Razvoj web aplikacije za upravljanje projektima

Semjaniv, Nikola

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

NIKOLA SEMJANIV

RAZVOJ WEB APLIKACIJE ZA UPRAVLJANJE PROJEKTIMA

ZAVRŠNI RAD

Pula, rujan, 2020.

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

NIKOLA SEMJANIV

RAZVOJ WEB APLIKACIJE ZA UPRAVLJANJE PROJEKTIMA

ZAVRŠNI RAD

JMBAG: 0303069708, redoviti student
Studijski smjer: Informatika

Predmet: Programsko inženjerstvo
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: izv. prof. dr. sc Tihomir Orehovački

Pula, rujan, 2020.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani _____, kandidat za prvostupnika _____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 nedozvoljen 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

_____

U Puli, _____, _____ godine

IZJAVA
o korištenju autorskog djela

Ja, _____ dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom

_____

_____

_____ 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, _____ (datum)

Potpis

_____

Sadržaj

1. Uvod

Strukture i sredstva upravljanja projektima iznimno su različite. Njihov izbor ovisi o području, vrsti, veličini i kompleksnosti projekta. Projekt može obuhvaćati jednu ili nekoliko tisuća osoba. Iz toga proizlazi da struktura upravljanja projektima može odgovarati jednostavnoj listi zadataka, ali i kompleksnoj organizaciji poduzeća čija je svrha izvođenje projekta uz podršku projektnog programa. Proces upravljanja projektom ili projektima iznimno je zahtjevan, stresan i iscrpljujući posao. Pogotovo ako osoba koja vodi tim ili čak više timova i više različitih projekata nije dobro organizirana. Međutim kada je sve transparentno, pregledno i organizirano sav taj stres i umor nestaje, jer se jasno može vidjeti u kojem stanju je proces i s obzirom na to koordinirati buduće korake. Motivacija za izradu aplikacije za upravljanje projektima proizašla je upravo iz tog shvaćanja da je ponekad potrebno omogućiti ljudima lakše i jednostavnije obavljanje posla, odnosno upravljanje projektima. Naročito je na to utjecala situacija u kojoj se svijet našao ove godine (Covid-19), tj. primoranost da veliki broj ljudi rade od kuće. Kako se dio tih ljudi bavi raznim projektima vezanih npr. uz informatiku, marketing i slično, tema ovog rada je razvoj web aplikacija za upravljanje projektima. Kako je tehnologija danas mnogo uznapredovala tako se i zahtjevi za web aplikacijama povećavaju. Aplikacije su složenije i veće, rješavaju kompliciranije probleme, poneke rješavaju i više probleme što je puno praktičnije, a postaje i sve učestalije. Jedna od ogromnih prednosti i svojstvo koje današnje web aplikacije moraju imati kako bi bile uspješne je istovremena komunikacija između poslužitelja i klijenta što omogućuje praktički automatsko ažuriranje podataka. Nema nepotrebnih prekomjernih slanja upita prema poslužitelju o promjeni stanja podataka, već ukoliko dođe do promjene klijent je odmah obaviješten te vrši određene funkcije sukladno s time. Tako da današnja trenutna razina na kojoj se nalaze web aplikacije ne samo da uključuje više mogućnosti, bolji dizajn sučelja, nego i istovremenu komunikaciju među ljudima koji su odvojeni kilometrima.

2. Istraživanje tržišta

Internet tržište se sve više puni web aplikacijama. Svaka je drugačija, ali mnoge su iste u smislu problema kojeg rješavaju. Postoje mnoge web aplikacije namijenjene organizaciji i upravljanju ili vođenju firme pa tako i projekata, unutar ili van firme. Aplikacije koje su već na tržištu i vezane su za upravljanje projektima imaju mogućnost od nekoliko dana pa do dva tjedna besplatnog korištenja. Nakon toga, ako je osoba zadovoljna može uključiti pretplatu te nastaviti koristiti aplikaciju. Aplikacije koje se ističu su:

Bitrix24 – platforma koja je tu kako bi pojednostavila upravljanje u velikim firmama s puno zaposlenika. Platforma se razlikuje od ove web aplikacije po tome sto uključuje puno više funkcionalnosti osim upravljanja projektima, a to su: CRM, komunikacija, marketing, izrada izvješća i mogućnost izrade web stranica u samoj platformi. Međutim, ono što je bitno napomenuti kod alata za zadatke i projekte, osim mogućnosti upravljanja grupnim zadacima kroz kanban, još je prisutan kalendarski prikaz zadataka, komunikacija između članova grupe kroz poruke ili video pozive te izrada izvještaja. Bitrix24 je na puno višoj razini te nudi puno više mogućnosti od Project Workbook aplikacije (Bitrix24, 2020).

Asana – mobilna i web aplikacija čija je namjena ista kao i kod Project Workbook aplikacije, upravljanje timom, zadacima i projektima. Može se upravljati kroz tri različita pregleda. Pregled kroz listu, što je na neki način slično tabličnom pregledu voditelja tima u Project Workbook aplikaciji, kroz vremensku listu, odnosno kalendar i zadnje kroz kanban. Osim toga, tu je još i moguće pogledati detaljno izvješće projekta što uz kalendarski prikaz nije moguće u web aplikaciji iz ovog rada. Jedna stvar još za izdvojiti je ta da svaki član grupe u Asani je na istoj razini, dok je kod Project Workbook aplikacije to podijeljeno u dvije razine, razina voditelj i razina članova tima (Asana, 2008).

OpenProject – *Open-Source* mobilna i web aplikacija koja rješava upravljanje projektima na izrazito detaljan način. Kombinira prikaz liste i kalendara u jedno gdje su tekstualno i vremenski opisani svi zadaci. Aplikacija pruža mogućnost prikaza statistike kroz graf, projektnog plana, korištenih resursa, zadataka dodijeljenih korisniku, zadataka kreiranih od strane korisnika, korisničke aktivnosti te sažetka događanja. Dosta je opširna

aplikacija te ide u dubinu kako bi se što lakše i transparentnije proveo projekt. U usporedbi s time Project Workbook aplikacija se dosta orijentira na upravljanje voditelja tima kroz interaktivni tablični prikaz tako da OpenProject aplikacija kao takva nudi puno više funkcionalnosti na samome početku.

To su neke od današnjih web aplikacija za upravljanje projektima koje se nalaze na visokom mjestu na tržištu. Sve one dijele iste osnovne funkcionalnosti s Project Workbook aplikacijom koja je još uvijek mlada, dok su se ostale imale vremena razvijati i prilagođavati tržištu. Bitno je napomenuti da je moderno i jednostavno korisničko sučelje vrlo važan, ako ne i jedan od najvažnijih čimbenika u uspješnosti aplikacije. Aplikacije poput Bitrix24 i Asana-e imaju jako dobro prilagođeno korisničko sučelje, dok druge imaju pomalo zastarjelo. Project Workbook aplikacija je na dobrom putu te ima vremena za razvoj u funkcionalnome smislu, dok u grafičkom je vrlo blizu modernom i jednostavnom.

3. Korišteni alati

Prije same implementacije aplikacije potrebno je odabrati odgovarajuće alate. Pod alate spadaju stvari kao što je alat za uređivanje teksta, odnosno razvojno okruženje[1] kojih danas postoji mnogo. Kako besplatnih koji u najmanju ruku služe osnovnim stvarima, prvenstveno uređivanju teksta kod implementacije, tako postoji i oni koji se plaćaju, jer u njihovom opsegu se nalazi puno više mogućnosti od čistog uređivanja teksta. Osim alata za uređivanje teksta koji čine najmanju o nikakvu ulogu u implementaciji, potrebno je odabrati odgovarajući jezik u kojem će se pisati kod. Dodatno, ukoliko postoji aplikacijski okvir za odabrani jezik, pogodno je koristiti ga ili odabrati najbolji ukoliko ih je više. Mnogi aplikacijski okviri se razlikuju u svom opsegu funkcionalnosti te samoj veličini paketa. Zato je bitno prije same implementacije i odabira paketa odrediti što je bitnije, brzina aplikacije ili njene funkcionalnosti.

Kod izrade ove aplikacije korišteni su sljedeći alati i paketi:

[1] IDE – Integrated development environment

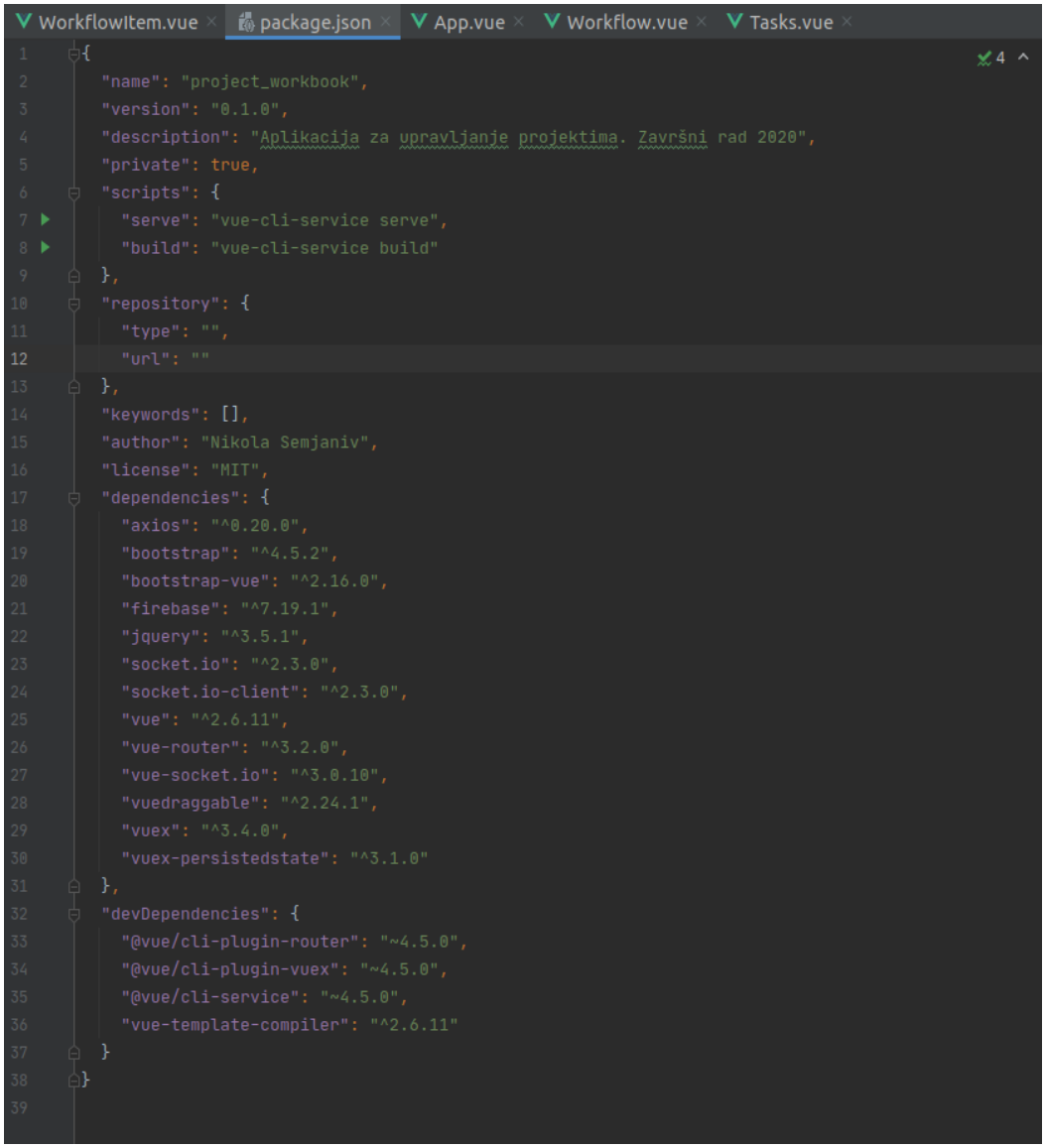
- Node.js – *Open-Source Cross-Platform JavaScript runtime* okruženje koje izvodi *javascript* kod izvan web pretraživača. *Node.js* je preduvjet za korištenje npm-a[2] pomoću kojeg se instaliraju potrebni paketi za razvoj aplikacije.
- Vue.js – *Open-Source Progressive JavaScript Framework*, znači aplikacijski okvir u *javascript* jeziku. Njegova prednost u odnosu na ostale aplikacijske okvire *javascript* obitelji je u njegovoj veličini i brzini izvođenja. Veličinom je najmanji te u kombinaciji s brzinom izvođenja se koristi kod izrade korisničkog sučelja i *single-page* aplikacija.

- Flask – *Micro Web Python Framewor*. Klasificiran je kao mikro aplikacijski okvir napisan u jeziku *python*, jer ne zahtijeva nikakve posebne alate ili pakete. U ovom slučaju korišten je kao *RESTful API*, odnosno predstavlja standardni dizajn izrade *server-side* trenutne aplikacije.
- DB Browser For SQLite – visoko kvalitetni *Open'source* alat za dizajniranje, izradu i uređivanje *SQL* baze podataka.
- Realtime Databese – *NoSQL* baza podataka. Na *Google-ovoj Firebase* platformi koja pomaže u brzom razvoju visoko kvalitetnih aplikacija, ponuđena je opcija spremanja podataka u *NoSQL* bazu koja je korištena u izradi ove aplikacije (Firebase platforma).
- WebStorm - snažno razvojno okruženje za moderni razvoj *javascript* aplikacija. Pruža potpunu podršku za *javascript, typescript, HTML, CSS* kao i za aplikacijske okvire *React, Angular* i *Vue.js* bez ikakve dodatne instalacije paketa.

Osim navedenih alata, korišteni su i određeni paketi zbog funkcionalnosti koje omogućuju. Kod *Vue.js* i *Flask* aplikacijskih okvira korišteni su drugačiji paketi. Kad se govori o *Vue.js* korišteni su paketi za lakše upravljanje klasama za uređivanje *HTML* elemenata (*bootstrap* i *bootstrap-vue*), za razmjenu podataka preko HTTP ili XMLHttpRequests zahtjeva (*axios*), za korištenje Google-ove Firebase Realtime baze podataka (*firebase*), za korištenje web servisa u svrhu izgradnje *real-time* aplikacije (*socket.io*, *vue-socket.io, socket.io-client*), za mogućnost izgradnje *drag and drop*

[2] Node Package Manager – menadžer za JavaScript programski jezik. Koristi se kod instaliranja paketa

elemenata (*vuedraggable*), za ruter za izradu SPA[3] (*vue-router*), za korištenje globalnih varijabli dostupnih kroz cijeli projekt te za privremeno, lokalno spremanje varijabli u pretraživaču (*vuex, vuex-persistedstate*) kako je navedeno u package.json dokumentu, što se može vidjeti na slici 1.



Slika 1. Datoteka package.json

[3] Single Page Aplications

Dok su u *Flask*-u korišteni paketi prvobitno za Flask aplikacijski okvir, *CORS*[4] pakete zadužene za provjeru komunikacije s korisničkim sučeljem, dobivanje trenutnog datuma i vremena, za operacije nad *SQLite* bazom podataka (sqlite3) te paket za dobivanje nasumične vrijednosti korištene iz *id* objekta (uuid[5]).

4. Programsko rješenje i implementacija

4.1. Kreiranje projekta

Prvi korak u implementaciji je instalacija *Vue-CLI* paketa za aplikacijski okvir preko komande sa slike 2, nakon toga kreira se novi *Vue* projekt pod nazivom Project Workbook, odnosno „project_workbook“, što prikazuje slika 3. Zatim dodajemo osnovne *Vue* pakete *Vue Store,* odnosno *vuex* zajedno s dodatnim opcionalnim paketom *vuex-persistedstate* i *vue-router*, kako je prikazano na slikama 4, 5 i 6 (Vue CLI, 2018). Nakon toga slobodno je dodavati pakete proizvoljno. Slike 7, 8, 9 i 10 prikazuju komande za instalaciju dodatnih paketa među kojima je i *vue-socket.io* koji je instalirane prema uputama iz literature (Vue-socket.io, 2016). Tako smo s time završili kreiranje projekta aplikacije za korisničko sučelje.

```
envy@Envy:~/Documents/Zavrsni$ npm install -g @vue/cli
```

Slika 2. Instalacija Vue-CLI paketa

```
envy@Envy:~/Documents/Zavrsni$ vue create project_workbook
```

Slika 3. Kreiranje novog projekta

```
envy@Envy:~/Documents/Zavrsni$ vue add vuex
```

Slika 4. Instalacija paketa vuex

[4] Cross-Origin Resource Sharing
[5] Universally unique identifier

```
envy@Envy:~/Documents/Zavrsni$ npm install --save vuex-persistedstate
```

Slika 5. Instalacija paketa vuex-persistedstate

```
envy@Envy:~/Documents/Zavrsni$ vue add vue-router
```

Slika 6. Instalacija paketa vue-router

```
envy@Envy:~/Documents/Zavrsni$ npm install --save bootstrap bootstrap-vue
```

Slika 7. Instalacija bootstrap paketa

```
envy@Envy:~/Documents/Zavrsni$ npm install --save axios
```

Slika 8. Instalacija axios paketa

```
envy@Envy:~/Documents/Zavrsni$ npm install --save firebase
```

Slika 9. Instalacija firebase paketa

```
envy@Envy:~/Documents/Zavrsni$ npm install --save vue-socket.io
```

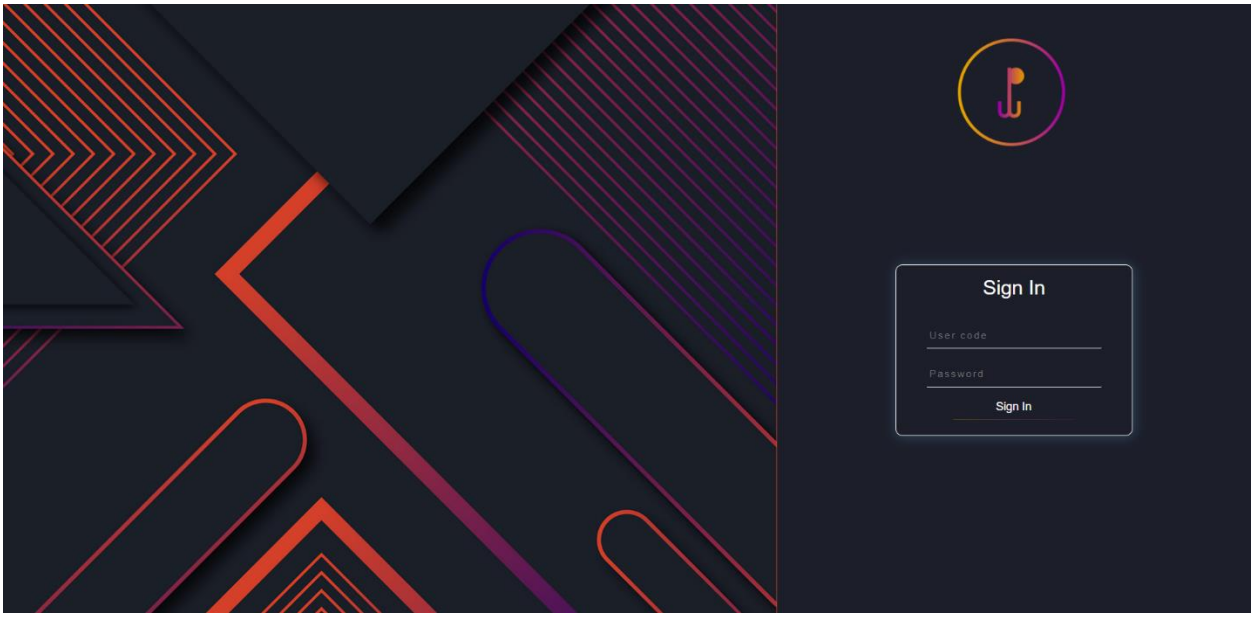
Slika 10. Instalacija socket.io paketa

Sljedeći korak je postaviti *server-side* dio aplikacije te *websocket* na način da se napravi novi direktorij posebno za tu svrhu, a u njega dva nova direktorija: websocket i api direktorij. U api direktorij se dodaju *python* datoteke „__init__.py“ gdje se inicijalizira app varijabla koja predstavlja flask server te se dodaje *CORS* u aplikaciju i prelazi se na sljedeću datoteku, a to je „routes.py“ gdje su postavljene aplikacijske rute koje se ponašaju kao *URL* rute koje upravljaju zahtjevima, poslanim prema njima na određeni način i prema određenim metodama. Zadnja datoteka ima svrhu odrađivanja operacija nad bazom podataka u smislu spremanja, pretraživanja i dohvaćanja podataka i naziva se „db_connection.py“. Na razini api direktorija se nalazi još jedna *python* datoteka koja predstavlja osnovu u pokretanju flask servera, „run.py“. U tu datoteku se uključuje app varijabla prethodno inicijalizirana u „__init__.py“ datoteci te se pokreće server samim pokretanjem te datoteke kroz *python* komandu „python run.py“. Osim toga, na istoj razini api direktorija i „run.py“ datoteke nalazi se i websocket direktorij koji sadrži „websocket.py“ datoteku u kojoj će se kasnije inicijalizirati rute za komunikaciju. Nakon toga slobodno je pokrenuti *Vue.js* aplikaciju komandom „npm run serve“, otvoriti *localhost* u pretraživaču na portu 8080 gdje je otvorena prvobitna vue aplikacija.

4.2. Stvaranje korisničkog sučelja

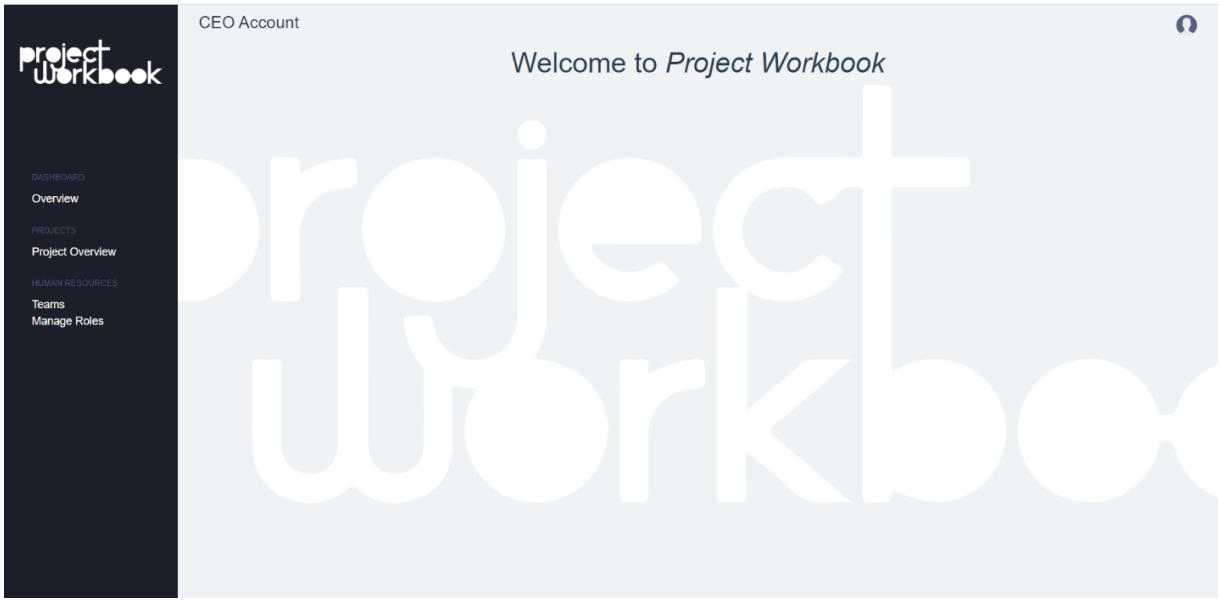
Prije samog početka potrebno je podsjetiti se da će biti tri razine pristupa aplikaciji. Administratorska razina koja ima pristup za dodavanje novog korisnika i upravljanje dopuštenjima, dodavanje novog tima koji će biti zadužen za određeni projekt te automatski dodavanje slobodnih članova u tim. Posljednje, stvaranje novog projekta i ubacivanje osnovnih informacija o projektu kao i dodavanje tima zaduženog za projekt. Sljedeća razina se odnosi na voditelje tima koji će moći uređivati i voditi interakciju s članovima tima kroz interaktivni tablični prikaz zadataka, pod zadataka i njihovih statusa. Najniža razina pristupa je developerska razina gdje developer vidi zadatke svog tima na projektu na kojem rade te po izvršenju zadatka ih pomakne na predodređeni stupac u interaktivnoj tablici gdje je taj zadatak onda spreman za pregled. Još jedna bitna stvar je ta da će se interaktivna tablica u developerskom pregledu i pregledu voditelja tima bitno razlikovati u izgledu i funkcionalnostima.

Kod stvaranja korisničkog sučelja bitno je naglasiti da se radi redom, odnosno kako bi se korisnik kretao aplikacijom. Imajući na umu tijek kojim se korisnik koristi aplikacijom prva stvar na listi je izrada sučelja za prijavu. To se rješava na način da se stvori nova *vue* komponenta koja se naziva *SignIn.vue*. Na kraju izgleda kao što je prikazano na slici 13.



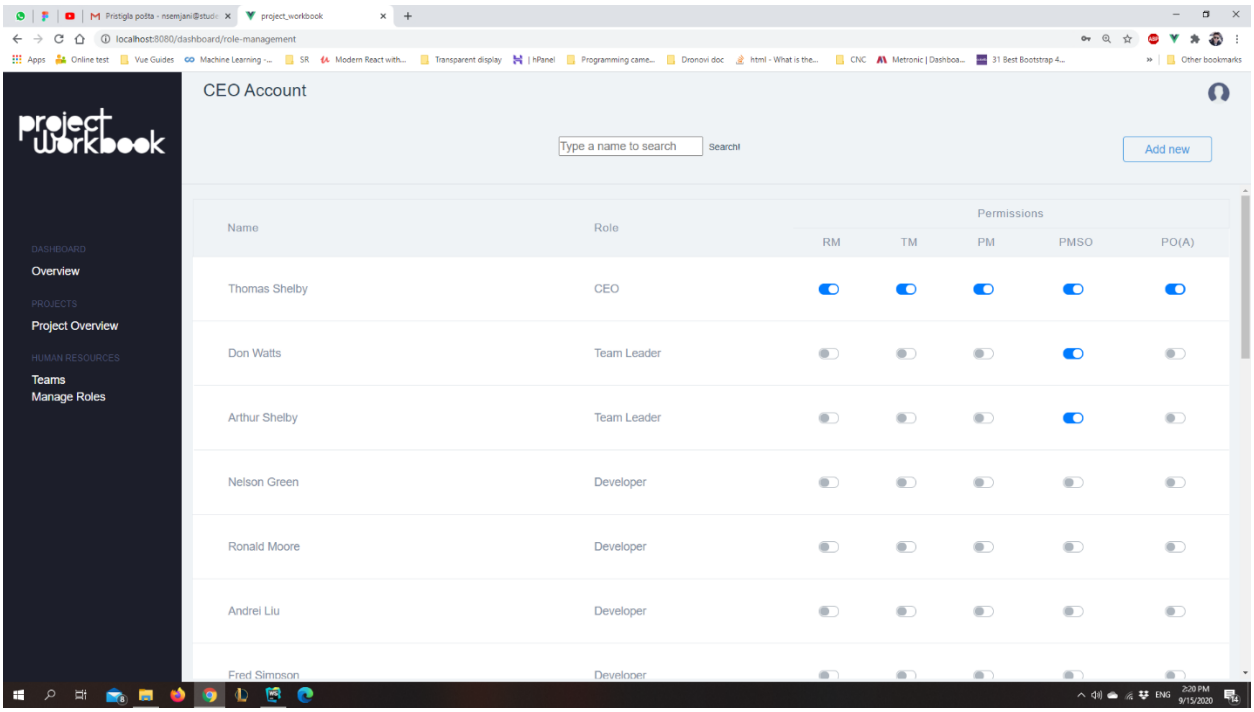
Slika 11. Korisničko sučelje za prijavu

Zatim dolazi na red sučelje koje će korisnik vidjeti kada se prijavi u aplikaciju, što se može vidjeti na slici 14. Aplikacija sadrži dvije glavne stavke, navigacijsku traku i prozor za pregled. Navigacijska traka se sastoji od dva dijela. Prvi dio koji je zadužen za navigaciju kroz aplikaciju te je položen okomito uz lijevi rub glavnog prozora, dok drugi dio prikazuje informaciju o razini pristupa aplikaciji te padajući izbornik na desnoj strani s opcijom za izlaz iz aplikacije što se vidi na slici dolje.



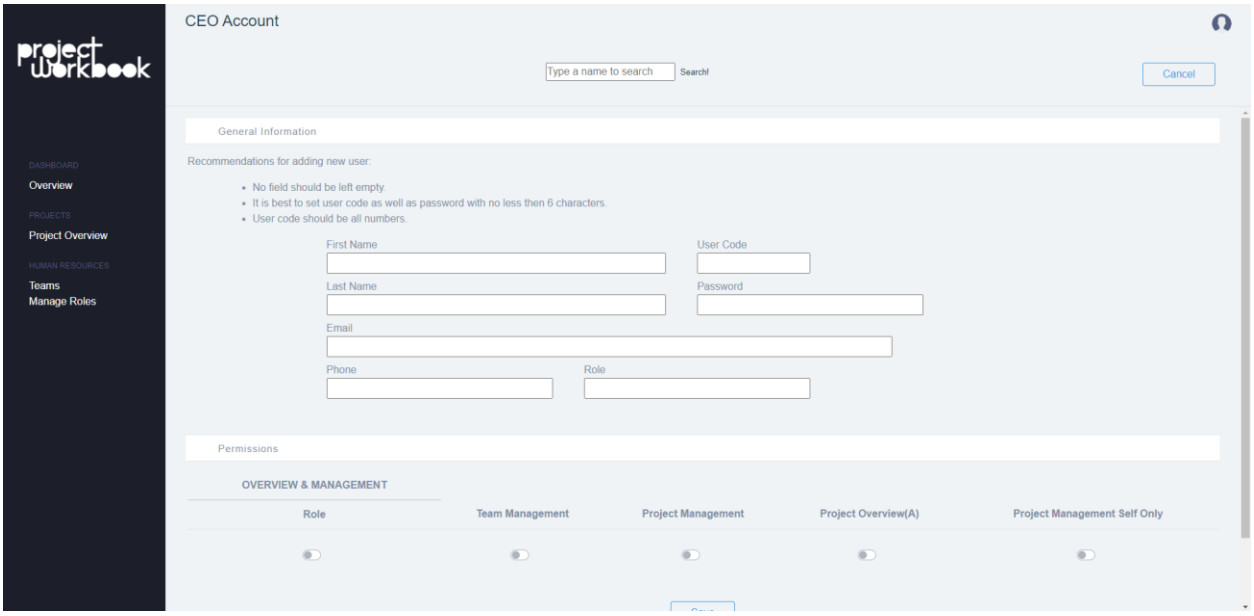
Slika 12. Početno korisničko sučelje unutar aplikacije

Slika 13 prikazuje sučelje za pregled svih aktivnih korisnika aplikacije karakteristično je samo za administratorsku razinu pristupa. Ovdje su funkcionalne mogućnosti osnove. Pregled korisnika, opcija za dodavanje novog te upravljanje korisničkim dopuštenjima.



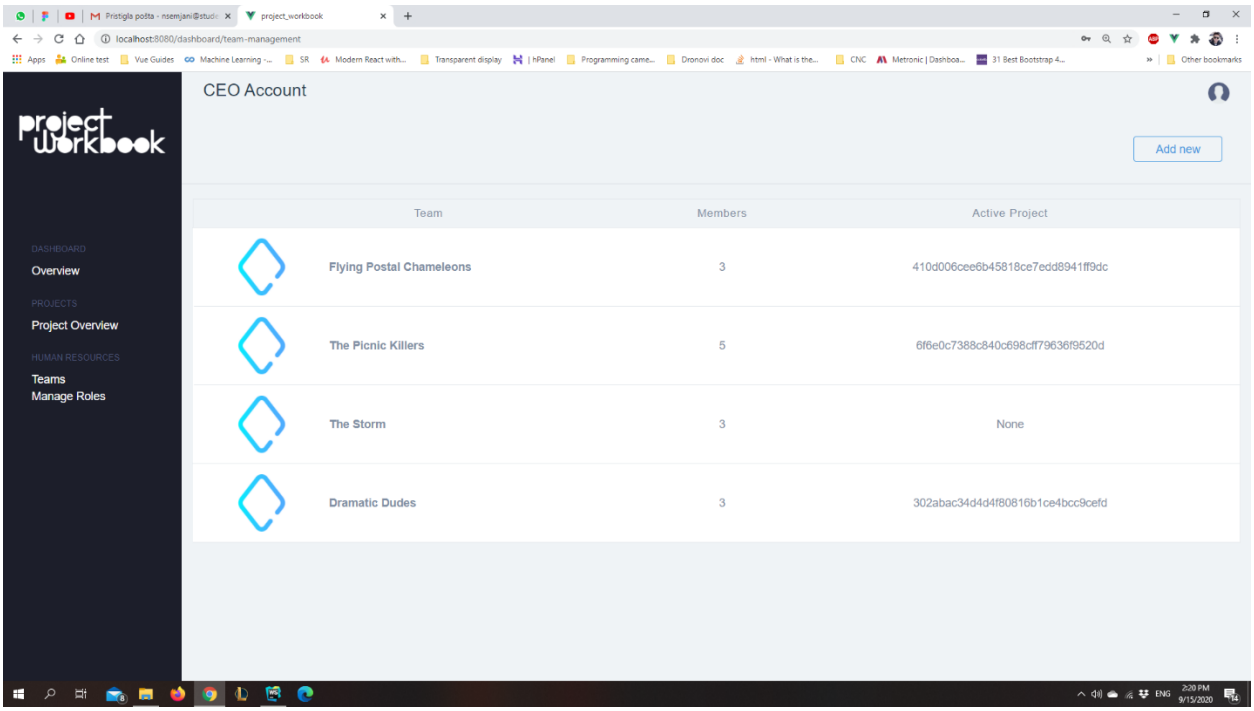
Slika 13. Prikaz svih korisnika aplikacije

Slika 14 prikazuje dodavanje novog korisnika nakon pritiska na gumb *Add new*. Otvara se novi pregled koji omogućuje unos informacija za novog korisnika. U informacije spadaju ime i prezime, korisnički kod i lozinka koji služe za prijavu, e-mail adresa, broj telefon i uloga u takozvanoj tvrtki ili timu. Nakon toga sljedeće je odabir dopuštenja: administratorsku razina dopuštenja, uloga voditelja tima ili developerska razina dopuštenja. Nakon toga se spremaju unesene informacije pritiskom na gumb za spremanje i novi korisnik je spreman za korištenje aplikacije u skladu s dopuštenjima.

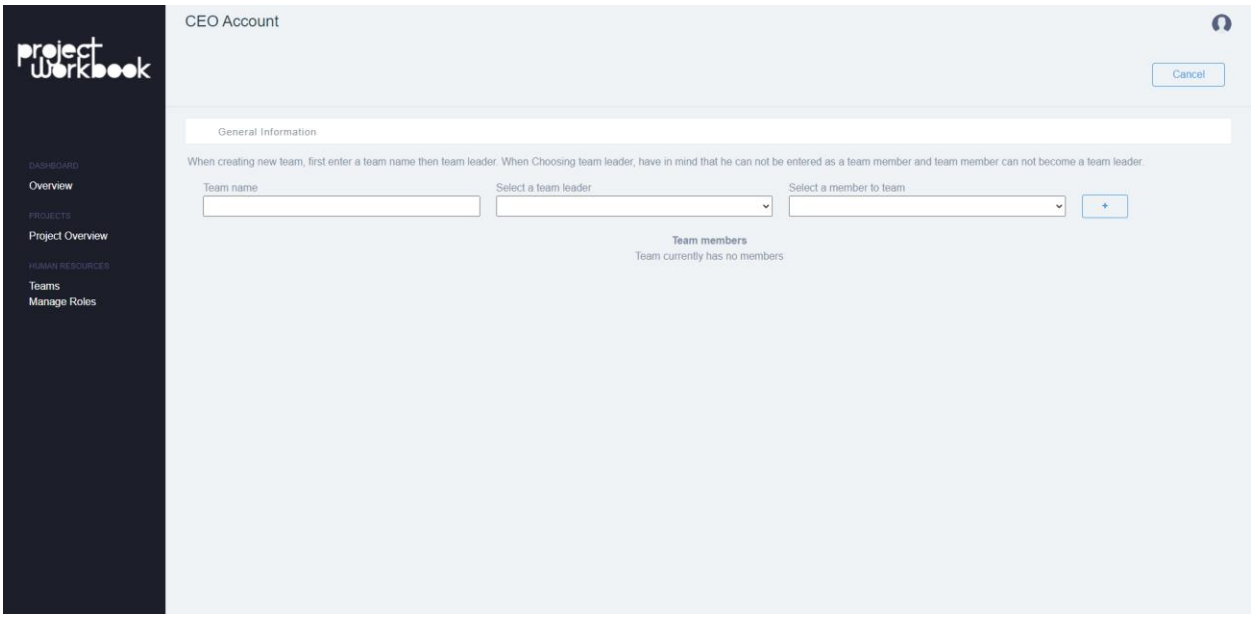


Slika 14. Prikaz dodavanja novog korisnika

Sučelje za prikaz timova, kako je vidljivo na slici 15, još jedna je od navedenih stvari koje spadaju u administratorske ovlasti. Osim što se mogu vidjeti osnovne informacije o timu (naziv, broj članova tima i aktivni projekt) pritiskom na opcije ili jednostavnije na red koji označava tim otvara se dodatan prikaz detaljnijih informacija o članovima tima. Dostupan je uvid u njihove informacije (ime i prezime, e-mail adresa i broj telefona) te opcija da se izbriše trenutni tim ukoliko je to potrebno. Nakon što se tim izbriše svi članovi tima postaju slobodni, odnosno bez tima te su im ovlasti na razini developera. Kao i kod prikaza svih korisnika, tako se i ovdje nalazi opciju za dodavanje novog tima. To se postiže pritiskom na gumb *Add new*. Nakon otvaranja prikaza za dodavanje novog tima , što prikazuje slika 16, predstavljene su tri stvari koje se mogu ispuniti podacima, a to su naziv tima, voditelj tima te članovi tima kojih naravno može biti više. Nakon unošenja, ako je sve ispravno, pritiskom na gumb za spremanje svi podaci se spremaju u bazu i tim je formiran.



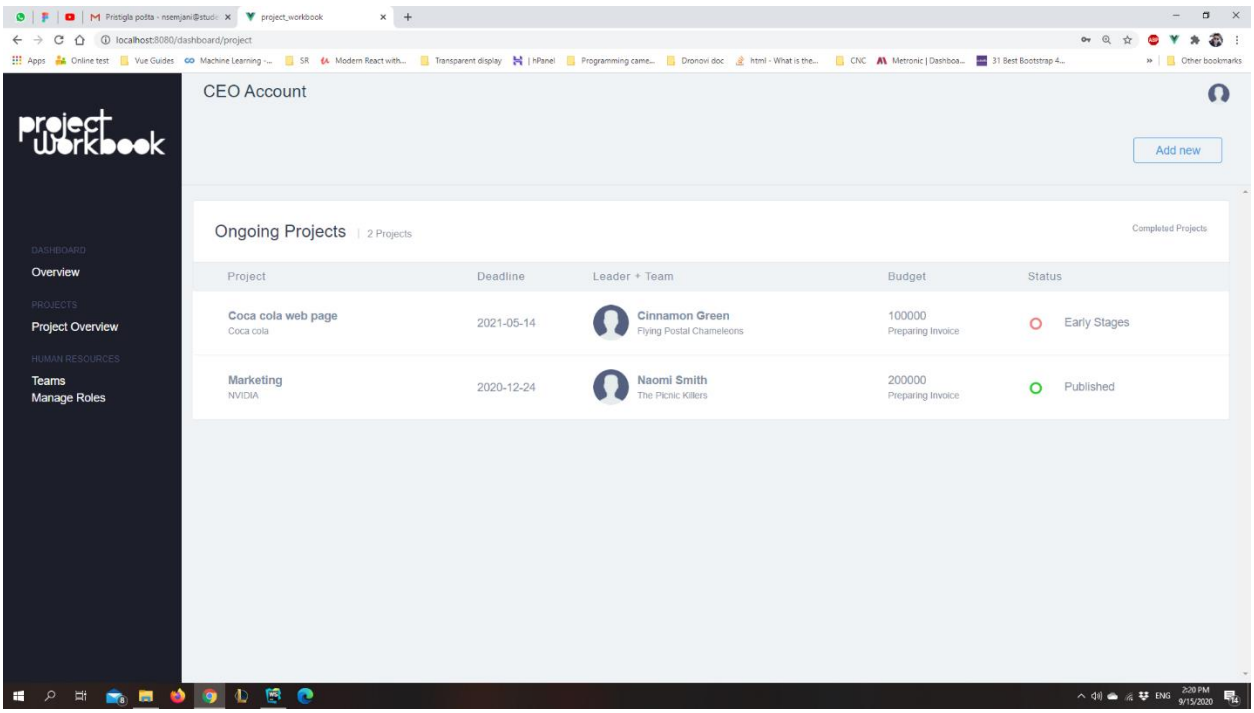
Slika 15. Pregled sučelja za prikaz liste timova



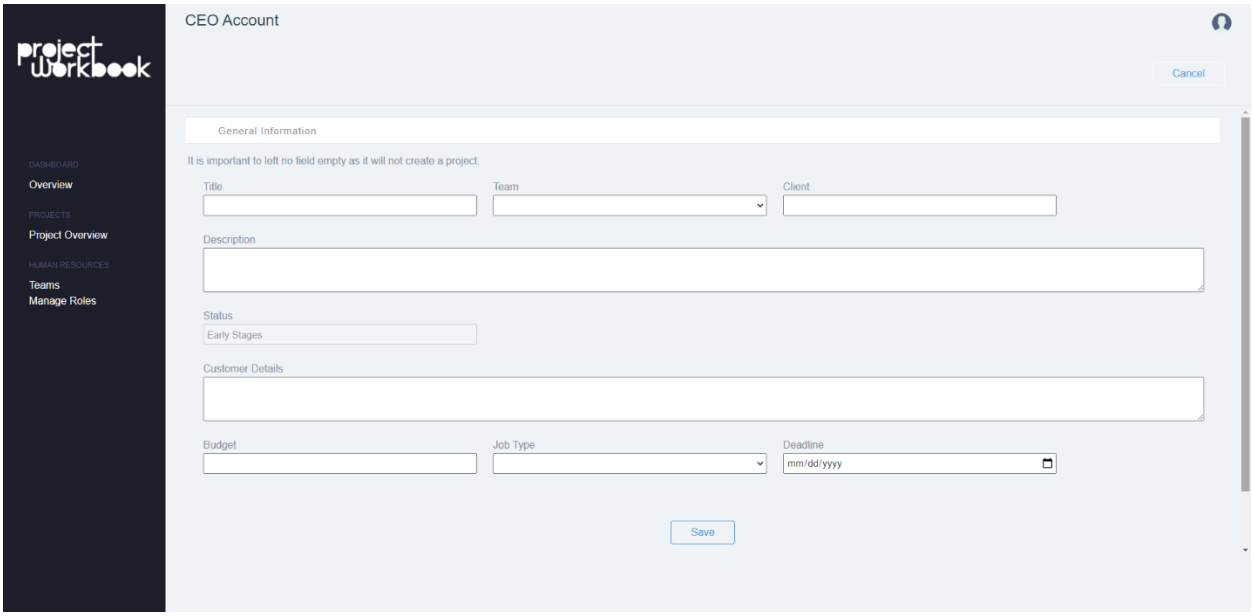
Slika 16. Korisničko sučelje za dodavanje novog tima

Slika 17 prikazuje pregled svih projekata projekata. Ovdje kao i u prethodnim prikazima su kroz listu prikazani svi aktivni projekti i njihove osnovne informacije kao što su naziv projekta i klijent, određeni krajnji rok, voditelj tima i naziv tima, budžet određen za projekt te trenutni status u kojem se projekt nalazi. Isto tako imamo i opciju za dodavanje novog

projekta. Pritiskom na gumb *Add new* otvara se novo sučelje kao što je vidljivo na slici 18 za dodavanje projekta. Kod dodavanja novog projekta prvo je potrebno unijeti osnovne informacije: naziv projekta, tim koji će biti zadužen za projekt, ime klijenta, opis, status i faza su unaprijed postavljeni, dodatne detalje koje je posebno klijent zatražio ukoliko ih ima, budžet, tip projekta, odnosno posla te posljednje u ovom djelu je odabir krajnjeg roka izvršenja projekta. Nakon osnovnih informacija slijedi unos dodatnih opcionalnih informacija, a one se dotiču faza u projektu. Ukoliko se administrator odluči unositi faze otvara se dodatni prikaz za unos informacija usko vezanih uz tu fazu. Nakon što je sve ispravno uneseno i pregledano pritiskom na gumb za spremanje podaci o projektu se spremaju u bazu te je svaki od članova tima, kao i sam voditelj slobodan ući u aplikaciju te krenuti s razvojem projekta.



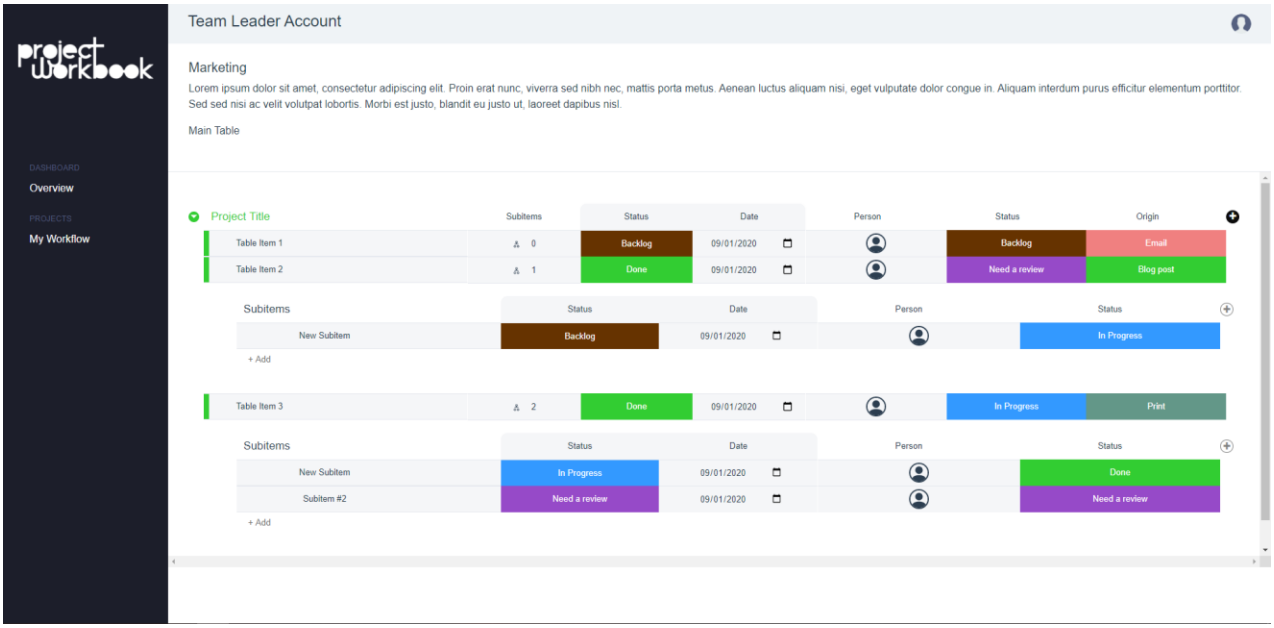
Slika 17. Pregled sučelja za prikaz liste projekata



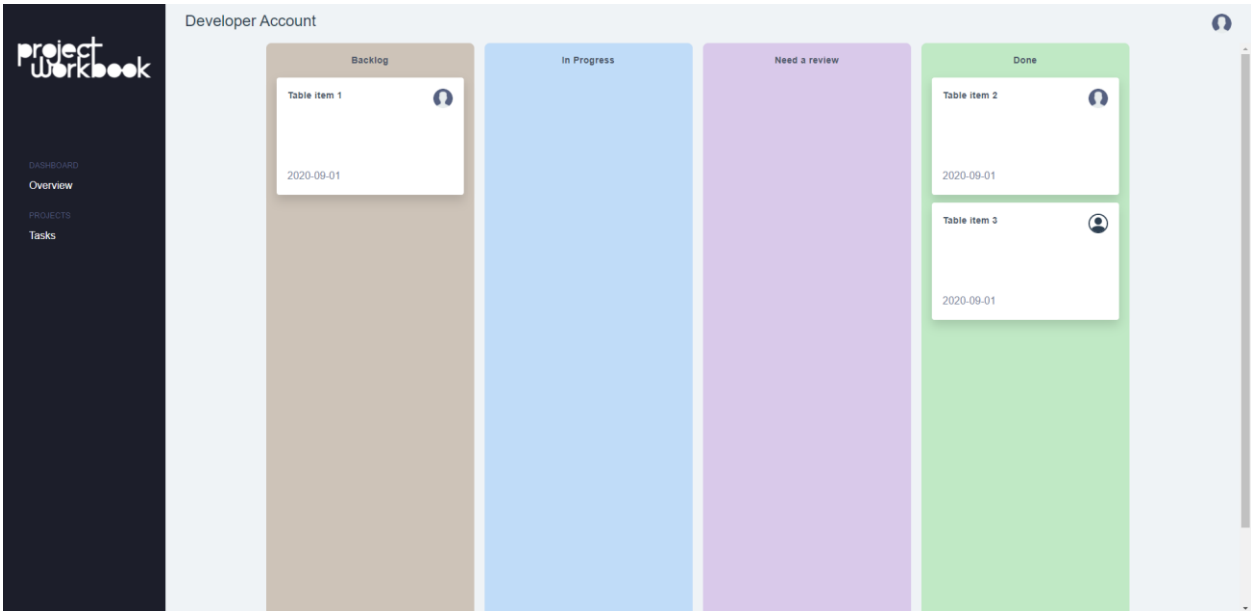
Slika 18. Korisničko sučelje za dodavanje novog projekta

Kada se voditelj tima priključi u aplikaciju glavna stvar oko koje se treba orijentirati je njegov *Workspace* kako je prikazano na slici 19, gdje je uključen trenutni projekt za koji je njegov tim zadužen. *Workspace* pruža mnoge mogućnosti za upravljanje projektom, zadacima u projektu i slično. Prvenstveno, velika je prednost interaktivni, bez limitni, tablični prikaz koji otvara mnoge dinamičke mogućnosti izgradnje zadanog koncepta. Tablica podataka se sastoji od stupaca i redaka što podsjeća na klasičan tablični prikaz, osim što se naslovi stupaca mogu mijenjati. Osim toga stupci se mogu dodavati po potrebi. Ponuđeno je šest različitih opcija stupaca, a to su status, datum, osoba zadužena za određeni dio, tip, kontekst i podrijetlo kao generalni tip sadržaja koji je određen trenutnim retkom u tablici. Status, sadržaj, prioritet i tip imaju dodatne opcije za odabir. Tako na primjer status ima opcije kao što su *backlog, in progress, need a review* i *done*, a sadržaj ima opcije *QA, backend, frontend, prototype, feature request* i *brainstorming*, dok za tip tu su *email, blog post, press release, web page/app, design* i *print* . Prioritet sadrži sljedeće opcije: *very low, low, normal, medium* i *high.* Datum, između ostalog, predstavlja samo datum odnosno rok za izvršenje zadatka, a osoba označava developera ili osobu zaduženu za taj dio zadatka. Osim što voditelj tima ima mogućnost dodavanja novih stupaca u tablicu točno koliko mu je potrebno, tako ima i mogućnost dodavanja novih redaka koji mogu predstavljati faze projekta ili određene zadatke. Dodatno, svaki

redak može imati unutarnje retke, odnosno pod zadatke koji mogu pomoći u lakšem postizanju glavnog rezultata. Bitno je za napomenuti da se unutarnji redci razlikuju od vanjskih po stupcima. Nije isključeno da mogu biti isti, naravno. Svih šest opcija za stupce ponuđeno je kod dodavanja novog stupca u unutarnjem retku. Još jedna stvar koja je bitna za naglasiti jest da se prilikom bilo kakvih statusnih, datumskih ili ostalih drugih promjena, mijenja i pregledna tablica zadataka developera. Odnosno mijenjaju se podaci zadataka na kojem je voditelj tima izvršio promjene. Ono što je *Workspace* voditelju tima, to je i *Workspace* developeru, kao na slici 20, samo manje interaktivniji i drugačije posložen. Zadaci su posloženi u stupce, a cijeli prikaz je posložen kao kanban. Bez obzira na povećanu jednostavnost interaktivnosti u pregledu ne nedostaje. Developer je sposoban zadatke premještati iz stupca u stupac te u samom procesu premještanja voditelj tima vidi statusne promjene zadatka, odnosno retka.

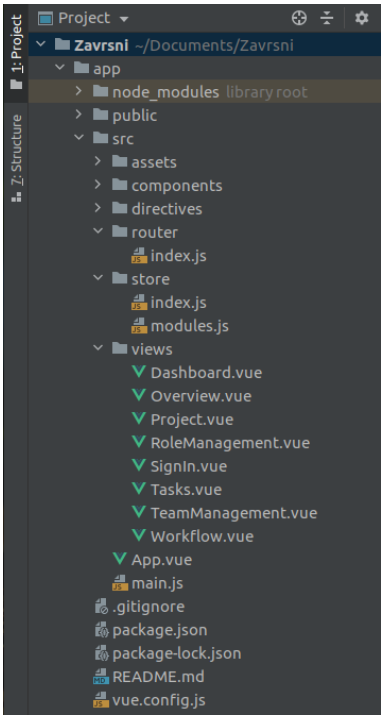


Slika 19. Prikaz sučelja voditelja tima



Slika 20. Prikaz sučelja developera

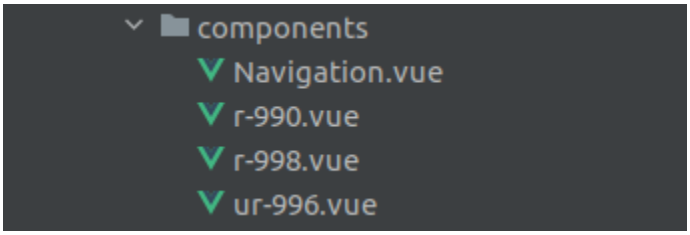
Nakon što smo postavili korisnička sučelja osnovni direktorij bi trebao izgledati kao na slici 21.



Slika 21. Osnovni direktorij

4.3. Rješenje problema razine pristupa

Problem razine pristupa, odnosno dopuštenja javlja se kada je aplikacija zadužena pružiti različite mogućnosti za različite korisnike kao što su ovdje administrator, voditelj tima te ostali članovi tima. To su tri korisnika aplikacije s različitim pristupnim mogućnostima korištenja aplikacije. Pazeći na to potrebno je sakriti određene stvari. Ono što je najbitnije, jest navigacije, jer se preko navigacije dolazi do ostalih korisničkih sučelja. Navigaciju je potrebno implementirati kroz četiri komponente kako je vidljivo na slici 22.



Slika 22. Navigacijske komponente

Prva je glavna komponenta „Navigation.vue“ u kojoj se stvara jedna od tri dodatne komponente s obzirom na to koji korisnik pristupa aplikaciji. Tri dodatne komponente su „ur-996.vue“ koja predstavlja najvišu razinu pristupa, „r-998.vue“ predstavlja razinu pristupa ograničenu samo za developere, odnosno članove tima i „r-990.vue“ je zadnja dodatna komponenta koja sadrži pristup za voditelje tima.

```
export default {
  name: 'Navigation',
  props: {
    navForRole: String,
    required: true
  },
  data () {
    return {
      nav: '',
    }
  },
  components: {
    'ur996': () => import('./ur-996'),
    'r998': () => import('./r-998'),
    'r990': () => import('./r-990')
  },
  computed: {
    currentNavComponent: function () {
      return this.navForRole;
    }
  }
}
```

Slika 23. Prikaz koda za dodavanje dodatnih navigacijskih komponenti

Kod koji prikazuje slika 23. predstavlja dodavanje komponenti po potrebi. Prikazana *navForRole* varijabla sadrži podatak o tome koju navigaciju je potrebno dodati u aplikaciju. Kada se korisnik prijavi, s obzirom na njegovu razinu pristupa, postavlja se varijabla s imenom jedne od tri dodatne komponente. Glavna navigacijska komponenta sadrži još jednu bitnu liniju koda koja omogućuje dinamičko stvaranje dodatne komponente je *<component :is="currentNavComponent"></component>*. Tag „component“ predstavlja dinamičku osnovu za dodavanje komponenti u osnovnu *Vue* datoteku. Kada se ova linija krene izvršavati poziva se funkcija *currentNavComponent()* koja vraća podatak, odnosno naziv dodatne komponente iz *navForRole* varijable te s obzirom na to se dodaje odgovarajuća dodatna navigacijska komponenta.

4.4. Osnovne funkcionalnosti klijenta

Među osnovnim funkcijama aplikacije podrazumijevaju se sve administratorske mogućnosti korištenja aplikacije uz prijavu u aplikaciju, odnosno pregled svih korisnika i dodavanje novog, pregled svih timova i dodavanje novog te posljednje je pregled svih projekata i dodavanje novog projekta.

4.5. Dodatne funkcionalnosti klijenta

U dodatne funkcionalnosti aplikacije se ubrajaju funkcionalnosti koje spadaju u korisnička sučelja za voditelja tima i developera.

4.5.1. Workspace.vue komponenta

```
<workflow-item
  v-for="w in workflowItems"
  :active-project="userInfo.team.activeProject"
  :key="w.id" :table-prop="w"
  :team="personnel"
  @update-table="updateTableData"
></workflow-item>
```

Slika 24. Integracija WorkspaceItem komponente

Za početak potrebno je objasniti od čega se sastoji *Workspace* voditelja tima. *Workspace* voditelja tima sastavljen je iz dva dijela: *Workspace.vue*, glavne komponente i *WorkspaceItem.vue*, dodatne komponente. U glavnoj komponenti, prilikom otvaranja, se prvo provjerava postoje li već dodatni podaci o projektu. Ukoliko postoje prosljeđuju se dodatnoj komponenti, što je vidljivo na slici 24, koja je zadužena za detaljan raspored podataka i izgled samog tabličnog prikaza. Kroz *v-for* određuje se koliko će biti dodatnih komponenti, a *key* je način, odnosno identifikacija pojedinih komponenti, a *active-project* je *id* projekta, odnosno pojedine dodatne komponente, jer ona predstavlja jedan projekt, dok je *team* objekt koji sadrži sve članove tima i njihove osnovne informacija (ime i prezime te put do slike). Osim toga definiran je i *Event listener* pod nazivom *update-table*. On sluša svako emitiranje funkcije pod ti nazivom u dodatnoj komponenti te poziva funkciju *updateTableData()*.

Nakon uvodnog objašnjenja prikaza voditelja tima, potrebno je shvatiti što se zapravo događa u glavnoj komponenti i čemu ona prvenstveno služi. Glavna komponenta je na neki način potpora na koju dograđujemo dodatne stvari, ono na što se dodatne komponente oslanjaju. Tu se prvenstveno misli na dohvaćanje i spremanje podataka iz *WorkspaceItem* komponenti kroz *requestProjectTable(), setNewProjectTable()* i *updateTableData()* metode koje direktno komuniciraju s *Firebase Realtime* bazom podataka. Metoda *requestProjectTable()*, sa slike 25, započinje stvaranje *Workspace.vue* komponente. Metoda dohvaća sve podatke projekta iz *Firebase* baze podataka (Firebase dokumentacija). ukoliko ih ima, a pronalazi ih preko *id*-ja projekta koji je pohranjen u *activeProject* atribut objekta *userInfo*. Ukoliko ih ima spremaju se u *workspaceItems* listu, a ukoliko ih nema poziva se metoda *setNewProjectTable().*

```
141   requestProjectTable () {
142     firebase.database().ref( path: 'P-' + this.userInfo.team.activeProject).once( eventType: 'value',  successCallback: (snap) => {
143       this.workflowItems = [];
144       this.workflowItems = snap.val();
145       this.checkSubItems();
146       if (this.workflowItems.length === 0) this.setNewProjectTable();
147     }, this);
148   },
```

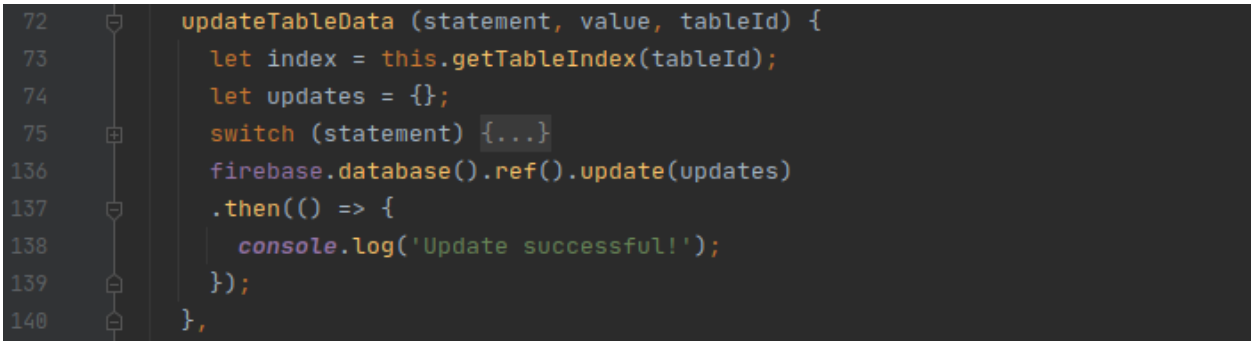
Slika 25. Metoda requestProjectTable()

Kod sa slike 26 predstavlja metodu *setNewProjectTable()*. Ona stvara novi projekt na način da spremi početne, zadane podatke u *Firebase* bazu podataka u osnovi. Prvo se izvršava poziv prema *Firebase* bazi podataka za *id* nove tablice koja se sprema. Nakon toga se određuju zadani podaci koje spremamo u novu listu te ju šaljemo prema *Firebase* bazi podataka na spremanje. Te ju ponovno dohvaćamo i spremamo u *workflowSpaceItems listu.*

```
149   setNewProjectTable () {
150     const tableId = firebase.database().ref().push().key;
151     let item = {id: tableId...};
261     let items = [];
262     items.push(item);
263     firebase.database().ref( path: 'P-' + this.userInfo.team.activeProject).set(items)
264     .then((r) => {
265       console.log('Project successfully created!');
266     })
267     firebase.database().ref( path: 'P-' + this.userInfo.team.activeProject).once( eventType: 'value',  successCallback: (snap) => {
268       this.workflowItems = snap.val();
269       this.checkSubItems();
270     });
271   },
```

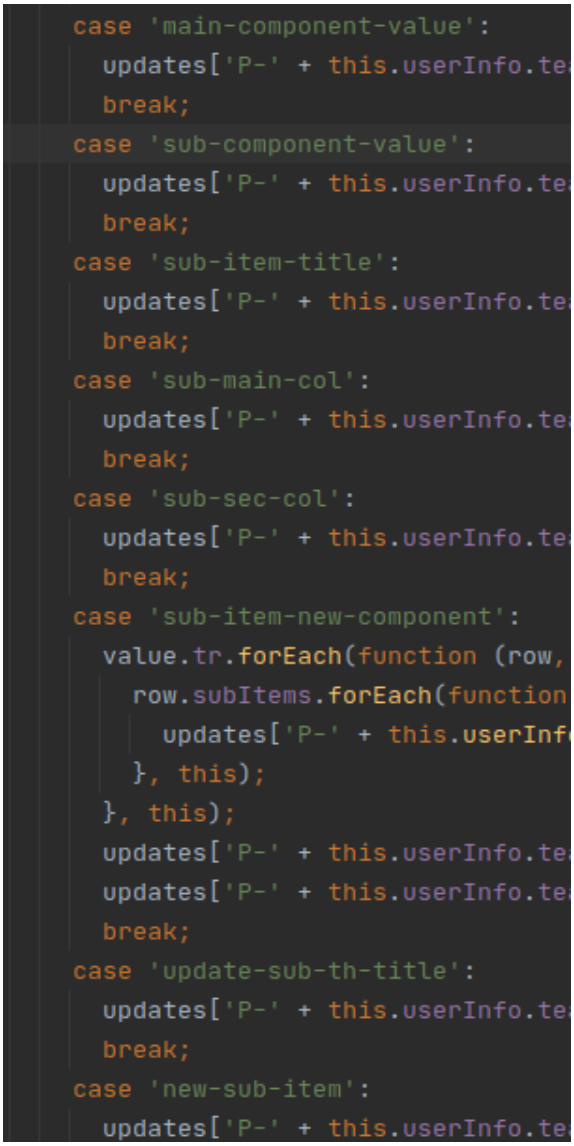
Slika 26. Metoda setNewProjectTable()

Slika 27 prikazuje zadnju bitnu metodu, *updateTableData().* Koja ažurira postojeće podatke za određeni projekt u *Firebase* bazi. Ovdje je važno za naglasiti da metoda prima tri bitna parametra. To su *statement* – string koji određuje što se sprema, *value* – varijabla ili objekt koji se sprema i *tableId* – identifikacijski broj kako bi mogli spremiti podatke u odgovarajuću tablicu. Definicije mogućih *statement* varijabli prikazane su na slikama 28.1. i 28.2. Isto tako, put do varijabli u bazi je određen prema slučaju koji je odabran, a napisan je prema sljedećem uzorku *updates['P-' + this.userInfo.team.activeProject + indeks + „put do atributa u kojeg se sprema varijabla ili objekta“] = value;*.



Slika 27. Metoda updateTableData()

Slika 28.1. Statement primjer 1.



Slika 28.2. Statement primjer 2.

4.5.2. WorkspaceItem.vue komponenta

Sljedeća stvar na popisu nakon glavne *Workspace* komponente je njena dodatna komponenta. Ona sadrži sedam konstanti prikazanih na slici 29. Konstante koje djeluju kao manje komponente te se uključuju u projekt po potrebi. Prvih šest predstavljaju podatkovno promjenjive ćelije u tablici, dok posljednja konstanta služi za prikazivanje pod redaka, odnosno zadataka u tablici. Kao što osnovna tablica uključuje prvih šest

komponenata u predložak, tako i *subItem* komponenta uključuje istih šest komponenata u svoj predložak.

```
85   const dateType = {name: 'date-type'...};
107  const originType = {name: 'origin-type'...};
156  const contentType = {name: 'content-type'...};
205  const priorityType = {name: 'priority-type'...};
254  const statusType = {name: 'status-type'...};
303  const personType = {name: 'person-type'...};
346  const subItem = {name: 'sub-item'...};
531
532  export default {
533    name: "WorkflowItem",
534    props: {tableProp: Object...},
540    data () {
541      return {
542        table: this.tableProp,
543        toggleNewCellDD: false,
544        newRowTitle: '',
545        colMenu: false,
546
547        subItem: 'sub-item',
548        dateType: 'date-type',
549        statusType: 'status-type',
550        originType: 'origin-type',
551        contentType: 'content-type',
552        priorityType: 'priority-type',
553        personType: 'person-type'
554      }
555    },
556    sockets: {...},
563    created () {...},
566    methods: {...},
642    components: {
643      'sub-item': subItem,
644      'date-type': dateType,
645      'status-type': statusType,
646      'origin-type': originType,
647      'content-type': contentType,
648      'priority-type': priorityType,
649      'person-type': personType
650    }
```

Slika 29. Prikazuje uključivanje dodatnih komponenti u WorkspaceItem

Metode koje se izvršavaju ovdje su *sendUpdateToTeam()* koja će biti objašnjena kasnije, *setNewDataValue(), newTableRow()*, *insertNewSubItemInRow()*, *newCell()*, *deleteColumn() i requestTableUpdate().*

Metoda *setNewDataValue()*, vidljiva na slici 30, prima dva parametra, *statement* i *value* te ih prosljeđuje glavnoj komponenti uz *table.id* vrijednost preko emitiranja *update-table* događaja.

```
570    setNewDataValue (statement, value) {
571      this.$emit( event: 'update-table', statement, value, this.table.id);
572    },
```

Slika 30. Metoda setNewDataValue()

Metoda *newTableRow(),* prikazana na slici 31, ima glavni zadatak ubaciti novi red sa zadanim podacima u tablicu što se vidi na liniji 575 sa slike 33. Nakon toga pozivamo *setNewDataValue()* metodu za ažuriranje podataka u *Firebase* bazi.

```
573    newTableRow () {
574      event.preventDefault();
575      this.table.tr.push({
576        id: this.table.tr.length + 1,
577        itemStatus: false,
578        title: this.newRowTitle,
579        subItems: [],
580        show: false,
581        components: this.table.tr[0].components
582      });
583      this.newRowTitle = '';
584      this.$refs['newRowInput'].blur();
585      this.setNewDataValue( statement: 'tr-new-item', this.table.tr);
586    },
```

Slika 31. Metoda newTableRow()

Metoda *insertNewSubItemInRow(),* vidljiva na slici 32, prima dva parametra *index* koja predstavlja indeks retka u tablici u kojeg korisnik želi dodati novi pod redak te *subitem* što je novi objekt kojeg se dodaje. Dodaje se na način da se objekt doda u *subItems* atribut

određen *index* podatkom te se nakon toga emitira *update-table* događaj i šalju potrebni parametri za ažuriranje podataka u *Firebase* bazi.

```
587    insertNewSubIteminRow (index, subitem) {
588      this.table.tr[index].subItems.push(subitem);
589      let updateData = {
590        asset: this.table.tr[index].subItems,
591        index: index
592      }
593      this.$emit( event: 'update-table',  args: 'new-sub-item', updateData, this.table.id);
594    },
```

Slika 32. Metoda insertNewSubItemInRow()

Metoda *newCell(),* prikazana na slici 33, je metoda koja dodaje novi stupac u tablicu s obzirom na to dodaje li korisnik novi stupac u glavnu tablicu ili unutarnju, odnosno na pod redak. Prima dva parametra *lvl* koji određuje dodaje li se stupac u glavnu tablicu ili unutarnju i *type* što je tip stupca koji se dodaje. Kada se dodaje novi stupac bez obzira u koju razinu, dodavanje se vrši nad *table* objektom, a postupak je sljedeći: prvo se dodaje nova stavka s tipom i nazivom u *th* atribut, povećava se *colNum* atribut za jedan, a on predstavlja sveukupan broj stupaca, nakon toga se sprema novi objekt u *tr* listu objekta s obzirom na tip u parametru. Nakon toga se poziva *requestTableUpdate()* metoda.

```
595    newCell (type, lvl) {
596      if (lvl === 'main') {
597        this.table.th.push({title: type, type: type});
598        this.table.colNum++;
599        this.table.tr.forEach(function (row) {
600          if (type === 'person')  row.components.push({id: row.components.length + 1, name: 'person-type', value: 'None'});
601          else if (type === 'date')  row.components.push({id: row.components.length + 1, name: 'date-type', value: '2020-09-01'});
602          else  row.components.push({id: row.components.length + 1, name: type + '-type', value: {text: '', statusColor: '245, 245, 245'}});
603        }, this);
604        this.requestTableUpdate( statement: 'tr-item-new-component', this.table.colNum, this.table.th);
605      } else {
606        this.table.sub.th.push({title: type, type: type});
607        this.table.sub.colNum++;
608        this.table.tr.forEach(function (row) {
609          if (row.components.length > 0) {
610            row.subItems.forEach(function (subRow) {
611              if (type === 'person') subRow.data.push({id: subRow.data.length + 1, name: 'person-type', value: 'None'});
612              else if (type === 'date' || type === 'link') subRow.data.push({id: subRow.data.length + 1, name: 'date-type', value: '2020-09-01'});
613              else subRow.data.push({id: subRow.data.length + 1, name: 'status-type', value: {text: '', statusColor: '245, 245, 245'}});
614            }, this);
615          }
616        }, this);
617        this.requestTableUpdate( statement: 'sub-item-new-component', this.table.sub.colNum, this.table.sub.th);
618      }
619    },
```

Slika 33. Metoda newCell()

Metoda *requestTableUpdate()*, kako je prikazano na slici, je na neki način slična *setNewDataValue()* metodi. Prima tri parametra: prvi parametar je *statement* kao i u

metodi sa slike 32, a druga dva parametra su *colNum* i *th* koji se spremaju kao atributi novog objekta koji se šalje u glavnu komponentu preko emitiranja događaja *update-table*.

```
639     requestTableUpdate (statement, colNum, th) {
640       let updateData = {
641         colNum: colNum,
642         th: th,
643         tr: this.table.tr
644       }
645       this.$emit( event: 'update-table', statement, updateData, this.table.id);
646     }
647   },
```

Slika 34. Metoda requestTableUpdate()

U metodi *deleteColumn()*, vidljivoj na slici 35, osnovna je funkcija obrisati stupac iz tablice. Prima tri parametra *col* objekt koji sadrži naziv i tip stupca koji se briše, *index* na kojem se nalazi u listi *th* te *type* koji označava briše li se stupac iz glavne ili unutarnje tablice. Postupak je sljedeći: prvo se briše objekt iz liste *tr*, zatim se briše naziv stupca iz liste *th* te se na kraju *colNum* atribut smanjuje za jedan i nakon toga se poziva funkcija *requestTableUpdate()*.

```
620   deleteColumn (col, index, type) {
621     if (type === 'main') {
622       this.table.tr.forEach(function (row) {
623         row.components.splice(index + 2, 1);
624       });
625       this.$delete(this.table.th, index);
626       this.table.colNum--;
627       this.requestTableUpdate( statement: 'tr-item-new-component', this.table.colNum, this.table.th);
628     } else {
629       this.table.tr.forEach(function (row) {
630         row.subItems.forEach(function (subRow) {
631           subRow.data.splice(index + 2, 1);
632         })
633       });
634       this.$delete(this.table.sub.th, index);
635       this.table.sub.colNum--;
636       this.requestTableUpdate( statement: 'sub-item-new-component', this.table.sub.colNum, this.table.sub.th);
637     }
638   },
```

Slika 35. Metoda deleteColumn()

4.5.3. *Task.vue* komponenta

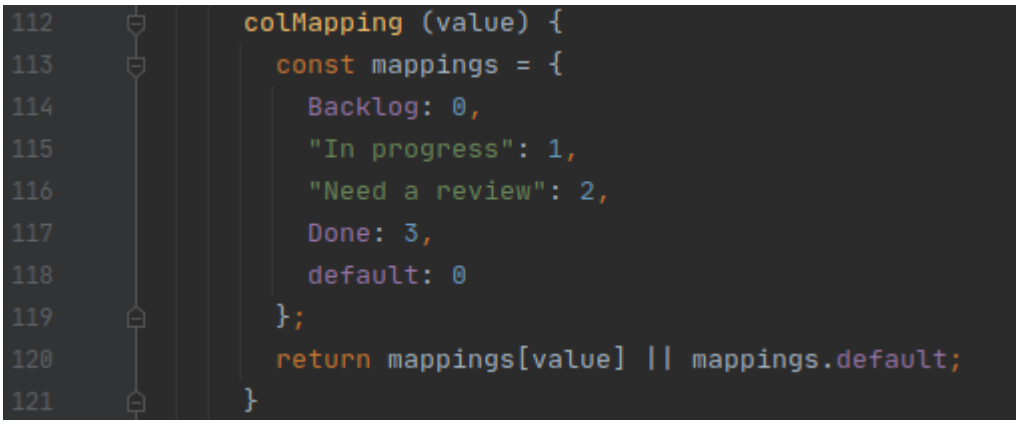
Prvi dio poglavlja govorio je o *Workspace*-u voditelja tima, a sada u drugom dijelu opisat će se dijelovi i karakteristike *Workspace*-a developera čija se komponenta naziva *Task.vue*. *Task.vue* je glavna komponenta na razini *Workspace.vue* komponente. Isto tako tu je i *TaskItem.vue* pod komponenta. *Task.vue* u predlošku uključuje dvije pod komponente: *TaskItem i draggable.*Pod komponenta *draggable* omogućuje *drag and drop* stavki unutar sebe, a te stavke su upravo *TaskItem* stavke. Metode koje se nalaze u ovoj glavnoj komponenti su *updateForLeader()* i *updateItemStatus()* koje će biti pojašnjene kasnije, *requestProjectTabe()* koja je ista kao ista kao i metoda sa slike 27, *sortDataToTable()* te *colMapping()*.

Metoda *sortDataToTable()*, vidljiva na slici 36, ima osnovnu svrhu da rasporedi retke i njihove glavne atribute (naziv, status, datum i osobu koja izvršava taj zadatak) u novi objekt nazvan *task*. Svi potrebni podaci se izvlače iz liste *tr* koja je atribut parametra *array* te se sortiraju u objekt *task* koji sadrži atribute: id, naziv, datum, boja i slika te se sprema u glavnu listu *columns* gdje se spremaju svi podaci koji su važni sučelje developera.

```
sortDataToTable (array) {
  let temp = this.columns;
  let index = 0;
  let img = null;
  let content = null;
  let color = null;
  array.tr.forEach(function (value) {
    index = this.colMapping(value.components[0].value.text);
    value.components.forEach(function (innerValue) {
      if (innerValue.name === 'person-type')  img = innerValue.value.img;
      else if (innerValue.name === 'content-type') {
        content = innerValue.value.text;
        color = innerValue.value.statusColor;
      }
    });
    temp[index].tasks.push({
        id: value.id,
        title: value.title,
        date: value.components[1].value,
        color: color,
        img: img, content: content
      })
  }, this);
  this.columns = temp;
},
```

Slika 36. Metoda sortDataToTable()

Metoda *colMapping()*, prikazana na slici 37, prima parametar *value* koji sadrži string koji predstavlja status retka glavne tablice iz *WorkspaceItem.vue* komponente. Na osnovu tog stringa vraća broj od nula do tri koji predstavljaju indeks objekta iz glavne liste *columns* gdje će se spremiti novi objekt *task*.



```
colMapping (value) {
  const mappings = {
    Backlog: 0,
    "In progress": 1,
    "Need a review": 2,
    Done: 3,
    default: 0
  };
  return mappings[value] || mappings.default;
}
```

Slika 37. Metoda colMapping()

4.5.4. TaskItem.vue komponenta

TaskItem.vue komponenta, sa slike 38, zadužena je za prikazivanje pojedinog retka, odnosno zadatka iz glavne tablice. Zadatak se prikazuje u obliku interaktivnog bloka kojeg korisnik, odnosno developer može pomicati iz stupca u stupac ovisno o stvarnom statusu zadatka. U predlošku sa slike 40 vidi se da se u bloku prikazuje *title*, odnosno naziv zadatka, atribut *img* koji sadrži put do profilne slike korisnika te ju prikazuju pozivom funkcije *profileImg()* koja prima taj atribut kao parametar te preko *require()* metode vraća konačnu sliku koja se prikazuje. Osim toga u bloku se prikazuje približan datum određen za izvršenje zadatka te tip zadatka.

```
<template>
  <div class="bg-white shadow rounded p-3 border border-white" :data-id="task.id">
    <div class="d-flex justify-content-between card-top-wrapper">
      <p class="text-gray-700 font-semibold font-sans tracking-wide text-sm">{{task.title}}</p>
      <b-icon v-if="task.img === undefined" icon="person-circle"></b-icon>
      <img v-else  class="avatar-small" :src="profileImg(task.img)" alt="Avatar">
    </div>
    <div class="d-flex text-justify justify-content-between" style="margin-top: 85px;">
      <span class="task-date">{{task.date}}</span>
      <badge v-if="task.content !== null" :content="task.content" :color="task.color">{{task.content}}</
    </div>
  </div>
</template>

<script>
const badge = {
  name: 'badge',
  props: {
    content: String,
    color: String
  },
  template: `<div class="badge-wrapper d-flex align-items-center"
                  :style="{backgroundColor: 'rgba(' + color + ', .2)', color: 'rgb(' + color + ')'}">
               <span class="badge-point" :style="{backgroundColor: 'rgb(' + color + ')'}"></span>
               <span><slot></slot></span>
             </div>`
};

export default {
  name: "TaskItem",
  props: {task: Object...},
  data () {...},
  methods: {
    profileImg (img) {
      return require(`../assets/photos/${img}`);
    }
  },
  components: {
    'badge': badge
  }
}
```

Slika 38. Detaljan prikaz koda TaskItem.vue komponente

4.6. Postavljanje funkcionalnosti poslužitelja

Poslužiteljski sloj aplikacije, u ovom slučaju *API*, vidljiv na slici 39, konkretnije sastoji se od nekoliko ruta koje sadrže *GET*, *POST*, *UPDATE* i *DELETE* metode. Neke rute sadrže sve metode, a neke jednu ili dvije ovisno o potrebi. Ruta */signup* koristi se kod prijave korisnika u aplikaciju. Prima dva parametra, korisnički kod i lozinku te ih uspoređuje s podacima iz baze. Ukoliko nije pronašla ništa slično vraća pogrešku, a ukoliko je sve u redu vraća korisniče podatke. Ruta */member* koristi se za dohvaćanje i spremanje korisnika, dok ruta */team* prvenstveno služi za spremanje novog tima te osim toga i za dohvaćanje korisnika iz baze podataka koji su slobodni, bez tima kako bi ih administrator mogao rasporediti u tim. Ruta */team/management/project* vraća *id* projekta preko *id* korisnika, a ruta */team/management/leader* vraća sve podatke o timu i njegovim članovima. Zadnja ruta je */project* koja se poziva kod spremanja novog projekta i dohvaćanja svih projekata i njihovih informacija iz baze podataka.

```
15  @app.route('/')
16
17  @app.route('/signup', methods=['GET', 'POST'])
18  def signup():
19
20  @app.route('/member/<state>', methods=['GET','POST', 'DELETE', 'UPDATE'])
21  def handle_base_member(state):
22
23  @app.route('/team/member/free', methods=['GET'])
24  def handle_teamless_member():
25
26  @app.route('/team/<state>', methods=['GET','POST'])
27  def handle_base_team(state):
28
29  @app.route('/team/management/project/<member_id>', methods=['GET'])
30  def handle_project_id_request(member_id):
31
32  @app.route('/team/management/leader/<leader_id>', methods=['GET'])
33  def handle_get_team_by_leader(leader_id):
34
35  @app.route('/project/<state>', methods=['GET','POST'])
36  def handle_base_project(state):
```

Slika 39. API funkcionalnosti

4.7. *Real time* aplikacijski sloj

Kako bi se uopće moglo razgovarati o *real time* aplikaciji *socket.io* paket je neophodan u klijentskom i poslužiteljskom sloju. Paket je instaliran prema uputama iz LITERATURE BROJ 3 Na slici 40 prikazana je implementacija *socket.io* paketa na poslužitelju. *@socketio.on* je događaj koji se pokreće kada klijent emitira naziv događaja koja je napisana u zagradi. To su ovdje *connect*, *update_to_team* i *update_to_leader*. Nakon što se događaj pokrene funkcija za upravljanje događajem prihvaća podatke u parametru *dana* te ih emitira dalje kroz sljedeći parametar i naziv događaja prema spojenim klijentima. Još je jedna stvar bitna za spomenuti kod emitiranja događaja prema klijentima. Atribut *broadcast* je ovdje postavljen na *true* što znači da se događaj emitira prema svim klijentima. U slučaju da je *false* ili da nije napisan, događaj se emitira samo prema klijentu koji je pokrenuo događaj na poslužitelju. Drugi dio sloja koji je potrebno namjestiti su *socket* funkcije na klijentu. Događaju se izmjenjuju između korisničkog sučelja voditelja tima, odnosno *WorkflowItem* komponente i korisničkog sučelja developera, tj. *Task* komponente. Događaji se odvijaju u smjeru od *WorkflowItem* komponente prema *websoc* poslužitelja do *Task* komponente i obratno. Bitno je dotaknuti se jednog i drugoga, jer nisu potpuno isti procesi primanja i slanja podataka.

U prvom procesu od korisničkog sučelja voditelja tima do sučelja developera događaj se pokreće kada voditelj tima promjeni podatke ćelije u statusnom stupcu. Ta promjena poziva metodu *setNewValue()*, prikazana na slici 41, koja prima tri parametra iz predloška: trenutni objekt status, novi objekt status te *id* retka koji zapravo u ovom slučaju nije potreban. U metodi se prvo vrši emitiranje događaja prema glavnoj komponenti s parametrima koji sadrže novi statusni objekt, a zatim se emitira događaj s potrebnim podacima za ažuriranje stanja sučelja developera (*id* retka uz novi objekt). Događaj se emitira do metode *sendUpdateToTeam()* gdje se izvršava sljedeća linija kod: *this.$socket.emit(„update_to_team“, value);* Znači preko *socket.emit* metode se emitira *update_to_team* događaj koji novi *value* sa sobom. Varijabla *value* predstavlja objekt iz *setNewValue()* metode. Nakon što poslužitelj prihvati događaj prenosi dalje podatke do klijenta, odnosno sučelja developera gdje se ti podaci prihvaćaju i obrađuju na način da

se kroz petlju provlače zadaci koji se već nalaze na sučelju klijenta te se traži zadatak s istim *id*-jem kao i kod novog objekta (*item.rowId*) kako je prikazano na slici 42. Kada se pronađe sprema se u novonastali objekt *tmp* te se sprema indeks na kojem se zadatak nalazi. Nakon toga ponovno se pretražuje, no ovaj put samo lista *columns* u svrhu pronalaska stupca jednakog imena kao i kod novog objekta (*item.text*). Kada se pronađe dodaje mu se objekt *tmp* te se stari objekt briše iz prvog stupca čime je prvi ciklus završen.

```python
from flask import Flask, render_template
from flask_socketio import SocketIO, send, emit
from flask_cors import CORS


app = Flask(__name__)
CORS(app)

app.config['SECRET_KEY'] = 'secret!'

socketio = SocketIO(app, cors_allowed_origins="*")

@socketio.on('connect')
def handle_connect():
    print('Connections established!')

@socketio.on('update_to_team')
def handle_team_update(data):
    emit('teamUpdate', data, json=True, broadcast=True)

@socketio.on('update_to_leader')
def handle_message_to_leader(data):
    emit('updateItemStatus', data, json=True, broadcast=True)


if __name__ == '__main__':
    socketio.run(app)
```

Slika 40. Flask socket.io kod, websoc.py

```
setNewValue (r, s, compId) {
  r.text = s.value;
  r.statusColor = s.color;
  this.toggleDD = false;
  this.$emit( event: 'update-table-values',  args: 'main-component-value', {
    index: this.parent,
    compIndex: compId - 1,
    asset: {text: r.text, statusColor: r.statusColor}
  })
  this.$emit( event: 'ws-canvas-update',  args: {rowId: this.parent + 1, text: r.text, statusColor: r.statusColor})
}
```

Slika 41. Metoda setNewValue()

```
sockets: {
  teamUpdate: function (data) {
    this.updateItemStatus(data);
  }
},
methods: {
  updateItemStatus (item) {
    let index = -1;
    let colIndex = -1;
    let tmp = null;
    this.columns.forEach(function (col, cIndex) {
      col.tasks.forEach(function (task, tIndex) {
        if (task.id === item.rowId) {
          index = tIndex;
          colIndex = cIndex;
          tmp = task;
        }
      })
    });
    this.columns.forEach(function (col) {
      if (col.title === item.text) {
        col.tasks.push(tmp);
      }
    });
    this.columns[colIndex].tasks.splice(index, 1);
  },
```

Slika 42. Prikaz prihvaćanja i obrade podataka poslanih s poslužitelja kroz metodu updateItemStatus()

Drugi ciklus u obrnutom smjeru, od korisničkog sučelja developera prema sučelju voditelja tima započinje kada developer prebaci blok iz jednog stupca u drugi, što je vidljivo na slici 43. Kada korisnik uhvati blok i krene ga pomicati okida se događaj koji sprema *id* bloka u globalnu varijablu *draggedItem*, a kada ga spusti pokreće metodu *updateForLeader()* koja je vidljiva na slici 44. Ona emitira *update_to_leader* događaj s objektom koji sadrži naziv novog stupca, boju i *id* bloka, odnosno zadatka nad kojim se radnja izvršila.

```
@drop="updateForLeader(column.title, column.color)" @dragstart="draggedItem = $event.target.dataset.id"
```

Slika 43. Prikaz koda koji pokreće drugi ciklus

```
80    updateForLeader (colTitle, color) {
81      this.$socket.emit('update_to_leader', {
82        colTitle: colTitle,
83        statusColor: color,
84        itemId: this.draggedItem
85      });
86    },
```

Slika 44. Metoda updateForLeader()

Nakon što poslužitelj prihvati događaj prosljeđuje podatke prema drugom klijentu, odnosno korisničkom sučelju. Slika 45 prikazuje metodu *updateItemStatus()* koja prihvaća događaj i preuzima podatke te ih sprema na odgovarajuću poziciju u tablici.

```
560  sockets: {
561    updateItemStatus: function (data) {
562      this.table.tr[data.itemId - 1].components[0].value.text = data.colTitle;
563      this.table.tr[data.itemId - 1].components[0].value.statusColor = data.statusColor;
564    }
565  },
```

Slika 45. Metoda updateItemStatus() u komponenti WorkspaceItem.vue

5. Zaključak

Web aplikacija za upravljanje projektima pruža mogućnost kvalitetnog upravljanja timom kroz razvoj raznih, na prvome mjestu, informatičkih, marketinških a i drugih projekata. Podijeljena je u tri razine pristupa što omogućava nesmetan pregled napretka projekta. Svaka razina ima svoje mogućnosti karakteristične za takav pristup. Administratorska razina omogućava pregled svih korisnika uz dodavanje novih, pregled svih timova s mogućnošću dodavanja novih te pregled projekata i stvaranje novih. S druge strane, razina voditelja tima ima mogućnost upravljanja članovima tima i njihovim zadacima do uspješnog završetka projekta te na kraju developerska razina koja ima pristup pregledu zadacima tima.

Ovo je prva verzija aplikaciju, no očekuju ju buduća poboljšanja koja dolaze u kasnijim verzijama. Poboljšanja će se odnositi na funkcionalnosti prvenstveno te paralelno s time optimiziranje koda, a možda i u korisničkom sučelju, međutim to je na kraju liste poboljšanja. Funkcionalna poboljšanja bit će usmjerena ponajprije prema interakciji voditelja s članovima tima. Primjerice otvaranje rasprave oko pojedinog zadatka ili čak na razini cijelog projekta gdje će svaki član imati mogućnost iznijeti svoje mišljenje o pojedinoj temi. Osim toga, pregled aktivnih korisnika te mogućnost razmjene internih poruka s članovima, dok kod sučelja developera bi se dodao detaljan prikaz zadatka, tj. bloka koji bi se otvorio klikom na pojedini blok. Možda bi se dodala još jedna razina pristupa koja bi imala mogućnost samo pregleda projekata i timova bez ikakvih dodatnih mogućnosti.

Aplikacija se nalazi u Github repozitoriju i moguće ju je preuzeti na adresi https://github.com/SemjanivNikola/project-workbook.git. Upute za korištenje aplikacije i korisničke podatke za prijavu u aplikaciju moguće je pronaći u *README.md* datoteci, a demo aplikacije moguće je pronaći na adresi https://project.workbook.h3.solutions.

Literatura

Internet izvori:

1. Asana (2008), Dostupno na:
   https://asana.com/
   [15.09.2020]
2. Bitrix24 (2020), Dostupno na:
   https://www.bitrix24.com/tools/tasks_and_projects/
   [15.09.2020]
3. Firebase dokumentacija, Dostupno na:
   https://firebase.google.com/docs
   [26.08.2020]
4. Firebase platforma, Dostupno na:
   https://firebase.google.com/
   [26.08.2020]
5. Flask socket.io, Dostupno na:
   https://flask-socketio.readthedocs.io/en/latest/
   [04.09.2020]
6. OpenProject (2020), Dostupno na:
   https://www.openproject.org/
   [15.09.2020]
7. Vue CLI (2018), Dostupno na:
   https://cli.vuejs.org/
   [18.08.2020]
8. Vue-socket.io (2016), Dostupno na:
   https://github.com/MetinSeylan/Vue-Socket.io
   [04.09.2020]

Popis slika

Sažetak

Web aplikacije svakim danom sve više rastu i sve više ih ima. Sve su složenije sa sve više mogućnosti, rješavaju više problema te je korisničko sučelje sve bolje i bolje. Jednostavno se stvari jako brzo razvijaju i sve više je čovjek usmjeren na Internet. Ovaj rad obuhvaća razvoj web aplikacije čija je tema upravljanje projektima. Kroz rad se raspravlja o načinu postavljanja glavnih alata i paketa za razvoj aplikacije te sam način na koji se aplikacija razvija. Od korisničkog sučelja, glavnih funkcionalnosti i detaljnog objašnjavanja metoda koje ih prate do poslužitelja i upravljanja podacima koji dolaze do njega.

Ključne riječi: web aplikacija, klijent, poslužitelj, upravljanje projektima, SPA, HTML, CSS, Vue.js, socket.io, Python, Flask

Abstract

Web applications grow exponentially and there are more and more of them with each day. They are more complexed with even more functionalities, solve even more problems, plus, user interface is better and better. Simply, things evolve rapidly and human is much more oriented to the Internet. This paper covers the development of web application whose topic is project management. Within the paper discusses how to set up the main tools and packages for application development and the way in which the application is developed. From user interface, main functionalities and detailed explanation of methods to the server and the management of the data from the client.

Keywords: web application, frontend, backend, project management, SPA, HTML, CSS, Vue.js, socket.io, Python, Flask