Razvoj RESTful web aplikacije korištenjem Laravela i Reacta

Livojević, Alen

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka, Faculty of Engineering / Sveučilište u Rijeci, Tehnički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:190:674807

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij računarstva

Završni rad

RAZVOJ RESTFUL WEB APLIKACIJE KORIŠTENJEM LARAVELA I REACTA

Rijeka, rujan 2022.

Alen Livojević

0069085057

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij računarstva

Završni rad

RAZVOJ RESTFUL WEB APLIKACIJE KORIŠTENJEM LARAVELA I REACTA

Mentor: Doc. dr. sc. Marko Gulić

Rijeka, rujan 2022.

Alen Livojević

0069085057

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 7. ožujka 2022.

Zavod: **Zavod za računarstvo**
Predmet: **Razvoj web aplikacija**
Grana: **2.09.06 programsko inženjerstvo**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Alen Livojević (0069085057)**
Studij: Preddiplomski sveučilišni studij računarstva

Zadatak: **Razvoj RESTful web aplikacije korištenjem Laravela i Reacta / The development of RESTful web application using Laravel and React**

Opis zadatka:

Razviti RESTful web aplikaciju za upravljanje aktivnostima i događajima unutar neke organizacije. Aplikacija mora podržavati uslugu dodavanja planiranih aktivnosti i događaja unutar organizacije od strane registriranih korisnika kao i uslugu upravljanja tim događajima. Također, potrebno je implementirati registraciju novih korisnika koji su članovi organizacije. Za razvoj poslužiteljskog dijela web aplikacije treba koristiti Laravel 8 radni okvir uz proizvoljno odabran sustav za upravljanje bazama podataka. Također, treba koristiti paket Laravel Sanctum za realizaciju autentifikacije (JWT token). Za razvoj klijentskog dijela aplikacije treba koristiti React JavaScript knjižicu za razvoj korisničkog sučelja uz učinkovito renderiranje aplikacije na uređajima s različitim veličinama zaslona.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 21. ožujka 2022.

Mentor:

Doc. dr. sc. Marko Gulić

Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Kristijan Lenac

Izjava o samostalnoj izradi rada

Izjavljujem da sam u potpunosti samostalno izradio ovaj rad.

Rijeka, rujan 2022.

Alen Livojević

ZAHVALA

Zahvaljujem mentoru, doc. dr. sc. Marku Guliću, na strpljenju za svako pitanje, pomoći koju mi je pružio i vremenu koje je izdvojio tijekom izrade završnog rada. Zahvaljujem svim profesorima i asistentima koji su zaslužni za prijenos znanja i vještina koje sam stekao pohađanjem Tehničkog fakulteta u Rijeci. Zahvaljujem obitelji i bližnjima koji su bili potpora tijekom cijelog studija.

Alen Livojević

SADRŽAJ

1. UVOD

Jedan od problema s kojim se susreće Udruga za mlade Agora je pregled svih aktivnosti i događaja na jednom mjestu. Svaki član udruge ima pravo i mogućnost osmisliti vlastite aktivnosti, odnosno događaje. Pošto udruga broji velik broj članova ni broj aktivnosti nije malen. Ova aplikacija služi kako bi se olakšao pregled svih aktivnosti koje su trenutno aktivne ili planirane. Svaki korisnik može stvarati nove aktivnosti koje bi pritom bile vidljive svim ostalim članovima. Korisnik koji je stvorio određenu aktivnost istu može ukloniti ili urediti.

Za stvaranje aplikacije korišten je Laravel radni okvir (eng. framework) i Vagrant program za izgradnju i održavanje prijenosnih virtualnih okruženja za izgradnju softvera u kombinaciji sa Laravel Homesteadom na serverskoj strani i Reactom na klijentskoj strani. React je besplatna JavaScript knjižnica za stvaranje korisničkih sučelja (eng. user interface). Za upravljanje bazom podataka korišten je MySQL sustav. Za pisanje i uređivanje koda korišten je besplatni uređivač (eng. editor) Visual Studio Code.

Aplikacija se temelji na REST arhitekturi. REST je kratica za engleski izraz *Representational state transfer* i označava stil arhitekture softvera kojeg karakterizira jedinstven sustav koji se sastoji od više odvojenih komponenata, u ovom slučaju odvojene poslužiteljske i klijentske strane.

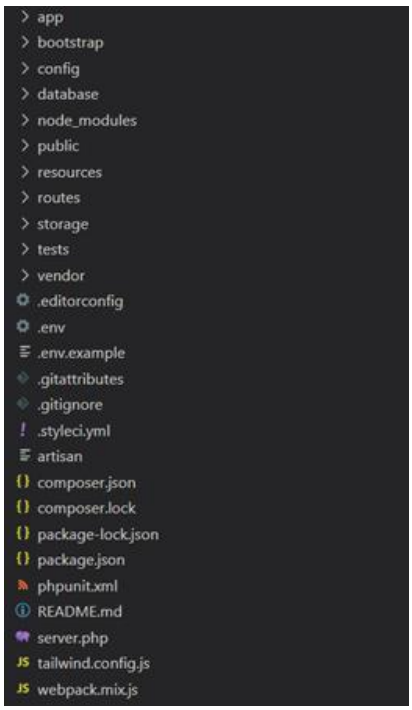
2. OPIS TEHNOLOGIJA

2.1. Laravel

Laravel programski okvir koristi se kako bi se pružio brži i lakši način razvoja aplikacijskih rješenja. Može se koristiti za razvoj serverske strane (eng. backend) i klijentske strane (eng. frontend). Razvoj aplikacija odvija se brže jer se programer može posvetiti razvoju funkcionalnosti aplikacija bez značajnog gubitka vremena na razvoj detalja niže razine aplikacije. Jedno je od najpopularnijih PHP (eng. Hypertext Preprocessor) razvojnih okruženja, ponajprije zbog jednostavnosti, jasne strukture (prikazano na *Slici 2.1.)* i vrlo detaljne dokumentacije. Tvorac Laravela je Taylor Otwel koji je htio napraviti alternativu okviru CodeIgniter koji nije pružao određene značajke kao što je ugrađena podrška za autentifikaciju i autorizaciju korisnika [1]. Prvo izdanje Laravela postalo je dostupno 9. lipnja 2011. godine, a kasnije toga mjeseca predstavljeno je izdanje Laravel 1. Laravel je već od prvog izdanja pružao ugrađenu podršku za autentifikaciju, modele, sesije, prikaze, usmjeravanje i još mnogo toga.

Laravel je temeljen na PHP-u. PHP je naširoko korišten programski jezik na poslužiteljskoj strani koji je tijekom godina postajao sve brži i moćniji [2]. Odlično radi u kombinaciji s HTML-om i bazama podataka što ga čini odličnim jezikom za svakoga tko je zainteresiran za izradu dinamičkih web aplikacija. Može se koristiti na svim operacijskim sustavima od Windowsa, Linuxa i raznih varijanti Unixa do MacOS-a [3]. Također ima podršku za većinu web poslužitelja kao što su Apache i IIS.

Velika prednost Laravela je njegova vrlo pristupačna i jasna dokumentacija koja se nalazi na službenim stranicama i organizirana je na način da se na početku nalaze upute za instalaciju, a kasnije prolaskom kroz sva poglavlja ulazi se u sve naprednije značajke sve do testiranja gotove aplikacije. Unatoč velikoj opsežnosti, kretanje kroz dokumentaciju je jednostavno zbog navigacijske trake pomoću koje se brzo nađe tražena informacija. U ovom projektu korišten je Laravel 8, a trenutno najnovija verzija je Laravel 9 koja je izdana 8. veljače 2022. godine, a za Laravel 8 još je uvijek dostupna podrška.



Slika 2.1 Struktura projekta kreiranog u Laravelu 8

2.2. Laravel Sanctum

Laravel Sanctum je paket koji pruža provjeru autentičnosti pomoću tokena. Svakom korisniku prilikom prijave dodijeli se token koji mu omogućuje pristup aplikaciji. Kada se korisnik odjavi token se izbriše (uništi). Instalacija Laravel Sanctuma vrlo je jednostavna i može se učiniti pomoću Composer package managera [4]. Laravel Sanctum omogućuje jednostavno kontroliranje pristupa zaštićenim rutama, automatsko generiranje kontrolera za upravljanje prijavom, registracijom i odjavom korisnika te generiranje tokena.

Zaštita se vrši na način da svi dolazni zahtjevi moraju biti autentificirani. Zaštita će osigurati da svi zahtjevi budu ili zahtjevi sa statusom, zahtjevi s autentifikacijom kolačića (eng. cookies) ili da sadrže valjano zaglavlje što je korišteno u izradi ovog projekta. Redoslijed provjere je takav da Laravel Sanctum prvo provjerava postoji li tipični kolačić, a ukoliko ne postoji, Sanctum će pokušati potvrditi autentičnost zahtjeva pomoću tokena u zaglavlju. Primjer slanja POST metodom s klijentske strane prikazan je na *Slici 2.2*.

```
const events = await axios.get('http://homestead.test/api/show/'+id, {
    headers: {
        "Authorization" : `Bearer ${token}`
    }
```

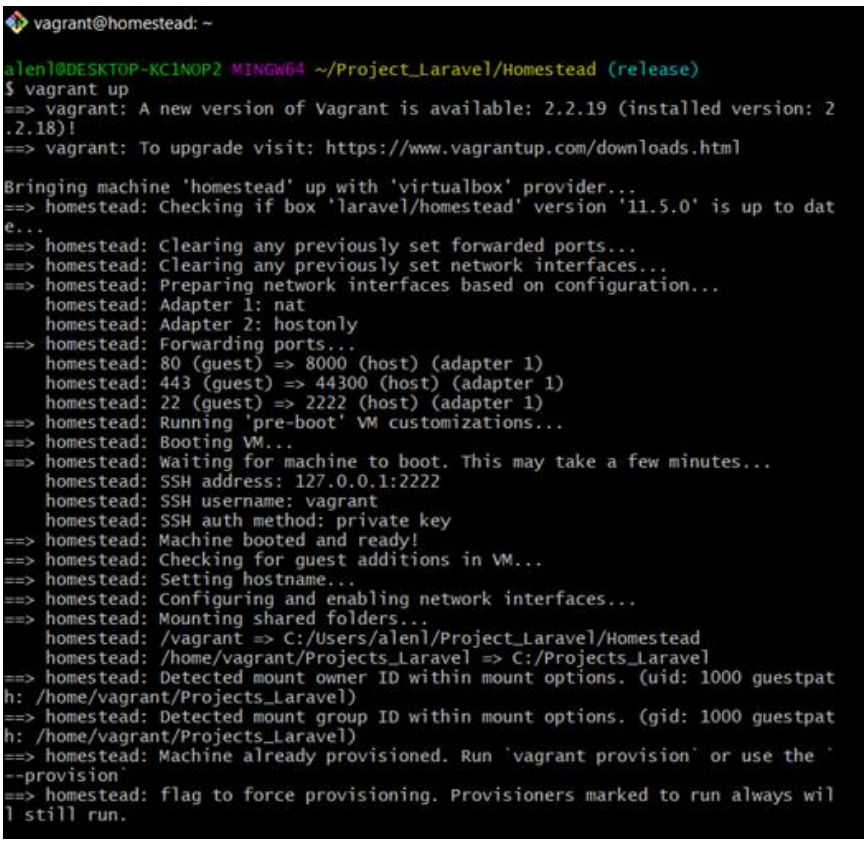
Slika 2.2. Primjer slanja tokena u zaglavlju zahtjeva

2.3. Vagrant

Vagrant je alat za virtualno upravljanje okruženjima [5]. Skraćuje vrijeme postavljanja razvojnog okruženja uz vrlo jednostavno korištenje. U ovom projektu Vagrant je pokrenut na VirtualBox softveru za virtualizaciju. VirtualBox razvijan je od strane Oraclea. Uloga mu je olakšati instalacije softvera, pojednostaviti testiranja, smanjiti troškove i omogućiti istovremeno pokretanje više različitih operacijskih sustava.

Prva verzija Vagranta objavljena je u ožujku 2010. godine, a prva stabilna verzija, Vagrant 1.0, objavljena je u ožujku 2012. godine. Vagrant je od samih početaka besplatan softver. U početku je bio vezan isključivo za VirtualBox, ali je već verzija Vagrant 1.1 pružila podršku za druge softvere za virtualizaciju kao što su VMware i KVM. Napisan je u Rubyju, a može se koristiti u projektima napisanim u drugim programskim jezicima kao što su PHP, Python, Java, JavaScript, C# [6].

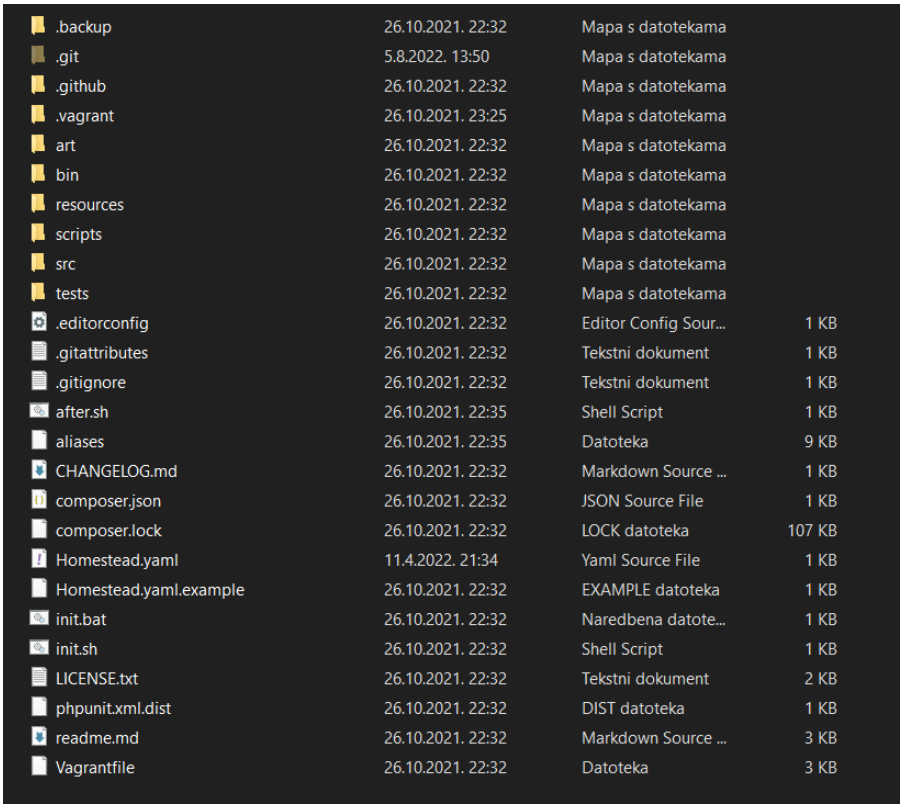
Nakon postavljanja korištene su tri osnovne naredbe za upravljanje Vagrantom: *vagrant up* za podizanje virtualnog stroja (eng. virtual machine), *vagrant ssh* za uspostavljanje SSH sesije u virtualnom stroju kako bi se omogućio pristup ljusci (eng. shell) i *vagrant halt* za gašenje virtualnog stroja. Naredbe se pokreću kroz terminal Git Bash u kojem se treba pozicionirati u Homestead mapu trenutnog projekta. Na *Slici 2.3.* vidi se lokacija Homestead mape kao i pokretanje virtualnog stroja pomoću naredbe *vagrant up.*



Slika 2.3. Pokretanje virtualnog stroja kroz Git Bash

2.4. Laravel Homestead

Laravel Homestead je unaprijed zapakirana Vagrant kutija (eng. Vagrant box) koja pruža razvojno okruženje bez potrebe za instalacijom PHP-a, web poslužitelja ili bilo kojeg drugog poslužiteljskog programa [7]. Radi na Windows, macOS i Linux sustavima. Uključuje razne tehnologije kao što su Nginx, PHP, MySQL, PostgreSQL, Redis, Memcached, Node… Instalacija se vrši kroz naredbeni redak kloniranjem GitHub direktorija, međutim nije komplicirana zahvaljujući detaljnoj dokumentaciji koja se nalazi na službenim stranicama Laravela. Sadržaj Homestead mape prikazan je na *Slici 2.4.*



Slika 2.4. Sadržaj Homstead mape

Postavke za instalaciju Homesteada postavljaju se u datoteci *Homestead.yaml*. Potrebno je odrediti tko je pružatelj (eng. provider): VirtualBox ili Parallels. Svojstvo (eng. property) *folders* sadrži sve mape koje će se dijeliti sa Homestead okruženjem. Kako se datoteke unutar ovih mapa mijenjaju, one će biti sinkronizirane između vašeg lokalnog računala i virtualnog okruženja. Moguće je konfigurirati koliko god dijeljenih mapa je potrebno. Ukoliko je potrebno povezati više aplikacija, preporučuje se mapiranje svake aplikacije pojedinačno, a ne cijele mape koja sadrži sve aplikacije iz razloga što bi takav način pogoršao performanse. Pomoću svojstva *sites* moguće je konfigurirati domenu aplikacije. Homestead sadrži nekoliko servisa koji se mogu omogućiti ili onemogućiti. Na primjer, moguće je omogućiti PostgreSQL i onemogućiti MySQL. Pomoću svojstva *features* moguće je omogućiti ili onemogućiti dodatni softver koji se koristi u aplikaciji. Sadržaj datoteke prikazan je na *Slici 2.5.*

```
ip: "192.168.10.10"
memory: 2048
cpus: 2
provider: virtualbox

authorize: ~/.ssh/id_rsa.pub

keys:
    - ~/.ssh/id_rsa

folders:
    - map: C:/Projects_Laravel
      to: /home/vagrant/Projects_Laravel

sites:
    - map: homestead.test
      to: /home/vagrant/Projects_Laravel/Project3/public

databases:
    - homestead

features:
    - mysql: true
    - mariadb: false
    - postgresql: false
    - ohmyzsh: false
    - webdriver: false

services:
    - enabled:
        - "mysql"
```

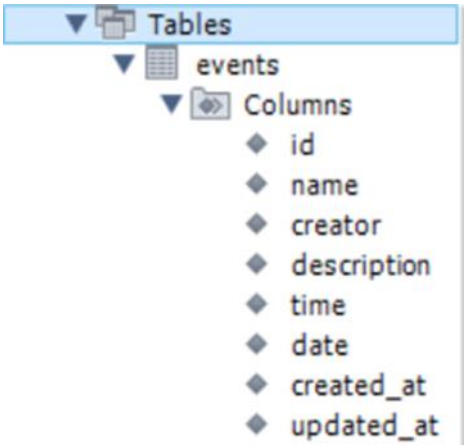
Slika 2.5. Sadržaj datoteke Homestead.yaml

2.5. MySQL

, MySQL je besplatan sustav za upravljanje bazom podataka. Primjer je relacijskog sustava upravljanja bazom podataka (eng. open-source relational database management system). Iz engleskog naziva dolazi često korištena kratica RDBMS. Relacijsku bazu podataka karakterizira organiziranje podataka u jednu ili više tablica čiji podaci mogu biti međusobno povezani. Takve veze čine strukturu baze podataka.

MySQL kreirala je švedska tvrtka MySQL AB. Razvoj je započeo 1994. godine, a prva verzija pojavila se 23. svibnja 1995. godine. Posljednja verzija pod nazivom 8.0 izašla je 2018. godine uz redovite nadogradnje. Trenutna verzija bit će podržana sve do travnja 2026 [8].

Osnovni element koji se pohranjuje u bazi naziva se entitet. Svi podaci pohranjuju se unutar tablica koje se sastoje od stupaca i redaka. Stupci se nazivaju još i atributima jer sadrže opise podataka pojedinog entiteta, a redci sadržavaju konkretne podatke. Na *Slici 2.6.* nalazi se primjer atributa tablice events korištene u projektu u programu MySQL Workbranch 8.0.



Slika 2.6. Atributi tablice events

Podacima spremljenim u bazi pristupa se pomoću SQL upitnog jezika (eng. SQL query language). SQL je dizajniran za dohvaćanje i upravljanje podacima u relacijskoj bazi podataka. Pomoću ključnih riječi mogu se izvoditi operacije dodavanja, uređivanja, brisanja i čitanja podatka. Na *Slici 2.7.* vidljivo je kako se pomoću naredbe SELECT prikažu podaci spremljeni u bazu. Prikazani su podaci koji su spremljeni u tablicu *users*.

```
1 • select * FROM users
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap

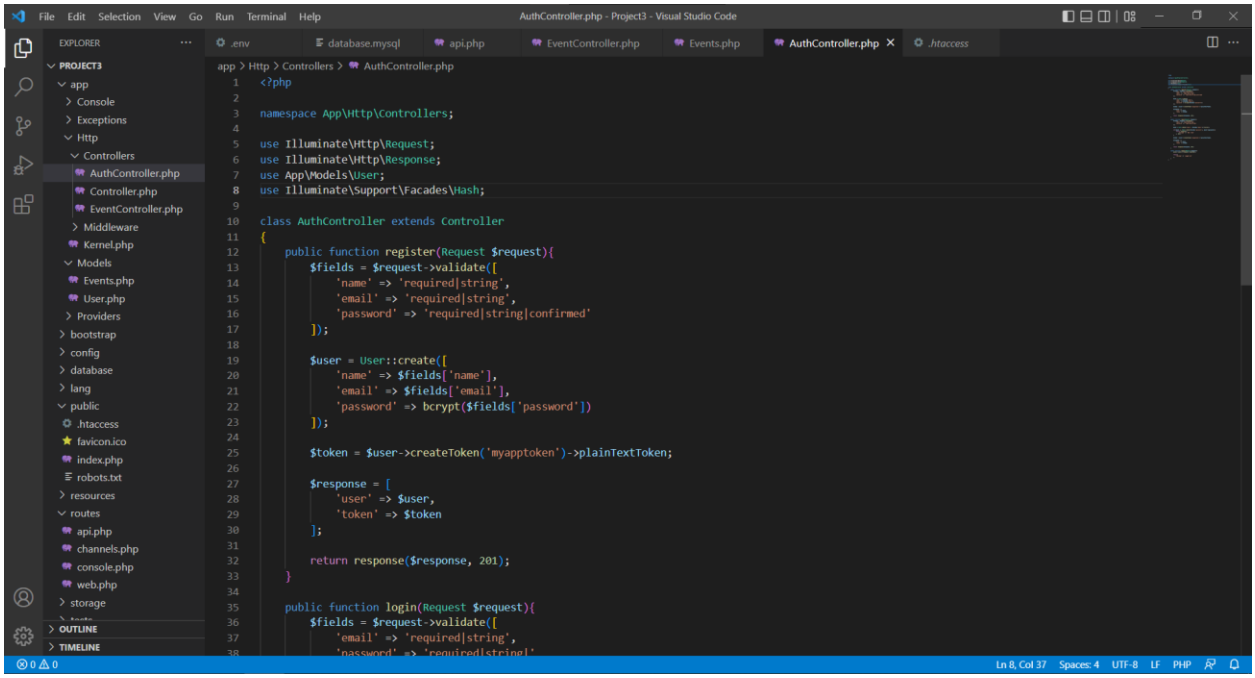
id	name	email	email_verified_at	password
1	Alen	alenliv12@gmail.com	NULL	$2y$10$FQ5Ake7PArYIm9kbvWzw2exRWBbnm…
3	Ivano	ivano@gmail.com	NULL	$2y$10$Goe9LvGDN6U7Aal6aKfA0OvT1C8.kJr…
NULL	NULL	NULL	NULL	NULL

Slika 2.7. Prikaz podataka u tablici users naredbom SELECT

2.6. Visual Studio Code

Visual Studio Code je uređivač izvornog koda razvijen od strane Microsofta. Dostupan je za operacijske sustave Windows, MacOS i Linux. Dolazi s ugrađenom podrškom za JavaScript, TypeScript i Node.js uz mogućnost proširenja na druge programske jezike i okruženja kao što su

C++, C#, Java, Python, PHP, Go, .NET [9]. Nudi značajke kao što su podrška za otklanjanje pogrešaka (eng. debugging), prepoznavanje i isticanje sintakse i ključnih riječi, ugrađenu podršku za Git sustav za upravljanje verzijama itd. [10] Korisna značajka je i ugrađeni terminal koji ubrzava izvođenje naredbi koje se inače izvršavaju u zasebnom prozoru Git Basha. Visual Studio Code omogućuje istovremeni prikaz strukture mapa zbog čega je vrlo jednostavno otvarati razne datoteke koje su potrebne u radu. Istovremeno može biti otvoreno više datoteka koje su pritom vidljive na traci iznad koda u obliku otvorenih kartica kroz koje se možemo vrlo brzo i jednostavno kretati. Otvorene kartice mogu biti prikazane i istovremeno na ekranu pomoću opcije dijeljenja ekrana. Sve značajke u kombinaciji tijekom rada (*Slika 2.8.*) izgledaju uredno i lako su dostupne što čini Visual Studio Code odličnim uređivačem s kojim nije teško raditi.



Slika 2.8. Primjer prikaza tijekom izrade projekta

Visual Studio Code razvijen je od strane Microsofta 2015. godine. U anketi za razvojne programere koju je proveo Stack Overflow 2021. godine, Visual Studio Code rangiran je kao najpopularniji alat za razvojno okruženje među 82 000 ispitanika, a 70% je izjavilo da ga koristi.

2.7. React

React je besplana JavaScript knjižnica za izgradnju i stvaranje korisničkih sučelja. Održavan je od strane Mete (Facebook) i zajednice individualnih programera i tvrtki. Nastao je 2013. godine

i primjer je deklarativnog programiranja [11]. Programeri dizajniraju prikaze za svako stanje aplikacije, a React ažurira i renderira komponente kada se podaci promijene. Deklarativni pogled čini kod predvidljivijim i lakšim za čitanje prilikom otklanjanja pogrešaka. Dva najčešća primjera komponenti su funkcionalne komponente (eng. function components) koje su prikazane na *Slici 2.9.* i komponente bazirane na klasi (eng. class based component).

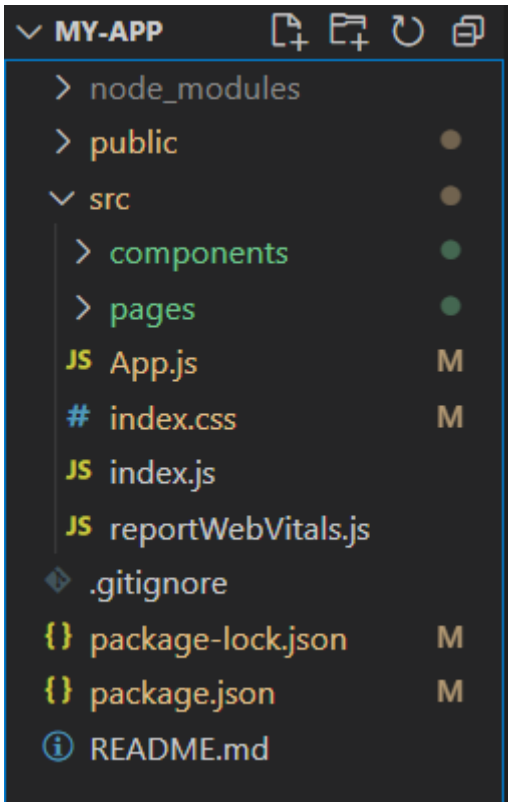
```
import 'bootstrap/dist/css/bootstrap.css'
import axios from 'axios'
import { useEffect } from "react";



function Logout(){

    useEffect(() => {
        function deleteToken(){
            const token = localStorage.getItem('token');
            const logout = axios.get('http://homestead.test/api/logout', {
                headers: {
                    "Authorization" : `Bearer ${token}`
                }
            }).then(function (response) {
                console.log(response);
                window.location.href = "/login";
              });
            console.log(logout);

        }
        deleteToken();
    },[]);


    return (null)
}


export default Logout
```

Slika 2.9. Primjer funkcionalne komponente

Popularnost Reacta danas je zasjenila popularnost svih ostalih razvojnih okvira korištenih za razvoj klijentske strane [12]. Razloga za to ima nekoliko. React olakšava izradu dinamičkih web aplikacija jer zahtijeva manje kodiranja i nudi više funkcionalnosti za razliku od JavaScripta gdje kodiranje često vrlo brzo postane složeno. Također koristi Virtual DOM (eng. Document Object Model), čime brže stvara web aplikacije. Virtualni DOM uspoređuje prethodna stanja komponenti i ažurira samo stavke u stvarnom DOM-u koje su promijenjene umjesto ponovnog ažuriranja svih komponenti, kao što to rade konvencionalne web aplikacije. Komponente su sastavni dio svake React aplikacije, a jedna se aplikacija obično sastoji od više komponenti. Sve komponente imaju svoju logiku i kontrole i mogu se ponovno koristiti bilo gdje u cijeloj aplikaciji što zauzvrat drastično smanjuje vrijeme razvoja aplikacije i pojednostavljuje njenu strukturu. React radi na principu jednosmjernog protoka podataka. To znači da prilikom dizajniranja React

aplikacije programeri često ugnijezde podređene komponente unutar nadređenih komponenti. Budući da podaci teku u jednom smjeru, postaje lakše ispravljati pogreške i znati gdje se problem pojavljuje u aplikaciji u dotičnom trenutku. Lako ga je naučiti jer uglavnom kombinira osnovne HTML-a i JavaScript koncepte s nekim korisnim dodacima. Ipak, kao što je slučaj i s drugim alatima i okvirima, treba potrošiti neko vrijeme kako bi se pravilno razumjelo Reactovu knjižnicu i snašlo se među mnogobrojnim mogućnostima čije korištenje ovisi o više različitih situacija koje je potrebno prepoznati prije odabira ispravne knjižnice. Osim što služi za razvoj web aplikacija, može se koristiti i za razvoj mobilnih aplikacija pomoću okvira React Native. Sastavni dio razvoja je i otklanjanje pogrešaka koje je u nekim situacijama vrlo teško pronaći. Zato je Facebook je izdao proširenje za internetski preglednik Google Chrome koje se može koristiti za traženje i uklanjanje pogrešaka u React aplikacijama što čini proces otklanjanja pogrešaka bržim i lakšim.

Struktura Reacta poprilično je jednostavna. Sastoji se od *node_modules*, *public*, i *src* mape. Najbitnija mapa za razvoj aplikacije je *src*. U njoj korisnik sam kreira mape u koje će rasporediti datoteke koje čine aplikaciju. Struktura je prikazana na *Slici 2.10.*



Slika 2.10. Struktura React projekta

U *components* mapi najčešće se nalaze implementacije komponenata kao što su navigacijska traka, različiti gumbi, polja za unos i slično. Mapa *pages* sadrži implementacije stranica od kojih se sastoji aplikacija. *App.js* je glavna komponenta koja ujedinjuje sve ostale komponente u aplikaciji.

React daje mogućnost instalacije dodatnih vanjskih knjižnica koje olakšavaju rad s osnovnim funkcionalnostima. Primjer takve knjižnice je Axios. On služi za slanje asinkronih HTTP zahtjeva na serversku stranu. Pomoću takvih zahtjeva možemo pohraniti podatke u bazu podataka ili ih pročitati i dalje ih obrađivati i koristiti. Primjer zahtjeva prikazuje *Slika 2.11*. Osim Axiosa, korisna je i React - Bootstrap knjižnica čijim se korištenjem izbjegava zasebno umetanje Bootstrapa u projekt. Ona olakšava vizualno uređenje stranice na način da omogućuje umetanje unaprijed definiranih komponenti kao što su gumbi ili navigacijska traka. Instalacija vanjskih knjižnica također je vrlo jednostavna. Moguće ju je izvršiti pomoću npm (eng. Node Package Manager) paketa kroz naredbeni redak. Na primjer, naredba za instalaciju Axiosa je *npm install axios.*

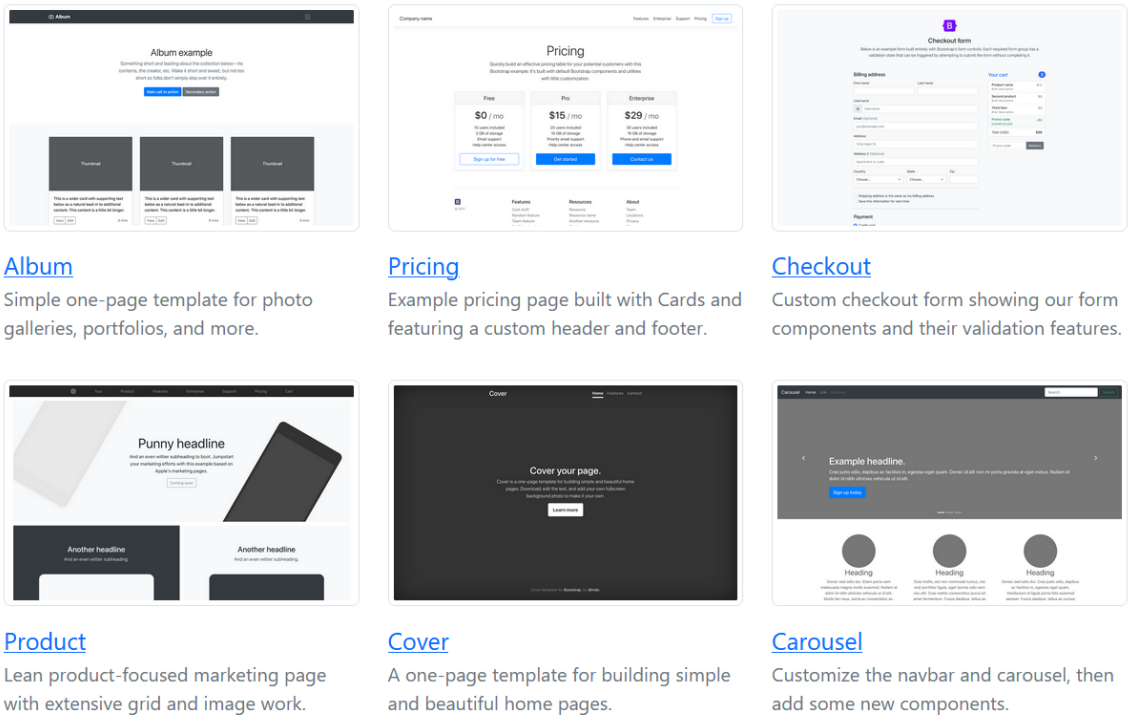
```
axios.post('http://homestead.test/api/register', payload, {
    headers: {
        'Content-Type': 'application/json',
    }
}).then(function (response) {
    console.log(response);
    localStorage.setItem('token', response.data.token);
    localStorage.setItem('user', response.data.user.name);
    console.log(localStorage.getItem('token'));
    console.log(localStorage.getItem('user'));
  }).catch(function (error) {
    console.log("error");
    window.location.href = "/invalidlogin";
  });

//resetira input polja na prazan string kad se submita forma
//setPayload({ name: "", time: "", description: "", date: "", creator: "" });
navigate('/about');
};
```

Slika 2.11. Primjer korištenja Axios knjižnice

2.8. Bootstrap

Bootstrap je snažan, proširiv set alata [13] koji nudi gotova dizajnerska rješenja u izgradnji aplikacije. Također je besplatna zbirka alata otvorenog koda za izradu responzivnih web stranica i web aplikacija. Koristi se jer omogućuje brži i lakši razvoj i dizajniranje neovisno o platformi na kojoj se aplikacija pokreće. Omogućuje stvaranje responzivnih web stranica i stranica koje se prikazuju na mobilnim uređajima [14]. Predlošci unaprijed kreiranih komponenata dostupni su na službenoj stranici. Mogu se iskoristiti izdvojeni dijelovi stranice kao što su zaglavlja, podnožja, bočni izbornici, navigacijske trake…, ili cijele stranice kao što je stranica za prijavu, album

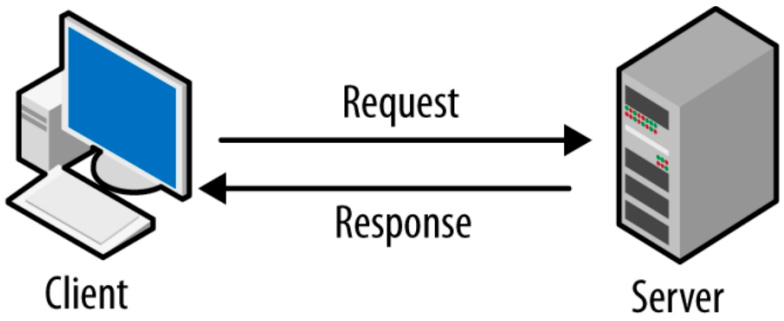
fotografija, blog… Na *Slici 2.12.* prikazana je službena stranica s velikim izborom dizajnerski riješenih predložaka.



Slika 2.12. Predlošci gotovih stranica na web stranici getbootstrap.com

2.9. Povezivanje tehnologija u krajnji proizvod

Česta arhitektura u izradi web aplikacija jest klijent – poslužitelj arhitektura. Serverska strana se procesuira na poslužitelju koji je fizički udaljen i daje odgovore na zahtjeve koje zaprimi od klijenta. Kada korisnik želi poslati, na primjer, zahtjev za registracijom profila na nekoj aplikaciji, on mora otvoriti internetski preglednik i pristupiti internet adresi željene stranice. Tada se na ekranu prikazuje stranica na kojoj korisnik unosi svoje podatke. Taj proces se odvija na klijentskoj strani i može biti izrađen u Reactu u kombinaciji sa Bootstrapom i ostalim knjižnicama. Nakon što korisnik pritisne gump za registraciju, zahtjev se šalje na poslužitelj ili server. On je fizički odvojen i komunicira s klijentom preko HTTP protokola (eng. Hypertext Transfer Protocol). On može biti implementiran koristeći Laravel, odnosno PHP razvojno okruženje i MySQL bazu podataka. Kada zaprimi zahtjev, poslužitelj će provjeriti prava pristupa korisnika, obraditi podatke koji su zaprimljeni, pohraniti ih u bazu podataka i vratiti odgovor poslužitelju. Isti ciklus se ponavlja za svaki zahtjev koji stigne sa strane klijenta (*Slika 2.13.)*. Arhitektura klijent - poslužitelj poznata je i kao REST arhitektura.



Slika 2.13. Princip rada klijent - poslužitelj arhitekture

2.10. REST arhitektura

REST (eng. REpresentational State Transfer) je arhitektonski stil za pružanje standarda između računalnih sustava na webu, čime se olakšava međusobna komunikacija sustava [15]. Sustavi usklađeni s REST-om, koji se često nazivaju RESTful sustavi, karakterizirani su time što nemaju stanja i dijele se na ulogu klijenta i poslužitelja. U REST arhitektonskom stilu implementacija klijenta i implementacija poslužitelja mogu se obaviti neovisno, a da jedna za drugu ne zna. To znači da se kod na strani klijenta može promijeniti u bilo kojem trenutku, a da ne utječe na rad poslužitelja, a kod na strani poslužitelja može se promijeniti bez utjecaja na rad klijenta. Sustavi su bez stanja, što znači da poslužitelj ne mora znati ništa o stanju u kojem se klijent nalazi i obrnuto. Na taj način i poslužitelj i klijent mogu razumjeti svaku primljenu poruku, čak i bez gledanja prethodnih poruka. Klijenti šalju zahtjeve za dohvat ili izmjenu resursa, a poslužitelji šalju odgovore na te zahtjeve. Zahtjev se može sastajati od HTTP metode, GET, POST, PUT ili DELETE, koja definira koju vrstu operacije poslužitelj mora izvesti, zaglavlja, koje omogućuje klijentu prosljeđivanje informacija o zahtjevu, putanje do resursa i dodatne poruke koja se nalazi u tijelu (eng. body) zahtjeva. Odgovori s poslužitelja sadrže statusne kodove koji obavještavaju klijenta o uspjehu operacije. Najčešći kodovi prikazani su na *Slici 2.14.*

Status code	Meaning
200 (OK)	This is the standard response for successful HTTP requests.
201 (CREATED)	This is the standard response for an HTTP request that resulted in an item being successfully created.
204 (NO CONTENT)	This is the standard response for successful HTTP requests, where nothing is being returned in the response body.
400 (BAD REQUEST)	The request cannot be processed because of bad request syntax, excessive size, or another client error.
403 (FORBIDDEN)	The client does not have permission to access this resource.
404 (NOT FOUND)	The resource could not be found at this time. It is possible it was deleted, or does not exist yet.
500 (INTERNAL SERVER ERROR)	The generic answer for an unexpected failure if there is no more specific information available.

Slika 2.14. Statusni kodovi

Klijent i poslužitelj prilikom komunikacije moraju znati pročitati poruku koju si međusobno pošalju i podatke koji se u njoj nalaze. Zato postoji JSON format kojeg obje strane znaju čitati. JavaScript Object Notation (JSON) standardni je format temeljen na tekstu za predstavljanje strukturiranih podataka na temelju sintakse JavaScript objekta [16]. On je jednostavan *string* koji sadrži svojstva koja opisuju neki objekt. Nazivi i vrijednosti moraju biti smješteni unutar dvostrukih navodnika i odvojeni dvotočkom, a svojstva se odvajaju zarezom. Može sadržavati polja, binarne (boolean) vrijednosti i sve ostale tipove podataka, čak omogućuje i formiranje hijerarhije podataka. Korištenje REST-a, kao konvencije za API, i JSON-a, kao medija za razmjenu, pokazalo se snažnom kombinacijom za balansiranje jednostavnosti, fleksibilnosti i dosljednosti. Postao je jedna od najvažnijih tehnologija koja se koristi u modernom softverskom okruženju. [17]. Primjer JSON-a prikazan je na *Slici 2.15.*

Slika 2.15. Primjer JSON formata

3. OPIS APLIKACIJE

3.1. Prijava

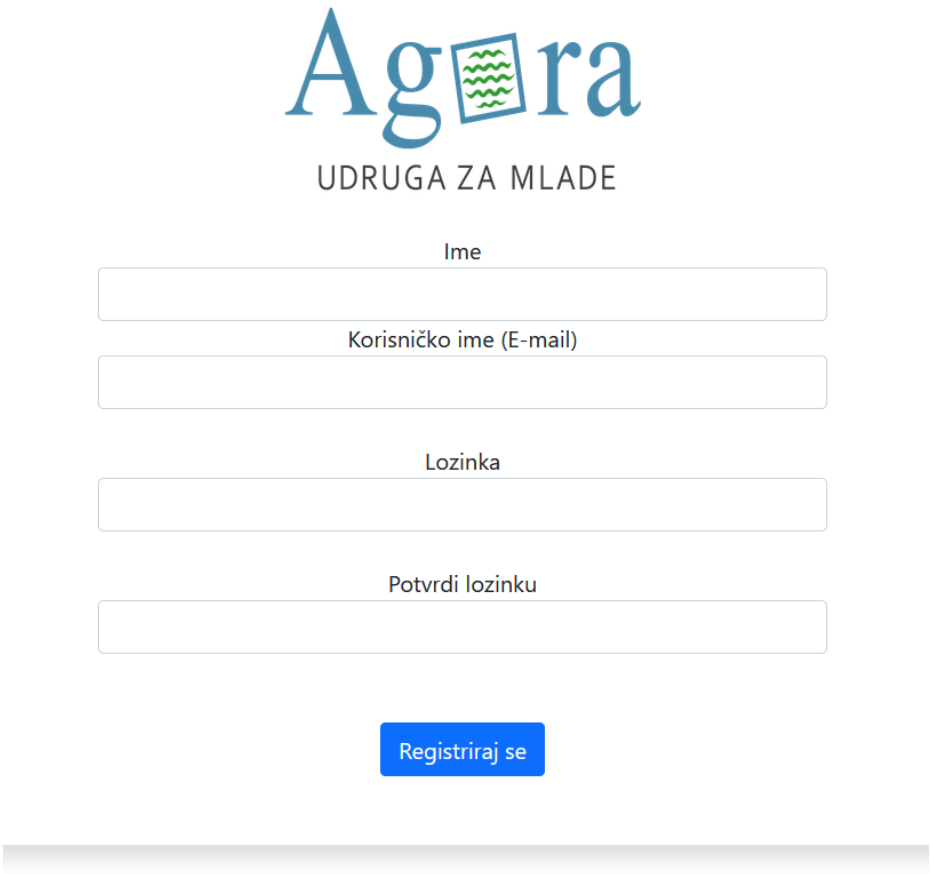
Ukoliko trenutno nema prijavljenog korisnika, odnosno ne postoji aktivni token, aplikacija će prikazati stranicu za prijavu (*Slika 3.1.)*. Ona se sastoji od dva prozora za unos – unos korisničkog imena, odnosno e-mail adrese, i unos lozinke. Nakon što korisnik unese podatke i pritisne gumb *Prijava* bit će preusmjeren na stranicu gdje se nalazi ispis svih trenutno otvorenih događaja. Ukoliko podaci koje je korisnik unio nisu ispravni, prikazat će se poruka *Neuspjela prijava*. Ako korisnik još nema kreiran profil, nudi mu se opcija *Registriraj se* gdje ga može izraditi.

Slika 3.1 Stranica za prijavu korisnika

3.2. Registracija

Korisnik, koji još nema kreiran profil na aplikaciji, može ga kreirati na stranici za registraciju. Na njoj se nalaze prozori za unos imena, korisničkog imena ili e-mail adrese, lozinke i potvrde lozinke. Sve vrijednosti korisnik može odrediti proizvoljno i služit će mu za buduću prijavu u

aplikaciju. Jedino polje na koje korisnik mora pripaziti je potvrda lozinke. Prilikom registracije potrebno je potvrditi unos lozinke kako bi se provjerila točnost unosa i izbjegla moguća pogreška. Ukoliko se lozinke ne podudaraju, korisniku će se prikazati poruka *Neuspjela prijava*, a u slučaju ispravne registracije, korisnik će biti odveden na stranicu *O Agori* gdje se može upoznati sa osnovnim informacijama o Udruzi za mlade Agora. Stranica za registraciju prikazana je na *Slici 3.2.*

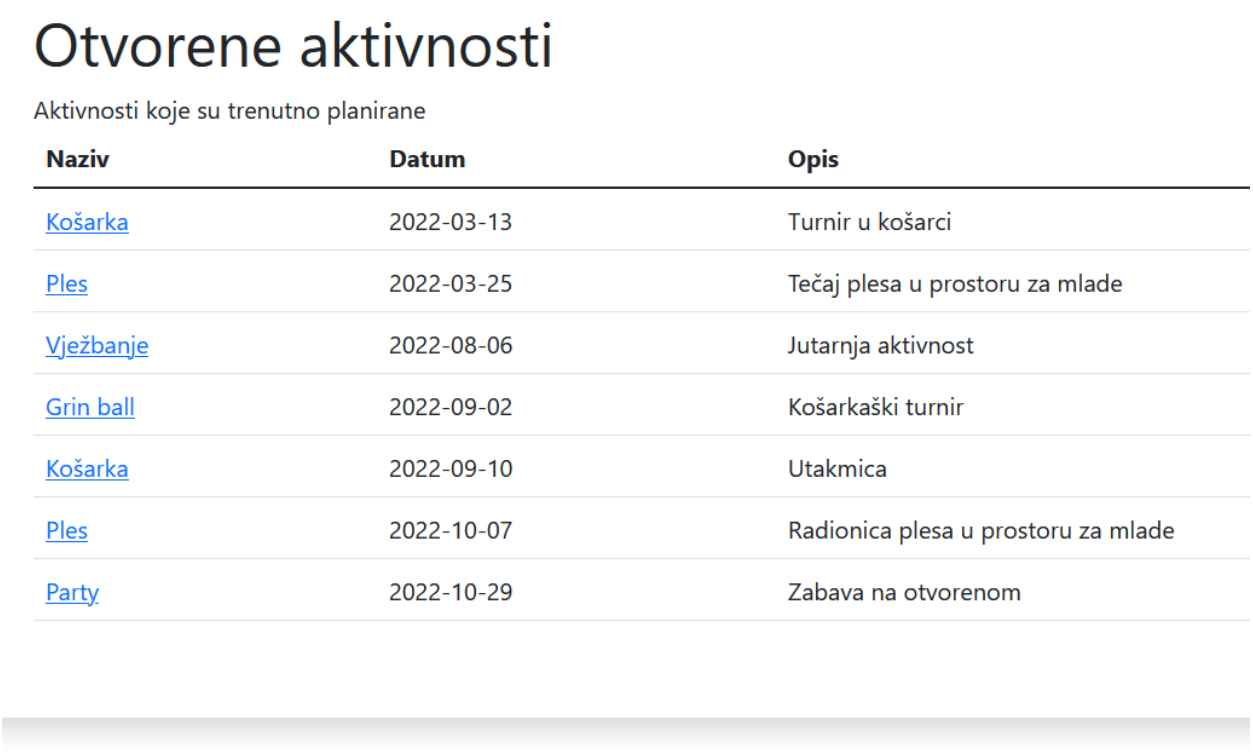


Slika 3.2. Registracija novog korisnika

3.3. Otvorene aktivnosti

Nakon što se korisnik prijavi u aplikaciju, nalazi se na stranici na kojoj se ispisuju sve aktivnosti koje su trenutno pohranjene u bazi podataka od strane svih korisnika. Aktivnosti se ispisuju prema datumu njihovog održavanja počevši od onih koje će prve biti održane. Ispis je formatiran pomoću tablice koju čine tri stupca i onoliko redova koliko ima ispisanih aktivnosti. Prvi stupac služi za prikaz naziva događaja, drugi za prikaz datuma predviđenog za održavanje aktivnosti i treći za ispis opisa aktivnosti kako bi se znalo o kojoj je aktivnosti riječ u slučaju da postoji više aktivnosti s istim nazivom. Klikom na naziv aktivnosti otvara se stranica na kojoj se

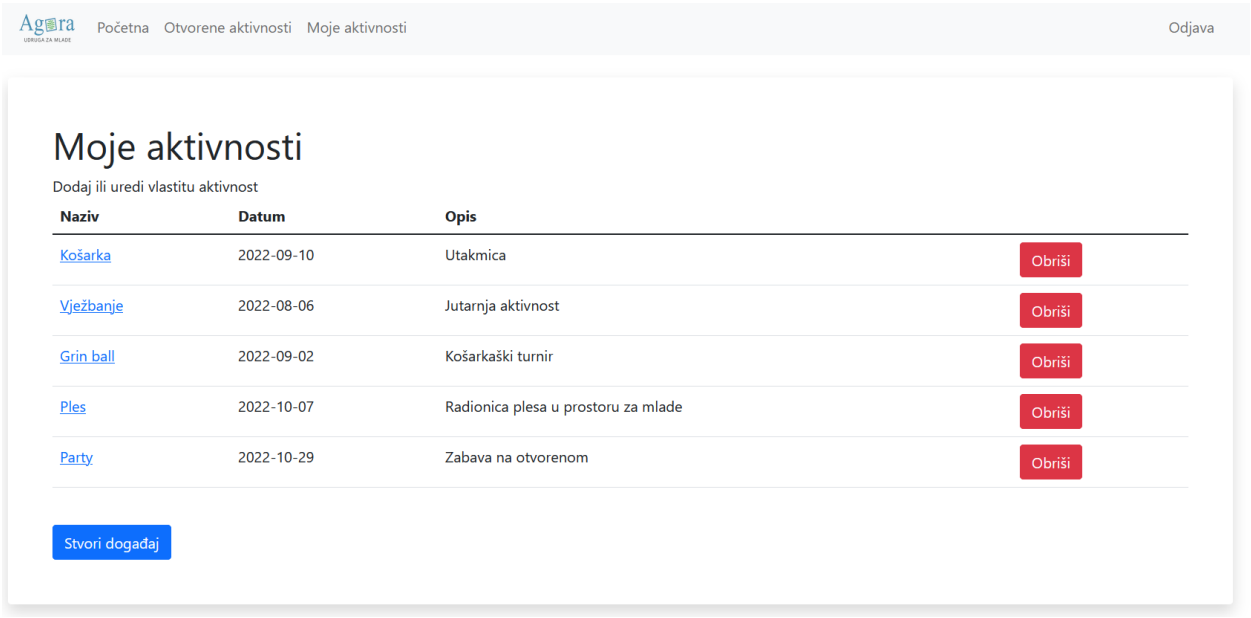
nalaze svi detalji vezani uz taj događaj. Na *Slici 3.3.* vidljiv je primjer aktivnosti i njihovog prikaza kao i navigacijska traka koja je zasebno izrađena komponenta uključena u prikaz svih stranica u aplikaciji.



Slika 3.3 Otvorene aktivnosti

3.4. Moje aktivnosti

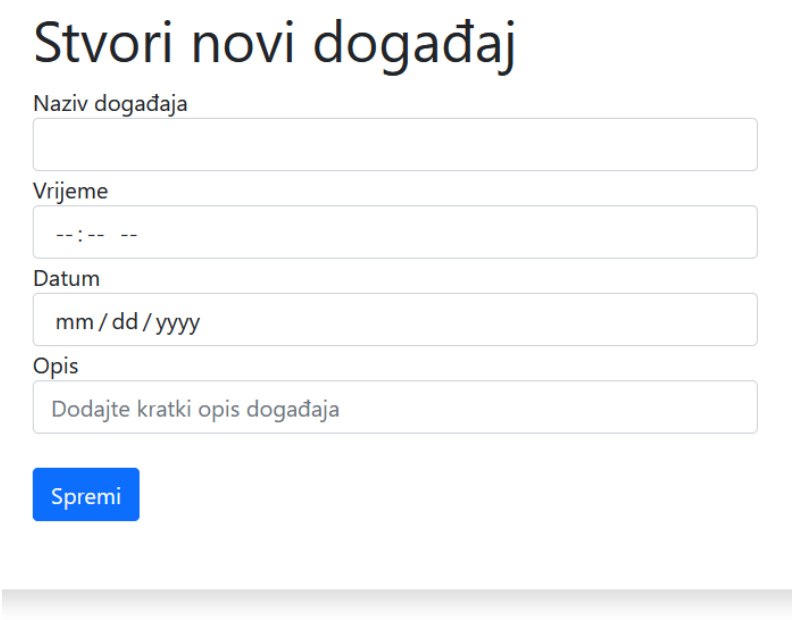
Slično kao i kod *Otvorenih aktivnosti*, u *Mojim aktivnostima* ispisuju se aktivnosti poredane po datumu održavanja, međutim samo one čiji je organizator trenutno prijavljeni korisnik. Ispod liste aktivnosti nalazi gumb *Stvori događaj* čijim odabirom korisnik može stvoriti novu aktivnost, a u desnom stupcu nalazi se gumb *Obriši* čijim pritiskom se briše aktivnost iz željenog reda. Ako korisnik klikom miša pritisne na naziv aktivnosti, u ovom se slučaju ispišu detalji odabrane aktivnosti, međutim u obliku polja za unos u kojima se informacije o odabranoj aktivnosti mogu urediti. Primjer prikaza *Mojih aktivnosti* prikazuje *Slika 3.4.*



Slika 3.4. Moje aktivnosti

3.5. Stvaranje novog događaja

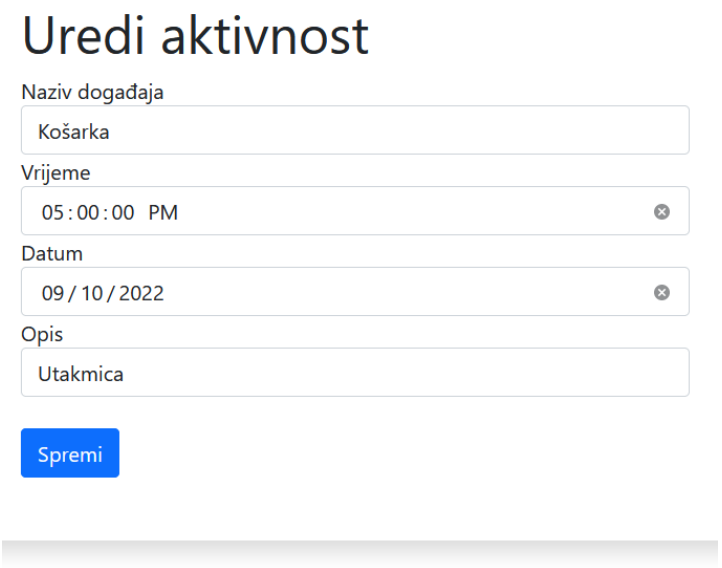
Nakon što na stranici *Moje aktivnosti* korisnik pritisne na gumb *Stvori događaj,* na ekranu će mu se prikazati polja za unos informacija o novom događaju kojeg želi stvoriti. Korisnik može događaju odrediti naziv, vrijeme njegova održavanja, datum pomoću kalendara koji se prikaže klikom na prikazano polje i opis aktivnosti kako bi se znalo o kojoj je aktivnosti riječ u slučaju da postoji više aktivnosti s istim nazivom (*Slika 3.5.)*. Polja za unos naziva i opisa događaja su tipa *text*, polje za unos vremena je tipa *time*, a polje za unos datuma je tipa *date*. Osim informacija koje unosi sam korisnik, klikom na gumb *Spremi* pohranjuje se i ime trenutno prijavljenog korisnika koji je kreator događaja kao i vrijeme kreiranja i posljednjeg uređivanja događaja.



Slika 3.5 Stvaranje novog događaja

3.6. Uređivanje postojećeg događaja

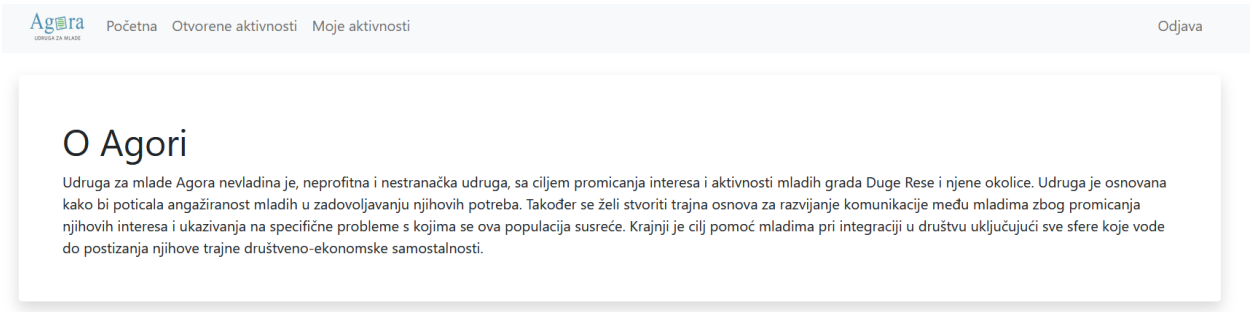
Slično kao kod *Stvaranja novog događaja*, kod *Uređivanja događaja* (*Slika 3.6.*) korisnik unosi podatke o događaju u polja za unos. Razlika je što su u ovom slučaju polja unaprijed ispunjena trenutnim podacima. Pritiskom na gumb *Spremi* korisnik pohranjuje izmjene i aplikacija ga preusmjerava na stranicu *Moje aktivnosti*. Automatski se mijenja vrijednost *updated_at* u bazi podataka.



Slika 3.6. Uređivanje postojećeg događaja

3.7. Početna stranica

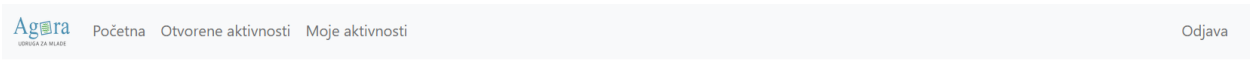
Početna stranica je stranica koja se učita nakon registracije korisnika. Kasnije se do nje može doći pritiskom klika miša na *Početna* na navigacijskoj traci ili klikom na logo udruge. Ona sadrži osnovne podatke o Udruzi za mlade Agora kako bi korisnici bili upućeni u rad udruge. Sadržaj stranice *O Agori* prikazan je na *Slici 3.7.*



Slika 3.7. Početna stranica O Agori

3.8. Navigacijska traka

Navigacijska traka posebno je izrađena Bootstrap komponenta koja se uključena u sve stranice aplikacije osim prijave i registracije zato što je za pristup njenim linkovima potreban token. Na lijevoj strani nalazi se logo Udruge za mlade Agora. Pritiskom na logo poveznica nas vodi na stranicu *O Agori*, isto kao i poveznica *Početna.* Poveznica *Otvorene aktivnosti* vodi nas na stranicu popisa otvorenih aktivnosti i poveznica *Moje aktivnosti* otvara stranicu s popisom aktivnosti koje je stvorio trenutno prijavljeni korisnik. Na desnoj strani nalazi se gumb *Odjava* čijim pritiskom se pokreće funkcija za brisanje (uništavanje) tokena i preusmjerava korisnika na stranicu za prijavu. Navigacijska traka prikazana je na *Slici 3.8.*



Slika 3.8. Navigacijska traka

4. DETALJAN OPIS FUNKCIONALNOSTI I POZADINSKA LOGIKA

4.1. Prijava

Slika 4.1. prikazuje programski kod stranice za prijavu koji se izvršava na klijentskoj strani, a pisan je u Reactu. U ovom primjeru korištena je funkcionalna komponenta.

```
8   function Login(){
9       const navigate = useNavigate();
10      const [payload, setPayload] = useState({
11          email: "",
12          password: "",
13          });
14
15          const getChange = (e) => {
16              console.log(e.target.value);
17              console.log(e.target.name);
18          setPayload({ ...payload, [e.target.name]: e.target.value});
19          };
20
21          const onSubmit = (e) => {
22          e.preventDefault();
23          axios.post('http://homestead.test/api/login', payload, {
24              headers: {
25                  'Content-Type': 'application/json',
26              }
27          }).then(function (response) {
28              console.log(response);
29              localStorage.setItem('token', response.data.token);
30              localStorage.setItem('user', response.data.user.name);
31              console.log(localStorage.getItem('token'));
32              console.log(localStorage.getItem('user'));
33              navigate('/openevents');
34            }).catch(function (error) {
35              console.log("error");
36              window.location.href = "/invalidlogin";
37            });
38          };
```

Slika 4.1 Funkcionalna komponenta stranice za prijavu

U osmoj liniji koda nalazi se deklaracija *useNavigate* kuke (eng. hook). *Hook* nam omogućuje praćenje stanja u funkcionalnoj komponenti. Kuka *UseNavigate* omogućuje programski poziv, odnosno navigaciju na neku drugu putanju kada aplikacija dođe u određeno stanje. Primjer korištenja vidljiv je u liniji 33 gdje je u slučaju uspješne prijave korisnik preusmjeren na stranicu *Open events.* U liniji 10 nalazi se još jedan „hook“, *useState*. Kuka *UseState* služi kako bi se pohranili podaci koji unose i poslali se na serversku stranu. Podaci se pohranjuju u „payload“ koji u sebi sadrži e-mail i lozinku. Njihova vrijednost mijenja se prilikom promjene stanja u prozoru za unos teksta što je implementirano u metodi *getChange* koja se nalazi u 15. liniji. Linije 16 i 17 su kontrolne linije u kojima se ispisuje trenutna vrijednost u prozoru za unos. *SetPayload* metoda u 18. liniji uzima unesene vrijednosti i postavlja vrijednosti e-maila i

lozinke u *Payload-u*. Prilikom pritiska gumba *Prijava*, poziva se metoda *onSubmit* (linija 21). Metoda *preventDefault* u liniji 22 sprječava da se obrada podataka izvrši ranije nego što bi trebala. U 23. liniji poziva se funkcija iz Axios knjižnice koja šalje podatke na serversku stranu. Njeni argumenti su adresa na koju se podaci šalju, zatim što se šalje, u ovom slučaju *payload*, i na kraju se definira sadržaj zaglavlja koji je u ovom slučaju *application/json*. Nakon od linije 27 pa sve do linije 37, aplikacija obrađuje odgovor koji vrati serverska strana. U slučaju uspješno poslanog zahtjeva, ispiše odgovor koji je stigao (linija 28). U odgovoru se nalaze vrijednosti tokena i ime korisnika koji se upravo prijavio u aplikaciju. Ti podaci se spremaju u lokalnu pohranu React-a pomoću funkcija *localStorage.setItem()*. U liniji 29 sprema se vrijednost tokena, a u liniji 30 ime prijavljenog korisnika. Kasnije se ti podaci koriste metodom *localStorage.getItem()*. Nakon toga vrijednosti se zbog kontrole ispišu u konzolu i korisnik se preusmjeri na stranicu *Otvoreni događaji*. U slučaju neuspješnog zahtjeva izvršava se *catch* dio koda koji se nalazi na liniji 34. U tom slučaju u konzolu se ispisuje da je došlo do greške (eng. error) i korisnik je preusmjeren na stranicu o neuspjeloj prijavi gdje se ispisuje poruka *Neuspjela prijava* gdje korisnik može ponoviti prijavu pritiskom na gumb *Pokušajte ponovno* ili *Registriraj se* ukoliko nema stvoren profil kako je prikazano na *Slici 4.2*.

Slika 4.2. Stranica koja se prikaže u slučaju neuspjele prijave

Osim gore objašnjenog logičkog dijela funkcionalne React komponente, u *return* dijelu koda (*Slika 4.3.)* nalazi se dio koji vraća prikaz stranice za prijavu.

```
50      return (
51       <div>
52       <div className="container">
53       <div className="row">
54              <div className="shadow p-5 mb-2 bg-white rounded text-center">
55              <img src={logo} width="250" height="150" alt="logo" />
56              <br></br><br></br>
57              <form className="justify-content-center">
58              <section className="aCenter">
59                  <div className="col-5 mx-auto">
60                      <label htmlFor="formGroupExampleInput">Korisničko ime (E-mail)</label>
61                      <input type="text" name = 'email' className="form-control" id="formGroupExampleInput" onChange={getChange} />
62                  </div>
63                  <div className="col-5 mx-auto">
64                      <label htmlFor="formGroupExampleInput2">Lozinka</label>
65                      <input type="password" name = 'password' className="form-control" id="formGroupExampleInput2" onChange={getChange} />
66                  </div>
67                  <br />
68                  <button type="submit" className="btn btn-primary" onClick={onSubmit}>Prijava</button><span>  </span>
69                  <a href="http://localhost:3000/register"><button type="button" className="btn btn-secondary">Registriraj se</button></a>
70              </section>
71              </form>
72              </div>
73              </div>
74      </div>
75  </div>
76      )
77  }
78
79
80  export default Login
```

Slika 4.3. Dio koda za prijavu korisnika koji formira prikaz komponenti na ekranu

Karakteristika funkcije *return* jest da vraća samo jednu komponentu, u ovom slučaju ona je sadržana u <*div*> oznaci. Unutar jedne, vanjske oznake, nalazi se još podjela unutar iste komponente. Tako je u 54. liniji definiran dio koji se na ekranu pokaže kao manji pravokutnik kojeg okružuje sjena i tako dodatno ističe sadržaj stranice. U liniji 55 pomoću *img src* oznake prikazuje se logo udruge. Nakon njega umetnuta su dva prazna reda (<*br*>) nakon čega je definirana sekcija koja je centrirana i u kojoj se nalaze dva polja za unos podataka. Prvo polje je tipa *text* i u njega korisnik upisuje korisničko ime, odnosno e-mail adresu, a drugo je tipa *password*. U drugo polje korisnik unosi lozinku koja je pritom skrivena na način da se umjesto znakova na ekranu prikazuju točke. U linijama 61 i 65 vidljiv je poziv funkcije *getChange* koja se poziva kada se vrijednost u polju promijeni. Na *Slici 4.3.* polja za unos definirana su od linije 59 do linije 66. Ispod polja za unos nalaze se dva gumba (linije 68 i 69). Pritiskom na prvi gumb *Prijava* poziva se funkcija *onSubmit*, a drugi gumb *Registracija* poveznica je na stranicu za registraciju korisnika implementirana pomoću oznake <*a*>. Svaka otvorena oznaka ima i odgovarajuću zatvarajuću oznaku koju karakterizira kosa crta (/) ispred naziva. U liniji 80 napravljen je tzv. *export* komponente kako bi se mogla koristiti u drugim JavaScript komponentama.

Slika 4.4. Rute u api.php datoteci u Laravelu

Nakon što korisnik unese tražene podatke i pošalje zahtjev na serversku stranu, Laravel, na temelju poslane rute (*Slika 4.4.*), pozove odgovarajući kontroler koji će obraditi zahtjev i vratiti odgovor na klijentsku stranu. Razlikujemo javne i zaštićene rute. Javnim rutama (eng. public routes) korisnik može pristupiti u svakoj situaciji dok je za pristup zaštićenim rutama (eng. protected routes) korisniku potreban token. Pošto korisnik dobije token tek nakon što se uspješno prijavi, za pristup stranicama za prijavu i registraciju on nije potreban. Nakon što s klijentske strane server zaprimi zahtjev na rutu koja sadrži putanju */login,* u liniji 23 pozove se *login* funkcija u kontroleru *AuthController* čija je implementacija prikazana na *Slici 4.5.*

```
35      public function login(Request $request){
36          $fields = $request->validate([
37              'email' => 'required|string',
38              'password' => 'required|string|'
39          ]);
40
41          $user = User::where('email', $fields['email'])->first();
42
43          if(!$user || !Hash::check($fields['password'], $user->password)){
44              return response([
45                  'message' => 'Bad creds'
46              ], 401);
47          }
48
49          $token = $user->createToken('myapptoken')->plainTextToken;
50
51          $response = [
52              'user' => $user,
53              'token' => $token
54          ];
55
56          return response($response, 201);
57      }
```

Slika 4.5. Login funkcija u AuthControlleru

Login funkcija zaprimi zahtjev (eng. request) u kojem se nalaze podaci poslani s klijentske strane u JSON formatu. JSON format je način zapisa podataka koji je ljudima lako čitljiv i temelji se na principu atribut – vrijednost. Funkcija *login* automatski je generirana od strane Laravel Sanctum paketa. U prvom koraku, u liniji 36, provjeri je li zahtjev ispravan i pohrani ga u varijablu *fields.* Nakon toga po kriteriju e-mail adrese pronađe korisnika u bazi podataka (linija 41). Ukoliko traženi korisnik ne postoji ili lozinka nije ispravna, vraća se poruka klijentskoj strani da prijava nije uspjela što označava kod 401. Ukoliko je korisnik unio ispravne podatke, generira se token u 49. liniji i server vraća odgovor koji sadrži vrijednosti imena korisnika koji se prijavio i tokena uz kod 201 koji označava uspješnu prijavu.

4.2. Stvaranje novog događaja

Za stvaranje novog događaja također je korištena funkcionalna React komponenta prikazana na *Slici 4.6.*

Slika 4.6. Funkcionalna komponenta korištena za stvaranje novog događaja

Payload u ovom slučaju sadrži više podataka. Za kreiranje novog događaja potrebno je u njega pohraniti naziv događaja, vrijeme, mjesto, datum i opis. Kreator događaja sprema se automatski na način da se sa *localStorage.getItem* dohvati ime trenutno prijavljenog korisnika što je vidljivo u liniji 15. Linija 18 prikazuje postavljanje vrijednosti *payload-a* prilikom promjene stanja polja za unos. Dobra je praksa ispisivati trenutne vrijednosti u konzolu radi lakšeg traženja i uklanjanja mogućih pogrešaka kao što je učinjeno u 22. liniji. Kako je ruta za kreiranje novog događaja na serverskoj strani definirana kao zaštićena, potrebno je iz lokalne pohrane dohvatiti token i poslati ga u zaglavlju zahtjeva. Token iz lokalne pohrane dohvatimo funkcijom *localStorage.getItem('token')* i pošaljemo ga POST metodom kao vrijednost atributa *Authorization* u zaglavlju zahtjeva, dok se ostali podaci šalju u tijelu zahtjeva (eng. body). Nakon što server vrati odgovor o uspješnosti zahtjeva, aplikacija odlučuje hoće li korisnik biti preusmjeren na stranicu Moji događaji (linija 35) ili će biti preusmjeren na stranicu za prijavu ukoliko nema aktivni token (linija 38). Te dvije linije prikazuju i dva različita načina preusmjeravanja. *Navigate* koristi *hook* dok *window.location.href* vraća trenutnu URL lokaciju i njenom promjenom se učitava željena stranica. Oba načina imaju istu ulogu, ali je razlika u logici izvršavanja.

Prikaz stranice za unos novog događaja realiziran je na način koji prikazuje *Slika 4.7.*

```
return (
<div>
    <div>
    <NavbarComp />
    </div>
    <div className="container">
    <div className="row">
    <div className="shadow p-5 mb-2 bg-white rounded">
    <h1>Stvori novi događaj</h1>
    <form>
        <div className="col-5">
            <label htmlFor="formGroupExampleInput">Naziv događaja</label>
            <input type="text" name = 'name' className="form-control" id="formGroupExampleInput"
            defaultValue={payload.name || ""} onChange={getChange}  />
        </div>
        <div className="col-5">
            <label htmlFor="formGroupExampleInput2">Vrijeme</label>
            <input type="time" name = 'time' className="form-control" id="formGroupExampleInput2"
            defaultValue={payload.time || ""} onChange={getChange}/>
        </div>
        <div className="col-5">
            <label htmlFor="formGroupExampleInput2">Datum</label>
            <input type="date" name = 'date' className="form-control" id="formGroupExampleInput2"
            defaultValue={payload.date || ""} onChange={getChange}/>
        </div>
        <div className="col-5">
            <label htmlFor="formGroupExampleInput2">Opis</label>
            <input type="text" name = 'description' className="form-control" id="formGroupExampleInput2"
            placeholder="Dodajte kratki opis događaja" defaultValue={payload.description || ""} onChange={getChange}/>
        </div>
        <br />
        <button type="submit" className="btn btn-primary" onClick={onSubmit}>Spremi</button>
    </form>
    </div>
    </div>
    </div>
</div>
```

Slika 4.7. Return funkcija komponente za stvaranje novog događaja

U *return* dijelu koda korištena je * * komponenta rastavljena na manje komponente unutar nje. React nam daje mogućnost umetanja postojećih komponenti koje su definirane u drugoj datoteci na više različitih mjesta. Primjer korištenja te metode je umetanje navigacijske trake koristeći oznaku *<NavbarComp />*. Kako bi se ta oznaka mogla koristiti, kao i ostale funkcije iz vanjskih knjižnica, potrebno ih je uvesti (eng. import) u komponentu u zaglavlju koda kako je prikazano na *Slici 4.8.*

```
import NavbarComp from "../components/Navbar"
import 'bootstrap/dist/css/bootstrap.css'
import axios from 'axios'
import React, { useState } from "react";
import {useNavigate} from 'react-router-dom';
```

Slika 4.8. Uvoz knjižnica i komponenata

Od linije 52 do linije 75 nalazi se implementacije forme (oznaka *<form>)* u kojoj su definirana polja za unos podataka. Polja u koja korisnik unosi naziv i opis događaja su tipa *text*, dok je za ostala polja korišten specifični tip za namjenu koja je potrebna. Tako je za unos vremena korišten tip *time* koji omogućuje jednostavniji unos u obliku 12-satnog formata, a za unos datuma korišten je tip *date* koji omogućuje odabir datuma pomoću kalendara koji se otvori klikom miša na to polje. U 74. liniji implementiran je gumb *Spremi* koji je tipa *submit* i čijim se pritiskom pozove funkcija *onSubmit* te podaci se pošalju na serversku stranu.

Na serverskoj strani poziva se, pomoću odgovarajućeg URL-a, funkcija *store* koja se nalazi u kontroleru *EventController*. Njegova implementacija prikazana je na *Slici 4.9.*

```
40      public function store(Request $request)
41      {
42          return Events::create($request->all());
43      }
```

Slika 4.9. Funkcija za kreiranje novog događaja

Ova jednostavna funkcija zaprima tijelo (eng. body) zahtjeva i napravi upit prema bazi koristeći metodu *Events::create* koja je sastavni dio Laravela i odličan primjer jednostavnosti njegova korištenja.

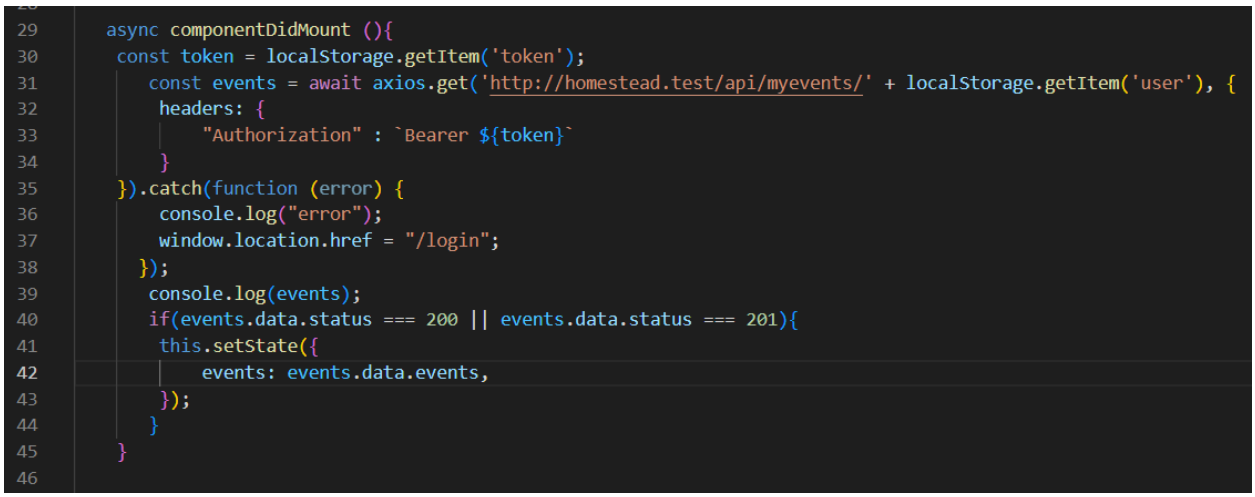
4.3. Ispis Mojih aktivnosti

Stranica na kojoj se prikazuje ispis svih aktivnosti koje je stvorio trenutno prijavljeni korisnik nije implementirana pomoću funkcionalne komponente kao prethodni primjeri već pomoću klase. Klasa omogućuje korištenje stanja (eng. state). *State* je ugrađeni React objekt koji se koristi za spremanje podataka. Implementacija stanja korištenog u *MyEvents* komponenti prikazana je na *Slici 4.10.* počevši od 8. linije koda.

```
7   class MyEvents extends Component {
8       state = {
9           events: [],
10      }
11      destroy(id){
12          console.log(id);
13          this.deleteOperation(id);
14
15      }
16
17      async deleteOperation(id) {
18          const token = localStorage.getItem('token');
19          await axios.delete('http://homestead.test/api/delete/' + id, {
20              headers: {
21                  "Authorization" : `Bearer ${token}`
22              }
23          });
24          console.log("test");
25          window.location.reload(false);
26      }
27
```

Slika 4.10. Stanje, brisanje događaja i zahtjev na poslužiteljsku stranu

U ovom primjeru u *state* se pohranjuju događaji koji su dohvaćeni iz baze podataka. Na početku polje *events* inicijalizira kao prazno polje jer će tek naknadno biti popunjeno. U 11. liniji nalazi se poziv funkcije *deleteOperation* prilikom kojeg se funkciji prosljeđuje *id* vrijednost događaja koji će biti izbrisan. U 17. liniji definirana je funkcija *deleteOperation.* Riječ *async* označava knjižnicu koja omogućuje deklarativno dohvaćanje podataka. Prije slanja zahtjeva potrebno je dohvatiti token iz lokalne pohrane što je učinjeno u liniji 18. Prilikom slanja zahtjeva *delete* metodom potrebno je koristiti oznaku *await*. Ona uzrokuje pauzu izvršavanja asinkrone funkcije dok poslužitelj ne vrati odgovor o zaprimljenom i izvršenom zahtjevu. Kada odgovor stigne, funkcija nastavlja s izvršavanjem. Nakon što se željeni događaj izbriše, potrebno je osvježiti prikaz stranice kako bi promjena bila vidljiva što je učinjeno u liniji 25. Korištena je metoda *window.location.reload* koja vrši ponovno učitavanje stranice na kojoj se korisnik trenutno nalazi. Argument *false* određuje da se ponovno učita verzija stranice koju je internetski preglednik već dohvatio, dok bi argument *true* odredio da se stranica ponovno dohvati sa servera. U ovoj situaciji nije potrebno preusmjeravati korisnika na stranicu za prijavu u slučaju da nema aktivni token jer je to učinjeno prilikom inicijalnog učitavanja stranice kada se šalje zahtjev na poslužitelj za dohvat događaja kako je prikazano na *Slici 4.11.*



Slika 4.11. ComponentDidMount metoda

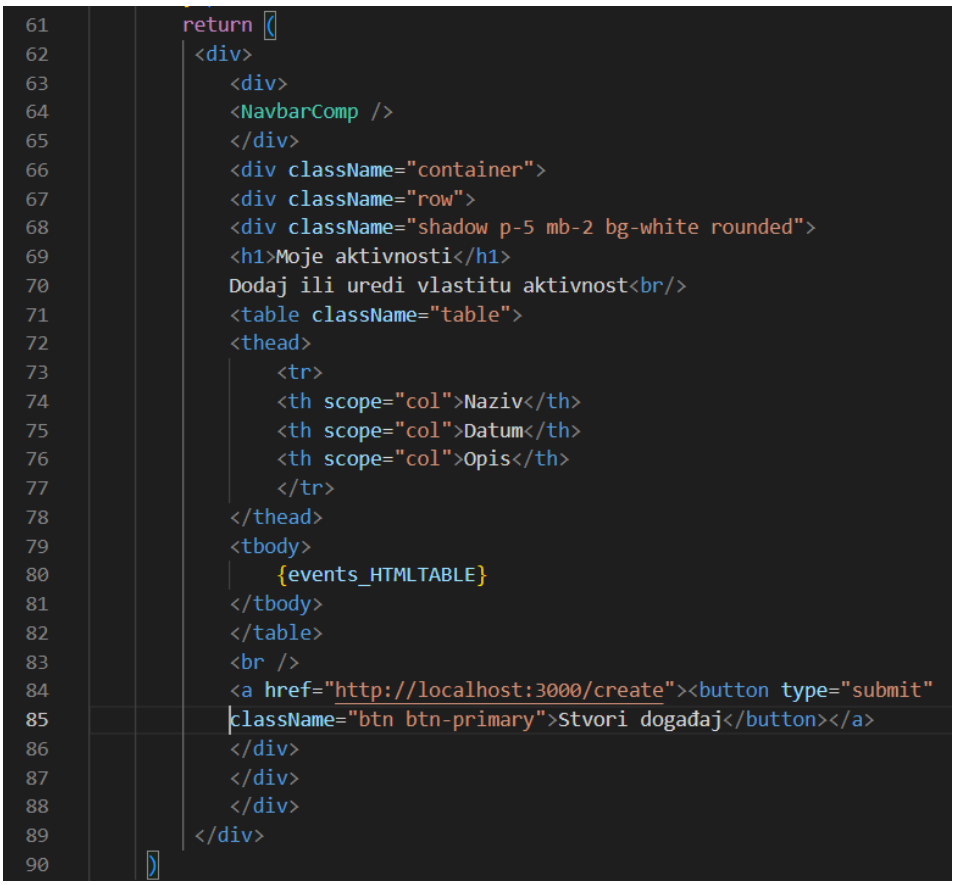
U 29. liniji nalazi se metoda *componentDidMount.* Njena karakteristika je da se poziva samo jednom prilikom učitavanja stranice. U njoj se nalazi algoritam za dohvaćanje podataka iz baze. U 30. liniji dohvaćen je token iz lokalne pohrane, a u sljedećoj se izvršava zahtjev prema poslužitelju i odgovor se pohranjuje u varijablu *events.* Ovaj zahtjev realiziran *get* metodom specifičan je zbog toga što se uz URL na poslužitelj šalje i ime trenutno prijavljenog korisnika kako bi se iz baze dohvatili samo oni događaji koje je stvorio trenutno prijavljeni korisnik. Ime se uzima iz lokalne pohrane i šalje se na način da se doda na kraj URL-a. U 37. liniji korisnik se preusmjerava na stranicu za prijavu ukoliko nema valjani token. Pošto se provjera vrši prilikom učitavanja stranice, nije ju potrebno raditi prilikom brisanja podataka jer bez valjanog tokena korisnik ne može pristupiti gumbu za brisanje jer se on prikazuje zajedno s podacima koji su pročitani iz baze. Nakon uspješnog zahtjeva i dobivenog odgovora polje *events* poprima sadržaj koji je pohranjen u odgovoru što je vidljivo u liniji 42.

Još jedna bitna razlika između klase i funkcionalne komponente jest način implementacije prikaza. U funkcionalnoj komponenti kod se nalazi unutar funkcije *return,* dok se kod klase koristi metoda *render* koja u sebi sadrži funkciju *return.* Ispis je formatiran u obliku tablice koja se sastoji od 4 supca. Tablica je implementirana kako je prikazano na *Slici 4.12.*

```
47      render(){
48          var events_HTMLTABLE = "";
49          events_HTMLTABLE =
50          this.state.events.map( (item) => {
51              return(
52                  <tr key={item.id}>
53                      <td><a href={"/edit/"+item.id}>{item.name}</a></td>
54                      <td>{item.date}</td>
55                      <td>{item.description}</td>
56                      <td><button type="submit" className="btn btn-danger"
57                      onClick={() => this.destroy(item.id)}>Obriši</button></td>
58                  </tr>
59              );
60          } );
```

Slika 4.12. Implementacija tablice

U *render* metodi definirana je varijabla *events_HTMLTABLE* koja pomoću funkcije *map* dohvaća podatke iz polja *events* koji su pohranjeni u stanju *state.* Na taj način izvodi se iteracija kroz polje iz kojeg se čitaju podaci i formira se tablica. Prvi stupac tablice je naziv aktivnosti koji je ujedno i link. Pritiskom klika miša na naziv aktivnosti korisnik se preusmjerava na stranicu Uredi aktivnost. Na isti način implementirana je i tablica na stranici Otvorene aktivnosti, međutim u tom slučaju korisnik se preusmjerava na stranicu gdje nije omogućeno uređivanje događaja. U liniji 53 vidljivo je da je u link potrebno ubaciti i *id* događaja koji se želi urediti kako bi se njegovi trenutni podaci mogli dohvatiti iz baze podataka. Drugi stupac prikazuje datum događaja, treći opis, a četvrti stupac sadrži gumb *Obriši*. Pritiskom klika miša na gumb poziva se prethodno opisana funkcija *destroy* kojoj se predaje i *id* vrijednost događaja kojeg je potrebno obrisati. Nakon što je definirana tablica koja sadrži podatke, potrebno ju je ubaciti u ostatak prikaza stranice koji se definira u funkciji *return* kao što je prikazano na *Slici 4.13.*



Slika 4.13. Return funkcija unutar render metode

U funkciji *return* nalazi se implementacija konačnog prikaza na ekranu. Unutar vanjske * * komponente umetnuta je *<NavbarComp>* komponenta (linija 64) koja sadrži implementaciju navigacijske trake definiranu u zasebnoj JavaScript datoteci. Ispod navigacijske trake ispisuje se naslov *Moje aktivnosti* s opisom funkcionalnosti stranice. Zatim se definira komponenta *<table>*. Ona se sastoji od zaglavlja kojeg označava oznaka *<thead>* (linija 72) i tijela (eng. body) kojeg označava oznaka *<tbody> (linija 79*). U zaglavlju se nalaze tri stupca koja predstavljaj prvi red tablice i sadrže nazive svakog od stupaca. U tijelu tablice se nalazi prethodno definirana *events_HTMLTABLE* koja sadrži podatke dohvaćene iz baze podataka. Na dnu stranice nalazi se gumb *Stvori događaj* koji je poveznica na stranicu za stvaranje novog događaja. Dostupan je samo na stranici *Moje aktivnosti*.

Funkcionalnosti stranice *Moje aktivnosti* zahtijevaju slanje dva zahtjeva prema poslužitelju. Prvi zahtjev zatraži podatke o događajima koje je kreirao trenutno prijavljeni korisnik, a drugi zahtjev zahtijeva brisanje određenog događaja. Prilikom učitavanja stranice na poslužitelj stigne zahtjev na URL „/myevents“ uz dodatak imena korisnika. Kako bi poslužitelj ispravno zaprimio ime, potrebno je u *api.php* datoteci dati do znanja da se na tom mjestu očekuje neka

vrijednost. To možemo učiniti stavljanjem očekivane varijable u vitičaste zagrade prilikom definiranja rute kako je prikazano na *Slici 4.14.* u liniji 29.

```
27  Route::group(['middleware' => ['auth:sanctum']], function () {
28      Route::get('/events', [EventController::class, 'index']);
29      Route::get('/myevents/{creator}', [EventController::class, 'index_my']);
30      Route::get('/logout', [AuthController::class, 'logout']);
31      Route::get('/show/{id}', [EventController::class, 'show']);
32      Route::post('/update', [EventController::class, 'update']);
33      Route::delete('/delete/{id}', [EventController::class, 'destroy']);
34      Route::post('/create', [EventController::class, 'store']);
35  });
```

Slika 4.14. Definiranje rute koja sadrži poslanu vrijednost

Nakon što poslužitelj zaprimi zahtjev, poziva funkciju *index_my* u kontroleru *EventController*. Implementacija funkcije *indeks_my* prikazana je na *Slici 4.15*.

```
24
25      public function index_my($creator)
26      {
27          $events = DB::select("SELECT * FROM events WHERE creator = '$creator'");
28          return response()->json([
29              'status'=>200,
30              'events'=>$events
31          ]);
32      }
33
```

Slika 4.15. Funkcija index_my

Funkcija zaprima varijablu *creator* koja sadrži ime korisnika čije aktivnosti moraju biti poslane natrag na klijentsku stranu. Kako bi aktivnosti bile spremljene u varijablu *events,* potrebno je napraviti upit na bazu podataka. To je učinjeno koristeći SQL upitni jezik. Riječ *SELECT* znači da se iz baze vrši čitanje, * govori da se čitaju svi stupci u tablici, *events* označava ime tablice iz koje se podaci čitaju i *WHERE creator = '$creator'* označava da se dohvaćaju samo redovi tablice u kojima je vrijednost stupca *creator* jednaka imenu trenutno prijavljenog korisnika.

Nakon što su podaci uspješno dohvaćeni, poslužitelj vraća na klijentsku stranu odgovor u JSON formatu koji sadrži kod *200* koji označava uspješnu radnju i vrijednost zapisanu u varijabli *events* u kojoj se nalaze traženi podaci.

4.4. Detalji o događaju

```
6   class EventDetails extends Component {
7       state = {
8           events: []
9       }
10
11      async componentDidMount (){
12       const token = localStorage.getItem('token');
13          var path = window.location.pathname
14          var id = path.split("/")
15         const events = await axios.get('http://homestead.test/api/show/'+id[2], {
16          headers: {
17              "Authorization" : `Bearer ${token}`
18          }
19      }).catch(function (error) {
20          console.log(error);
21          window.location.href = "/login";
22        });
23          console.log(events);
24          if(events.data.status === 200){
25           this.setState({
26              events: events.data.events[0]
27           });
28           //console.log(events.data.events[0].name)
29          }
30      }
```

Slika 4.16. Klasa EventDetails

Slika 4.16. prikazuje implementaciju jednostavne klase *EventDetails.* Klasa se sastoji od stanja *state* u 7. liniji i funkcije *componentDidMount* koja započinje u 11. liniji. Stanje se sastoji od inicijalno praznog polja *events.* Funkcija *componentDidMount* ima ulogu slanja zahtjeva na poslužiteljsku stranu isto kao u prethodnim primjerima, ali uz jednu bitnu razliku. Prilikom pozivanja URL-a na stranici Otvoreni događaji, na kraju URL-a nalazi se *id* događaja čiji se detalji prikazuju. Kako bi aplikacija mogla znati koji *id* poslati na poslužiteljsku stranu potrebno je iz URL-a iščitati vrijednost identifikatora. To je učinjeno koristeći funkciju *window.location.pathname.* Na taj način dohvati se *string* koji sadrži trenutni URL. Vrijednost se pohranjuje u varijablu *path.* Koristeći funkciju *split,* kao što je to učinjeno u liniji 14, moguće je razdvojiti vrijednost zapisanu u varijabli prema nekom kriteriju. U ovom slučaju *string* je razdvojen pomoću kose crte (/). Time varijabla *id* postaje polje. Konkretno, u ovom primjeru je skup znakova *localhost:3000/eventdetails/8* postao podijeljen na članove polja *[localhost:3000, eventdetails, 8]*. Traženi *id* nalazi se na trećem mjestu u polju, odnosno na mjestu s indeksom dva. Zbog toga URL u zahtjevu na kraju sadrži vrijednost *id[2].*

Kako bi bilo moguće koristiti slanje *id* vrijednosti sa stranice Otvorene aktivnosti na stranicu *Detalji o događaju*, potrebno je na odgovarajući način definirati rutu u *React Routeru* koji se nalazi u datoteci *App.js*. *App.js* glavna je komponenta u Reactu koja se ponaša kao spremnik za sve ostale komponente. Njena implementacija prikazana je na *Slici 4.17.*

```
1  import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
2  import About from "./pages/About";
3  import EditEvent from "./pages/EditEvent";
4  import EventDetails from "./pages/EventDetails";
5  import MyEvents from "./pages/MyEvents"
6  import NewEvent from "./pages/NewEvent";
7  import OpenEvents from "./pages/OpenEvents";
8  import Login from "./pages/Login";
9  import Register from "./pages/Register";
10 import Logout from "./pages/Logout";
11 import InvalidLogin from "./pages/InvalidLogin";
12
13 const App = () => {
14   return (
15     <Router>
16       <Routes>
17       <Route path="/" element={<OpenEvents />}></Route>
18       <Route path="/myevents" element={<MyEvents />}></Route>
19       <Route path="/openevents" element={<OpenEvents />}></Route>
20       <Route path="/about" element={<About />}></Route>
21       <Route path="/edit/:id" element={<EditEvent />}></Route>
22       <Route path="/create" element={<NewEvent />}></Route>
23       <Route path="/eventdetails/:id" element={<EventDetails />}></Route>
24       <Route path="/login" element={<Login />}></Route>
25       <Route path="/register" element={<Register />}></Route>
26       <Route path="/logout" element={<Logout/>}></Route>
27       <Route path="/invalidlogin" element={<InvalidLogin/>}></Route>
28       </Routes>
29     </Router>
30
31   )
32 }
33
34 export default App
```

Slika 4.17. App.js

Za razliku od Laravela, gdje se varijabla unutar URL-a zapisuje u vitičastim zagradama, u Reactu ona se označava dvotočkom (:) što je vidljivo u liniji 23 gdje se nalazi definicija rute za stranicu Detalji o događaju. Sve rute definirane su unutar *Routera* koji je sastavni dio paketa *React Router DOM* koji je u *App.js* uvezen u liniji 1, a od linije 2 do linije 11 uvezene su sve komponente aplikacije. U liniji 34 izvezena je komponenta *App* koja sadrži sve značajke aplikacije.

```
51      public function show($id)
52      {
53          $event = DB::select("SELECT * FROM events WHERE id = $id");
54          return response()->json([
55              'status'=>200,
56              'events'=>$event
57          ]);
58      }
59
```

Slika 4.18. Funkcija show

Na poslužiteljskoj strani zaprima se zahtjev koji u sebi sadrži *id* događaja koji se dohvaća. Poziva se funkcija *show* (*Slika 4.18*) koja se nalazi u kontroleru *EventController*. Odrađuje se poziv na bazu definiran SQL upitnim jezikom i odgovor se sprema u varijablu *event* čija se vrijednost šalje u JSON formatu natrag na klijentsku stranu uz status kod. Kako bi upit bio obavljen prema ispravnoj bazi podataka i kako bi se moglo koristiti *DB* naredbe, potrebno je i na poslužiteljskoj strani u kontroleru definirati kakvo će se nazivlje koristiti (eng. namespace). Ono se definira ključnom riječi *use* kako je prikazano na *Slici 4.19.*

```
use Illuminate\Http\Request;
use DB;
use App\Models\Events;
```

Slika 4.19 Definiranje aliasa "namespacea"

5. ZAKLJUČAK

Aplikacija za raspored aktivnosti rađena je za potrebe Udruge za mlade Agora kako bi riješila problem organizacije aktivnosti i njihova prikaza na jednom mjestu. Do sada se koristio način dijeljenog dokumenta u oblaku koji je nepregledan i često kompliciran za pristup sa različitih uređaja. Ova aplikacija omogućuje vrlo jednostavan, pregledan i intuitivan pristup organizaciji i upravljanju aktivnostima.

Korištene tehnologije također omogućuju programeru da jednostavno realizira ideje bez previše razmišljanja o pozadini rada tehnologija i alata. Za implementaciju proizvoda nije potrebna ni dodatna oprema jer korisnik sve može implementirati virtualno na vlastitom računalu bez velikih i teških prilagodbi softvera uz jednostavne instalacije. Laravel pruža vrlo jednostavnu integraciju PHP razvojnog okruženja na serverskoj strani uz pomoć mnogobrojnih paketa i knjižnica koje se mogu instalirati kako bi se olakšao razvoj. Kombinacija s Laravel Homesteadom i Vagrant virtualnim strojem osigurava brzo i jednostavno pokretanje virtualnog poslužitelja i baze podataka. Na klijentskoj strani React također omogućuje korištenje raznih knjižnica koje se vrlo lako dodaju u aplikaciju i višestruko pojednostave ionako već jednostavan i moćan JavaScript.

Sve spomenute tehnologije ukomponirane u REST arhitekturu čine stabilan sustav koji je uspješan u izvršavanju svojih zadaća. Komunikacija između klijenta i poslužitelja je jednostavna i laka za implementirati.

LITERATURA

[1] Laravel, https://en.wikipedia.org/wiki/Laravel, 13.9.2022.

[2] Learn PHP, https://www.codecademy.com/learn/learn-php, 13.9.2022.

[3] Što je PHP?, http://www.netakademija.hr/sto-je-php/, 13.9.2022.

[4] Laravel Sanctum, https://laravel.com/docs/9.x/sanctum#introduction, 10.9.2022.

[5] Getting started with Vagrant, https://semaphoreci.com/community/tutorials/getting-started-with-vagrant, 10.9.2022.

[6] Vagrant (software), https://en.wikipedia.org/wiki/Vagrant_(software), 13.9.2022.

[7] Laravel Homestead, https://laravel.com/docs/9.x/homestead, 10.9.2022.

[8] MySQL, https://en.wikipedia.org/wiki/MySQL, 13.9.2022.

[9] Documentation for Visual Studio Code, https://code.visualstudio.com/docs, 13.9.2022.

[10] Visual Studio Code, https://en.wikipedia.org/wiki/Visual_Studio_Code, 10.9.2022.

[11] React (JavaScript library), https://en.wikipedia.org/wiki/React_(JavaScript_library), 10.9.2022.

[12] The Best Guide to Know What Is React, https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs#what_is_react, 13.9.2022.

[13] Bootstrap, https://getbootstrap.com/, 11.9.2022.

[14] Bootstrap, https://www.geeksforgeeks.org/bootstrap/, 13.9.2022.

[15] What is REST?, https://www.codecademy.com/article/what-is-rest, 13.9.2022.

[16] Working with JSON, https://developer.mozilla.org/enUS/docs/Learn/JavaScript/Objects/JSON, 13.9.2022.

[17] What is JSON? The universal data format, https://www.infoworld.com/article/3222851/what-is-json-a-better-format-for-data-exchange.html, 13.9.2022.

POPIS SLIKA

SAŽETAK

Aplikacija za upravljanje događajima omogućuje svakom korisniku stvaranje i uređivanje aktivnosti i pregled svih aktivnosti koje su kreirali ostali korisnici. Aplikacija je izrađena na principu REST arhitekture. Poslužiteljska strana izrađena je u Laravelu 8, a klijentska u Reactu. U uvodu ovog završnog rada opisana je uloga aplikacije i nabrojane su tehnologije korištene u izradi. Zatim su korištene tehnologije detaljnije opisane i prikazane su glavne funkcionalnosti aplikacije. Funkcionalnosti *Prijava*, *Stvaranje novog događaja*, *Ispis Mojih aktivnosti* i *Detalji o događaju* opisane su na razini koda. U zaključku je navedeno zašto je aplikacija korisna i koje su prednosti tehnologija korištenih u izradi projekta.

Ključne riječi: React, Laravel, REST arhitektura, poslužitelj, klijent, web aplikacija, upravljanje događajima

ABSTRACT

Event management application allows each user to create and edit activities and view all activities created by other users. The application is built on the principle of REST architecture. The server side is made in Laravel 8 and the client side in React. In the introduction of this paper, the role of application is described and the technologies used in the creation are enumerated. Then the technologies used are described in more detail and the main functionalities of the applications are presented. Functionalities *Login*, *Create a new even*t, *List of My Activities* and *Event Details* are described at the code level. In the conclusion, it is stated why the application is useful and what are the advantages of the technologies used in the creation of the project.

Keywords: React, Laravel, REST architecture, server, client, web application, event management