Integracija React razvojnog kostura u Laravel web aplikaciji

Križanić, Dario

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:916272

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-04**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Stručni studij

INTEGRACIJA REACT RAZVOJNOG KOSTURA U LARAVEL WEB APLIKACIJI

Završni rad

Dario Križanić

Osijek, 2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Osijek, 14.09.2022.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Ime i prezime Pristupnika:	Dario Križanić
Studij, smjer:	Preddiplomski stručni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	AR4671, 26.07.2018.
OIB Pristupnika:	97095262876
Mentor:	Izv. prof. dr. sc. Irena Galić
Sumentor:	Dr. sc. Hrvoje Leventić
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc. dr. sc. Tomislav Galba
Član Povjerenstva 1:	Dr. sc. Hrvoje Leventić
Član Povjerenstva 2:	Dr. sc. Krešimir Romić
Naslov završnog rada:	Integracija React razvojnog kostura u Laravel web aplikaciji
Znanstvena grana završnog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada	Zadatak završnog rada je opisati načine integracije React razvojnog kostura u Laravel web aplikaciju. Istražiti i opisati Laravel i React razvojne kosture, te pomoćne alate za integraciju (webpack, yarn, npm). U praktičnom dijelu rada potrebno je izraditi web aplikaciju koja će demonstrirati integraciju Reacta i Laravela. Tehnologija: PHP, Laravel, JS, React Tema rezervirana za: Dario Križanić Sumentor s FERIT-a: Hrvoje Leventić
Prijedlog ocjene pismenog dijela ispita (završnog rada):	Vrlo dobar (4)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 1 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	14.09.2022.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 22.10.2022.

Ime i prezime studenta:	Dario Križanić
Studij:	Preddiplomski stručni studij Računarstvo
Mat. br. studenta, godina upisa:	AR4671, 26.07.2018.
Turnitin podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Integracija React razvojnog kostura u Laravel web aplikaciji**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Irena Galić

i sumentora Dr. sc. Hrvoje Leventić

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

1. UVOD 1

2.Pregled klijentskih razvojnih kostura 2

2.1.React 2

2.2. Alternative 3

3.Korisničko i programsko sučelje 4

3.1. React.js 4

3.2. Funkcijska i razredna komponenta 5

3.3. Validacije i notifikacije 8

3.4. REST API 9

4.Integracija klijentskog razvojnog kostura u Laravel mrežnu aplikaciju 13

4.1. Laravel kao poslužiteljska tehnologija 13

4.2. Struktura aplikacije 13

4.3. MVC 14

4.4. Alati za upravljanje paketima 15

4.5. Opis aplikacije 16

5.ZAKLJUČAK 21

6.LITERATURA 22

1. UVOD

Cilj ovog završnog rada je upoznavanje s izradom *fullstack* web aplikacije koristeći Laravel i PHP na serverskoj strani i React i JavaScript na strani korisničkog sučelja. Za potrebe demonstracije napravljena je aplikacija za organizaciju poslova s dva tipa korisnika, administratora odnosno poslodavca i zaposlenika. Administrator nema nikakvih restrikcija i on ima sva prava zaposlenika. Aplikacija je fokusirana na organizaciju radnika, projekata i poslova, najveći dio cijele logike vezan je za administratorsku stranu. Na strani aplikacije zaposlenika nalaze se kartice poslova koje bi radnik prema odgovarajućem opisu i u odgovarajućem vremenskom periodu trebao odraditi.

U sljedećim poglavljima bit će objašnjeni svi bitni dijelovi i koncepti izrade *fullstack* aplikacije od korisničkog sučelja preko API-ja do serverske strane. Bit će objašnjeni pojmovi kao što su MVC i REST, te najbolje prakse njihove uporabe kao i sve ostale tehnologije korištene pri izradi aplikacije.

2. Pregled klijentskih razvojnih kostura

2.1.React

Korisničko sučelje izrađeno koristeći isključivo Laravel građeno je od velikog broja HTML stranica odnosno Laravelovih *blade*-ova. Konstanto učitavanje novih stranica aplikacije spor je proces, stoga sve moderne aplikacije koriste jedan od *frameworka* za izradu korisničkog sučelja.

Korištenjem *frameworka* za izradu korisničkog sučelja velik broj stranica smanjuje se na samo jednu stranicu odnosno jedan *blade*.

```
<!DOCTYPE html>
    <html lang="{{ app()->getLocale() }}">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <meta name="csrf-token" content="{{ csrf_token() }}">
        <link href="{{ asset('css/app.css') }}" rel="stylesheet">
    </head>
    <body>
        <div id="app"></div>
        <script src="{{ asset('js/app.js') }}"></script>
    </body>
    </html>
```

Kod 2.1 Blade.

```
if (document.getElementById('app')) {
    ReactDOM.render(<App />, document.getElementById('app'));
}
```

Kod 2.2 React render.

Kod 2.1. prikazuje *blade* u čijem se tijelu prikazuje *App* komponenta. U *App* komponenti se ovisno o interakciji korisnika prikazuje različita komponenta.

2.2. Alternative

Najčešće korištene alternative React-a su Angular i Vue. Angular je MVC *framework* za izradu korisničkog sučelja, namijenjen izradi vrlo kompleksnih aplikacija. Angular je najkompleksniji i najveći *framework* s velikim brojem pred-instaliranih paketa. Koristi pravi DOM za razliku React-a i Vue. Vue je *framework* MVVM (model-view-view model) arhitekture, također se kao i React može pohvaliti svojim perfomansama i u suprotnosti s Angularom, ove se dvije tehnologije oslanjaju na korištenje velikog broja vanjskih biblioteka, što nije nužno nikakav nedostatak jer će Angular sigurno imati paketa koji se nikada neće niti koristiti. React ima najveću fleksibilnost, dok je Angular najviše ograničen i kontroliran.

3. Korisničko i programsko sučelje

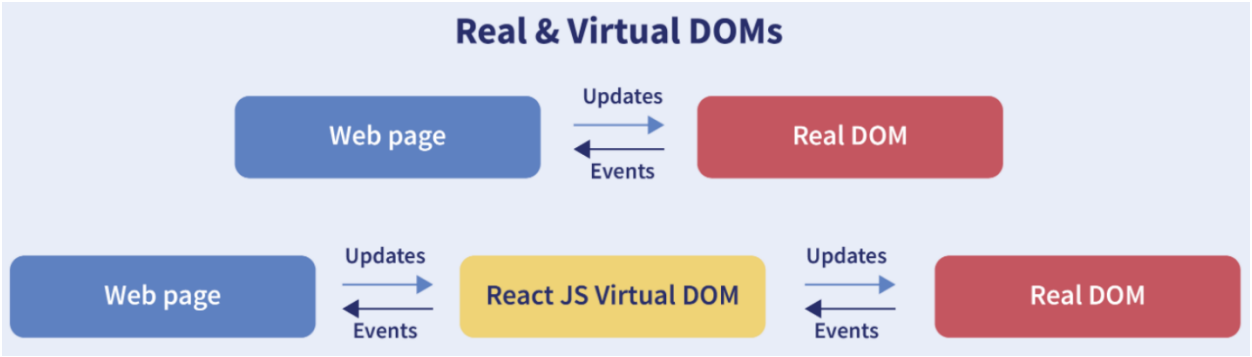
3.1. React.js

Za izradu korisničkog sučelja korištena je JavaScript biblioteka React.js. React je najčešće korištena tehnologija za izradu korisničkog sučelja, uz Angular i Vue kao konkurencije. React se koristi za izradu jednostraničnih web aplikacija kao i mobilnih aplikacija koristeći React Native. Prednost korištenja ovih tehnologija je što se sve baziraju na jednostraničnoj metodologiji, gdje nema primitivnog učitavanja svake zasebne web stranice kada postoji potreba za promjenom sadržaja i prelaska na drugu hipervezu stranice. Kako sam pojam glasi SPA - *Single Page Application,* u jednostraničnoj aplikaciji sve se odvija na jednoj web stranici gdje se ovisno o korisničkoj interakciji i unosima prikazuju specifične komponente koje su učitane pri prvom pokretanju web aplikacije. SPA na taj način pruža znatno bržu interakciju korisnika i sučelja. Najveća prednost React.js- je korištenje *Virtual* DOM-a [1].

Virtualni DOM
1. Brže ažuriranje
2. Bez gubitaka u memoriji
3. Jednostavne manipulacije
4. Nije moguće direktno ažurirati HTML

Slika 3.1 Razlika stvarnog i virtualnog DOM-a

Virtual DOM virtualna je reprezentacija pravog DOM-a. Svaki put pri promjeni podataka aplikacije dolazi do novog kreiranja virtualnog DOM-a, njegovo kreiranje je brzo, stoga virtualni DOM povećava efikasnost i brzinu aplikacije.



Slika 3.2 Dom način rada [1]

Za svaki DOM objekt postoji njegova virtualna kopija. React koristi dvije virtualne kopije, dva virtualna DOM-a. Jedna kopija sadrži trenutno stanje objekta, dok druga sadrži prethodno stanje objekta. React uspoređuje te dvije kopije kako bi pronašao razliku između verzija, nakon što odredi

razliku između objekata, React te promjene ažurira u pravi DOM, stoga se pravi DOM ne mora ponovno učitavati već se samo obrade promijenjeni objekti.

3.2. Funkcijska i razredna komponenta

Prije uvođenja *Hooks*-a funkcijske komponente su se zvale *stateless* i bile su ograničene funkcionalnostima u odnosu na razredne komponente koje su imale mogućnost manipuliranja stanjima (*state*) Uvođenjem Hooks-a funkcijske komponente dobile su sposobnost manipuliranja stanjima, te su s tom novom mogućnosti zamijenile prijašnji superioritet razrednih komponenti. Lista prednosti funkcijskih komponenti velika je u odnosu na razredne, sa znatno manje koda može se implementirati ista funkcionalnost. Korištenje funkcijskih komponenti rezultira kvalitetnijim kodom, također njihovo korištenje povećava perfomanse aplikacije.

U sam React ugrađeno je desetak različitih *Hooks*-a, najčešće korišteni su `useState` i `useEffect`.

- **`useState`** - omogućuje manipuliranje stanjima unutar funkcijske komponente.
- **`useEffect`** - uglavnom se koristi pri dohvaćanju podataka sa servera, slanje ajax ili axios *request*, dohvaćanje liste podataka.

Često korišteni Hooks-i iz biblioteke react-router-dom:

- **`useParams`** - omogućuje čitanje podataka iz URL-a. Tako se može prenijeti neki podatak, npr. *Id*, koji će se koristiti u drugoj komponenti. Praktično u slučaju brisanja gdje nema smisla slati cijeli *state* podataka, već je potreban samo id.
- **`useNavigate`** - omogućuje odlaske na prethodne URL poveznice iz povijesti, npr. *navigate(-1)* će otići na prethodno korištenu poveznicu, što je vrlo korisno u raznim situacijama brisanja, ažuriranja podataka nakon kojeg se vraća na prethodnu poveznicu, npr. nakon završetka postavljanja nove šifre. Također omogućuje odlazak na specifičnu URL poveznicu.
- **`useLocation`** - omogućuje slanje *state*-a preko poveznice do neke druge komponente, npr. na početnoj komponenti dohvaća se cijela lista korisnika sa svim njihovim pripadajućim podacima, preko poveznice se može od te liste doći do specifičnog korisnika. Nema smisla ponovno raditi *get request* dohvaćanja podataka koji su već dostpuni na prethodnoj glavnoj

komponenti, pa se stoga cijeli *state* specifičnog korisnika proslijedi na sljedeću komponentu.

Primjer korištenja `useState`-a i `useEffect`-a:

Dohvaćanje liste grupa iz baze, ako je *request* uspješan lista iz *response-a* spremit će se u listu preko *state*-a, ako *response* nije dobar u konzoli će biti ispis o kakvoj se grešci radi.

```
const [group, setListOfGroups] = useState<IState["group"]>([]);
interface IState {
group: {
        group_name: string;
        id : number;
}[]
}
```

Kod 3.1 useState-a i useEffect-a.

```
useEffect(()=>{
    axios.get("/api/group")
    .then((response) =>{
    if(response.data.status === 200){
        console.log(response.data);
        setListOfGroups(response.data.group_list);
        console.log(response.data.message);
    }else{
        console.log(response.data.message);
    }
    })

}, []);
```

Kod 3.2 useEffect-a za grupu.

```
{status ? (
      <>
          <Route path='*' element={<Home />} />
          <Route path='/' element={<Home />} />
          <Route path='/addWorkers' element={<AddWorkers/>} />
          <Route path='/project-settings' element={<ProjectSettings/>}
/>
          <Route path='/groups' element={<Groups/>} />
          <Route path='/groups/:id/:group_name'
element={<GroupOperations/>} />
          <Route path='/workers' element={<Workers/>} />
          <Route path='/workers/worker' element={<Worker/>} />
          <Route path='/create-project' element={<CreateProject/>} />
          <Route path='/user-profile' element={<UserProfile/>} />
          <Route path='/user-profile/change-password'
element={<ChangePassword/>} />
          <Route path='/project' element={<Project/>} />
          <Route path='/project/create-task/:id' element={<AddTask/>} />
          <Route path='/workers/worker/password-reset/:id'
element={<PasswordReset/>} />
          <Route path='/project/task' element={<Task/>} />
      </>
    ) : (
      <>
          <Route path='*' element={<Home />} />
          <Route path='/' element={<Home />} />
          <Route path='/user-profile' element={<UserProfile/>} />
          <Route path='/user-profile/change-password'
element={<ChangePassword/>} />
          <Route path='/project' element={<Project/>} />
          <Route path='/worker-task' element={<WorkerTask/>} />
      </>
  )}
  </Routes>
</Router>
```

Kod 3.3 rute.

```
<div>
    <Link to={`/workers/worker/password-reset/${location.state.id}`}
    className="btn btn-primary fadeIn third">Password reset</Link>
</div>
```

Kod 3.4 React Link.

Na kodu 2.3 vidi se popis svih poveznica koji postoje. U slučaju izrađene aplikacije, postoje dvije skupine korisnika, administrator i običan korisnik, odnosno korisnik radnik. Radnik nema pristup administratorskim poveznicama. Prava na linkove se određuju preko kriptiranog statusa.

Kod 2.4 pokazuje jednu od poveznica koji vodi na specifičnu komponentu. React traži podudarnost url-a s rutom kako bi mogao prikazati specifičnu komponentu, tako u ovom primjeru prikazuje komponentu PasswordReset.

3.3. Validacije i notifikacije

Kako bi korisničko sučelje bilo intuitivno, nužno je dobro organizirati sustav validacija i obavijest. Svaki bi unos trebao imati dinamički sustav obavijesti, bilo to u obliku *alerta*, *popouta* ili jednostavnih ispisanih poruka.

```
const updatePassword = () =>{
      if(password.length<5){
          alert("Password too short");
      }
      if(validatePassword(password, password2) === true){
          axios.put('../api/change-password',{
              old_password: old_password,
              new_password: password2
          })
          .then((response) =>{
            if(response.data.status === 200){
              console.log(response.data.message);
              navigate('/');
            }else{
              setMessage(response.data.message);
            }

          });
      }else{
          setMessage("Password missmatch");
      }
 }
```

Kod 3.5 validacije.

Na kodu 2.5 primjer je validacije lozinke pri registraciji. Prvo postoji provjera za duljinu lozinke, koja zbog sigurnosnih razloga ne bi smjela biti kraća od 5 znakova, u slučaju kraće lozinke iskočiti će prozor s obavijesti. Nakon što je osigurana minimalna duljina lozinke, dalje se korištenjem *TypeScript*-a osigurava da je lozinka tipa *String* i da je ponovljena lozinka jednaka početnoj, čime se osigurava da se željena lozinka pohrani u bazi. U slučaju da se lozinke ne podudaraju, u korisničkom sučelju će se pojaviti poruka koja će to dojaviti. Nakon što se naprave sve nužne provjere, potrebno je na samom serveru odraditi iste provjere. Ako dođe do greške u serveru, korisnik će biti obavješten zašto je greška nastala. Nije idealno da se ikad ispisuju serverske greške, u ovom slučaju će jako teško ikada i doći do serverske greške jer je napravljena rigorozna provjera u samom korisničkom sučelju. U slučaju pokušaja manipulacije korisničkog sučelja i zaobilaženja provjera, napadač neće moći napraviti nikakvu štetu na serveru i bazi već će dobiti prikladnu poruku servera kako nema prava za to ili da serverska validacija nije uspješna.

3.4. REST API

REST (*REpresentational State Transfer*) je web standard za sučelje aplikacijskog programa (API-a) koji koristi HTTP zahtjeve za pristup podacima. Pristup podacima moguć je kroz nekoliko vrsta HTTP zahtjeva, najpoznatije i najkorištenije četiri vrste HTTP zahtjeva su GET, PUT, POST i DELETE koji se odnose na dohvaćanje, ažuriranje, kreiranje i brisanje [5].

API je web kôd koji omogućuje komunikaciju između korisničkog sučelja i servera.

Da bi API bio *RESTful* mora zadovoljavati niz propisanih standarda. API podržava prijenos nekoliko vrsta podataka, odnosno resursa, od kojih su najpoznatiji formati XML i JSON. JSON je široko korišten format zbog svojih prednosti, tako da je postao standardni format resursa za prijenos između korisničkog sučelja i servera. Velika većina tehnologija korisničkog sučelja i servera ima odličnu podršku za JSON format. Resurs u REST-u jako je sličan objektu u objektno orijentiranom programiranju ili entitetu u bazi.

RESTful web servisi koriste HTTP protokole kao posrednike komunikacije između klijenta i

```
{
    "id" :  1,
    "firstName"  :   "Dario",
    "lastName"  :   "Krizanic",
}
```

Kod 3.6 JSON.

servera. Klijent šalje "poruku" u obliku HTTP zahtjeva a server odgovara u obliku HTTP odgovora HTTP zahtjev sastoji se od pet ključnih dijelova.

1) *Verb* koji se odnosi na jedan od HTTP zahtjeva (GET, POST..).
2) URI koji služi za identificiranje resursa na serveru
3) HTTP verzija
4) *Request Header* koji sadrži meta-podatke kao što su vrsta klijentskog preglednika, formati, *cache*, postavke..
5) *Request body* podaci odnosno resursi koje prenosimo od klijenta do servera

HTTP odgovor (*response*)sastoji se od četiri ključna dijela.

1) Status/Response code je odgovor servera u obliku brojčanog statusa
    - 1xx - predstavlja informacijski odgovor
    - 2xx - predstavlja uspješan odgovor
    - 3xx – predstavlja preusmjerenje (redirect)
    - 4xx - predstavlja grešku klijenta
    - 5xx - predstavlja grešku servera
    - Najčešći odgovor kodovi:
        - 200 - uspjeh, OK
        - 201 - uspješno kreirano koristeći POST ili PUT metodu
        - 400 - loš zahtjev koji može nastati zbog loše validacije ili nedostatka poslanih podataka
        - 401 - loša autorizacija
        - 403 - zabrana pristupa, korisnik nema pristup resursima
        - 404 - nije pronađena, resurs nije pronađen
        - 500 - greška servera, izuzetak na serveru

2) HTTP verzija
3) Response Header koji sadrži meta-podatke kao što su vrsta servera, formati, *cache,* postavke, vrijeme odgovora..
4) Response body podaci odnosno vraćeni resursi klijentu ili vraćena poruka servera do klijenta
5) Uniform resource indentifier URI (uniformni resursni identifikator)

Svaki resurs u REST arhitekturi je identificiran preko URI-a. Dvije vrste na koje se URI dijeli su URL i URN [6].

URN ne specificira lokaciju resursa, već ima ulogu identifikatora resursa. Dva objekta na internetu mogu imati identično ime, pa ih se nekako mora razlikovati.

URL specificira lokaciju resursa i mehanizme dohvaćanja [7]
Standardizirani URI mora poštivati sljedeća pravila:

1) korištenje množine za naziv resursa
2) ako resurs ima dugačak naziv praksa je između naziva koristiti znak _ ili -, npr autorizirani_korisnici ili autorizirani-korisnici
3) URI je osjetljiv na velika i mala slova, ali je praksa koristiti mala slova
4) HTTP metode nije poželjno koristiti u nazivu URI-a, npr. za dohvaćanje informacije o korisniku koristi se `/korisnici/{id}` umjesto `dohvatiKorisnika/`
5) Koristi se znak / za hirarhiju npr. `/korisnici/{id}/adresa` gdje je cilj dohvatiti adresu specifičnog korisnika

Prilikom dohvaćanja podataka iz servera također se mora uzeti u obzir količina i veličina podataka. Odgovor (*response*) ne bi trebao vraćati klijentu cijelu veliku listu podatak. Nužno je ograničiti dohvaćanje podataka uvođenjem filtriranja i paginacije kako bi se smanjilo opterećenje na serveru i znatno ubrzao spomenuti proces.

Sigurnost je uvijek prioritet svake aplikacije. Nužno je osigurati API pristup i ograničiti njegov pristup različitim korisnicima. Običan korisnik nikad ne bi trebao imati pristup procesima koji bi trebali biti dopušteni samo administratorskoj strani. API rute za prijavu i registraciju dostupne su uvijek bez ikakvih ograničenja. Sve ostale rute koriste sanctum posrednik (*middleware*), te je takvim rutama moguć pristup samo ako se uspješno prijavi u aplikaciju. Rute koje bi trebale biti dozvoljene samo administratoru zaštićene su *middleware*-om za status. On uzima trenutno prijavljenog korisnika i provjerava njegov status u bazi, ako mu status odgovara statusu administratora, API *request* će se uspješno izvršiti.

```
Route::post('/createUser', [UserController::class, 'store']);
Route::post('/loginUser', [UserController::class, 'login']);

Route::group(['middleware' => ['auth:sanctum']], function () {
    Route::get('/projects', [ProjectController::class, 'projectList']);
    Route::put('/projects', [ProjectController::class, 'updateProject'])-
>middleware("status");
    Route::delete('/projects/{id}', [ProjectController::class,
'deleteProject'])->middleware("status");
    Route::delete('/task/{id}', [ProjectController::class, 'deleteTask'])-
>middleware("status");
    Route::post('/projects', [ProjectController::class, 'store'])-
>middleware("status");
    Route::post('/task', [ProjectController::class, 'storeTask'])-
>middleware("status");
    Route::put('/task', [ProjectController::class, 'updateTask'])-
>middleware("status");
    Route::get('/tasks/{id}', [ProjectController::class, 'taskList'])-
>middleware("status");
    Route::get('/workers/{id}', [ProjectController::class, 'taskWorkers'])-
>middleware("status");
    Route::put('/status', [ProjectController::class, 'taskUndoneStatus'])-
>middleware("status");
    Route::get('/tasks', [ProjectController::class, 'taskListForWorker']);
    Route::put('/task-done-status', [ProjectController::class,
'taskDoneStatus']);
```

Kod 3.7 API rute.

```
public function handle(Request $request, Closure $next)
  {
      if($request->user()->status === 1){
          return $next($request);
      }else{
          return response()->json([
              'message' => '403'
          ]);
      }
  }
```

Kod 3.8 status middleware.

4. Integracija klijentskog razvojnog kostura u Laravel mrežnu aplikaciju

4.1. Laravel kao poslužiteljska tehnologija

Laravel je robustan, brz, siguran, jednostavan i lako razumljiv PHP *framework* koji prati MVC uzorak arhitekture [3]. Laravel iskorištava postojeće komponente drugih *framework*-a. Ima ugrađen velik broj funkcionalnosti koji značajno skraćuju vrijeme potrebno za izradu aplikacije. Uz Symfony, Laravel je najčešće korišten PHP *framework*.

Koristeći Laravel, vrlo je jednostavno napraviti sigurnu aplikaciju, čak i bez upotrebe trećih biblioteka. Laravel Sanctum omogućuje autentifikaciju SPA, mobilnih aplikacija i token baziranih API-a.

Sanctum je jednostavan paket za autentifikaciju API tokena, njegovo kreiranje i manipuliranje bez potrebe za korištenjem OAuth-a. Prilikom prijave u aplikaciju, generira se token koji se šalje u *Authorization header* kao *Bearer* token u pregledniku, zatim se token kriptira SHA-256 *hash*-om, te se kao takav pohranjuje u bazu. Prilikom svakog API zahtjeva (request) koji ima Santcum *middleware* uspoređuje se korisnikov kriptirani token iz baze sa tokenom iz preglednika.

4.2. Struktura aplikacije

Uz Laravel vrlo je jednostavno krenuti s izradom aplikacije jer se pomoću jedne naredbe kreira cijela struktura datoteka i njegovih pod-datoteka koje su nužne za izradu aplikacije.

App datoteka sadrži sve potrebno za izradu serverske strane aplikacije, kao što su: konzola, izuzeci, modeli, kontroleri u kojima se nalazi sva serverska logika, *middleware* kojeg se poziva u API-u, a u kojem se nalazi logika koju se može izvršiti prije izvršavanja glavne logike iz kontrolera [4].

Baza datoteka sadrži *seeds* datoteku namijenjenu za *unit* testove, *factories* gdje se može generirati velik broj podataka za bazu koja će biti potrebna za testiranje aplikacije, *migrations* datoteku gdje se nalaze sve migracije kreirane prema modelima preko kojih se generira kompletna baza sa svim potrebnim poljima i atributima.

Resources datoteka sadrži sve vezano za korisničko sučelje i API putanje. JavaScript datoteka sadrži sve React komponente, CCS datoteke, preglede(views) gdje je u slučaju SAP-a (*single-*

page-application) potrebna samo jedna datoteka gdje će se prikazivati sav JavaScript kod, odnosno React komponente.

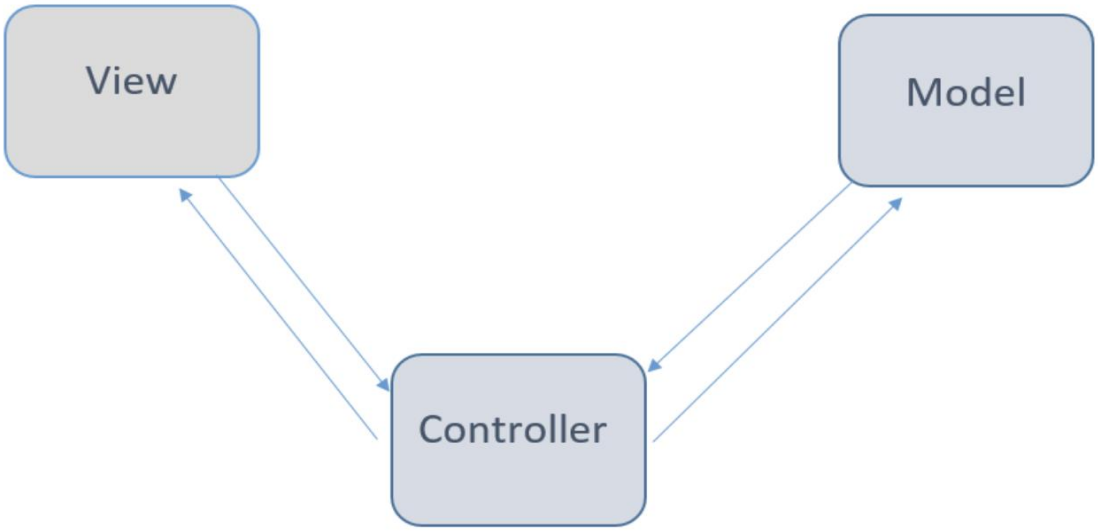
Storage datoteka sadrži logove, *cache*-eve i druge podatke koji se često koriste.

Vendor datoteka sadrži sve potrebno za uspješno funkcioniranje servera i u njemu se nalaze sve biblioteke navedene u composer.json odakle se preko terminala i composer paket managera može mijenjati verzije biblioteka, brisati, ažurirati ih ili dodavati druge.

Kao što je u Vendoru smješteno sve potebno za uspješno funkcioniranja serverske strane tako se u Node modulu nalazi sve potrebno za uspješno funkcioniranje klijentske strane aplikacije, zapisi o korištenim bibliotekama nalaze se u package.json datoteci kojom se manipulira pomoću NPM ili Yarn upraviteljem paketa.

4.3. MVC

Trygve Reanskaug osmislio je MVC arhitekturu 1978. Cilj je bio riješiti problem kompleksnih i velikih projekata koji do tad nisu bili kvalitetno strukturirani. Kako su korisnički zahtjevi rasli, tako se i složenost projekata i aplikacija znatno povećala. Kvalitetna standardizirana arhitektura je stoga postala nužna. Danas su sve srednje do visoko kompleksne aplikacije napravljene po arhitekturi MVC-a. MVC je dobro rješenje za velik broj aplikacija, dok je one najkompleksnije ipak bolje implementirati koristeći DDD odnosno Domain Driven Design (domenski vođen dizajn) arhitekturu.



Slika 4.1 MVC arhitektura [2]

MVC (*model*, *view*, *controller*) arhitekturni je obrazac koji se sastoji od 3 komponente: modela, pogleda i upravitelja.

Model je najniža razina koja je odgovorna za manipuliranje podacima. Preko modela se kreiraju i ažuriraju podatci koji se spremaju u bazu, te se također i dohvaćaju. Upravljač svojom logikom implementira CRUD zahtjeve, odnosno zahtjeve za kreiranje, čitanje (dohvaćanje podataka), ažuriranje i brisanje. U pogledu odnosno korisničkom sučelju prezentiraju se dohvaćeni podatci koje je upravljač proslijedio u odgovor *(response)*.

4.4. Alati za upravljanje paketima

Paket upravitelji znatno ubrzavaju svakodnevno programiranje, za puno toga što je potrebno implementirati već postoji gotova implementacija. Ponekad gotovi dijelovi u potpunosti odgovaraju potrebama programera, ali češće je slučaj da gotovi dijelovi samo djelomično odgovaraju traženome, u oba slučaja znatno štedimo na vremenu izrade aplikacije

NPM paket menadžer korišten je u izradi front JavaScript djela aplikacije, dok je composer korišten kao serverski paket menadžer PHP-a.

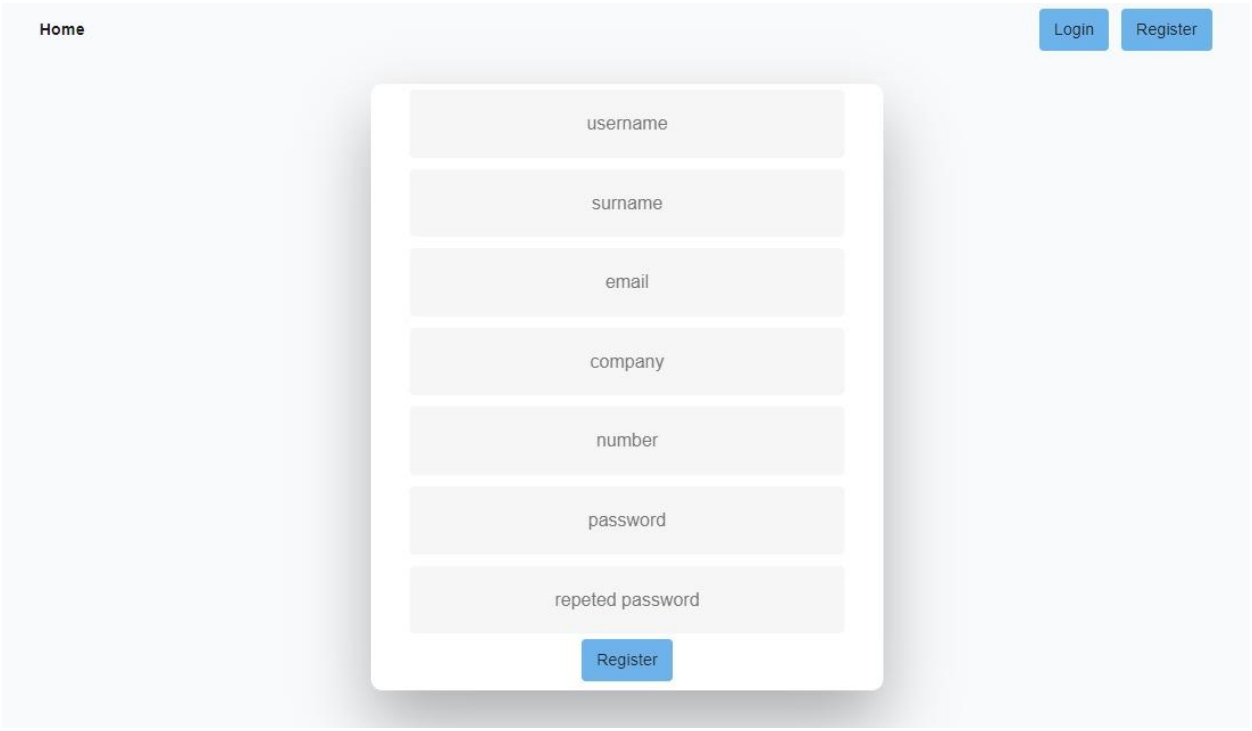
Neke od naredba:

- `Npm run watch` - gleda promjene u kodu, u slučaju promjene u kodu korisničkog sučelja ponovno pokreće server
- `Npm install <package>` - instaliranja specifičnog paketa
- `Npm uninstall <package>` - brisanje paketa
- `Npm update` - ažuriranje svih paketa iz package.json datoteke
- `Php composer install <package>` - instaliranja specifičnog paketa
- `Php composer update <package>` - ažuriranje specifičnog paketa

4.5. Opis aplikacije

Za potrebe demonstracije izrađena je aplikacija za organizaciju poslova. Aplikacija je razdvojena na dva djela, stranu poslodavca i stranu radnika. Velika većina logike odvija se na administratorskoj strani koja je zadužena za kreiranje projekata, kreiranje i dodjeljivanje poslova, vođenje kompletne evidencije radnika. Na radničkoj strani aplikacije nalazi se lista svih poslova koje trebamo obaviti u definiranom vremenu. Radnik može mijenjati statuse zadataka pomoću kojih poslodavac može pratiti progres.

Prvo što se može vidjeti kada se otvori aplikacija je početna registracija administratora određene firme.

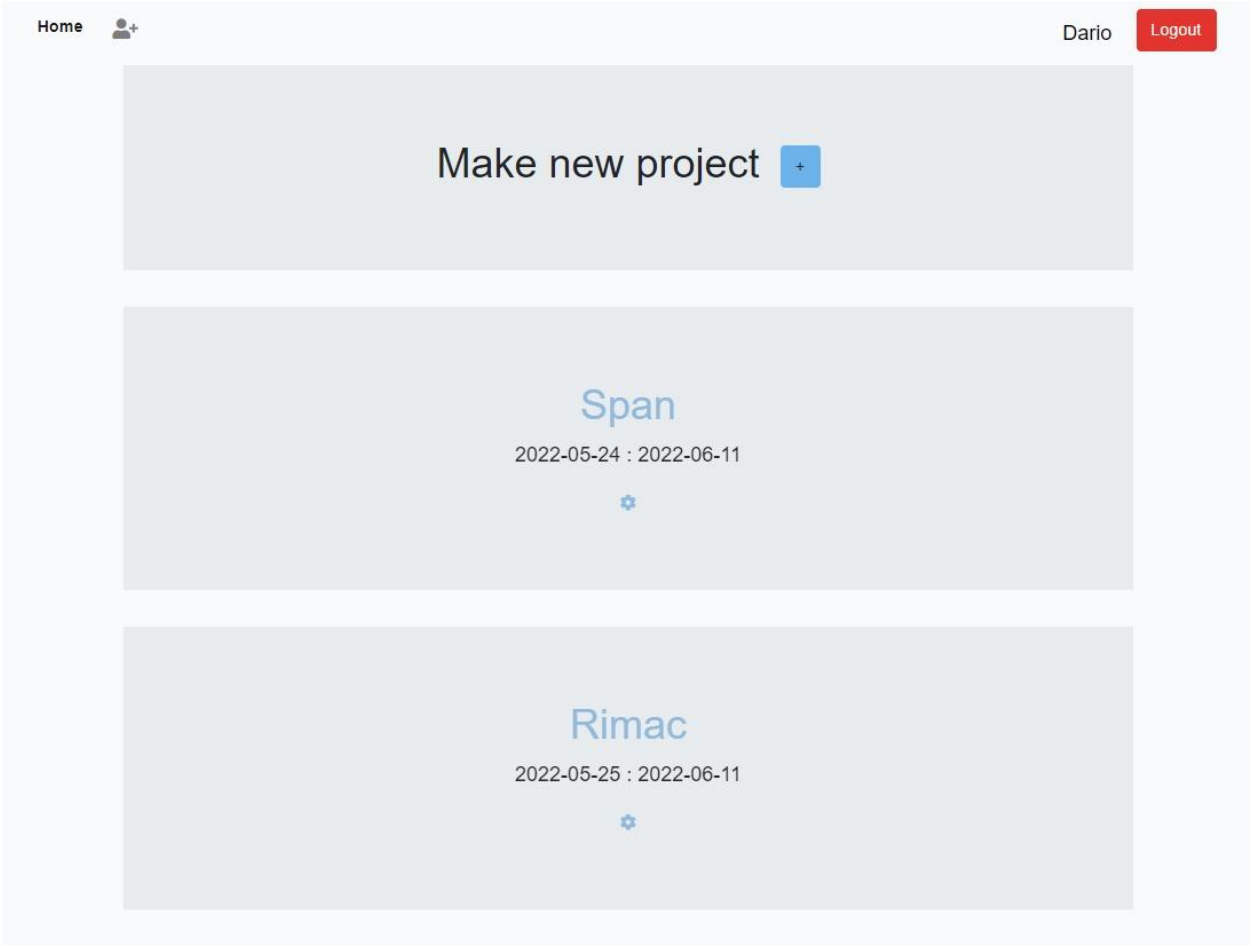


Slika 4.2 Registracija

Email, ime firme i broj su jedinstveni za svakog korisnika, te u slučaju da se pokuša kreirati novi korisnik koji je već unesen u sustav, u formi će se izbaciti prikladna poruka koja će precizno ukazati gdje je nastao problem. Registracijska forma uvijek je prikazana kao početna stranica jer

korisnik neće imati potrebu koristiti Login formu osim ako se odjavi iz sustava. Nakon uspješne registracije korisnik će odmah biti prijavljen u sustav i neće morati odraditi prijavu.

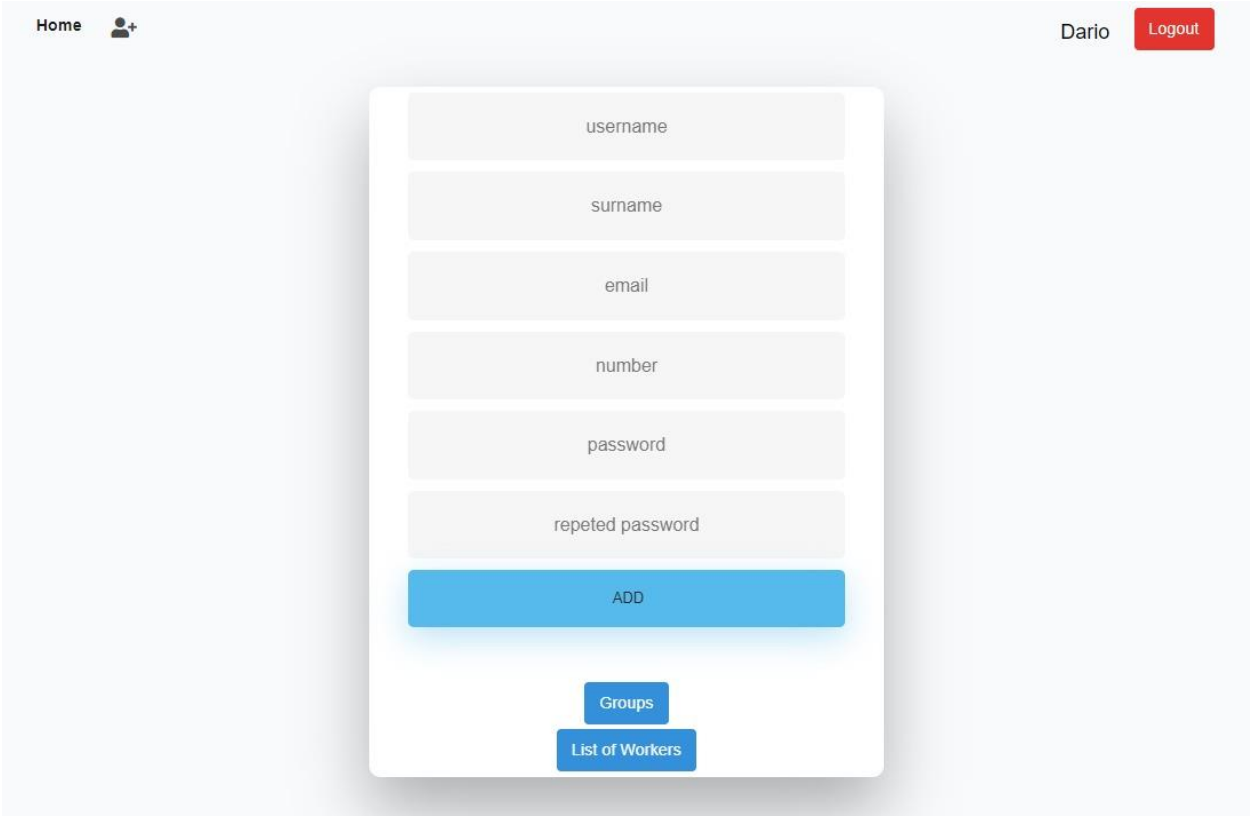
Zaposlenici neće nikada imati potrebu registracije jer njih poslodavac unosi u sustav, te će oni morati odraditi samo inicijalnu prijavu u sustav s odgovarajućim podacima.



Slika 4.3 Projekti

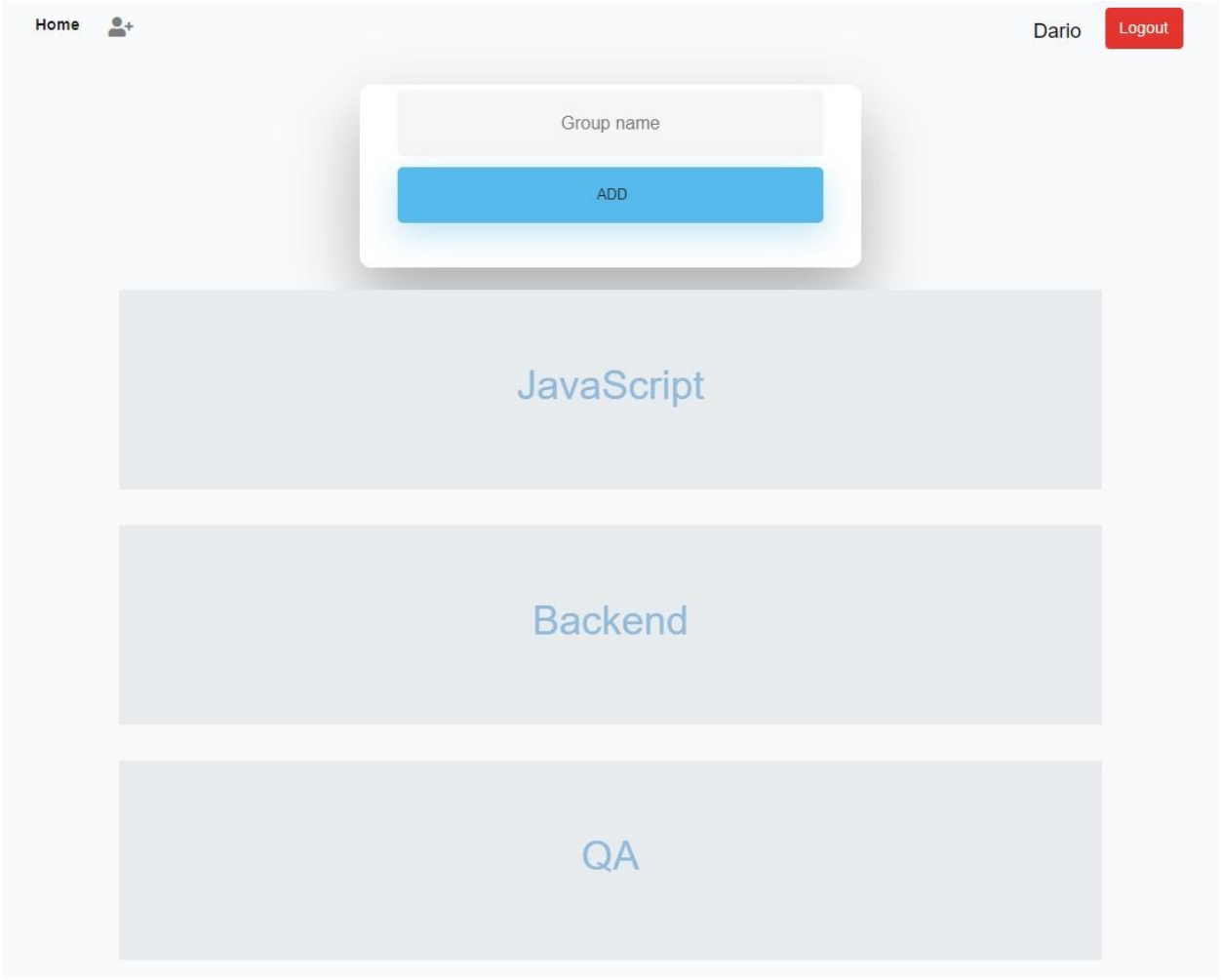
Nakon uspješne registracije ili prijave u sustav, na prvoj stranici aplikacije kreiraju se novi projekti i dostupan je uvid u sve projekte. U listi projekata postoji gumb „opcije“ gdje se mogu uređivati informacije o projektu i mogu se dodati nove grupe radnika u projekt. Kada se otvori projekt, prikazuje se stranica gdje se može dodavati, uređivati, brisati i mijenjati statuse zadataka, te dodjeljivati poslove zaposlenicima iz grupa dodanih na projekt. U gornjem lijevom kutu nalazi se ikona osobe koji nas vodi na stranicu dodavanja novih radnika u sustav. U gornjem desnom kutu nalazi se ime prijavljenog korisnika koje vodi na stranicu s osobnim podacima gdje se sve informacije mogu ažurirati. Osim ažuriranja informacija, također se može promijeniti lozinka i

postoji mogućnost brisanja cijele firme iz sustava gdje će se svi podaci koji imaju ikakvu poveznicu s firmom trajno obrisati. Rijetko se dešava scenarij brisanja kompletnog sustava, pa je dodana sigurnosna potvrda koja će osigurati da se cijeli sustav ne obriše slučajno.



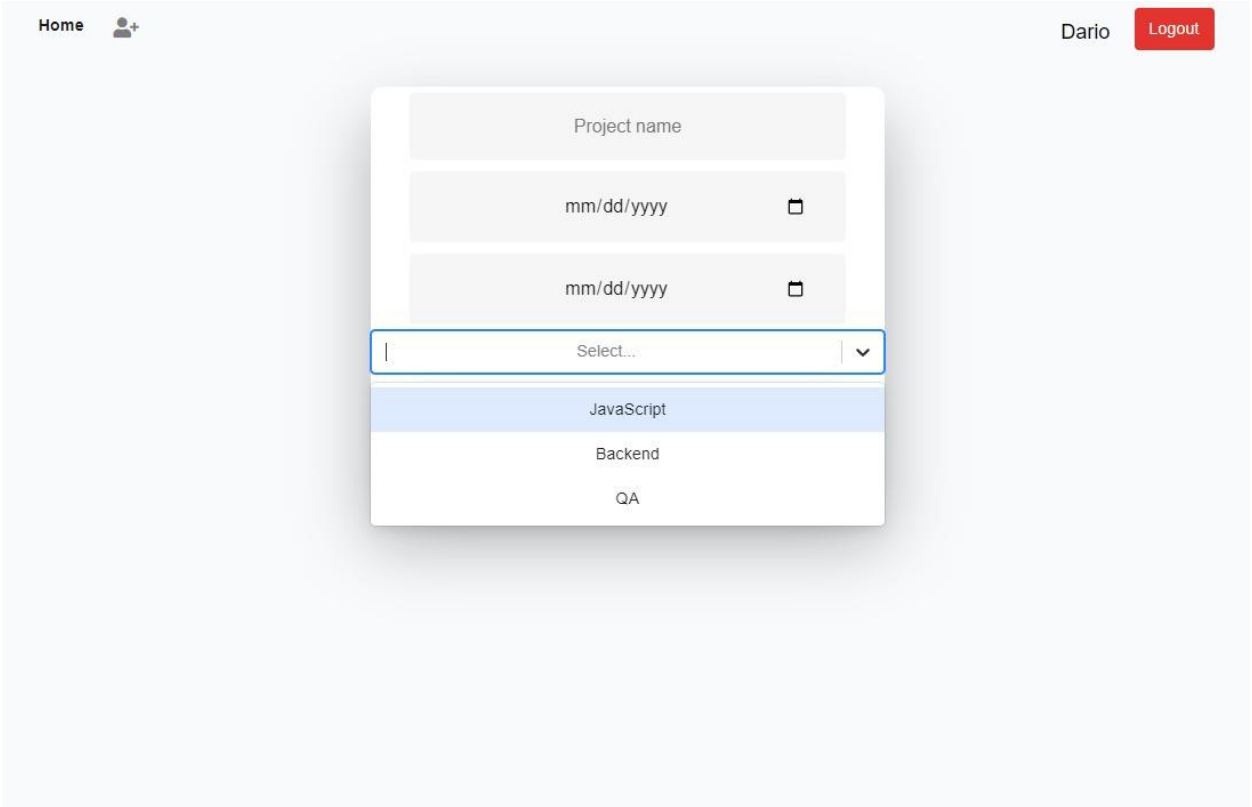
Slika 4.4 Dodavanje zaposlenika

Zaposlenike u sustav može dodati isključivo administrator. Zaposlenik se prijavljuje koristeći podatke predane od strane poslodavca. Zaposlenik nakon inicijalne prijave u sustav mijenja lozinku u svoju privatnu. U slučaju zaposlenikove zaboravljene lozinke, administrator ima mogućnost resetiranja lozinke. Osim dodavanja zaposlenika u sustav, može se otići na popis svih radnika ili popis svih grupa gdje se mogu kreirati nove grupe i gdje se postojeći zaposlenici mogu dodavati u grupe.



Slika 4.5 Dodavanje grupa

Nakon kreiranja grupe, svakoj grupi mogu se preko padajućeg izbornika dodati dostupni zaposlenici.



Slika 4.6 Kreiranje projekata

U kreiranju projekta, osim dodavanja imena i planiranog datuma izrade, postoji mogućnost dodavanja prethodno kreiranih grupa koje će biti uključene u izradu projekta. Iz tih grupa moći će se dodavati zadatci radnicima.

5. ZAKLJUČAK

Integracijom *frameworka* korisničkog sučelja u Laravel riješen je problem primitivnog učitavanja zasebnih stranica. Takve web aplikacije moderne su, visokih perfomansi i mogu se pohvaliti s činjenicom da su izuzetno interaktivne. Kombinacija Laravela i React-a osigurava pisanje dugotrajnog stabilnog koda kojeg ne bi trebalo biti problematično održavati i ažurirati. Pisanje testova jednostavno je i brzo kako u Laravelu tako i u React-u.
Obje tehnologije imaju velik broj korisnika, stoga je potpora zajednice odlična. Njihovi paket menadžeri znatno olakšavaju i ubrzavaju proces izrade aplikacije.

6. LITERATURA

[1] K. G. Labs, “Coding interview questions,” *InterviewBit*. [Online]. Available: https://www.interviewbit.com/. [Accessed: 14-Jul-2022].

[2] “Everything you need to know about MVC architecture.” [Online]. Available: https://towardsdatascience.com/everything-you-need-to-know-about-mvc-architecture-3c827930b4c1. [Accessed: 14-Jul-2022].

[3] “The PHP framework for web artisans,” *Laravel*. [Online]. Available: https://laravel.com/. [Accessed: 14-Jul-2022].

[4] “Laravel - application structure,” *Tutorials Point*. [Online]. Available: https://www.tutorialspoint.com/laravel/laravel_application_structure.htm. [Accessed: 14-Jul-2022].

[5] A. S. Gillis, “What is Rest Api (restful API)?,” *SearchAppArchitecture*, 22-Sep-2020. [Online]. Available: https://www.techtarget.com/searchapparchitecture/definition/RESTful-API. [Accessed: 14-Jul-2022].

[6] “URL and URN - understanding the key difference,” *Software Testing Help*, 25-Oct-2022. [Online]. Available: https://www.softwaretestinghelp.com/url-and-urn/. [Accessed: 14-Jul-2022].

[7] “RESTful web services - addressing,” *Tutorials Point*. [Online]. Available: https://www.tutorialspoint.com/restful/restful_addressing.htm. [Accessed: 14-Jul-2022].

Sažetak

U završnom radu opisan je način rada aplikacije od unosa podataka u korisničkom sučelju preko API-a sve do servera koji je zaslužan za obradu dobivenih podataka te njihovo spremanje u bazu podataka. Opisana je arhitektura aplikacije, REST arhitektura korisničkog sučelja te MVC arhitektura servera. Velik fokus završnog rada je sama aplikacija koja je izrađena u svrhu demonstracije izrade *fullstack* aplikacije koristeći Laravel i React. Ključne riječi: aplikacija, MVC, REST, PHP, Laravel, React.

Abstract

The final thesis describes how the application works, starting with data entry in the user interface through the API all the way to the server responsible for processing the received dana and theirs save to database. The architecture of the application, REST architecture of the user interfaces and MVC server architecture is described. A big focus of the final thesis is the application itself, which is created for the purpose of demonstrating the creation of a fullstack application using Laravel and React.

Key words: aplikacija, MVC, REST, PHP, Laravel, React