User* модел се креира позивом *mongoose.model* методе и помоћу овог модела могуће је извршити интеракцију са колекцијом *users* у *MongoDB* бази.

```
userSchema.virtual("groups", {
  ref: "Group",
  foreignField: "members.member",
  localField: "_id",
});

userSchema.virtual("expenses", {
  ref: "Expense",
  foreignField: "members",
  localField: "_id",
});

userSchema.virtual("expensePaid", {
  ref: "Expense",
  foreignField: "paidby",
  localField: "_id",
});

userSchema.set("toJSON", { virtuals: true });
userSchema.set("toObject", { virtuals: true });

const User = mongoose.model("User", userSchema);
module.exports = User;
```

Слика 6.4.3.2. Пример виртуелних референци и креирања модела

#### 6.4.4 Имплементација метода контролера

Контролери омогућавају централизацију пословне логике апликације. Логика се пише унутар контролера уместо у осталим деловима апликације. На тај начин омогућена је боља организација апликације. Контролери врше обраду долазних *HTTP* захтева и генерисање одговарајућих *HTTP* одговора. Сваки контролер је фокусиран на одређени део апликације и њену функционалност. У наставку приказана је имплементација неких метода контролера *FairShare* апликације.

На слици 6.4.4.1 приказана је имплементација методе *getFriends* која се користи за прибављање информација о пријатељима за одређеног корисника апликације. Функција је асинхронна и извршава се у позадини како не би дошло до блокирања главне нити извршења. На почетку из *query* стринг-а прибавља се идентификатор корисника чије је пријатеље неопходно прибавити. Користи се *Mongoose* модел „*UserModel*“ помоћу којег се коришћењем методе *findById* из базе прибавља корисник са датим идентификатором. Метода *populate* се користи за попуњавање референци подацима, у овом случају референцу *friends.friend* попуњавамо подацима. Добра ствар је та што је могуће изабрати које податке је потребно прибавити и тиме спречити прибављање целог документа већ прибавити само захтевана поља.

```
exports.getFriends = async (req, res) => {
  try {
    const { userId } = req.query;
    const userWithFriends = await UserModel.findById(userId)
      .populate("friends.friend", "name username email photo")
      .exec();

    if (!userWithFriends) {
      return res.status(404).send({ message: "User not found" });
    }
    const friends = userWithFriends.friends.map((friend) => ({
      friend: {
        id: friend.friend._id,
        username: friend.friend.username,
        email: friend.friend.email,
        name: friend.friend.name,
        photo: friend.friend.photo,
      },
    }));
    res.send({ friends });
  } catch (error) {
    console.error(error);
    res
      .status(500)
      .send({ message: "Internal Server Error", error: error.message });
  }
};
```

Слика 6.4.4.1. Пример имплементације контролера за прибављање пријатеља

Једна од јако битних метода ове апликације јесте функција за заштиту ресурса од неовлашћеног приступа. Она се користи као *middleware* у апликацији и њена сврха је да провери да ли корисник који је послао захтев има валидан *JWT (JSON Web Token)* пре него што приступи одређеним рутама и ресурсима апликације. На почетку проверава се да ли је уз захтев пристигао и *jwt cookie* у којем се налази токен. Уколико није, то значи да корисник није пријављен на апликацију и функција ће послати поруку о грешци назад кориснику. Ако *cookie* постоји онда ће се обавити његово декодирање и верификација. Уколико је токен валидан, добиће се објекат који садржи идентификатор корисника који је послао захтев и извршиће се провера да ли корисник постоји у бази. Уколико корисник не постоји у бази, функција ће послати поруку о грешци, а ако постоји онда ће се његов објекат додати у *req* објекат као *req.user* и позваће се *next()* помоћу којег се прелази на извршење следећег *middleware-a* у ланцу.

```
exports.protect = async (req, res, next) => {
  try {
    if (!req.cookies.jwt) {
      return next(new Error("Niste ulogovani, nemate token", 401));
    }
    const decoded = await promisify(jwt.verify)(
      req.cookies.jwt,
      process.env.JWT_SECRET
    );

    const currentUser = await User.findById(decoded.id);
    if (!currentUser) {
      return next(new Error("Korisnik sa datim tokenom ne postoji", 401));
    }
    req.user = currentUser;
    next();
  } catch (error) {
    console.error(error);
    res.status(400).json({
      status: "fail",
      message: "Greska prilikom autentikacije",
      error: error.message,
    });
  }
};
```

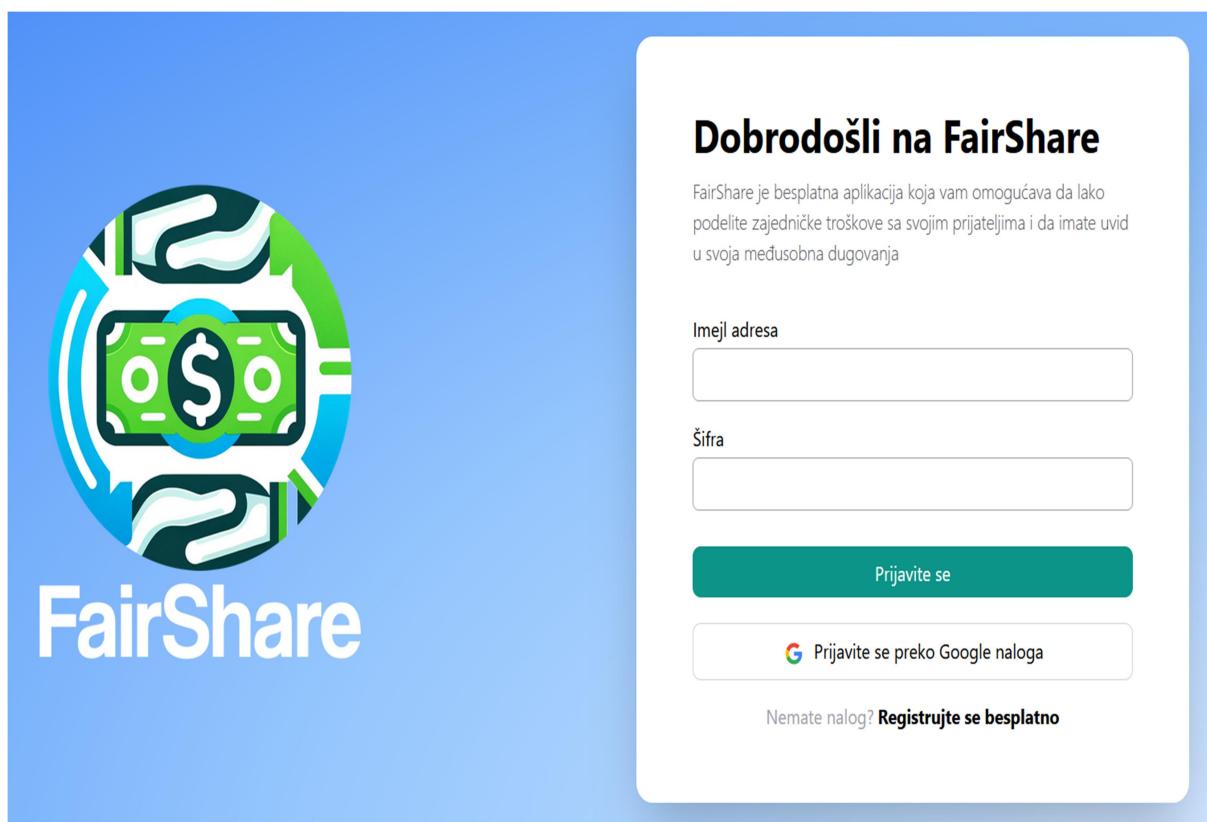
Слика 6.4.4.2. Имплементације функције за заштиту пута

## 7. Приказ рада апликације

У овом поглављу приказан је рад апликације и кориснички интерфејс којем корисник приступа. Дизајн апликације рађен је коришћењем *Tailwind* програмског оквира за *CSS* као и коришћењем додатних библиотека које нуде готове компоненте, као што су: *React Icons*, *React Google Charts* и *React Leaflet*.

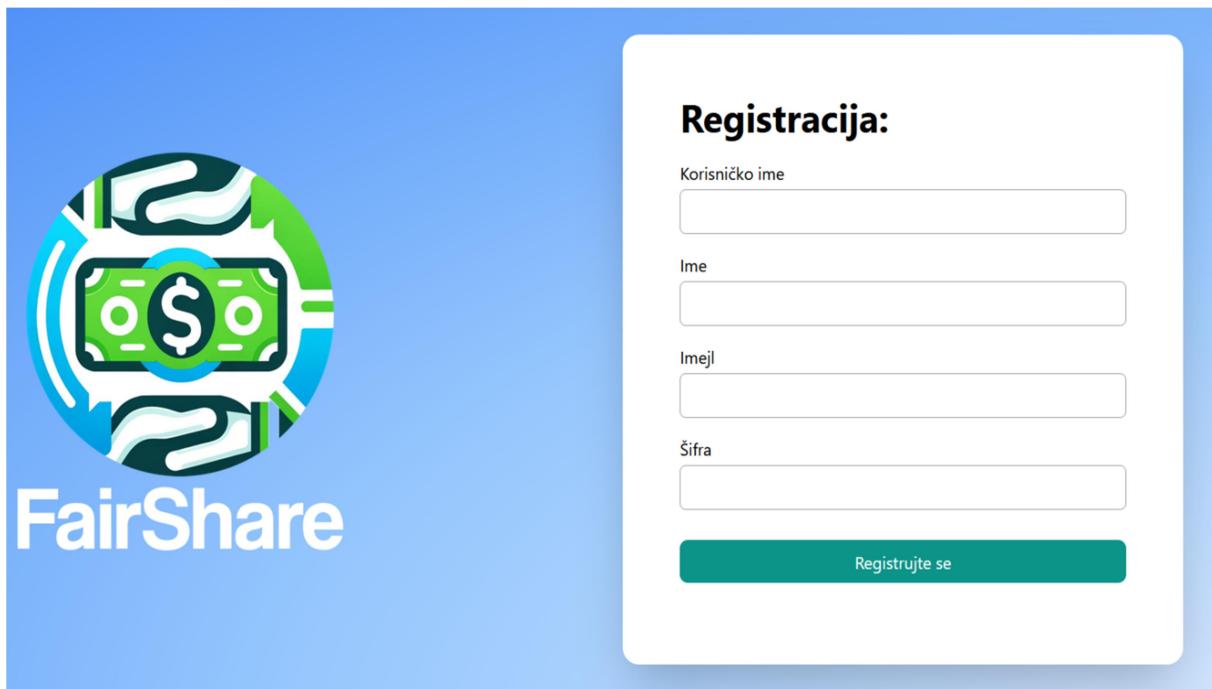
### 7.1 Почетна страница

Када корисник приступа апликацији, прва страница која се приказује је страница за пријаву на апликацију. На овој страници налазе се основне информације о апликацији, лого апликације и форма коју корисник попуњује како би приступио. Уколико корисник има налог, може се пријавити путем имејл-а и шифре или се пријавити коришћењем *Google* налога. Приказ почетне странице и форме за пријављивање налази се на слици 7.1.1.



Слика 7.1.1. Приказ почетне странице апликације.

Уколико корисник нема налог онда има опцију за регистровање. Кликом на линк “*Registrujte se besplatno*” отвориће се страница за регистрацију (Слика 7.1.2). Како би се корисник успешно регистровао неопходно је да попуни сва поља. Поља која се налазе на форми за регистрацију су: корисничко име, име, имејл адреса, лозинка. Корисничко име и имејл адреса које се наводе морају бити јединствене. Уколико је регистрација прошла успешно, кориснику ће се направити налог, доделиће му се *JWT* токен и он може приступити садржају апликације.



Слика 7.1.2 Приказ странице за регистрацију

## 7.2 Главна страница апликације и администрација

Након успешне пријаве на апликацију, кориснику се приказује главни садржај апликације. Са леве стране налази се навигациони бар преко којег корисник може приступити различитим деловима апликације. На навигационом бару налазе се различите опције. Кликом на сваку од ових опција отвориће се нова форма са десне стране на којој ће се, у зависности од тога на коју је корисник кликнуо, приказати одређени садржај апликације.

Слика 7.2.1 Приказ главне странице апликације

На почетку, кориснику се отвара профил његовог налога где он има увид у своје основне податке ( слика 7.2.1). Корисник може променити своју профилну слику и изабрати неку од многобројних слика коју Giphy API нуди. Уколико је корисник админ, приказаће му се дугме „*Administracija*“ преко којег може приступити заштићеној страници апликације. Ова страница је скривена корисницима који нису админ-и. Корисници који су админ-и могу приступити администрацији кликом на дугме „*Administracija*“ и на тај начин приступити листи свих корисника који су регистровати на апликацију. Админ може извршити претрагу корисника и резултат претраге може сортирати у растућем или опадајућем редоследу на основу различитих критеријума. Такође он може и обрисати налог било ког корисника који није админ. Приказ садржаја администрације налоги се на слици 7.2.2. Навигациони бар је могуће приказати или уклонити кликом на дугме у доњем десном углу.

The screenshot shows the FairShare application interface. On the left is a dark sidebar with white text and icons for navigation: Prijatelji, Dodaj prijatelja, Grupe, Napravi grupu, Troškovi, Dodaj trošak, Analitika, Mapa troškova, and Logout. At the bottom of the sidebar is the copyright notice © FairShare 2024.

The main content area has a light gray background. At the top, there are two input fields: 'Korisnici:' with a placeholder 'Pretraga...' and 'Sortiranje:' with a dropdown menu set to 'Korisnik'. Below these are two buttons: '↑ Rastuće' and '↓ Opadajuće'.

A table lists five users:

KORISNIK	KORISNIČKO IME	IMEJL	DATUM KREIRANJA	POZICIJA	OPCIJA
Andrija Miletic	Andrija	andrijamiletic@yahoo.com	2/17/2024	user	<button>Obriši</button>
Dimitrije Stankovic	Dimitrije	dimitrijestankovic@gmail.com	2/14/2024	user	<button>Obriši</button>
Ivan Lukic	Ivan	ivanlukic@yahoo.com	2/11/2024	user	<button>Obriši</button>
Jelena Jovanovic	Jelena	jelenajovanovic1@gmail.com	2/13/2024	user	<button>Obriši</button>
Jovan Jovanovic	Jovan123	jovanjovanovic@yahoo.com	2/11/2024	user	<button>Obriši</button>

At the bottom of the main content area, it says 'Stranica 1 od 5 Ukupno 24 Korisnika'. Below the table are two buttons: 'Nazad' and 'Sledeca'. A dark circular icon is located in the bottom right corner of the main content area.

**Слика 7.2.2. Приказ администрације**

## 7.3 Опција за приказ пријатеља

Сваки пријављени корисник има увид у своју листу пријатеља. Корисник може имати неограничен број пријатеља. Сваки пријатељ је неки постојећи корисник апликације. Уколико се из неког разлога налог корисника избрише, тај корисник ће бити изbrisан из листа пријатеља свих осталих корисника апликације. Листе пријатеља заштићене су од неауторизованог приступа, тако да листи неког корисника само тај корисник може приступити. Корисник има увид у дуговања са својим пријатељима. Уколико корисник дугује неком пријатељу, текст ће бити црвене боје и биће исписана сума коју корисник дугује. Уколико пријатељ дугује кориснику, онда ће текст бити зелене боје и биће исписана сума коју пријатељ дугује. Корисник може избрисати пријатеља из своје листе и на тај начин ће се такође из базе избрисати и све трансакције које су ове две особе имале. Апликација нуди опцију претраге пријатеља на основу имена пријатеља или њиховог корисничког имена. Такође постоји и опција сортирања у растућем или опадајућем редоследу на основу следећих критеријума: име, корисничко име, имејл, дуговање. Пrikaz листе пријатеља налази се на слици 7.3.1.

KORISNIK	KORISNIČKO IME	IMEJL	DUGOVANJE	OPCIJA	
	Marko Cvetković	Marko	mare123@yahoo.com	Duguješ 763 RSD	Obrisi prijatelja
	Milica Jovanović	Milica	milica97@gmail.com	Duguje ti 1118 RSD	Obrisi prijatelja
	Nikola Nikolić	Nikolica	nikola123@gmail.com	Duguješ 500 RSD	Obrisi prijatelja
	Sanja Aleksić	Sanja	sanjica@yahoo.com	Duguje ti 375 RSD	Obrisi prijatelja
	Uroš Stanojević	Uroš	urkestanoje@yahoo.com	Duguje ti 300 RSD	Obrisi prijatelja

Слика 7.3.1. Пrikaz корисникове листе пријатеља

## 7.4 Опција додај пријатеља

Сваки корисник има опцију додавања пријатеља (других корисника) у своју листу пријатеља. Додавање се врши на основу имејл адресе корисника. Уколико пријатељ већ постоји у његовој листи, исписаће се порука да пријатељ постоји и да га није могуће поново додати. Ако не постоји корисник у бази са датом имејл адресом исписаће се порука да корисник не постоји. Уколико додавање прође успешно, исписаће се порука да је корисник успешно додат у његову листу пријатеља. Приказ форме за додавање пријатеља налази се на слици 7.4.1.

The screenshot shows the FairShare mobile application's user interface. On the left, there is a vertical navigation menu with the following items: 'Prijatelji' (Friends), which is highlighted in blue; 'Dodaj prijatelja' (Add friend), which is also highlighted in blue; 'Grupe' (Groups); 'Napravi grupu' (Create group); 'Troškovi' (Expenses); 'Dodaj trošak' (Add expense); 'Analitika' (Analytics); 'Mapa troškova' (Expense map); and 'Logout'. At the bottom of the menu, it says '© FairShare 2024'. On the right, a modal window titled 'Unesite email prijatelja:' (Enter friend's email address:) is displayed. It contains a text input field and a dark blue 'Dodaj prijatelja' (Add friend) button.

Слика 7.4.1. Приказ форме за додавање пријатеља.

## 7.5 Опција за приказ група

Сваки корисник може бити члан једног или већег броја група. Групе су осмишљене како би корисник лакше могао да дели своје трошкове са одређеним скупом пријатеља. Корисник има увид у укупну потрошњу која је остварена у групи као и у чланове те групе. Такође корисник може напустити групу, а уколико је он последњи члан те групе она ће аутоматски бити изbrisана. Апликација нуди претрагу група по њиховом називу као и сортирање у растућем и опадајућем редоследу по критеријумима као што су: назив, категорија, опис, укупан трошак. Корисник такође може видети и колико је сваки члан групе учествовао у укупној потрошњи. Приказ листе група налази се на слици 7.5.1.

ČLANOVI	NAZIV	KATEGORIJA	OPIS	UKUPAN TROŠAK	OPCIJA
Članovi	Društvo iz srednje	Izlazak	Grupa namenjena izlascima sa društvom iz srednje škole.	2500.00 RSD	<button>Napusti grupu</button>
Članovi	Kolege sa posla	Posao	U ovoj grupi nalaze se kolege sa posla.	11103.00 RSD	<button>Napusti grupu</button>
Članovi	Putovanje2024	Putovanje	Grupa u kojoj delimo troškove sa putovanja u Italiju.	9373.17 RSD	<button>Napusti grupu</button>
Članovi	Superbowl2024	Sport	Gledanje finala američkog fudbala.	5701.00 RSD	<button>Napusti grupu</button>

Слика 7.5.1. Приказ корисникова листе група

## 7.6 Опција направи групу

Сваки корисник апликације има могућност креирања нове групе. Како би креирао групу он мора да попуни сва поља у форми. Форма садржи следећа поља: Име, Категорија, Опис и листа пријатеља. Корисник из своје листе пријатеља бира оне пријатеље који ће бити чланови нове групе. Уколико корисник заборави да попуни неко поље или дође до грешке, биће приказана порука о грешци. Ако је група успешно направљена, приказаће се порука да је група успешно направљена. Такође уколико неки члан те групе нема остале чланове групе као пријатеље, апликације ће аутоматски додати те кориснике у његову листу пријатеља. Приказ форме за креирање група налази се на слици 7.6.1.

The screenshot shows the FairShare mobile application interface. On the left, there's a sidebar with the following menu items:

- Prijatelji
- Dodaj prijatelja
- Grupe
- Napravi grupu** (highlighted in blue)
- Troškovi
- Dodaj trošak
- Analitika
- Mapa troškova
- Logout

Below the sidebar, the copyright notice reads: © FairShare 2024.

The main content area displays a form for creating a group:

- Ime (Name): An empty input field.
- Kategorija (Category): An empty input field.
- Opis (Description): A large empty input field.
- Članovi (Members): A list of five users with their names and profile icons:
  - Sanja
  - Marko
  - Mateja
  - Uroš
  - Milica

At the bottom right of the form is a large blue button labeled "Napravi grupu" (Create group).

*Слика 7.6.1. Приказ форме за креирање група*

## 7.7 Опција за приказ трошкова

Сваки корисник апликације има увид у своје трошкове. Сваки трошак има категорију, вредност, опис, датум настанка, корисника који је платио, потрошаче (особе које учествују у трошку) и групу уколико је трошак начињен на нивоу групе. Корисник има могућност претраге трошкова или сортирање у растућем или опадајућем редоследу на основу различитих критеријума. Такође, он може видети све потрошаче тог трошка и колики је свачији део. Постоји и могућност брисања трошка. Уколико се трошак избрише, сва дуговања са његовим пријатељима ће се ажурирати.

POTROŠAČI	KATEGORIJA	VREDNOST	DATUM	KO JE PLATIO	GRUPA	OPIS	OPCIJA
Potrošači	Bioskop	2000.00 RSD	2/13/2024	Vukašin Branković	Društvo iz srednje	Vukašin je platio karte za sve nas!	Obrisi trošak
Potrošači	Bioskop	1555.00 RSD	2/14/2024	Zoran	-/-	Zoran je platio karte za bioskop.	Obrisi trošak
Potrošači	Hrana	5672.00 RSD	2/17/2024	Vukašin Branković	Kolege sa posla	Vukašin je naručio hranu za sve nas!	Obrisi trošak
Potrošači	Hrana	4231.00 RSD	2/17/2024	Ivan	Kolege sa posla	Ivan je preko Wolt-a naručio hranu za sve	Obrisi trošak

Stranica 1 od 4 Ukupno 18 Troškova

Nazad Sledeca

**Слика 7.7.1.** Приказ корисникова листе трошкова

## 7.8 Опција за додавање трошка

Корисник апликације може додавати и делити различите трошкове. Кликом на опцију „*Dodaj trošak*“ са десне стране отвориће се форма коју корисник попуњава. Неопходно је попунити сва поља како би се трошак успешно додао и поделио. Форма садржи следећа поља: *Kategorija* (категорија трошка), *Valuta* (избор једне од мноштво валута које апликација нуди), *Vrednost* (вредност трошка), *Vrednost računa u izabranoj valuti* (у зависности коју валуту изабере апликација ће аутоматски конвертовати износ у динаре како би се трошак лакше поделио), *Lokacija* (корисник на мапи може изабрати локацију где је трошак настао), *Grupa* (овде корисник може одабрати једну од група уколико трошак дели са групом), *Datum* (датум настанка трошка, уколико није изабрано узима се данашњи датум), *Lista prijatelja* (уколико група није изабрана, кориснику ће се приказати његова листа пријатеља из које може бирати једну или више особа са којом ће поделити свој трошак). *Ko je platilo* (овде корисник бира особу која је платила трошак, уколико је трошак на нивоу групе приказаће се сви чланови групе, ако трошак дели са пријатељима, приказаће се сви селектовани пријатељи), *Opis* (овде корисник наводи опис трошка), *Svačiji deo* (апликација ће аутоматски поделити трошак на једнаке делове и корисник може видети колики је свачији део). Приказ форме за додавање трошка налази се на слици 7.8.1.

The screenshot shows the FairShare mobile application interface. On the left is a dark sidebar with the following menu items:

- Prijatelji
- Dodaj prijatelja
- Grupe
- Napravi grupu
- Troškovi
- Dodaj trošak** (highlighted in blue)
- Analitika
- Mapa troškova
- Logout

On the right is the main content area for adding a transaction:

**Kategorija:** [Empty input field]

**Valuta:** RSD [Dropdown menu]

**Vrednost računa u izabranoj valuti:** 0 [Input field with up/down arrows]

**Vrednost računa konvertovana u RSD:** 0.00 [Input field]

**Lokacija nastanka računa:** [Large button]

**Grupa:** BEZ GRUPE [Dropdown menu]

**Datum:** mm / dd / yyyy [Input field with calendar icon]

**Koristite Ctrl+click za selekciju više prijatelja:**

A list of friends with small profile icons next to their names:

- Sanja
- Marko
- Mateja
- Uroš
- Milica

**Ko je platio?** Vukašin Branković [Dropdown menu]

**Opis:** [Large input field]

**Svačiji deo:** 0.00 RSD [Input field]

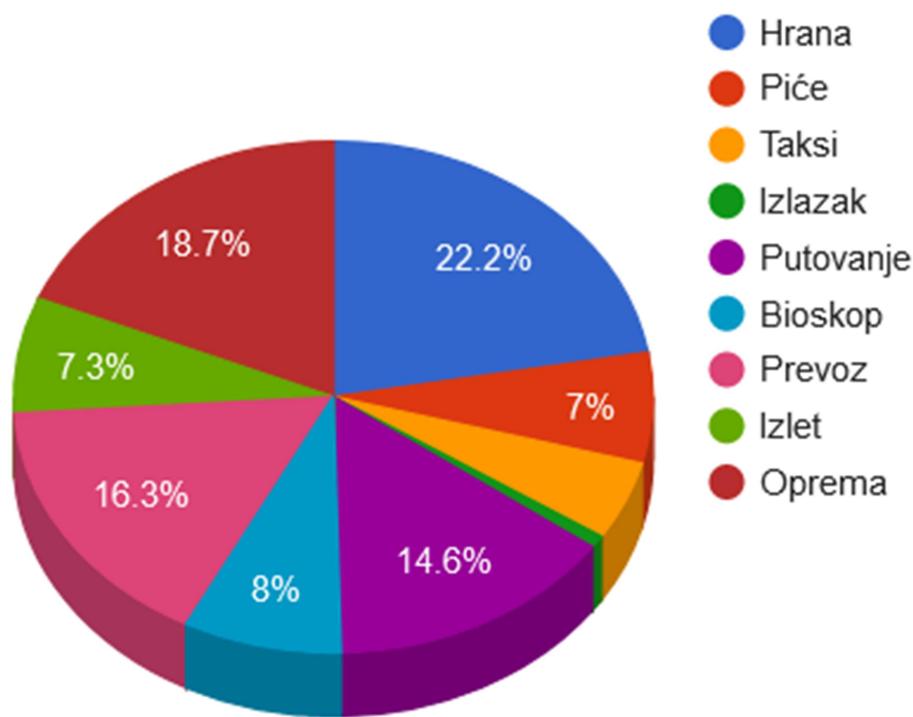
**Podeli trošak** [Large button]

*Слика 7.8.1. Приказ форме за додавање трошка*

## 7.9 Опција за аналитику

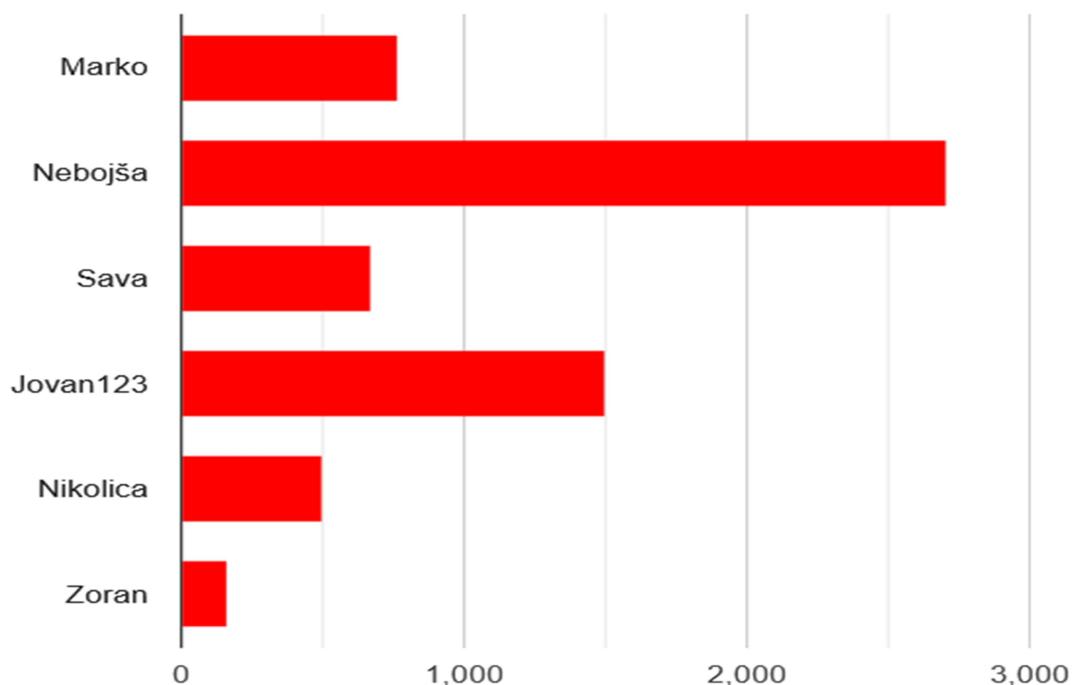
Корисник има увид у графички приказ дуговања и потрошње по групама путем различитих графика. Библиотека *React Google Charts* је коришћена како би се омогућио графички приказ. Кориснику се приказују четири графика. Преко првог графика корисник може видети процентуалну расподелу трошкова по категоријама и колико је пара потрошено на свакој од категорија. На другом графику корисник има приказ његових дуговања према пријатељима. На трећем графику је приказ дуговања који његови пријатељи имају према њему. Док на четвртом графику корисник има приказ трошкова по групама. Он може видети укупну потрошњу у групи, део потрошње коју је он платио и део коју су остали чланови групе платили. Приказ свих графика налазе се на сликама: (7.9.1, 7.9.2, 7.9.3 и 7.9.4).

**Procentualna raspodela vaših troškova po kategorijama**



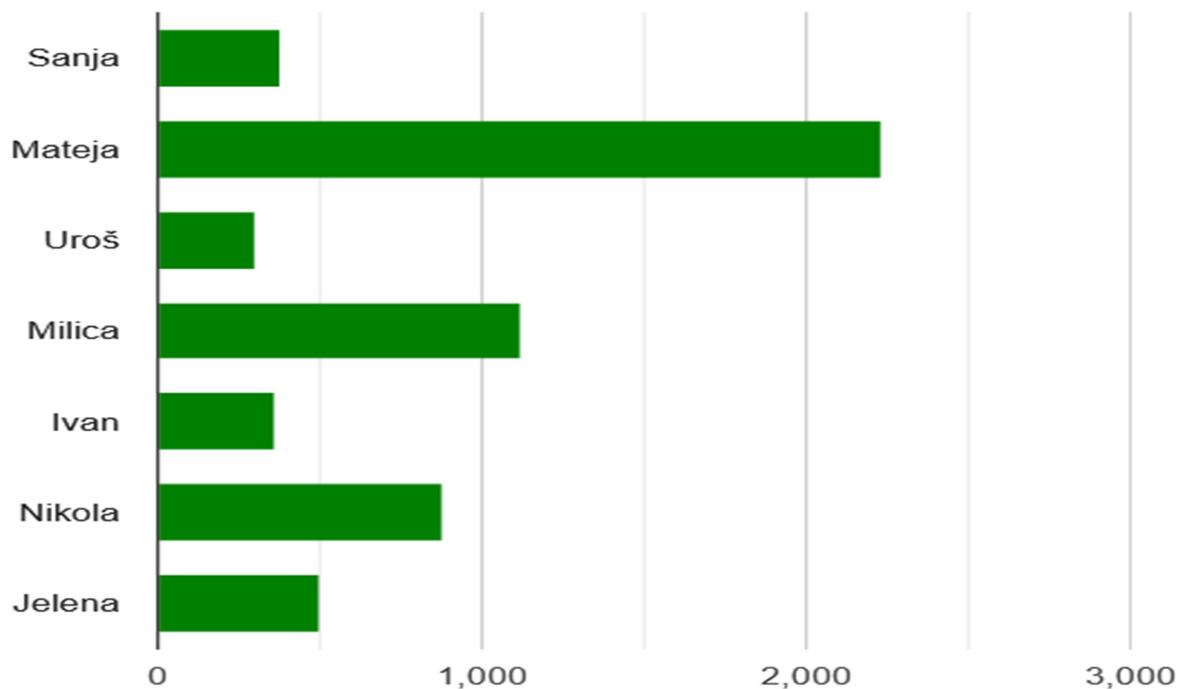
*Слика 7.9.1. Графикон за приказ процентуалне расподеле трошкова*

**Prikaz vaših dugovanja prema prijateljima**

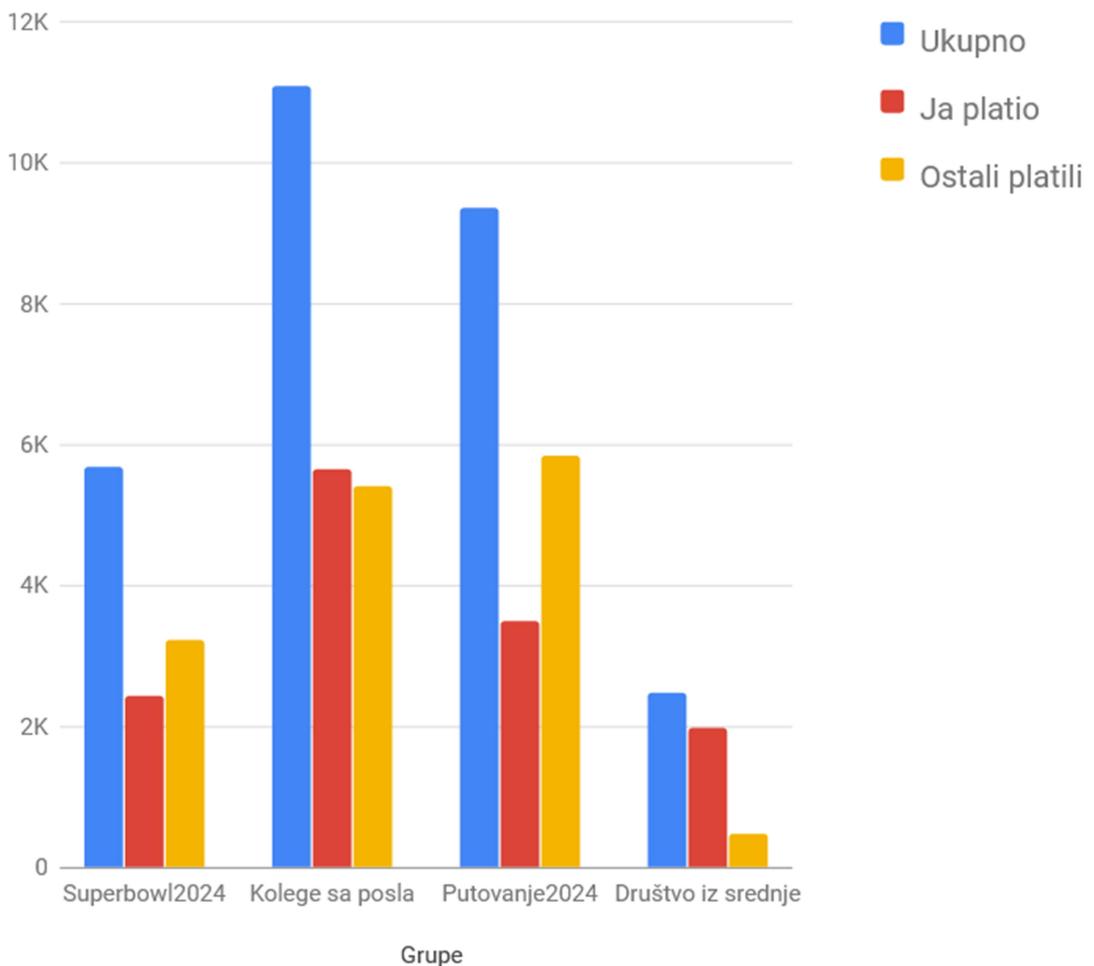


*Слика 7.9.2. Графикон за приказ дуговања корисника према пријатељима*

**Prikaz dugovanja vaših prijatelja**



*Слика 7.9.3. Графикон за приказ дуговања пријатеља према кориснику*



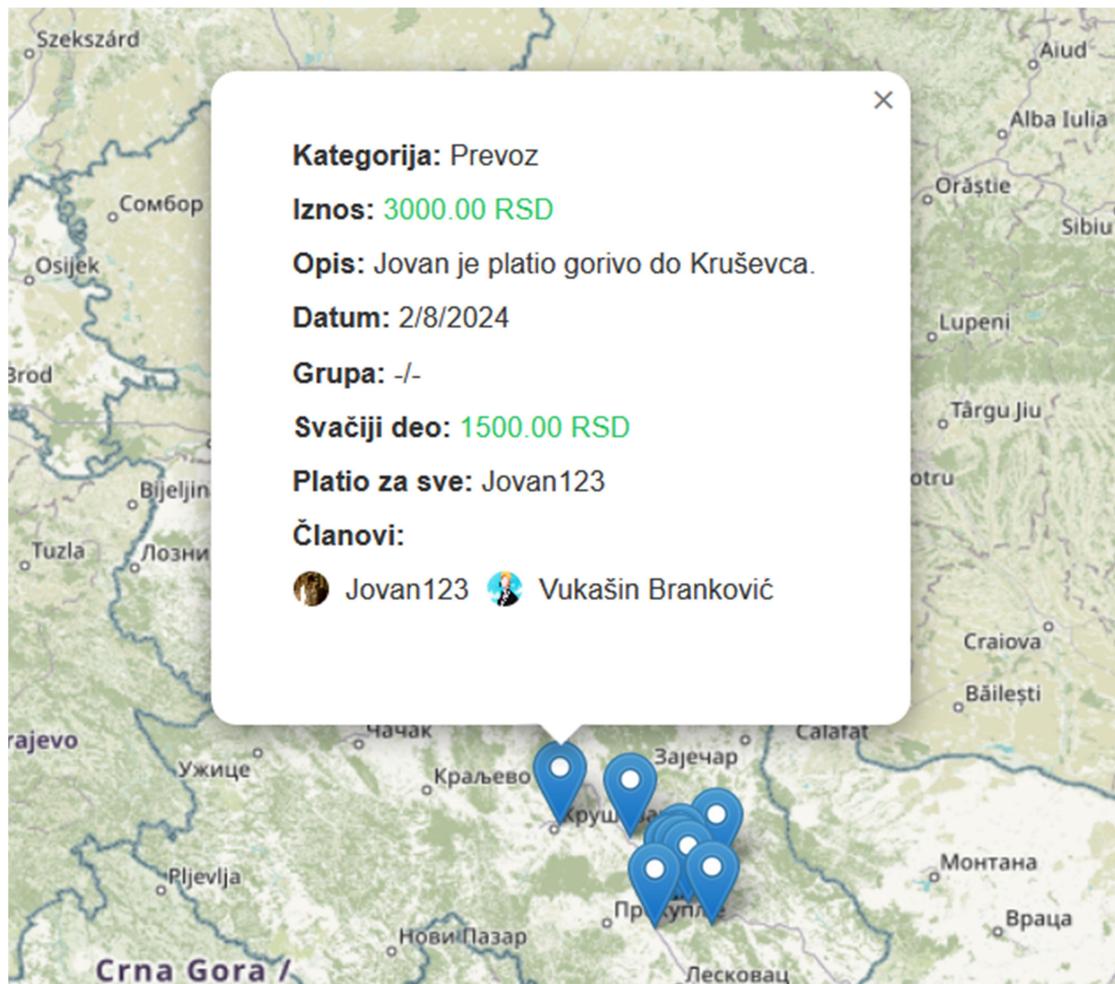
*Слика 7.9.4. Графикон за приказ потрошње по групама*

## 7.10 Опција за приказ мапе трошкова

Сваки корисник апликације приликом уноса трошка уноси и локацију настанка трошка. Корисник кликом на опцију „*Мапа трошкова*“ може приступити интерактивној мапи и на њој може видети све локације на којима је био и делио своје трошкове са пријатељима. Библиотека *React Leaflet* је коришћена за приказ интерактивне мапе. На мапи, за сваки унети трошак приказан је један маркер (слика 7.10.2). Корисник може кликнути на тај маркер и онда се отвора прозор у којем се налазе информације везане за сам трошак, као што су: категорија, износ, опис, датум, да ли је трошак групни или није, колики је свачији део, ко је платио и ко су потрошачи. На слици 7.10.1. приказан је изглед интерактивне мапе.

The image shows the FairShare mobile application's main screen. On the left, a vertical navigation bar lists various features: Prijatelji, Dodaj prijatelja, Grupe, Napravi grupu, Troškovi, Dodaj trošak, Analitika, and Mapa troškova (which is highlighted with a blue background). At the bottom of this bar are Logout and Copyright information. The right side of the screen displays a map of Europe with a blue location marker centered over Serbia. The map includes labels for countries and regions in multiple languages.

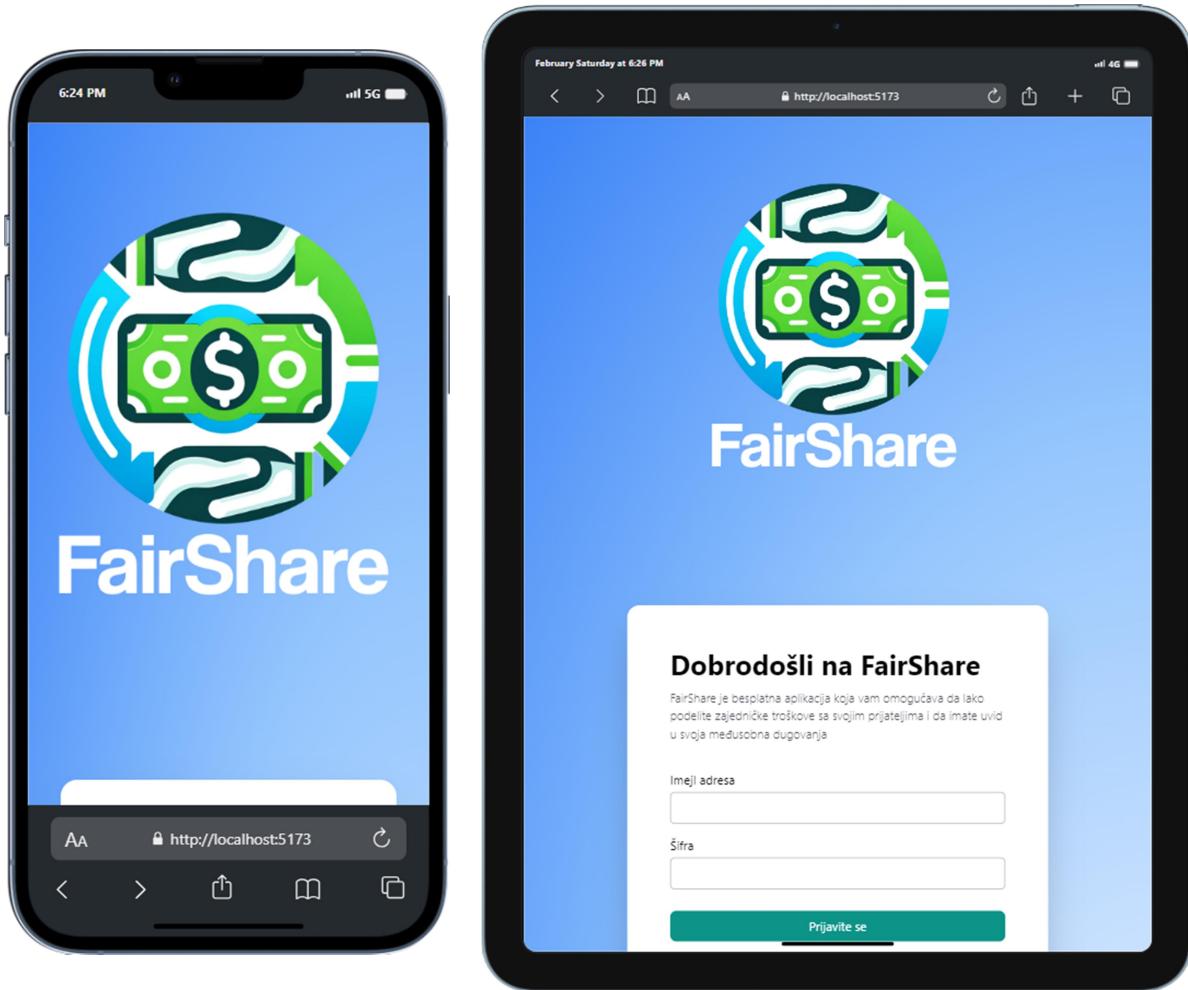
**Слика 7.10.1.** Приказ интерактивне мапе трошкова



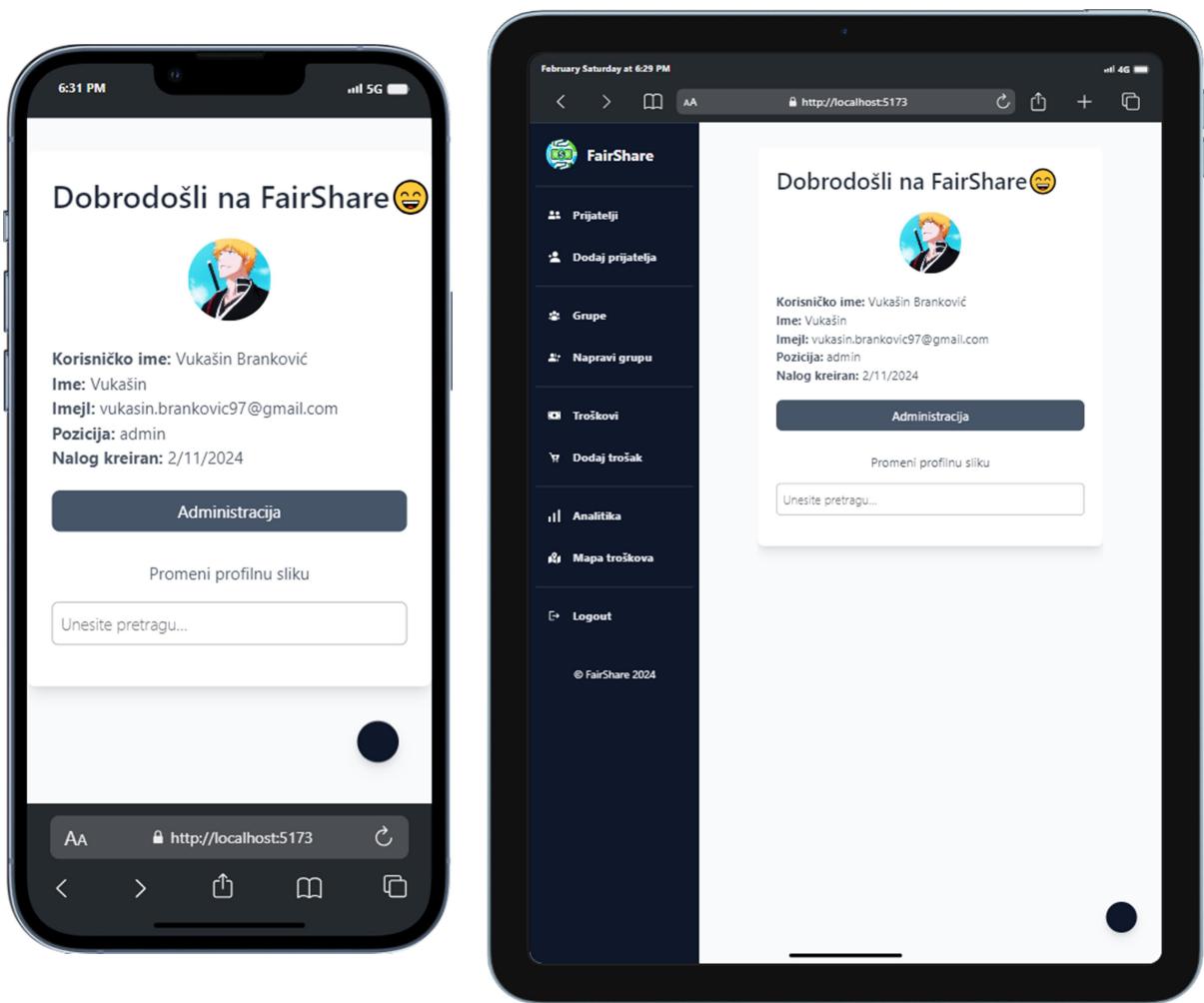
*Слика 7.10.2. Приказ прозора који се отвара кликом на плави маркер*

## 7.11. Интерфејс апликације на малим екранима

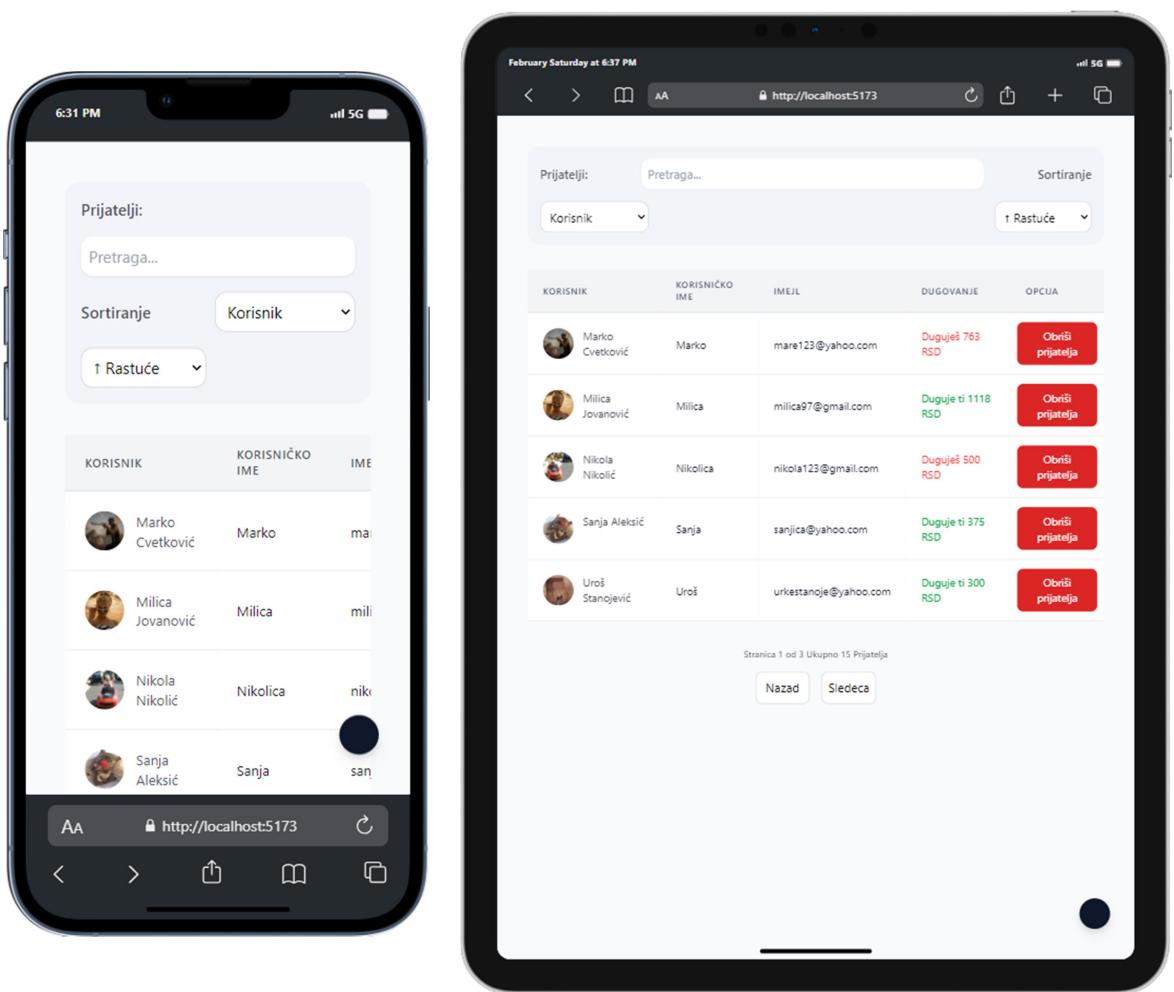
Захваљујући програмском оквиру *Tailwind*, апликација пружа леп и динамичан интерфејс и на малим екранима. *Tailwind* омогућава коришћење такозваних „преломних тачака“. Односно омогућава дефинисање стилова који ће се применити када ширина прозора буде прешла одређену преломну тачку. Постоје пет преломних тачака међутим могуће је убацити и нове преломне тачке и модификовати постојеће. *FairShare* апликација је рађена тако да је корисници могу користити на било којим уређајима и резолуцијама екрана. У доњем десном углу налази се дугме уз помоћ којег се навигациони бар приказује или склања. Уклањање навигационог бара омогућује лепши приказ на мањим екранима. На следећим сликама приказан је изглед одређених делова апликације на мобилним и таблет уређајима. За приказ коришћена је *Google Chrome* екstenзија *Mobile simulator*.



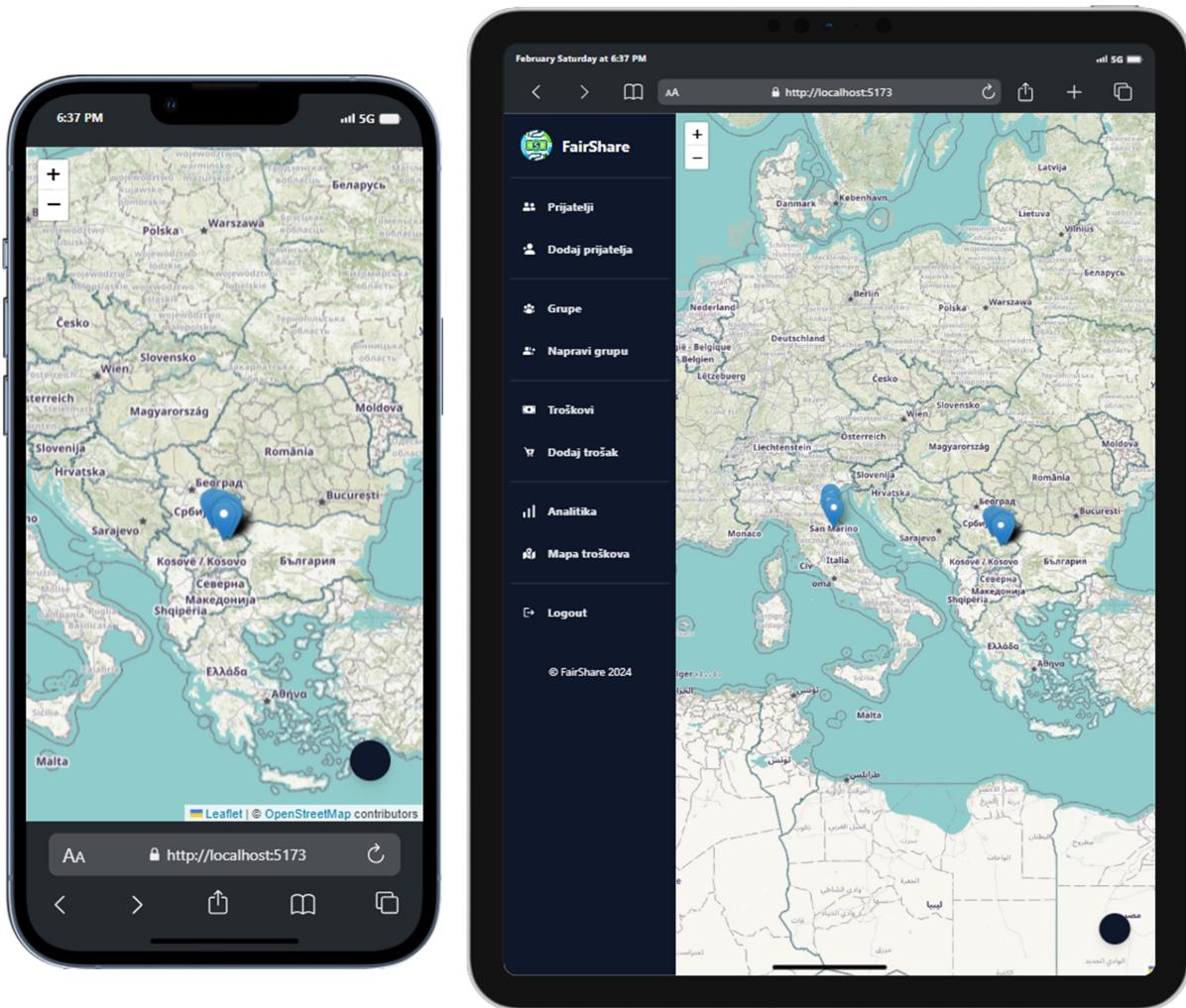
**Слика 7.11.1.** Приказ странице за пријављивање на мобилним и таблет уређајима



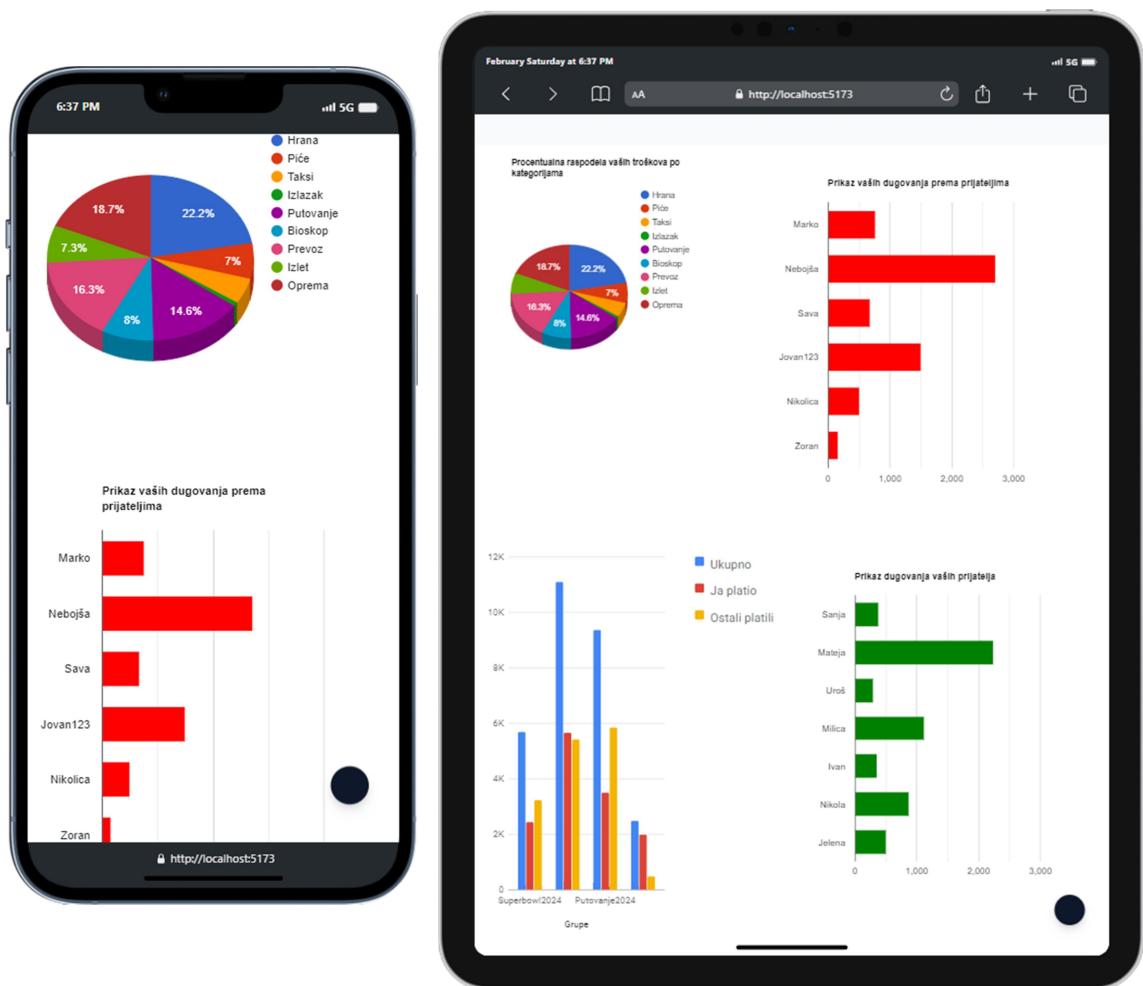
Слика 7.11.2. Приказ почетне странице апликације на мобилним и таблет уређајима



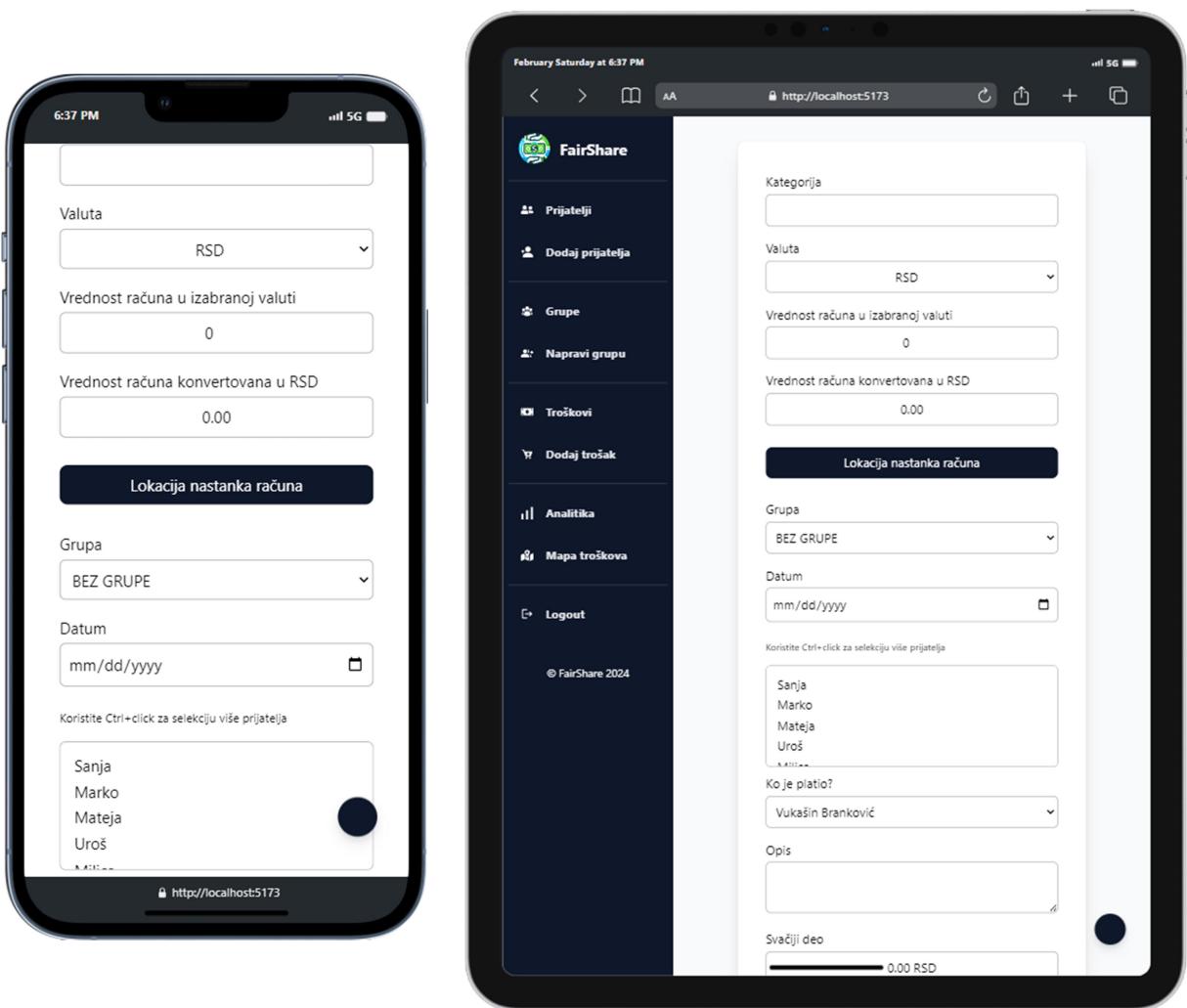
**Слика 7.11.3.** Приказ листе пријатеља и дуговања на мобилним и таблет уређајима



**Слика 7.11.4.** Приказ мапе трошкова на мобилним и таблет уређајима



*Слика 7.11.5. Приказ аналитике на мобилним и таблет уређајима*



**Слика 7.11.6.** Приказ форме за додавање трошкова на мобилним и таблет уређајима

## 8. Закључак

У овом раду представљена је апликација за поделу трошкова која је имплементирана коришћењем популарног скупа *MERN* технологија. Извршена је анализа коришћених технологија и техника које су употребљене у реализацији пројекта.

Ова апликација омогућава корисницима лаку поделу трошкова са пријатељима у различитим валутама, поред тога корисници имају увид у своја међусобна дуговања и детаље везане за трошкове. Ово олакшава корисницима управљање њиховим финансијама. Са друге стране, апликација ће омогућити велику транспарентност приликом поделе трошкова, јер свако може видети ко је колико допринео у заједничком трошку. Такође апликација може унапредити пријатељске односе међу корисницима тако што ће смањити могућност настанка конфликата приликом поделе заједничких трошкова.

Приказану апликацију могуће је унапредити и побољшати. Један од корака за даљи развој могао би бити додавање могућности дописивања путем чета између корисника апликације. На тај начин корисници би се лакше организовали око поделе заједничких трошкова. Такође, могуће је побољшати интерфејс апликације коришћењем неких додатних библиотека за *Tailwind* које нуде готове компоненте, као што су *MaterialUI* и *daisyUI*. Још један од начина за даљи развој јесте интеграција са платним системом где ће се корисницима омогућити измиривање дуговања путем разних платних начина, на пример употребом *Sripe* микросервиса.

## 9. Референце

- [1] “MERN Stack in Web App Development | Ramotion Branding Agency,” *Web Design, UI/UX, Branding, and App Development Blog*, Dec. 20, 2022.  
<https://www.ramotion.com/blog/what-is-mern-stack/> (pristupano 19.1.2024).
- [2] A. Gillis, “What is MongoDB? A definition from WhatIs.com,” *SearchDataManagement*.  
<https://www.techtarget.com/searchdatamanagement/definition/MongoDB> (pristupano 21.1.2024).
- [3] Vasan Subramanian, and Springerlink (Online Service. *Pro MERN Stack : Full Stack Web App Development with Mongo, Express, React, and Node*. Berkeley, Ca, Apress, 2019.
- [4] Udemy course, Node.js, Express, MongoDB: The complete Bootcamp, Jonas Schmedtmann, <https://www.udemy.com/course/nodejs-express-mongodb-bootcamp>, (pristupano 22.1.2024)
- [5] “The client/server model,” *www.ibm.com*. <https://www.ibm.com/docs/en/cics-ts/6.1?topic=programs-clientserver-model> (pristupano 24.1. 2024).
- [6] T. Nguyen, “HTTP Status Codes,” *Auto Test With Tuyen*, May 14, 2021.  
<https://www.automatedtestingwithtuyen.com/post/http-status-codes> (pristupano 25.1. 2024).
- [7] A. Banks and E. Porcello, *Learning React : modern patterns for developing React apps*. Sebastopol, CA: O'Reilly Media, 2020.
- [8] Udemy, The Ultimate React Course, Jonas Schmedtmann,  
<https://www.udemy.com/course/the-ultimate-react-course>, (pristupano 27.1.2024)
- [9] “angular vs react vs vue | npm trends,” *npmtrends.com*. <https://npmtrends.com/angular-vs-react-vs-vue> (pristupano 28.1. 2024).
- [10] “Quick Start,” *react.dev*. <https://react.dev/learn> (pristupano 29.1.2024).
- [11] Wieruch, Robin. *The Road to Learn React*. United States] [Createspace Independent Publishing Platform, 2019.
- [12] “Getting Started | Vite,” *vitejs.dev*. <https://vitejs.dev/guide/> (pristupano 29.1.2024).
- [13] “A Simple Explanation of React.useEffect( ),” *Dmitri Pavlutin Blog*, Oct. 13, 2020.  
<https://dmitripavlutin.com/react-useeffect-explanation/> (pristupano 28.1.2024).
- [14] “Feature Overview v6.4.1,” *reactrouter.com*.  
<https://reactrouter.com/en/main/start/overview> (pristupano 30.1. 2024).
- [15] “useNavigate v6.11.2,” *reactrouter.com*. <https://reactrouter.com/en/main/hooks/use-navigate> (pristupano 31.1.2024).

- [16] “Explain new Context API in React,” *GeeksforGeeks*, Feb. 07, 2022. <https://www.geeksforgeeks.org/explain-new-context-api-in-react/> (pristupano 1. 2.2024).
- [17] “How to use the React Context API and avoid prop drilling,” *Scrimba blog*, Feb. 06, 2023. <https://scrimba.com/articles/react-context-api/> (pristupano 2.2.2024).
- [18] “React Icons,” *react-icons.github.io*. <https://react-icons.github.io/react-icons/> (pristupano 3.2.2024).
- [19] tailwindcss, “Tailwind CSS - Rapidly build modern websites without ever leaving your HTML.,” *tailwindcss.com*, 2023. <https://tailwindcss.com/> (pristupano 4. 2.2024).
- [20] “Introduction to Node.js,” *Node.js*. <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs> (pristupano 5.2.2024).
- [21] “What is the relationship between Node.js and V8 ?,” *GeeksforGeeks*, Nov. 02, 2021. <https://www.geeksforgeeks.org/what-is-the-relationship-between-node-js-and-v8/> (pristupano 6.2.2024).
- [22] OpenJS Foundation, “Express - Node.js web application framework,” *Expressjs.com*, 2017. <https://expressjs.com/> (pristupano 8.2.2024).
- [23] “Express.js,” *GeeksforGeeks*. <https://www.geeksforgeeks.org/express-js/> (pristupano 9.2.2024).
- [24] L. Gupta, “What is REST – Learn to create timeless REST APIs,” *Restfulapi.net*, Jun. 13, 2019. <https://restfulapi.net/> (10.2.2024).
- [25] A. Gillis, “What is MongoDB? A definition from WhatIs.com,” *SearchDataManagement*. <https://www.techtarget.com/searchdatamanagement/definition/MongoDB> (pristupano 11.2.2024).
- [26] freeCodeCamp.org, “Introduction to Mongoose for MongoDB,” *freeCodeCamp.org*, Feb. 11, 2018. <https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/> (pristupano 12.2.2024).
- [27] “Comparing Document Databases Versus Relational Databases | Couchbase,” *developer.couchbase.com*. <https://developer.couchbase.com/comparing-document-vs-relational/> (pristupano 14.2.2024).