Izrada web servisa za upravljanje timovima u uredskom poslovanju informatičara

Valek, Alen

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Alen Valek

Izrada web servisa za upravljanje timovima u uredskom poslovanju informatičara

Završni rad

Pula, kolovoz, 2022.

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Alen Valek

Izrada web servisa za upravljanje timovima u uredskom poslovanju informatičara

Završni rad

JMBAG: 0303092444, redoviti student

Studijski smjer: Informatika

Kolegij: Informatizacija uredskog poslovanja

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: doc. dr. sc. Snježana Babić

Pula, kolovoz, 2022.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani ALEN VALEK, kandidat za prvostupnika INFORMATIKE ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljeni način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

Alen Valek

U Puli, 12.09.2022.

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, ALEN VALEK dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj Završni rad pod nazivom IZRADA WEB SERVISA ZA UPRAVLJANJE TIMOVIMA U UREDSKOM POSLOVANJU INFORMATIČARA koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 12.09.2022.

Potpis

Alen Valek

Sadržaj

1. Uvod

Prilikom razvoja programskog proizvoda koji bi služio svrsi upravljanja timova potrebno je bilo istražiti hijerarhiju samih odnosa unutar većine informatičkih tvrtki. Hijerahijski model korišten za samu implementaciju je dosta pojednostavljen kako bi pokrio što generalniju namjenu te bio spreman za primjenu u što više informatičkih ureda bez obzira na to kako je sama tvrtka koja koristi programski proizvod podijeljena. Programski proizvod namijenjen upravljanju timovima mogao bi biti iznimno kompliciran te povezan s tisuću malih mikroservisa, a ujedno mogao bi biti iznimno jednostavan i još uvijek riješiti problem upravljanja timovima u informatičkim uredima. Bitna stvar za projekte ovakve prirode je mogućnost skalabilnosti samog programskog proizvoda i mogućnost prilagodbe iste za funkcionalnost u više različitih tipova ureda. Tijekom pandemijske krize 2019. godine znatno se povećao broj ljudi koji su obavljali svoje poslove iz udobnosti svojeg doma te nisu imali mogućnost direktne odnosno fizičke kolaboracije sa svojim kolegama. To je rezultiralo velikom potražnjom programskih rješenja koje nude mogućnosti kolaboracije i upravljanja drugim različitim aspektima informatičkog poslovanja. Samim time, nakon povratka svega u normalu, trend rada od kuće se nastavio te i dalje postoji potreba za različitim alatima za kolaborativan i timski rad. Jedan od boljih načina implementacije takvih servisa i aplikacija je implementacija web aplikacije. Razlog zašto su web aplikacije jedne od najboljih načina za implementacije takvih servisa je jednostavnost korištenja i dostupnost. Svi koji imaju pristup internetu mogu koristiti web aplikaciju bez obzira na opremu kojom to odluče te im se iskustvo ne bi trebalo znatno razlikovati od ostalih korisnika. Korištenje takvih aplikacija na mobilnim uređajima poput mobitela i tableta je također jedna velika prednost. Da bi se takva aplikacija koristila nije potrebno nikakvo preuzimanje, već je dovoljno znati web adresu. Sama web aplikacija nije dovoljna za gradnju jedne dinamičke stranice. Web aplikacija predstavlja klijenta u vezi klijenta i poslužitelja odnosno servera. Na serveru postoje krajnje točke koje omogućavaju klijentu da kreira, ažurira, briše i čita podatke prema potrebi (engl. *CRUD*). Sve to je bila neka vrsta motivacije koja je utjecala na temu ovog rada, a to je izrada web servisa i web aplikacije koja će se služiti tim servisom kako bi se olakšalo upravljanje timovima u poslovanju informatičara u uredima.

1.1. Cilj projekta

Cilj projekta je omogućiti korisnicima upravljanje timovima, pregleda napretka timova po projektima te vizualizacija istih. Uz svrhu upravljanja timovima također je cilj predočiti alate za praćenje aktivnosti korisnika te dinamičke pozicije i uloge unutar sustava.

1.2. Konkurencija

Istraživanjem lokalnog tržišta najsličnija aplikacija predstavljenoj u ovom projektu je aplikacija *Osana,* koju razvija tvrtka *Tilio*. Njihova aplikacija se uglavnom bazira na upravljanju projektima i koordiniranju sastanaka kroz jednostavno sučelje. Ta aplikacija je nažalost limitirana na 15 ljudi po timu za besplatne korisnike. Na globalnoj razini jedna od najpopularnijih solucija za upravljanje timovima i projektima je *Jira*. To je aplikacija široke primjene i nude se besplatne verzije. Jira nudi mogućnost upravljanja timovima, postavljanja sudionika u projekte te praćenje istih. Jedna od ključnih značajki te aplikacije je mogućnost dodavanja dodataka trećeg lica. To znači da postoje pojedinci van tvrtke koji rade na poboljšavanju korisničkog iskustva pa se oko same aplikacije stvorio jedan mali eko sustav mini aplikacija i dodataka. Nažalost besplatna verzija Jira solucije također predstavlja svoje limitacije i nudi mogućnost dodavanje samo 10 korisnika u besplatnom planu. Prema posljednjem izvješću njihovu soluciju koristi preko 65,000 poduzeća, a rade na aplikaciji od 2002. godine. Navedene i opisane su dvije popularne solucije najsličnije aplikaciji koja je tema ovog projekta, no u suštini postoji puno solucija za upravljanje timova, ali svaka nosi svoje unikatne značajke. Shodno tome potrebno je spomenuti *Microsoft Teams*, popularan alat za kolaboraciju timova, *Slack* koji služi za komunikaciju i organizaciju timova, *Flock*, kanal za komunikaciju i kolaboraciju timova, *Monday.com* koji nudi mogućnosti upravljanja timskih projekata i mogućnost automatizacije te brojni drugi alati. Za potrebe ovog rada spomenuti su samo najrelevantiji alati. Prema navedenim aplikacijama jasno je da se svaka aplikacija fokusira na pojedinu svrhu, dok ovaj projekt pokušava integrirati što više značajki potrebno za rad informatičkog ureda u isti sustav.

1.3. Problem koji se rješava

Očito je da postoje već solucije koje rješavaju problem upravljanja timovima no većina tih solucija je skupo te stavljaju velike limite na korisnike koji koriste njihov programski proizvod besplatno. Aplikacija u sklopu ovog projekta rješava takve probleme jednostavnim optimiziranjem podataka kako bi se mogla nuditi besplatno uz što manje limitacije. Svrha je spojiti upravljanje timovima i projektima na kojima ti timovi rade, upravljanje uredom i upravljanje generalnim potrebama ljudskih resursa.

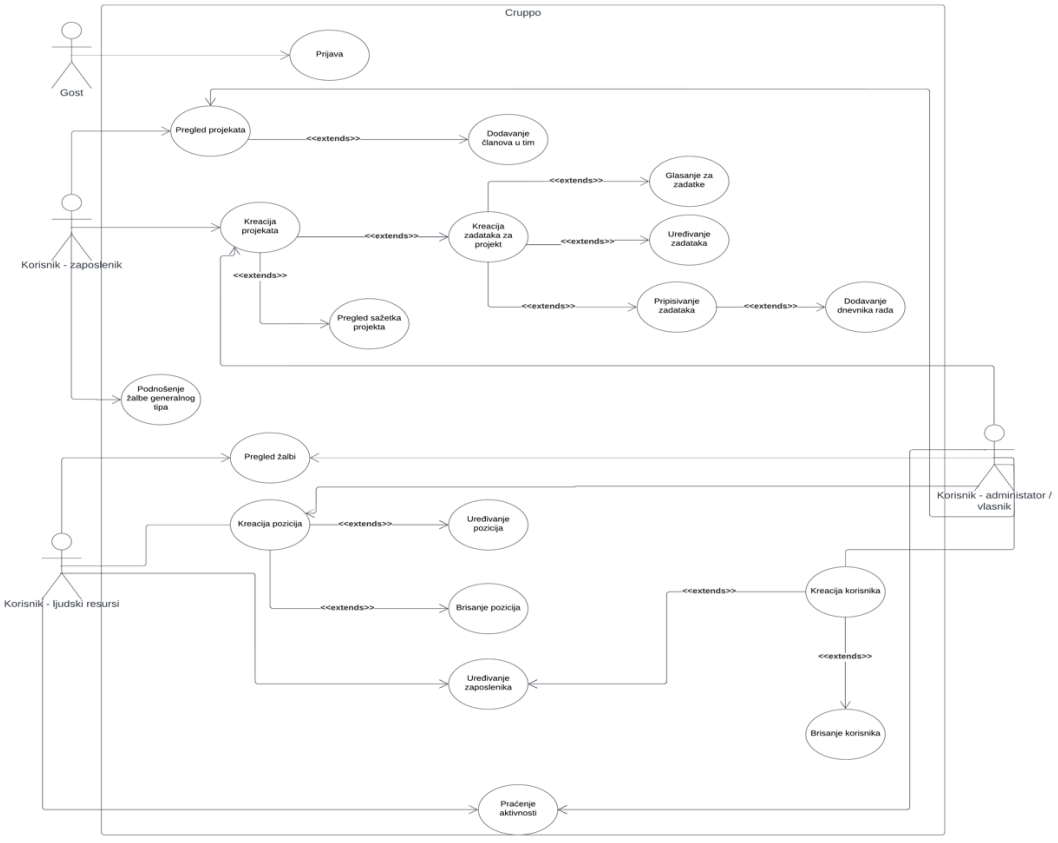
2. SWOT analiza

Kao što je prikazano na slici 1, snage ove solucije su mogućnost širenja na globalno tržište s obzirom na to da je samo sučelje na engleskom jeziku. Aplikacija je također zamišljena za rad s velikim količinama podataka te je vrlo dinamična i lako prilagodljiva raznim situacijama. Taj model otvara vrata mogućnosti monetizacije različitih planova i različitih pogodnosti te se ujedno stvara prilika za proširivanje već gotovih i dodavanje potpuno novih funkcionalnosti. Što tiče slabosti, aplikacija pati od nedostatka kolaborativnih alata te mogućnosti direktne komunikacije između korisnika unutar istog sustava. Od prijetnji tu je standardna prijetnja odanosti korisnika ustanovljenim brendovima, mali broj korisnika te nepredviđeni troškovi za brz rast.

Slika 1. SWOT Analiza

3. Dijagram obrasca uporabe

Na slici 2 prikazan je dijagram obrasca uporabe koji prikazuje kakve mogućnosti korisnici imaju i kako se služe sustavom. Prvo je korisnik tipa gost kojem je jedina opcija prijaviti se, ako uopće ima račun. Nakon gosta slijedi korisnik koji ima ovlasti običnog zaposlenika, on ima pravo pregledavati projekte, kreirati ih u skladu sa svojim ovlastima te to naravno uključuje i dodavanje članova u tim, pregled sažetka projekta, kreaciju zadataka koji sa sobom također nose aktivnosti poput uređivanja zadatka, glasanje za zadatak te pripisivanje zadatka. Korisnici koji si pripišu zadatak također imaju opciju dodavanja radnih dnevnika unutar zadatka. Taj tip korisnika također može pisati žalbe. Korisnici koji su dio ljudskih resursa imaju ovlasti pregleda žalbi te kreacije pozicija. Kreacija pozicija sa sobom nosi i mogućnost uređivanja pozicija i brisanje pozicija. Korisnici tog tipa također mogu uređivati sve informacije zaposlenika i nadzirati aktivnosti koje se dešavaju unutar sustava. Administrator i vlasnik mogu sve što i oba tipa korisnika skupa te dodatno mogu kreirati korisnike te samim time ih brisati.



Slika 2. Dijagram obrasca upotrebe

4. Korištene tehnologije

4.1. NodeJS

Node.js je asinkrono programsko okruženje bazirano na događajima koje omogućava izvršavanje JavaScript koda van preglednika te je dizajniran za izgradnju skalabilnih mrežnih aplikacija [1]. Prema službenoj stranici pri svakom povezivanju klijenta i Node.js programa poziva se funkcija koja je u tom vremenu potrebna te se pokreće, dok Node.js miruje ako nema nikakvih poziva (Node.js, n.d.). Node.js također nije baziran na principu više dretvi pa nije potrebno baviti se upravljanjem dretvi pri kreaciji programa. Jedna od bitnijih značajki Node.js-a koja je utjecala na odabir ove tehnologije je mogućnost spajanja velikog broja klijenata istodobno bez nekih blokada ili zastoja bez potrebe instanciranja novih dretvi.

4.2. ExpressJS

Express.js je minimalistička i fleksibilna biblioteka bazirana na Node.js-u koja služi za izgradnju web aplikacija i API-a (Express.js, n.d). Shodno tome, odlična je podloga za izgradnju REST API-a. Bitna je činjenica da postoje mnogi gotovi paketi i mogućnost kreacije spomenutih paketa koji olakšavaju rad s različitim dijelovima izrade web servisa.

4.3. MongoDB

MongoDB je nerelacijska odnosno baza bazirana na kolekcijama te dokumentima koji su dio tih kolekcija. Napušta sve principe pravilne forme odnosno normalizacije podataka koja vrijedi za relacijske ekvivalente. MongoDB podržava te ustvari potiče duplikaciju podataka odnosno denormalizaciju. Način na koji se modeliraju baze poput MongoDB-a je prilagodba svih podataka koji su potrebni pri pozivu. To se može postići referenciranjem dokumenta iz druge kolekcije preko njegovog objektnog id-a ili jednostavno direktnim gniježdenjem podataka u isti dokument. MongoDB je izuzetno brz s obzirom na to da je građen na principu ključ-vrijednost te tim principom hvata podatke iz baze što rezultira izuzetno brzim čitanjem podataka.

4.4. Rest API

Aplikacijsko programsko sučelje (engl. *Application programming interface*) je način na koji su različiti oblici programskih proizvoda i biblioteka dani programerima na korištenje (Myers, B. A., & Stylos, J., 2016). Postoji nekoliko vrsti korisnika API-a s različitim ciljevima (Myers, B. A., & Stylos, J., 2016). Prva grupa su dizajneri API-a, čiji je cilj dizajnirati i postaviti API za javno ili privatno korištenje od strane programera koji će taj API pretvoriti u krajnji prozivod kojeg će koristiti posljednja grupa, a to su potrošači (Myers, B. A., & Stylos, J., 2016).

Prema Roy Fielding-u, izumitelju REST-a, API koji implementira RESTful arhitekturni stil dizajna strukture i ponašanja trebao bi biti bez stanja tako da bilo kakva promjena u implementaciji ne šteti programeru koji koristi API (Myers, B. A., & Stylos, J.,2016b). API koji je RESTful sadrži točna pravila kako će biti korišten. Programer treba samo znati URL, odnosno mjesto gdje poslati i operaciju koja će se vršiti nad dokumentom. Operacije se šalju pomoću HTTP glagola. HTTP glagoli koji se koriste za operacije kreacije, čitanja, ažuriranja i brisanja su: get(čitanje), post(kreiranje), put/patch(ažuriranje) i delete(brisanje). Iako postoji dogovoren set pravila za kreaciju REST API-a, još uvijek nije u potpunosti standardizirano pa postoje neke nesuglasice oko naziva i URL-a koji bi se trebali koristiti.

4.5. ReactJS

React.js je JavaScript biblioteka za izgradnju korisničkih sučelja. Omogućava bržu i kvalitetniju gradnju interaktivnih sučelja. React kao temelj gradnje sučelja koristi takozvane komponente koje su samo elementi stranice koji bi se mogli ponovno koristiti u drugim projektima ili je njihovo korištenje jednostavno vrlo redundantno u trenutnom projektu. Što dodatno poboljšava lakoću implementacije je JSX, što je proširenje sintakse običnog JavaScript-a. JSX daje mogućnost korištenja JavaScript koda i HTML koda u istom dokumentu. Iako .jsx ekstenzija datoteke postoji, nije potrebna za korištenje JSX sintakse unutar React projekta. Većina projekata koristi običnu .js ekstenziju te popunjavaju sintaksu raznim dodacima njihovog preferiranog okruženja za razvoj koda.

4.6. Redux

Redux je JavaScript biblioteka za upravljanje i centraliziranje stanja aplikacije (Redux, n.d.). Redux je odličan dodatak React.js-u za spremanje stanja kojem je potrebno pristupiti iz više komponenti. Implementacija kompleksnih aplikacija bez Redux-a je u potpunosti moguća, no moć i probleme koje rješava je teško zanemariti. Redux koristi akcije koje ovisno o namjeni čitaju i ažuriraju reduktore.

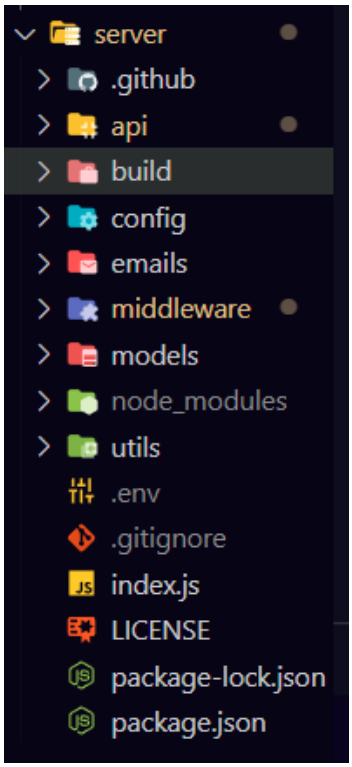
4.7. MUI

MUI je niz preddefiniranih komponenti i drugih elemenata za korisnička sučelja (MUI, n.d.). Pri gradnji svojih komponenti MUI se jako fokusira na principe kvalitetnog dizajna. Pomaže pri brzom isporučivanju novih funkcionalnosti i gradnji modernih korisničkih sučelja. Komponente koje se nude, iako su preddefinirane sve se mogu prilagoditi potrebi i situaciji u kojoj se koriste.

5. Implementacija web servisa

5.1. Struktura datoteka web servisa

Kako bi se što lakše baratalo i snalazilo podacima koji se nalaze na serveru prate se pravila dobre arhitekture. Svaka datoteka i svaka mapa imaju svoju namjenu te je to izraženo deskriptivnim imenovanjem svake mape i svake datoteke. Mapa „api“ sadrži sve krajnje točke te je svaka datoteka imenovana prema svrsi za koju je namijenjena. U „build“ mapi se nalaze CSS datoteke koje su namijenjene za mail šablone koje se šalju korisnicima. „Config“ mapa sadrži postavke za povezivanje baze podataka sa serverom. „Emails“ mapa sadrži sve mail šablone koje se šalju korisnicima. „Middleware“ mapa je namijenjena za sve dodatne provjere koje se trebaju obaviti prije pozivanja funkcionalnosti krajnje točke kojoj se želi pristupiti. „Models“ sadrži sve sheme MongoDB-a koje se koriste. „Utils“ mapa se koristi za sve funkcionalnosti čiji je kod dosta širok, ali koristan pri obavljanju specifičnih zadataka unutar servisa. „index.js“ datoteka je zaslužna za spajanje svih funkcionalnosti ostalih datoteka u projektu u jedno centralno programsko rješenje.



Slika 3. Struktura datoteka web servisa

5.2. MongoDB sheme

Kako se u bazu ne bi slali nestrukturirani podaci, pri izradi servisa koriste se sheme koje definiraju točan tip podataka i pravila vezana uz iste. Definiraju se modeli pomoću mongoose biblioteke koji će baratati svim dokumentima unutar kolekcije istog naziva. Mongoose je jednostavna JavaScript biblioteka koja služi za modeliranje podatak koji će se nalaziti u samoj bazi podataka.

5.2.1. User shema

„User“ shema definira kako bi trebao izgledati dokument korisnik unutar MongoDB baze. Kao što je vidljivo iz slike 4, definirano je polje za mail, koje je nužno za dokument te mora biti jedinstveno. Nakon toga slijede ime, prezime i plaća kao osnovni podaci svakog korisnika te su također nužni. Zatim se pojavljuje polje pozicije koje je direktna referenca na shemu pozicije. To polje omogućava dinamičku promjenu pozicije svakog korisnika. Polje za lozinku je također nužno i treba određena je minimalna veličina od 6 znakova. Za kraj su tu polje „hasTempPassword“ koje samo provjerava je li korisnik potvrdio svoj korisnički račun te nakon toga rola koju korisnik ima. Role služe za određivanje prava pristupa određenim funkcionalnostima. Ima četiri tipa rola: vlasnik, ljudski resursi, administrator i zaposlenik. Ako ni jedno od toga nije uneseno, korisniku se direktno pripisuje rola zaposlenika.

```
const userSchema = new mongoose.Schema({
   email: {
      type: String,
      required: true,
      unique: true,
   },
   firstName: {
      type: String,
      required: true,
   },
   lastName: {
      type: String,
      required: true,
   },
   salary: {
      type: Number,
      required: true,
   },
   position: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Job role",
      required: true,
   },
   password: {
      type: String,
      required: true,
      min: 6,
   },
   hasTempPassword: {
      type: Boolean,
      default: true,
   },
   role: {
      type: String,
      enum: ["owner", "hr", "admin", "employee"],
      required: true,
      default: "employee",
   }
```

Slika 4. Izvorni kod sheme User

5.2.2. Job Role shema

„Job role“ shema određuje neku poziciju na poslu koju korisnik može okupirati. Prvo polje predstavljeno u ovoj shemi je „positionName“ koje određuje ime pozicije te je nužno za popuniti i zahtjeva minimalno 3 znaka pri unosu. Sljedeće je polje „recommendedSalary“ koje služi za unos optimalne plaće za tu istu poziciju. Nakon toga imamo polje „canStartProject“ koje određuje ima li osoba s tom pozicijom pravo kreiranja projekta i upravljanja tima. Na kraju tu je opcionalno polje „description“ koje služi za slobodan unos teksta u kojem se opisuje pozicija.

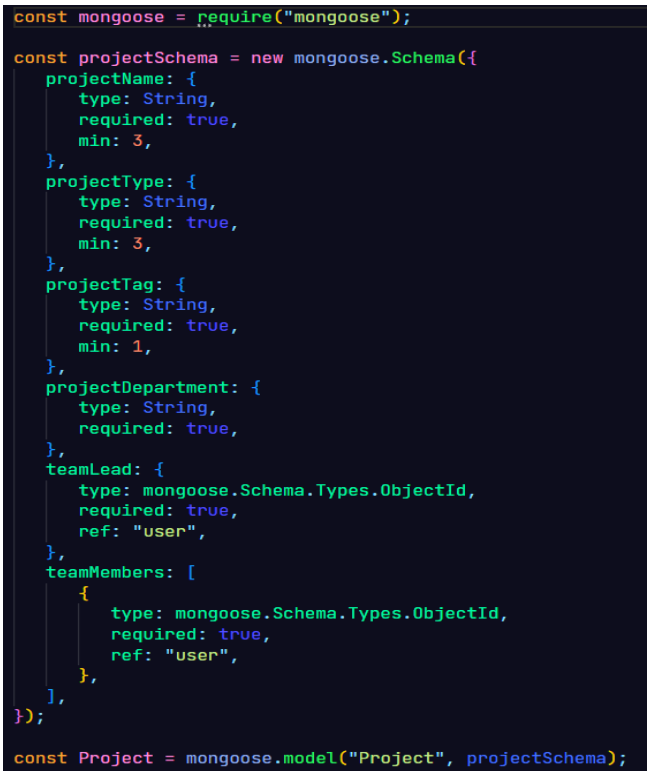
```
const mongoose = require("mongoose");

const jobRoleSchema = new mongoose.Schema({
   positionName: {
      type: String,
      required: true,
      min: 3,
   },
   recommendedSalary: {
      type: Number,
      required: true,
   },
   canStartProject: {
      type: Boolean,
      required: true,
      default: false,
   },
   description: {
      type: String,
   },
});

const JobRole = mongoose.model("Job role", jobRoleSchema);

module.exports = JobRole;
```

Slika 5. Izvorni kod sheme Job Role

5.2.3. Project shema

„Project“ shema je jedna od kompleksnijih shema te određuje projekt koji ujedno i definira tim koji radi na tom projektu. Prvo polje sheme je „projectName“ koje određuje samo ime projekta te je nužno i zahtjeva minimalno tri znaka. Iduće polje „projectType“ određuje tip odnosno sferu projekta te je također nužno i zahtjeva minimalno tri znaka. Polje „projectTag“ je kratica projekta te je nužna i zahtjeva bar jedan znak pri unosu.

Sljedeće polje je „projectDepartment“ polje koje označava odjel koji radi na projektu. Sljedeće su polja „teamLead“ koje je direktna referenca na korisnika koji vodi tim na projektu te polje „teamMembers“ koje sadrži polje svih referenci korisnika koji sudjeluju u timu.



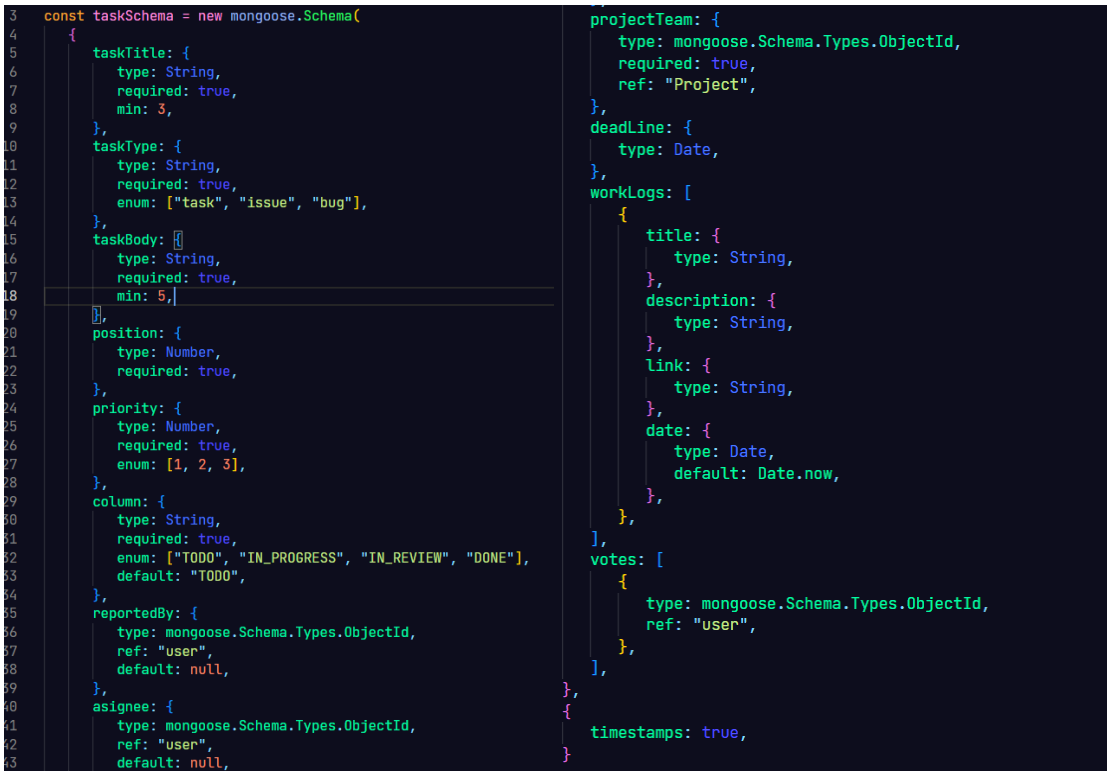
```
const mongoose = require("mongoose");

const projectSchema = new mongoose.Schema({
   projectName: {
      type: String,
      required: true,
      min: 3,
   },
   projectType: {
      type: String,
      required: true,
      min: 3,
   },
   projectTag: {
      type: String,
      required: true,
      min: 1,
   },
   projectDepartment: {
      type: String,
      required: true,
   },
   teamLead: {
      type: mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "user",
   },
   teamMembers: [
      {
         type: mongoose.Schema.Types.ObjectId,
         required: true,
         ref: "user",
      },
   ],
});

const Project = mongoose.model("Project", projectSchema);
```

Slika 6. Izvorni kod sheme Project

5.2.4. Task shema

„Task“ shema je najkompleksnija shema unutar projekta te sadržava informacije o zadaćama koje se trebaju obaviti unutar jednog projekta. Prvo polje definira naziv zadatka koji će se izvršavati te je nužno i zahtjeva minimalno 3 znaka pri unosu. Nakon toga slijedi polje „taskType“ koje određuje je li zadatak na kojem će se raditi bug, problem ili samo generalan zadatak te je nužno. Polje „taskBody“ sadrži dodatan opis samog zadatka te zahtjeva minmalno 5 znakova pri unosu te je također nužno. Sljedeće je polje „position“ koje određuje poziciju unutar kanban ploče na web aplikaciji te je shodno tome tipa broj. „Prirority“ označava razinu prioriteta zadatka, rangira se brojevima od 1-3 s time da je 1 najmanji prioritet, a 3 najveći prioritet. „Column“ polje određuje stupac u kojem se nalazi zadatak unutar kanban ploče na web aplikaciji. Polje „reportedBy“ sadrži direktnu referencu na korisnika koji je prijavio zadatak.

„Asignee“ je direktna referenca na korisnika koji trenutno radi na zadatku. Polje „projectTeam“ određuje kojem timu odnosno projektu zadatak pripada te sadrži direktnu referencu na shemu projekta. Polje „deadLine“ je opcionalno polje čija je namjena prikaz roka do kojeg bi zadatak trebalo obaviti. Nakon toga je polje „workLogs“ koje sadrži niz objekata koji predočavaju jedan radni dnevnik unutar određenog zadatka. Svaki radni dnevnik sadrži informacije poput imena dnevnika, opisa rada, ako postoji link na dio rada na koji se odnosi te datum i vrijeme unosa dnevnika rada. Na kraju se nalazi polje „votes“ koje samo sadrži niz referenci korisnika koji su označili zadatak sa „sviđa mi se“ u znak potpore.



Slika 7. Izvorni kod sheme Task

5.2.5. Activity log shema

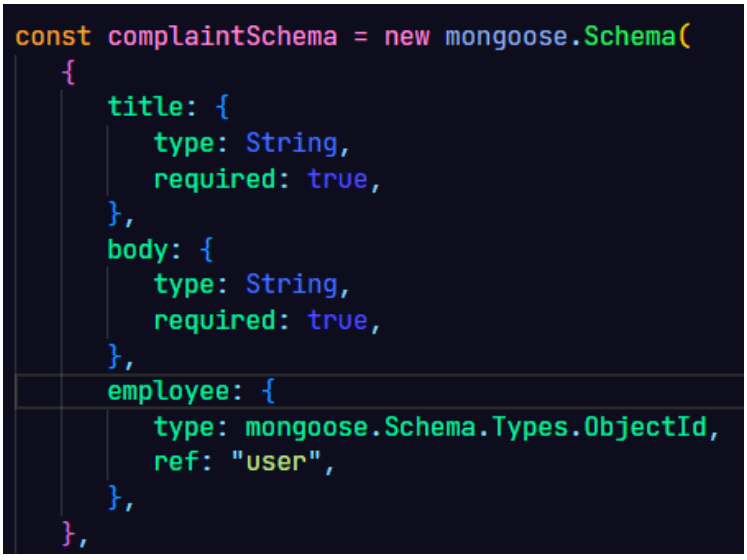
„Activity log“ shema služi za bilježenje aktivnosti korisnika pri pozivanju različitih akcija nad različitim tipovima dokumenata. Prvo polje je polje „actionType“ koje označava tip akcije koja je izvršena, a može nam označiti kreiranje, ažuriranje i brisanje dokumenta. Polje „actionEffect“ određuje na kojem tipu dokumenta se izvršila određena akcija. Tipovi dokumenata čije se akcije mogu bilježiti su redom: pozicija, projekt, korisnik i zadatak. Pred kraj se nalazi polje „user“ koje je direktna referenca na korisnika koji je

izvršio neku aktivnost. Kako bi ti podaci bili kronički izredani automatski se generiraju polja „createdAt“ i „updatedAt“ pomoću poziva „timestamps“ kao dijela sheme.

Slika 8. Izvorni kod sheme Activity Log

5.2.6. Complaint shema

„Complaint“ shema predstavlja žalbu unutar organizacije. Žalba sadrži polje „title“ za naslov same žalbe te polje „body“ za detaljan opis žalbe te su oba polja obavezna. Nakon toga slijedi polje „employee“ koje sadrži direktnu referencu na korisnika koji je podnio žalbu, no to polje je u potpunosti opcionalno s obzirom na to da korisnik ima mogućnost poslati anonimnu žalbu.



Slika 9. Izvorni kod sheme Complaint

5.3. Varijable okruženja

Varijable okruženja su objekti koji sadrže dodatne informacije za pokretanje programskog koda. U ovom programskom rješenju varijable okruženja se koriste kao spremnici za osjetljive podatke koji ne bi trebali biti dostupni korisniku iz sigurnosnih razloga. U varijable okruženja su svrstani tajni ključevi i lozinke za pristup različitim krajnjim točkama vanjskih API-a.

5.4. Konfiguracija krajnjih točaka

Na slici 10 prikazana je datoteka koja je zaslužna za povezivanje svih paketa u jednoj centralnoj lokaciji. Prije pokretanja servera preko express.js-a definira se varijabla „PORT“ koja određuje koji port će se koristiti u trenutnom okruženju. Ako u trenutnom okruženju nije prisutna definicija porta, koristi će se zadani port 5000. Kada je port određen potrebno je odrediti koja domena i koji port s te domene ima pristup krajnjim točkama. Za potrebe razvoja aplikacije koristi se lokalna adresa i zadani port React.js aplikacija, a to je port 3000. S tim kreće povezivanje baze kako bi krajnje točke mogle pristupati bazi i shodno funkcionalnosti za koju su namijenjene čitati i pisati podatke u istu. Uz povezivanje baze treba omogućiti čitanje notacije JavaScript objekata s obzirom na to da su svi podaci unutar baze JavaScript objekti. Na kraju svega definiraju se krajnje točke koje svoje imenovanje dobivaju prema dokumentima koje uređuju. U imenovanje ulazi i sama verzije API-a koja se koristi. To se koristi kao dobra praksa u slučaju da se promjene funkcionalnosti krajnjih točaka i više ne obavljaju zadaće na isti način ili s istim podacima. Pri samom kraju poziva se slušatelj koji će čekati neki poziv od klijenta i shodno tome pokrenuti odgovarajuću funkciju.

```
const PORT = process.env.PORT || 5000;

// creating the server instance
const app = express();
app.use(cors({ origin: "http://localhost:3000" }));

// configuration and database connection
connectToDB();
app.use(express.json());

// api endpoints
app.use("/api/v1/users", userRouter);
app.use("/api/v1/authentication", authenticationRouter);
app.use("/api/v1/roles", jobRoleRouter);
app.use("/api/v1/activity", activityRouter);
app.use("/api/v1/complaints", complaintRouter);
app.use("/api/v1/tasks", taskRouter);
app.use("/api/v1/projects", projectsRouter);

app.listen(PORT, () ⇒
    console.log(`[SERVER] Listening for requests at: http://localhost/${PORT}`)
);
```

Slika 10. Izvorni kod index.js datoteke

5.5. Middleware

Middleware je tip funkcije koja se pokreće prije pokretanja funkcije koja se nalazi na krajnjoj točki koja se poziva. Prvi *middleware* koji se koristi je „verifyUser“, koji je zaslužan za provjeru valjanosti json web tokena. Json web token je kompaktan i siguran spremnik koji se koristi za enkripciju podataka tajnim ključem. Tajni ključ se može predstaviti kao digitalni potpis. Svaki token koji je poslan s web servisa je prije slanja potpisan tajnim ključem te će biti pozitivno potvrđen samo ako tajni ključ odgovara ključu koji je spremljen u varijabli okruženja na web servisu. Ovisno o svrsi za koju je namijenjen, koriste se različiti algoritmi. U ovom slučaju koristi se sha256 algoritam za enkripciju podataka. *Middleware* „verifyUser“ prvo iz zahtjeva vadi zaglavlje autorizacije u kojoj se nalazi nositelj tokena. S obzirom na to da to zaglavlje vraća string s dvije različite riječi, prvotno se trebaju razdvojiti. Nakon toga se provjerava valjanost tokena. Valjanost tokena ovisi o starosti tokena i valjanom potpisu. Ako ne zadovoljava i jedan od ta dva slučaja, klijent će dobiti odgovor o ne valjanom tokenu. U suprotnom slučaju se u zahtjev stavlja korisnikov jedinstveni id.

```
const jwt = require("jsonwebtoken");

module.exports = verifyUser = (req, res, next) => {
   try {
      if (!req.headers.authorization) {
         return res.status(401).json({ msg: "No token present." });
      }
      const headerType = req.headers.authorization.split(" ")[0];
      if (!headerType === "Bearer") {
         return res.status(401).json({ msg: "Invalid token type." });
      }
      const token = req.headers.authorization.split(" ")[1];
      const data = jwt.verify(token, process.env.JWT_SECRET, {
         maxAge: "14d",
      });

      if (!data) {
         return res.status(401).json({ msg: "Invalid token" });
      }

      req.userID = data.uid;
      next();
   } catch (error) {
      console.error(error);
      res.status(401).json({ msg: "Invalid token" });
   }
};
```

Slika 11. Izvorni kod verifyUser middleware-a

Sljedeći i posljednji *middleware* korišten u projektu je „verifyRole“ middleware koji je zadužen za provjeru razine pristupa koju ima određeni korisnik. Pokreće se uvijek nakon „verifyUser“ *middleware-a* i nikad se ne pokreće sam po sebi. Razlog tome je to što zahtjeva korisnikov jedistveni id kojeg „verifyUser“ postavlja unutar samo zahtjeva. Administratori i vlasnik imaju najveće ovlasti unutar sustava pa su oni svrstani u kategoriju super korisnika, nakon toga slijedi odjel za ljudske resurse te rola običnih zaposlenika. U slučaju administatora i vlasnika postavlja se „isSuperUser“ unutar zahtjeva s pozitivnom vrijednošću. Ako je korisnik dio odjela za ljudske resurse postavlja se „isHR“ unutar zahtjeva s pozitivnom vrijednošću, dok se ostali postavljaju na negativnu vrijednost. Krajnji slučaj je običan korisnik i on nema nikakva elevirana prava pa se obje vrijednosti postavljaju u zahtjev kao negativne.

```
const User = require("../models/User");

module.exports = verifyRole = async (req, res, next) => {
   try {
      const user = await User.findById(req.userID);
      if (user.role === "admin" || user.role === "owner") {
         req.isSuperUser = true;
         req.isHR = false;
      } else if (user.role === "hr") {
         req.isSuperUser = false;
         req.isHR = true;
      } else {
         res.isSuperUser = false;
         req.isHR = false;
      }
      next();
   } catch (error) {
      console.error(error);
      res.status(401).json({ msg: "Server Error" });
   }
};
```

Slika 12. Izvorni kod verifyRole middleware-a

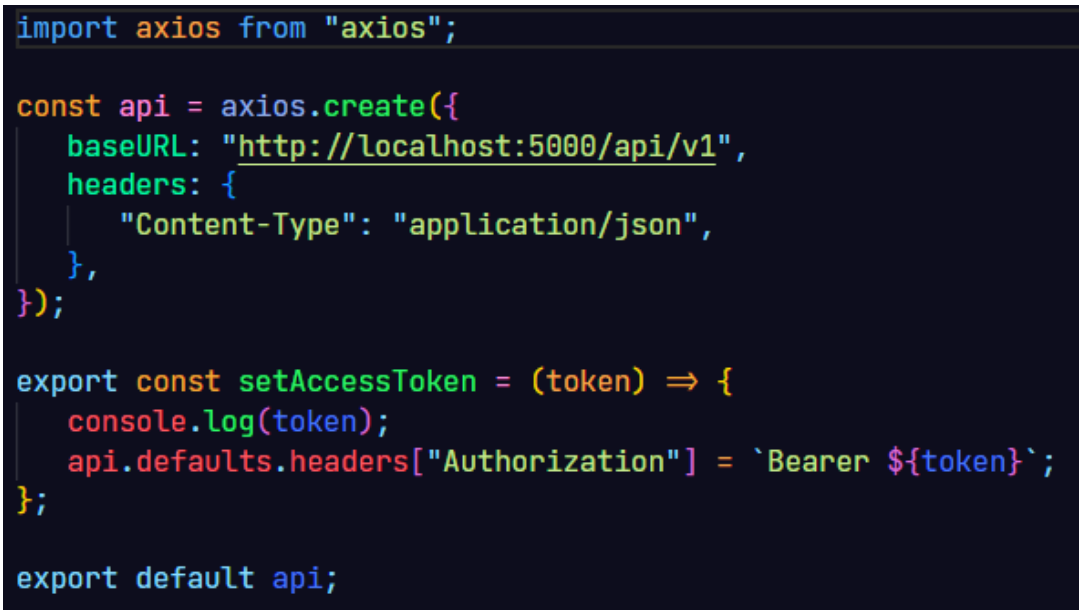
6. Implementacija web aplikacije

6.1. Struktura mapa i datoteka

Za lakši i brži razvoj aplikacija unutar React.js biblioteke potrebno je imati čistu organizaciju mapa i datoteka. U ovom projektu mape su podijeljene prema svrsi. Mapa *api* sadržava sve konfiguracije za API. Od konfiguracija koje se nalaze u toj mapi su instanciranje axios-a prema temeljnom URL-u i funkcije za postavljanje tokena u zahtjeve prema web servisu odnosno serveru. Mapa *components* sadrži sve komponente koje se koriste pri gradnji vizualnih elemenata aplikacije. Komponente koje se tu nalaze su obično redundantan ili opsežan kod koji je odvojen od ostatka zbog lakše čitljivosti, smanjenja redundantnosti i mogućnosti ponovnog korištenja unutar više od jednog pogleda. Sve datoteke vezane uz implementaciju i rukovanje *redux* elementima stanja nalazi se u *store* mapi. Pogledi su prikazi koji se prikazuju unutar različitih ruta te se shodno tome nalaze unutar *views* mape. Svaka mapa unutar *views* mape je imenovana prema objektu ili funkcionalnosti na koju se odnosi.

6.2. Axios instanca

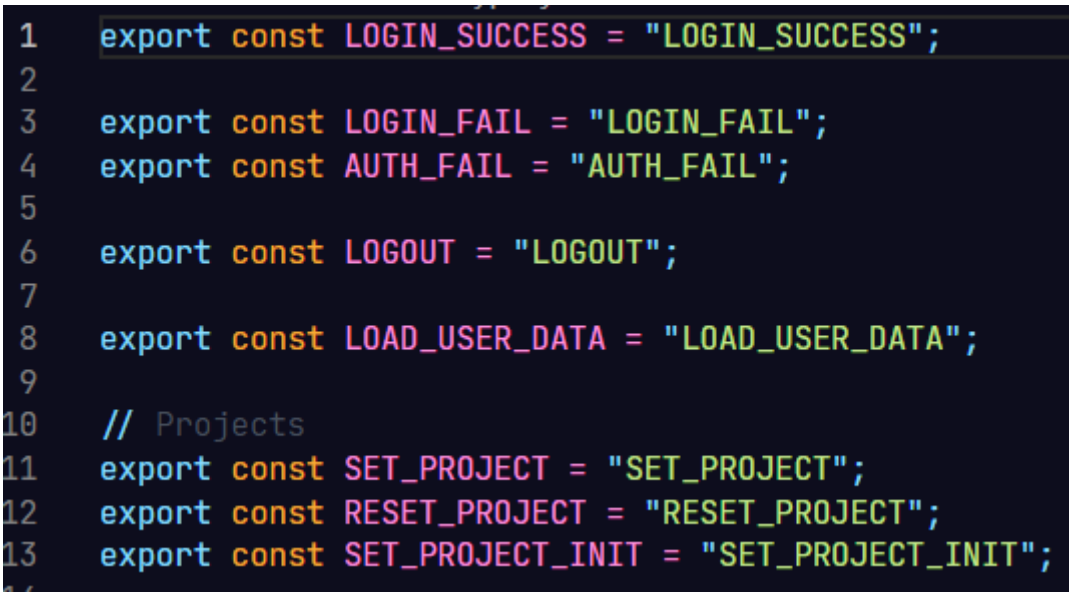
Kako bi klijent mogao komunicirati i razmjenjivati podatke sa serverom, koristi se axios. *Axios* je Javascript HTTP klijent baziran na obećanjima (Axios, n.d.). Obečanja su najbolje objašnjena kao dio koda koji čeka odgovor. Sve odgovore i zahtjeve automatski pretvara u notaciju JavaScript objekta. Kako ne bi bilo potrebno mijenjati kod na više mjesta kod promjene okruženja ili promjene verzije API-a korištenog za aplikaciju, napravljena je posebna instanca axios-a kojoj se definira temeljni URL koji će se odraziti u kodu na ostalim dijelovima aplikacije gdje se koristi bez da ga se eksplicitno naglašava. Toj axios instanci se također naglašava u zaglavlje tipa sadržaja kojeg će primiti da se radi isključivo o notaciji JavaScript objekata. Kao što je vidljivo iz slike 13, tu je i pomoćna funkcija „setAccessToken“ koja samo postavlja token za autorizaciju gdje god se koristi i gdje god je potreban.



Slika 13. Izvorni kod axios instance

6.3. Redux implementacija

Redux se koristi za upravljanje globalnim stanjem kojem je potrebno pristupiti iz više od jedne komponente unutar same aplikacije. Kroz ovu aplikaciju redux je korišten za implementaciju dosljednog stanja korisnika i praćenja kojem projektu je određeni korisnik pristupio. Shodno tome u redux-u su definirani odgovarajući reduktori i akcije. Prije samih akcija i reduktora bitno je definirati moguće tipova akcija, pošto su tipovi akcija obični stringovi te su skloni pogreškama. Idealna solucija za smanjenje te sklonosti pogreškama je postavljanje tipova u konstante s obzirom na to da se nikad ne mijenjaju. Tipovi akcija nisu nužno vezani uz samo jednu akciju. Na slici 14 prikazane su sve definicije tipova koji se koriste kroz projekt.



Slika 14. Izvorni kod tipova akcija

6.3.1. Auth reduktor

Prije definiranja bilo kojeg reduktora, potrebno je unutar iste datoteke uvesti sve tipove akcija koje se planiraju vršiti nad stanjem samog reduktora. Na početku samog reduktora se definira njegovo početno stanje. Na slici 15 vidljivo je da unutar stanja reduktora „auth“ spremamo autorizacijski token, stanje učitavanja i podatke o samom korisniku. Token se hvata is lokalnog spremnika koji je prisutan u svim preglednicima. Ako token postoji stanje će biti taj isti token, a ako ne postoji token će biti jednak vrijednosti null. Iduće je stanje učitavanja kojem je početna vrijednost istinita, jer se pretpostavlja da svaki put kad korisnik učita stranicu, njegovi podaci još uvijek nisu prisutni što se ujedno i odražava u posljednjem stanju „user“ koje drži podatke o trenutnom korisniku. Nakon definicije početnog stanja potrebno je definirati kako će se reduktor mijenjati kroz različite tipove akcija koji su mogući. Svaka akcija vraća svoj tip i *payload* u kojem su obično spremljeni neki korisni podaci odnosno koristan teret. Kako bi sustav bio što dinamičniji bez previše gniježdenja, za samu implementaciju koristi se sklopka koja se aktivira ovisno o tipu akcije koja se pozove. Prvi slučaj na koji sklopka može naletjeti je „LOAD_USER_DATA“ koji je zaslužan za učitavanje korisničkih podataka. S tim na umu mijenjaju se početne vrijednosti za učitavanje i spremnik korisničkih podatakad dok ostala stanja jednostavno vraćamo. Sljedeći slučaj, „LOGIN_SUCCESS“, zaslužan je za prijavu korisnika u sustav. Koristan teret ovog slučaja je token i putem tog tereta se postavlja token u lokalni spremnik

preglednika s ključnom vrijednošću „token“. Na kraju se učitavanje postavlja na negativnu vrijednost. Iduća tri slučaja, „LOGIN_FAIL“, „AUTH_FAIL“ i „LOGOUT“, rezultiraju istom akcijom. S obzirom da sva tri zahtijevaju da se trenutnog korisnika odjavi, sva tri rezultiraju uklanjanjem tokena iz lokalnog spremnika preglednika te postavljanjem stanja korisničkih podataka i tokena na null vrijednost i postavljanje učitavanja na negativnu vrijednost.

```
import {
    LOGIN_SUCCESS,
    AUTH_FAIL,
    LOGIN_FAIL,
    LOGOUT,
    LOAD_USER_DATA,
} from "../actions/types";

const initialState = {
    token: localStorage.getItem("token"),
    loading: true,
    user: null,
};

export default (state = initialState, action) => {
    const { type, payload } = action;

    switch (type) {
        case LOAD_USER_DATA:
            return {
                ...state,
                loading: false,
                user: payload,
            };
        case LOGIN_SUCCESS:
            localStorage.setItem("token", payload.token);
            return {
                ...state,
                ...payload,
                loading: false,
            };
        case LOGIN_FAIL:
        case AUTH_FAIL:
        case LOGOUT:
            localStorage.removeItem("token");
            return {
                ...state,
                user: null,
                token: null,
                loading: false,
            };
        default:
            return state;
    }
};
```

Slika 15. Izvorni kod auth reduktora

6.3.2. Project reduktor

Za razliku od „auth“ reduktora, „project“ reduktor je puno jednostavniji. U početnom stanju se nalazi spremnik za trenutni projekt te stanje učitavanja istog. Slučajevi koji mogu zadesiti ovaj reduktor su „SET_PROJECT_INIT“ koji služi za postavljanje stanja učitavanja prema korisnom teretu kojeg prima. Nakon toga je slučaj „SET_PROJECT“ koji služi za postavljanje stanja trenutnog projekta prema primljenom korisnom teretu koji ciljano sadrži podatke o trenutnom projektu. Posljednji slučaj je „RESET_PROJECT“ koji služi za ponovno incijaliziranje stanja

projekta na početnu vrijednost. Shodno tome se trenutni projekt postavlja na null vrijednost, a stanje učitavanja odnosno inicijaliziranja na pozitivnu vrijednost.

```
const initialState = {
   currentProject: null,
   isProjectLoading: false,
};

export default (state = initialState, action) => {
   const { type, payload } = action;

   switch (type) {
      case SET_PROJECT_INIT:
         return {
            ...state,
            isProjectLoading: payload,
         };
      case SET_PROJECT:
         return {
            ...state,
            currentProject: payload,
         };
      case RESET_PROJECT:
         return {
            isProjectLoading: true,
            currentProject: null,
         };
      default:
         return state;
   }
};
```

Slika 16. Izvorni kod project reduktora

6.3.3. Glavni reduktor

Kako bi oba reduktora bila dostupna za korištenje unutar aplikacije, postoji centralan reduktor koji kombinira, u ovom slučaju oba reduktora, pod nekim pseudonimom. U ovom slučaju pseudonimi su nazivi samih reduktora kao što je vidljivo iz slike 17.

```
import { combineReducers } from "redux";
import auth from "./auth";
import project from "./project";

export default combineReducers({ auth, project });
```

Slika 17. Izvorni kod glavnog reduktora

6.3.4. Auth akcije

Redux akcije su funkcije koje reduktorima šalju tip akcije i koristan teret. Obično su pozvane iz različitih komponenti i pogleda, ali često su korištene i unutar samih akcija.

Prva akcija koja je implementirana je „loginUser“. Ta funkcija kroz argumente prima email i lozinku te ih šalje kao objekt detalja prijave pomoću naše *axios* instance na rutu web servisa zaslužnu za autentifikaciju. U slučaju da su podaci točni i sve prođe kako treba, *auth* reduktoru se šalje akcija tipa „LOGIN_SUCCESS“ s tokenom kao korisnim teretom te se nakon toga poziva druga akcija koja se nalazi u istoj datoteci, a to je „loadUserData“ akcija. U slučaju da je došlo do pogreške šalje se „LOGIN_FAIL“ tip akcije bez ikakvog korisnog tereta. U „loadUserData“ funkciji, vadi se podatak o tokenu unutar lokalnom spremniku preglednika te se pomoću pomoćne funkcije stavlja u zaglavlje ako postoji. Na kraju se šalje zahtjev na rutu web servisa zaslužnu za autorizaciju te se u uspješnom slučaju *auth* reduktoru šalje akcija tipa „LOAD_USER_DATA“ s korisnim teretom u kojem se nalaze korisnički podaci koji su dobiveni od strane web servisa. U slučaju da dođe do neke greške šalje se akcija tipa „AUTH_FAIL“ bez ikakvog korisnog tereta. Posljednja akcija je „logoutUser“ koja jednostavno šalje tip akcije „LOGOUT“ bez ikakvog korisnog tereta te služi za odjavu korisnika.

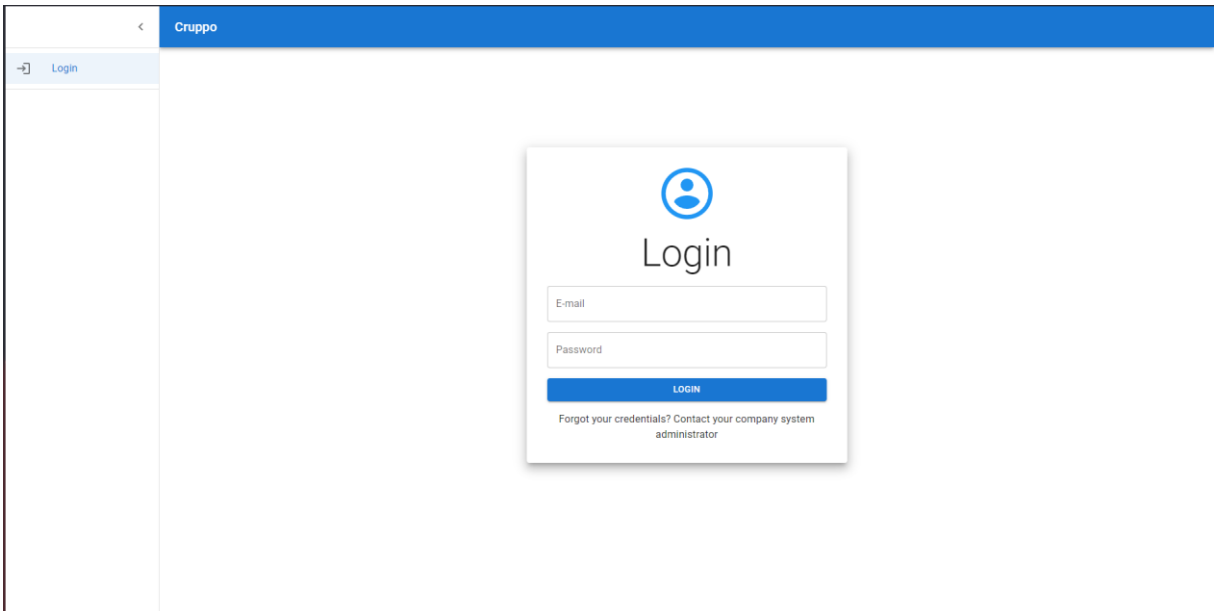
```
export const loginUser = (email, password) => async (dispatch) => {
   const loginDetails = { email, password };

   try {
      const res = await api.post("/authentication", loginDetails);

      dispatch({
         type: LOGIN_SUCCESS,
         payload: res.data,
      });
      dispatch(loadUserData());
   } catch (error) {
      dispatch({
         type: LOGIN_FAIL,
      });
   }
};

export const loadUserData = () => async (dispatch) => {
   const token = localStorage.getItem("token");
   if (token) {
      setAccessToken(token);
   }

   try {
      const res = await api.get("/authentication");

      dispatch({
         type: LOAD_USER_DATA,
         payload: res.data,
      });
   } catch (error) {
      dispatch({
         type: AUTH_FAIL,
      });
   }
};

export const logoutUser = () => (dispatch) => {
   dispatch({
      type: LOGOUT,
   });
};
```

Slika 18. Izvorni kod auth akcija

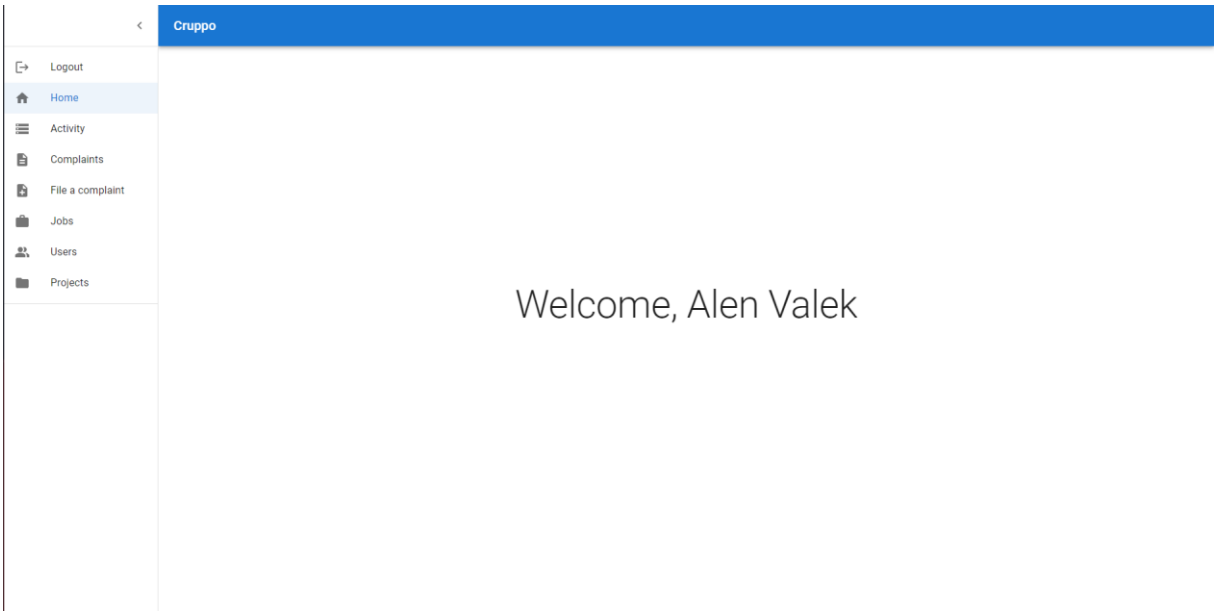
6.4. Pogledi aplikacije

6.4.1. Početni pogled

Pri prvom posjetu stranici korisnik je predstavljen s formom za prijavu. Aplikacija nema mogućnost registracije, već nove korisnike može dodati administrator. Početni pogled može varirati s obzirom na stanje prijave korisnika. S obzirom na to da korisnik može, a i ne mora biti već prijavljen, u slučaju korisnika koji nije prijavljen vidljivo je korisničko sučelje kao na slici 19. U slučaju da je korisnik već prijavljen, kao što je vidljivo na slici 20, prikazan mu je pozdrav s imenom i prezimenom te su mu sad dostupne opcije u navigacijskoj ladici shodno razini pristupa koju posjeduje.



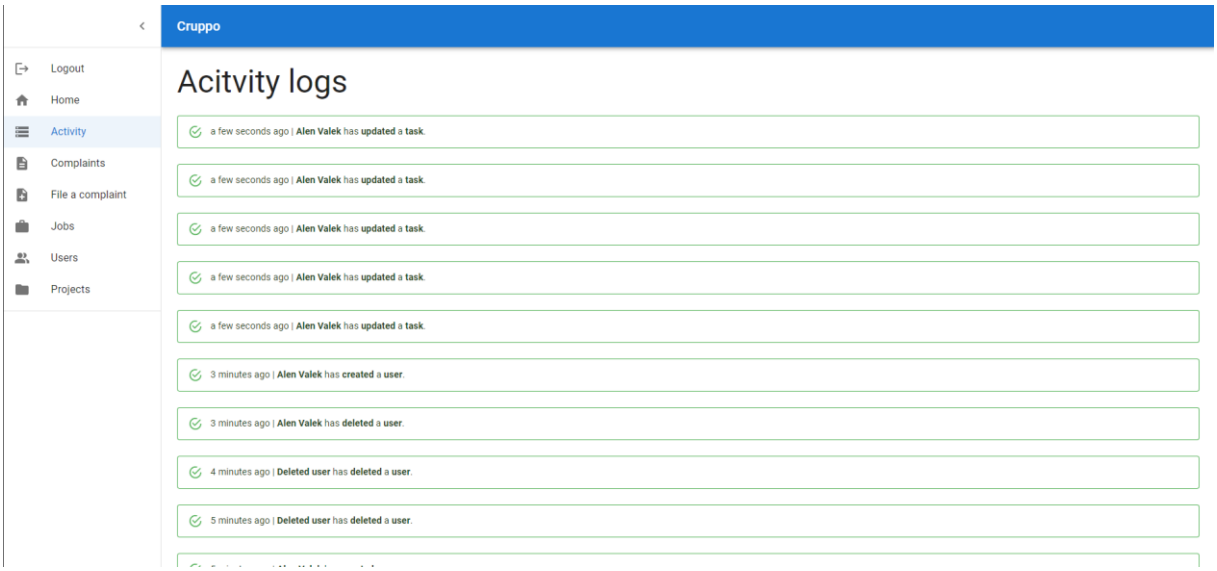
Slika 19. Početni pogled – Gost



Slika 20. Početni pogled – korisnik

6.4.2. Pogled aktivnosti

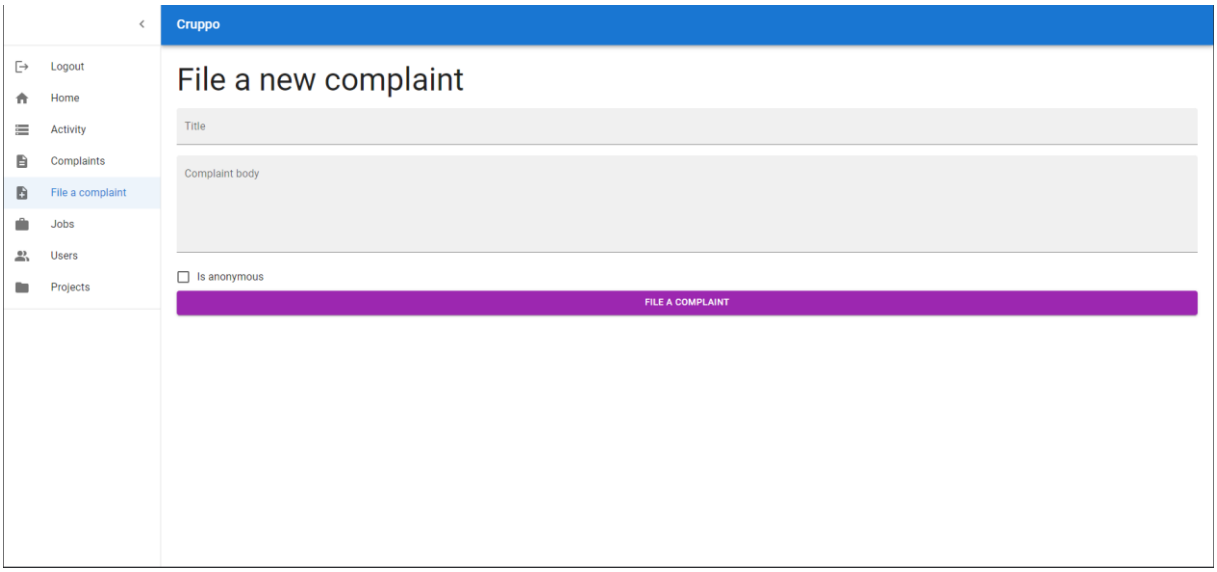
Pogled aktivnosti omogućava korisnicima s višim pravom pristupa pregled akcija koje su se odvile od početka rada sustava pa do trenutka kad je otvoren pogled. Prema slici 21 jasno se vidi da se bilježi vrijeme relativno prema trenutnom vremenu, ime osobe, tip akcije i dokument nad kojim se akcija izvršila. U slučaju da je korisnik u međuvremenu izbrisan ili je došlo do neke druge greške, njegovo ime i prezime. U slučaju da je došlo do neke greške ili korisnik više ne postoji, svaka aktivnost će to odraziti tako da ispiše da je korisnik izbrisan.



Slika 21. Pogled aktivnosti

6.4.3. Pogled kreiranja žalbi i pogled pregleda žalbi

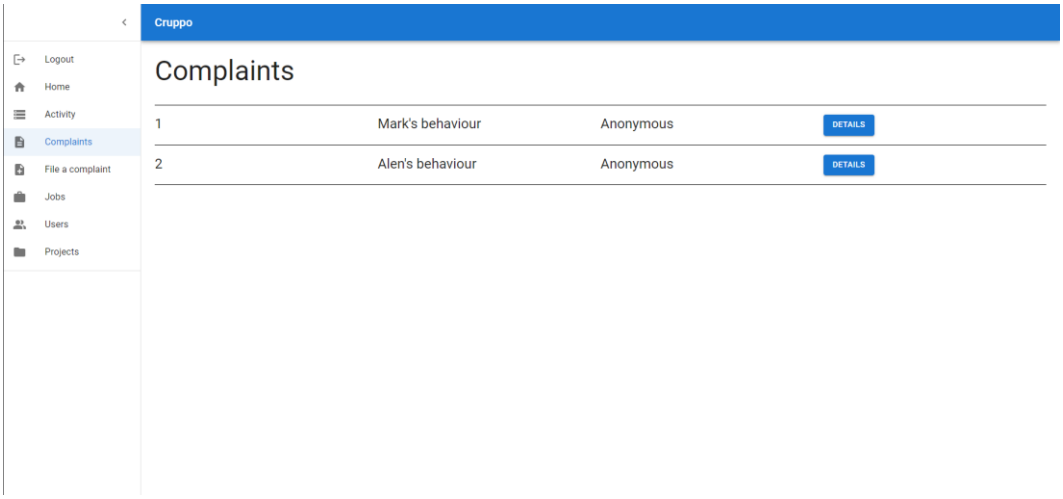
Svaki korisnik ima pravo kreirati žalbu. Potrebno je unijeti naslov žalbe i opis. Opcionalno korisnik može žalbu predati kao anonimnu. Pritiskom na gumb „File a complaint“ kreirat će se žalba i spremiti u bazu.



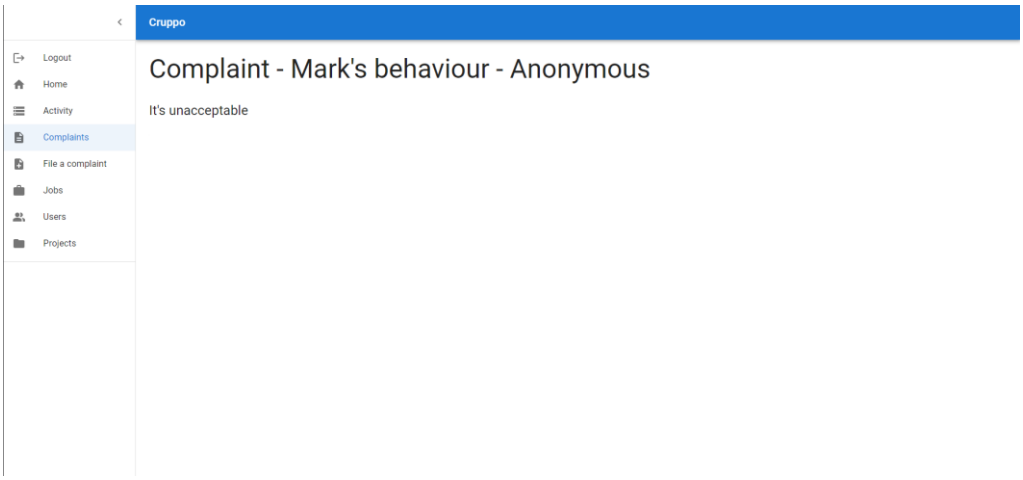
Slika 22. Pogled kreiranja žalbi

6.4.4. Pogled pregleda žalbi

Svaka žalba koja je kreirana dostupna je na pregled korisnicima eleviranih ovlasti i korisnicima s ovlastima ljudskih resursa. Svaka žalba sadrži redni broj, naslov i ime podnositelja ako podnositelj nije anoniman. Pored žalbi također postoji gumb koji će dinamički prikazati detalje žalbe prema jedinstvenom ID-u. Jedinstveni ID žalbe bit će proslijeđen kroz parametre URL-a te će se pogled zaslužan za detalje pobrinuti da se učitaju detalji upravo te žalbe.



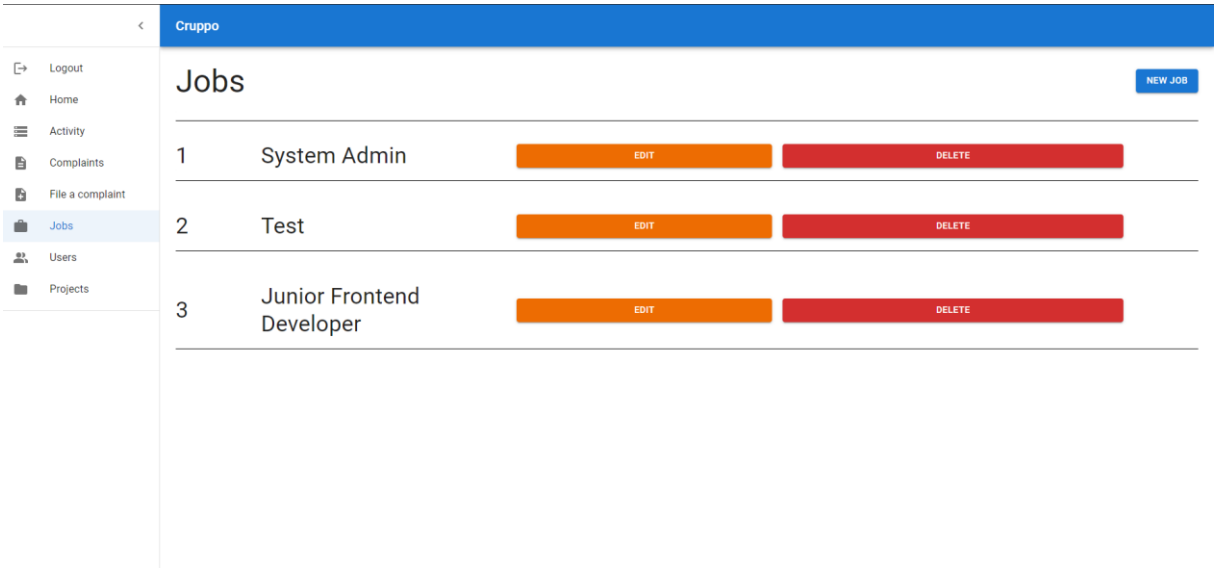
Slika 23. Pogled pregleda žalbi



Slika 24. Pogled detaljnog prikaza žalbe

6.4.5. Pogled poslova

Pregledu poslova imaju pristup samo elevirani korisnici. Prva razina pregleda poslova izlista sve poslove koji se trenutno nalaze unutar organizacije. Svaki posao sadrži redni broj i naziv posla. Dodatno imaju dva gumba, jedan koji će odvesti korisnika na pogled za uređivanje, a drugi koji će trajno izbrisati posao. Važno je naglasiti da ako je posao u upotrebi potrebno je i ažurirati korisnike kojima taj posao pripada inače će postojati korisnici koji nemaju određen posao.

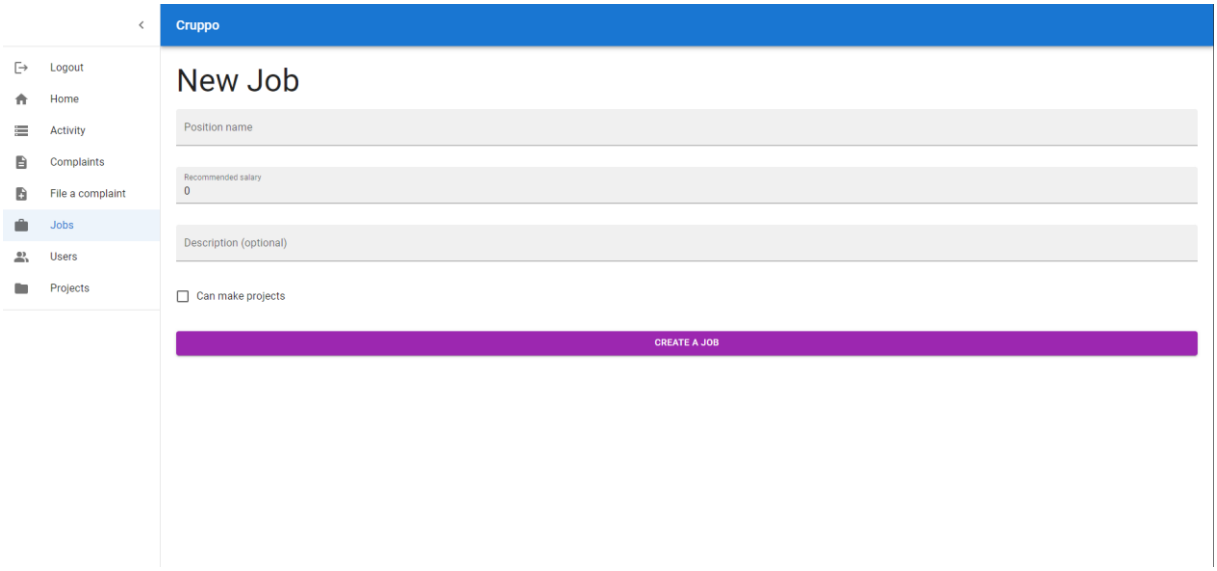


Slika 25. Pogled pregleda poslova

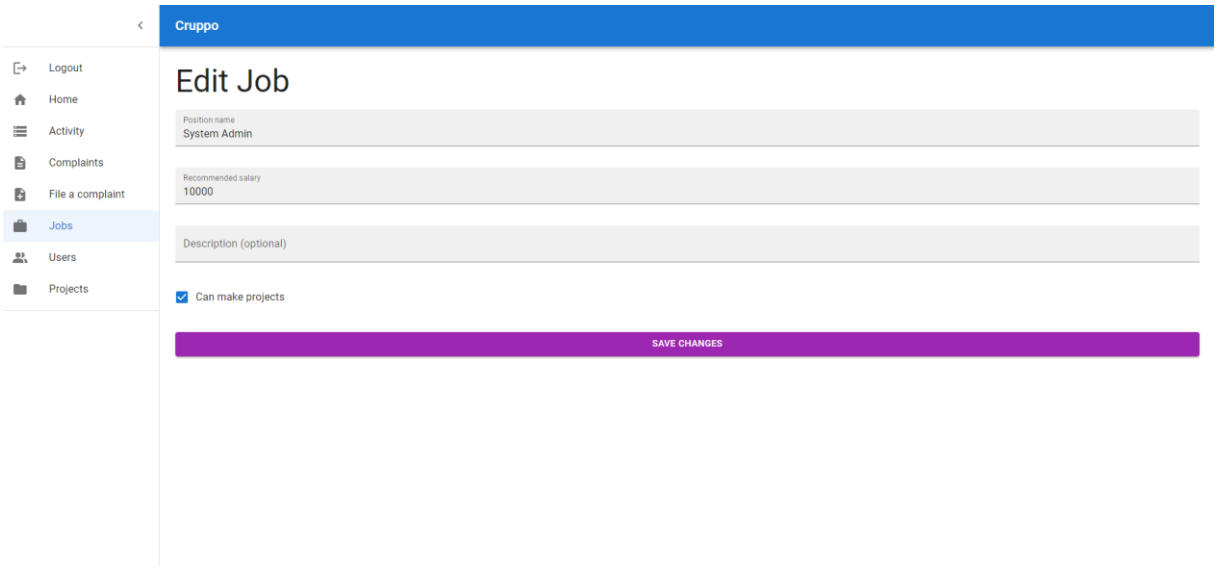
6.4.6. Pogled kreiranja i pogled uređivanja poslova

Pogled kreiranja i pogled uređivanja poslova su skoro pa identični. Sadrže ista polja koja se trebaju popuniti. Jedina postojana razlika je da su kod uređivanja polja već popunjena s obzirom na to da se uređuje postojeća stavka. Kako bi se kreirao novi posao, potrebno je ispuniti nekoliko polja. Prvo je naziv pozicije koju će zadani posao okupirati, nakon čega slijedi polje za preporučenu plaću i opcionalno polje za opis same pozicije. Na kraju korisnik može odabrati želi li da osoba koja će okupirati ovu poziciju bude ovlaštena za kreiranje projekata odnosno vođenje svoga tima. Na

posljetku pritiskom na „create a job“ gumb, korisnik pokreće proces kreacije posla. U slučaju da je došlo do greške, aplikacije će to javiti u stilu *toast* notifikacije.



Slika 26. Pogled kreacije poslova

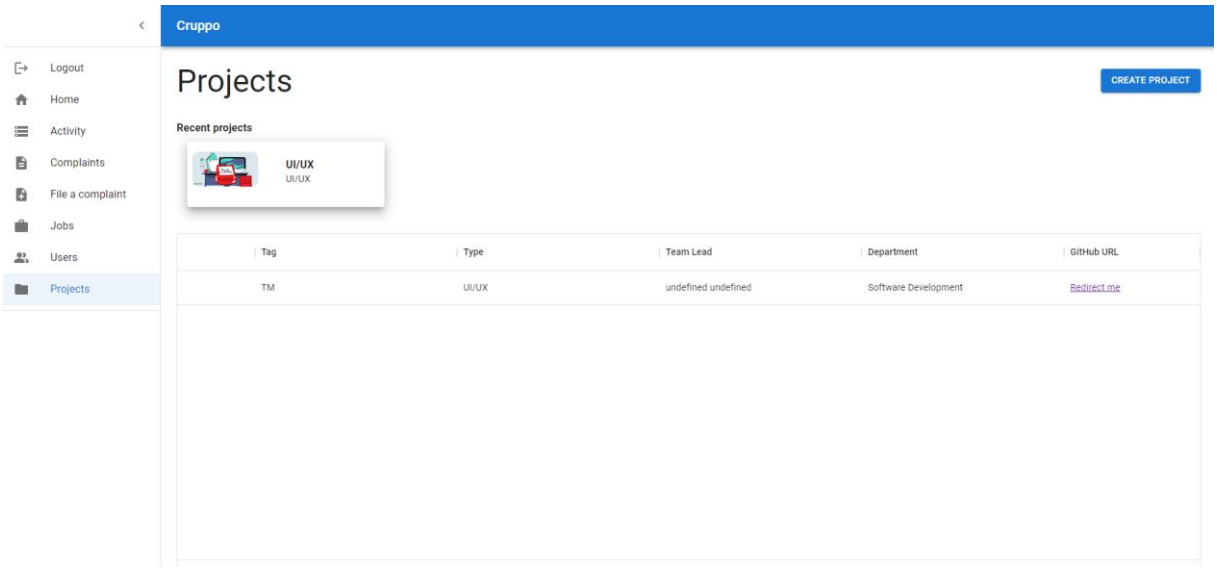


Slika 27. Pogled uređivanja poslova

6.4.7. Pogled pregleda projekata

Pri pogledu projekata korisnik je predočen s trenutnim projektima koji su započeti. Ako korisnikova pozicija to dozvoljava, korisnik će moći i kreirati novi projekt. Korisnik uz projekte koji su trenutno aktivni može vidjeti i do tri projekta koje je posljednje posjetio.

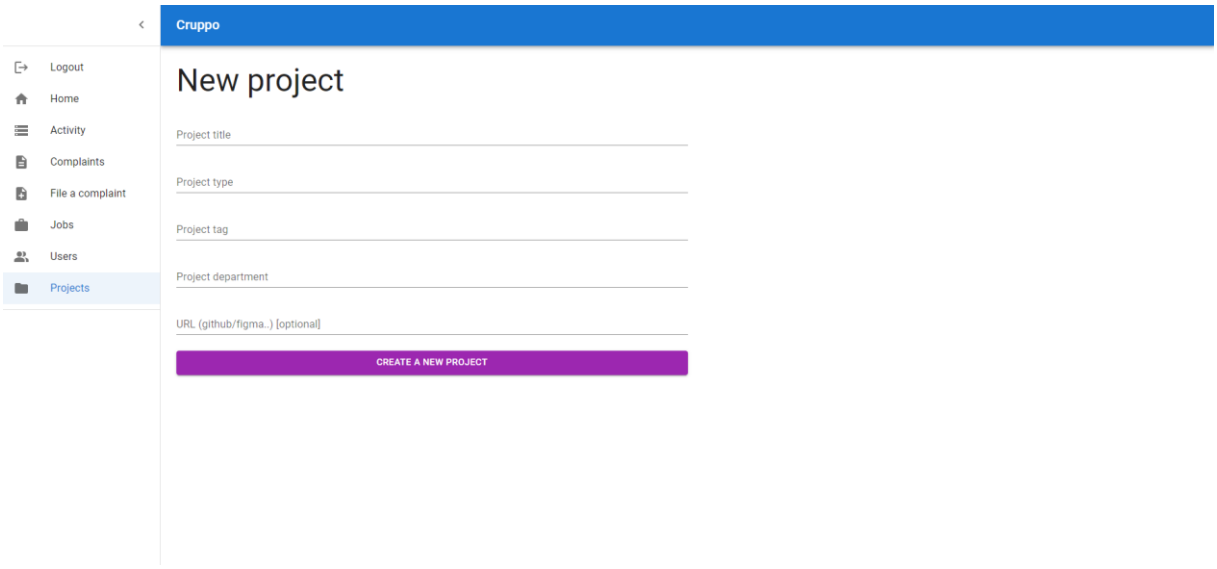
Trenutno aktivni projekti su izlistani unutar podatkovne tablice. Svako polje je moguće sortirati. Unutar same podatkovne tablice nalaze se informacije o projektu. Te informacije sadrže ime projekta, oznaku projekta, tip projekta, vođu tima na projektu, odjel kojem pripada projekt te ako postoji URL projekta.



Slika 28. Pogled pregleda projekata

6.4.8. Pogled kreacije projekata

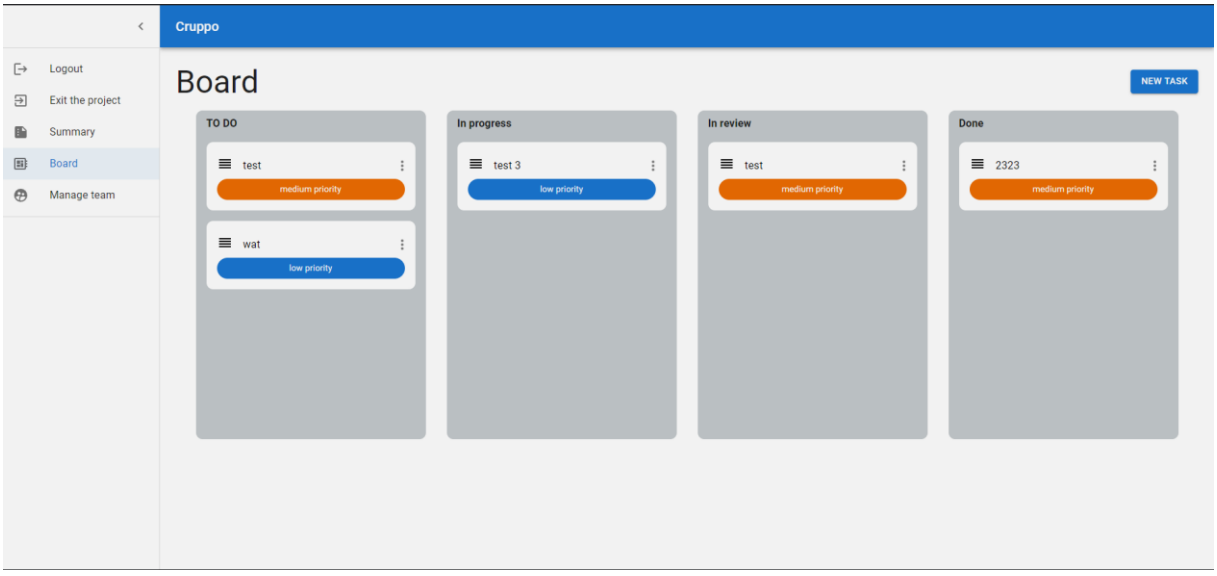
Pogled kreacije objekata sadrži jednostavnu formu gdje je potrebno unijeti naziv projekta, tip projekta, oznaku projekta, odjel projekta te opcionalno URL projekta. Osoba koja kreira projekt, postaje vođa tima istog projekta.



Slika 29. Pogled kreacije projekata

6.4.9. Pogled projektne ploče

Nakon ulaska u jedan od projekata korisnik je usmjeren na pogled projektne ploče. Ploča je inspirirana popularnom kanban metodom koja se bavi tijekom rada. Kanban ploče su korištene za vizualizaciju i organizaciju timskog rada. Stupci pokazuju sekvence aktivnosti gdje kartice prikazuju značajke projekta na kojem se radi (Ahmad, M. O., Dennehy, D., Conboy, K., & Oivo, M., 2018). Projektna ploča je jedan od kompleksnijih pregleda unutar same aplikacije. Što je posebno za projektnu ploču je to da sudionici tima mogu vizualno predočiti koji zadatak se nalazi u kojoj fazi razvoja projekta kao što je vidljivo iz slike 30. Ploča se sastoji od fiksnog broja stupaca. Svaki stupac predstavlja jednu fazu razvoja. Prva faza i faza u koju se stavljaju novokreirani zadaci je faza „to do“ odnosno za napraviti. Nakon te faze, zadatak se premješta u fazu „in progress“ odnosno u fazu razvoja. Nakon faze razvoja, zadatak ide u fazu „in review“ gdje će vrlo vjerojatno voditelj tima pregledati zadatak i dnevnik rada te zaključiti je li potrebno dalje razvijati ili povući zadatak u posljednju fazu „done“ gdje se nalaze svi završeni zadaci.



Slika 30. Pogled projektne ploče

Zadaci se mogu vući i ispuštati u bilo koji stupac hvatanjem elementa na mjestu koje je predviđeno za to. Korisnicima to mjesto i vizualno daje povratnu informaciju da se može vući. Svaki zadatak se može vući i unutar istog stupca u kojem se već nalazi. Razlika je u tome da se pri vući unutar istog stupca ažurira samo pozicija elementa dok se u slučaju vuče prema drugom stupcu, mora ažurirati i pozicija i stupac. Kako bi

se pravilno ažurirale pozicije i stupci bilo je potrebno razviti algoritam koji će sve to pratiti. Prva moguća situacija koju treba pokriti je vuča prema istom stupcu. U slučaju da se kartica nije maknula s trenutne pozicije, ostavljamo je gdje je i bila te nema potrebe ažurirati cijeli niz kartica. Nakon toga treba provjeriti je li kartica koja se trenutno provjerava unutar pozicija koje će biti utjecane promjenom te provjeriti jesu li manje ili veće od pozicije na koju se stavlja kartica. Prema tome se može odrediti treba li poziciju smanjiti za 1 ili povećati za 1. U slučaju da ništa od toga nije zadovoljeno samo se vraća kartica u stanju kakvom je i bila.

```
let modifiedArray = tasks.map((task) => {
   // ako je drag prema istom column-u
   if (isInSameColumn) {
      if (task._id === draggableId) {
         task.position = destination.index;
         api.patch(`/tasks/${task._id}/${id}`, {
            newPosition: task.position,
            newColumn: task.column,
         });
         return task;
      } else if (affectedRange.includes(task.position)) {
         if (isGreater) {
            task.position = task.position - 1;
            api.patch(`/tasks/${task._id}/${id}`, {
               newPosition: task.position,
               newColumn: task.column,
            });
            return task;
         } else {
            task.position = task.position + 1;
            api.patch(`/tasks/${task._id}/${id}`, {
               newPosition: task.position,
               newColumn: task.column,
            });
            return task;
         }
      } else {
         return task;
      }
```

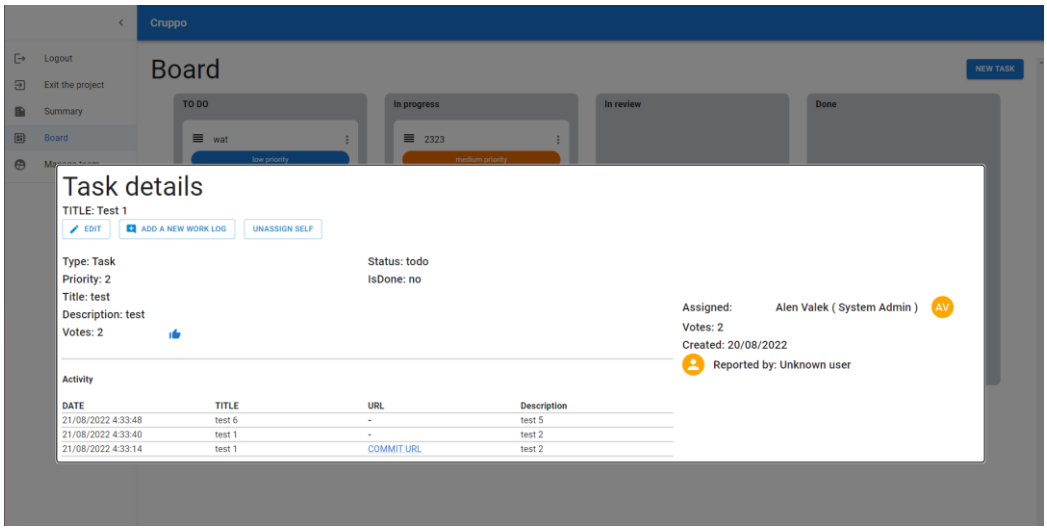
Slika 31. Izvorni kod za ažuriranje pozicija prema istom stupcu

U slučaju da je kartica povučena u stupac u kojem se prije nije nalazila, potrebno je zapamtiti stupac u kojem se nalazila prije kako bi se ažurirale pozicije kartica na koje je promjena utjecala. Također je bilo potrebno paziti na krajnje situacije kod kojih bi kartica bila jedina kartica u stupcu, posljednja kartica u stupcu ili prva kartica u stupcu. U slučaju da se vuče kartica iz sredine treba ažurirati sve pozicije manje na prema originalnoj poziciji kartice koja se vuče. Pozicije se u tom slučaju smanjuju za 1. Isti je i proces kad kartica dolazi u novi stupac, potrebno je ažurirati sve kartice na čije je pozicije utjecala nova pozicija kartice koja dolazi.

Slika 32. Izvorni kod za ažuriranje pozicija u različitim stupcima

6.4.10. Pogled detalja zadatka

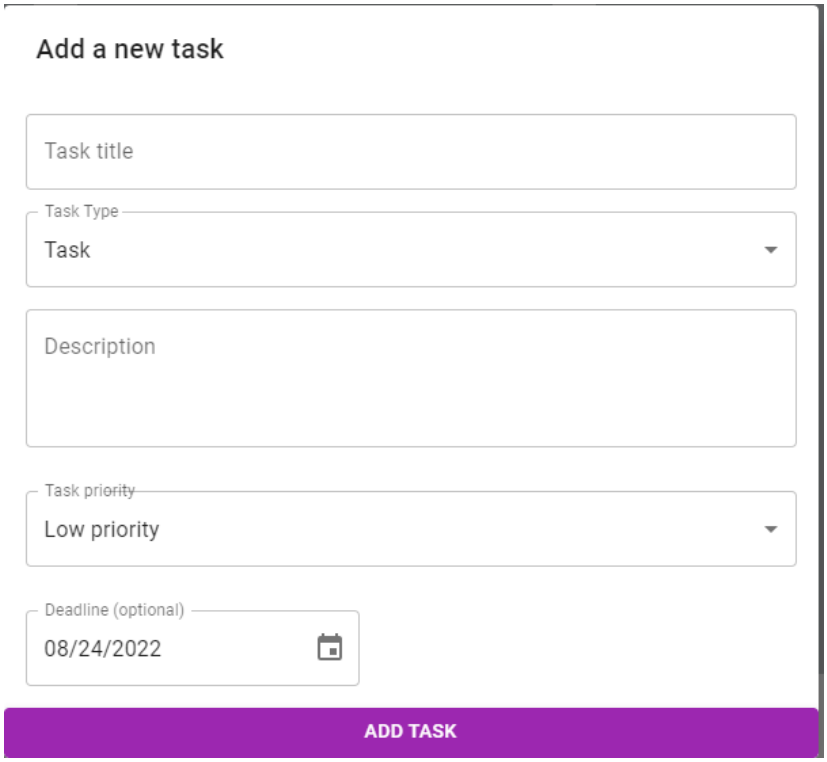
Svaki zadatak ima svoj detaljan prikaz u kojem su vidljivi radni dnevnici i dodatne informacije. Uz sve opće informacije o zadatku vidljivo je tko je trenutno postavljen za rješavanje zadatka, koliko ljudi je glasovalo za taj zadatak, detaljniji opis zadatka, datum kad je zadatak nastao te ako postoji rok završetka istog. Pritiskom na gumb „add a new work log“ otvara se forma za dodavanje novog dnevnika rada. U toj formi korisnik može priložiti naslov, opis i URL ako postoji.



Slika 33. Pogled detalja zadatka

6.4.11. Pogled kreacije zadatka

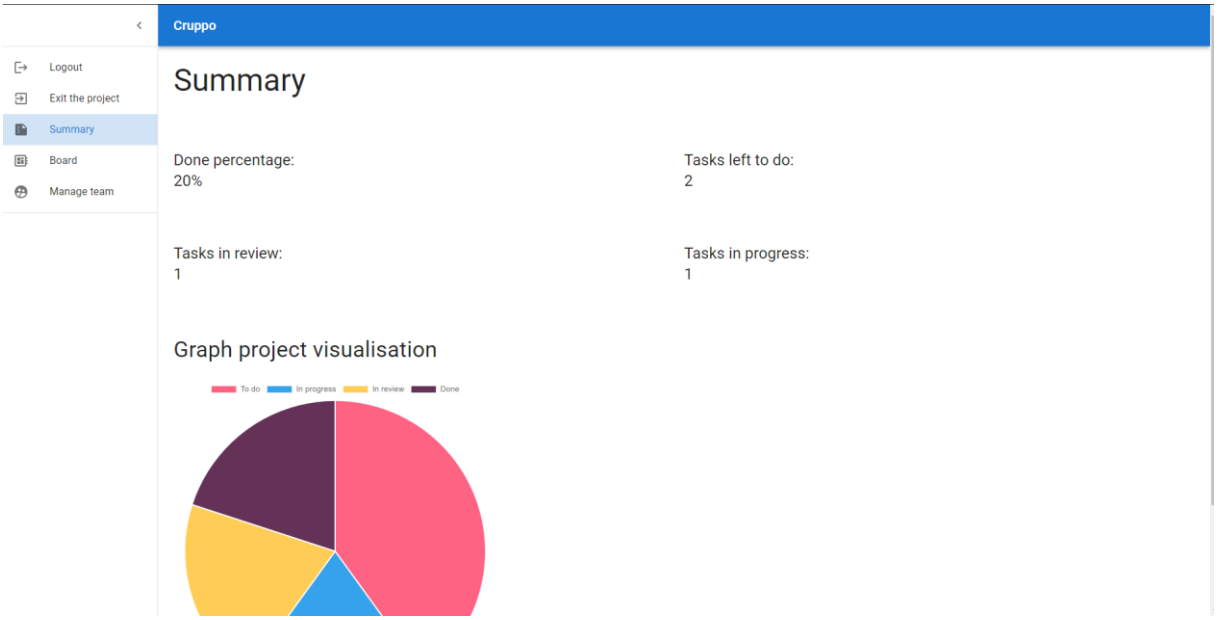
Kod pogleda kreacije zadatka, korisnik je dužan popuniti naslov zadatka, tip zadatka, opis ako želi, te prioritet i opcionalno rok do kojeg se zadatak treba izvršiti.



Slika 34. Pogled kreacije zadatka

6.4.12. Pogled sažetka projekta

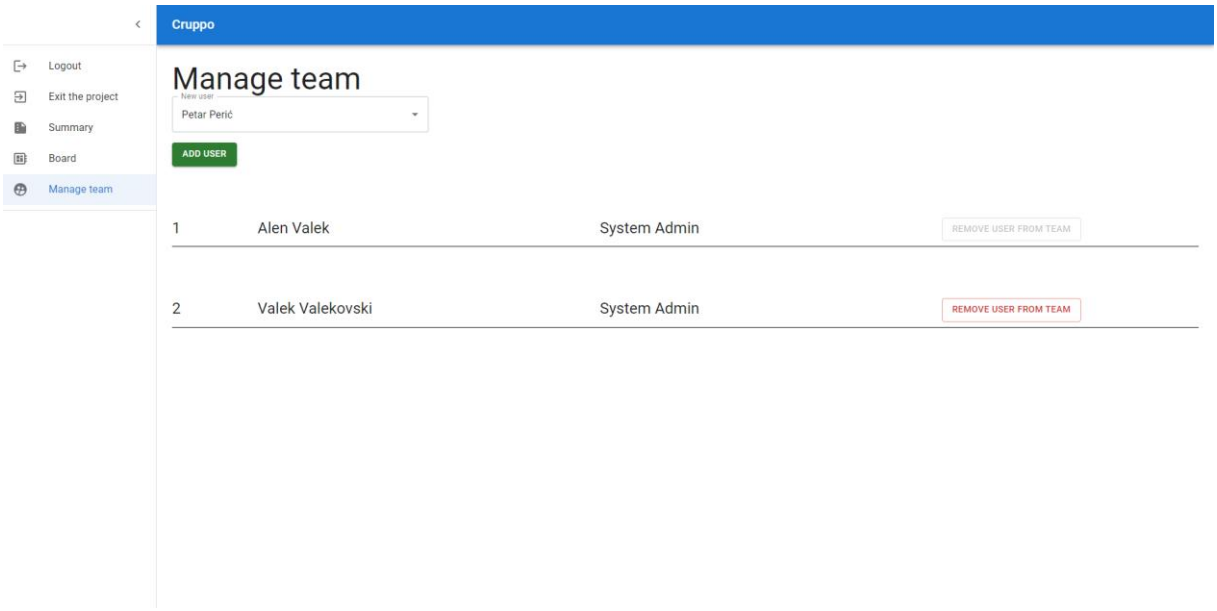
Svaki projekt ima svoj jedinstven sažetak. Sažetak je generiran iznova svaki put kad se učita ruta na kojoj se prikazuje. Generiran je pomoću informacija o zadacima u trenutnom projektu. Za pregled ovog pogleda nisu potrebne elevirane ovlasti. Kod ovog pogleda može se vidjeti koliki je postotak projekta gotov, koliko zadataka još ima za napraviti, koliko ih čeka pregled te na koliko ih se trenutno radi. Postotak završetka projekta se kalkulira dinamično prema ukupnom broju zadataka i broju zadataka koji su označeni kao završeni. Na kraju svih podataka i vizualno je prikazan dinamičan graf koji svaki stupac stavlja kao zasebnu stavku.



Slika 35. Pogled sažetka projekta

6.4.13. Pogled za upravljanje timom projekta

Pogled za upravljanje timom je dostupan vođi projekta. U ovom pogledu vođa ima pristup jednostavnoj kontrolnoj ploči gdje može odabrati novog korisnika koji već nije u timu te ga dodati u tim. Korisnik također može početi upisivati ime korisnika kojeg traži kako bi ga lakše pronašao. Kad ga odabere, vođa tima pritišće na gumb za dodavanje korisnika u tim.



Slika 36. Pogled za upravljanje timom projekta

7. Zaključak

Razvoj web aplikacije koja omogućava upravljanje timovima otvara vrata mnogim mogućnostima proširenja samog sustava. Tim je samo jedan od puno malih dijelova koji čine jednu funkcionalnu cjelinu. S tim na umu ova aplikacija je ukomponirala dinamičke pozicije na poslu, kreaciju projekata, zasebno kreiranje timova za svaki projekt, dinamičke ploče i sažetke svakog projekta te dodavanje novih korisnika odnosno zaposlenika unutar sustava. Kad bi se u sklopu ovog projekta bavilo samo i isključivo samo upravljanjem timovima, aplikacija bi bila vrlo jednostavna i ne bi riješila nikakav stvaran problem. U sklopu same aplikacije dolaze i različite razine pristupa koje su za sad statičke no služe svrsi. Sama hijerarhija razina pristupa je vrlo jednostavna u srži, ali je također vrlo moćna kod ograničavanja funkcionalnosti. Razine koje su trenutno u opticaju su razina administratora, razina ljudskih resursa, razina vlasnika i razina zaposlenika. Najviše ovlasti imaju administratori i vlasnik te se smatraju super korisnicima. Ljudski resursi su drugi po razini ovlasti, dok zaposlenici imaju pristup samo generičkim dokumentima i ovisno o poziciji privilegiju kreiranja projekata kroz koje će voditi svoje timove. S obziromna to da je ovo prva verzija aplikacije možda ne izgleda primamljivo poput nekih aplikacija koje se razvijaju godinama. Planirano je poboljšanje aplikacije u bliskoj budućnosti. Uz nadolazeća poboljšanja, planira se voditi politikom otvorenog koda te će biti dostupna svima koji je žele probati. Jedno od bitnijih ažuriranja koje su potrebne aplikaciji je povećanje razine interakcije unutar projekata, a pod tim se ciljano misli na timsku interakciju. S obzirom na to da se fokus vrši na interakciju timova, bilo bi dobro da sama aplikacija sadrži i mjesto gdje kolege mogu razmijeniti koju poruku bez da napuštaju ekosustav aplikacije. Aplikacija također treba popravak nekih nedosljednosti i optimizacija. Bez obzira na sve nedostatke, aplikacija je dizajnirana s ciljem proširenja te kod dozvoljava brzo dodavanje novih značajki.

Aplikacija je podijeljena na dva GitHub repozitorija. Prvi repozitorij je repozitorij web servisa koji se nalazi na adresi https://github.com/alenvalek/cruppo-backend te web aplikacija na adresi https://github.com/alenvalek/cruppo-frontend. Sve značajke i upute za instalaciju aplikacije lokalno na računalu se nalaze u *README.md* datoteci oba direktorija. Demonstracija aplikacije ja također dostupna na adresi

https://cruppo.netlify.app/, a podaci za prijavu demo računom su također dostupni u README.md datoteci na github-u.

Literatura

1. Node js. (n.d.). *Documentation*. Node.Js. Retrieved August 24, 2022, Dostupno na: https://nodejs.org/en/docs/ [Pristupljeno 22.08.2022]

2. Express.js (n.d.). - *Node.js web application framework*. Express JS. Dostupno na: http://expressjs.com [Pristupljeno 22.08.2022]

3. Myers, B. A., & Stylos, J. (2016). Improving API usability. *Communications of the ACM*, *59*(6), 62–69. Dostupno na: https://doi.org/10.1145/2896587 [Pristupljeno 22.08.2022]

4. Myers, B. A., & Stylos, J. (2016b). Improving API usability. Communications of the ACM, 59(6), 62–69. Dostupno na: https://doi.org/10.1145/2896587 [Pristupljeno 22.08.2022]

5. Redux (n.d.). - A predictable state container for JavaScript apps. | Redux. Redux. Dostupno na: https://redux.js.org/ [Pristupljeno 22.08.2022]

6. MUI (n.d.). MUI: The React component library you always wanted. Dostupno na: https://mui.com/ [Pristupljeno 22.08.2022]

7. Axios (n.d.) | Axios Docs. Axios. https://axios-http.com/docs/intro

8. Ahmad, M. O., Dennehy, D., Conboy, K., & Oivo, M. (2018). Kanban in software engineering: A systematic mapping study. *Journal of Systems and Software*, *137*, 96–113. Dostupno na: https://doi.org/10.1016/j.jss.2017.11.045 [Pristupljeno 22.08.2022]

Popis slika

Sažetak

Mnogi moderni uredi informatičara zahtijevaju prilagođene programske proizvode za komunikaciju i razmjenu podataka o projektu s kolegama unutar određenog tima. Prisutne solucije su u redu, no obuhvaćaju pre specifičnu nišu unutar cijelog jednog poduzeća te nemaju mogućnost integracije više od jednog odjela u istom sustavu, iako bi svi beneficirali od toga. Ovaj rad se fokusira na tematiku upravljanja timova informatičara unutar poduzeća odnosno ureda, ali ujedno se fokusira i na druge odjele i druge funkcionalnosti koje nisu podržane na platformama sličnog tipa. Cilj je bio zbližiti i zadržati što više funkcionalnosti unutar istog eko sistema. Kroz ovaj rad se raspravlja o glavnim značajkama web servisa i načinima integracije funkcionalnosti istih. Diskutira se o prednostima i manama trenutne implementacije te razlozima iza tih odabira kroz cijeli rad. Na kraju se sve spaja opisom i vizualnim prikazima web aplikacije, njezinim značajkama i metodama povezivanja s web servisom.

Ključne riječi: web aplikacija, web servis, upravljanje timovima, upravljanje projektima, upravljanje zaposlenicima, React.js, Node.js, Express.js, MongoDB

Abstract

A lot of modern IT offices require custom software for communication and data exchange about projects between colleagues in the same team. The existing solutions are fine, but they cover a rather specific niche within an entire company and they do not have the posibility of integrating more than one department in the same system, although everyone would benefit from it. This paper focuses on the topic of managing teams of IT professionals within companies or offices, but at the same time it focuses on the other departments and other functionalities that are not supported on platforms of a similar type. The goal was to bring together and keep as much funcionality as possible within the same ecosystem. The advantages and disadvantages of the current implementation and the reasons behind these choices are disscussed throughout the paper. In the end, everything comes together with a description and visual representations of the web application, its features and methods of connecting to the web service.

Keywords: web application, web service, team management, project management, employee management, React.js, Node.js, Express.js, MongoDB