Web aplikacija za upravljanje projektima

Jahaj, Bljeona

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

Sveučilišni diplomski studij računarstva

Web aplikacija za upravljanje projektima

Diplomski rad

Bljeona Jahaj

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Bljeona Jahaj
Studij, smjer:	Sveučilišni diplomski studij Računarstvo
Mat. br. pristupnika, god.	D-1125R, 13.10.2020.
JMBAG:	0165069167
Mentor:	doc. dr. sc. Tomislav Galba
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Alfonzo Baumgartner
Član Povjerenstva 1:	doc. dr. sc. Tomislav Galba
Član Povjerenstva 2:	prof. dr. sc. Tomislav Keser
Naslov diplomskog rada:	Web aplikacija za upravljanje projektima
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Potrebno je napraviti aplikaciju koja će omogućiti kompletno praćenje statusa nekog projekta i projektnih timova. Koristiti Spring framework. Tema je rezervirana za: Bljeona Jahaj

Datum ocjene pismenog dijela diplomskog rada od strane mentora:	20.09.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	30.09.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	30.09.2024.

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

IZJAVA O IZVORNOSTI RADA

Osijek, 30.09.2024.

Ime i prezime Pristupnika:	Bljeona Jahaj
Studij:	Sveučilišni diplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1125R, 13.10.2020.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija za upravljanje projektima**

izrađen pod vodstvom mentora doc. dr. sc. Tomislav Galba

i sumentora

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

Potpis pristupnika:

SADRŽAJ

1. Uvod

U ovom diplomskom radu razvija se web (eng. *World Wide Web*) aplikacija za praćenje projekata i projektnih timova te pojedinih članova unutar pojedinog tima. Cilj rada je prikazati kako na brz i efikasan način napraviti potpuno besplatnu i prilagođenu verziju postojećih alata koji se koriste u firmama, a koji su često plaćeni, ali nisu prilagođeni potrebama članova unutar projekta ili tima.

U prvom poglavlju opisane su funkcionalnosti već postojećih alata za praćenje projekata i timova te kako implementirana aplikacija može poboljšati efikasnost i produktivnost timova unutar projekta kao i već postojeći alati. U drugom poglavlju ovog rada opisat će se sve korištene tehnologije, uključujući *Spring Framework* i *Spring Boot*, koji su sve popularniji za razvoj web aplikacija zbog svojih prednosti. Pomoću primjera je prikazano kako dijelovi *Spring Framework-a* funkcioniraju i kako se što lakše upoznati s tim programskim okvirom. Korištenjem *Hibernate ORM-a* i *PostgreSQL* baze podataka, može se omogućiti laka interakcija s bazom podataka, implementiranje CRUD (*Create, Read, Update, Delete*) metode, osnovne operacije koje se izvode nad podacima te pomoću *Postman-a* izvršiti testiranje. *Axios* će biti korišten za slanje *HTTP* zahtjeva s frontenda prema backendu. Za *frontend*, koristi se React zajedno s *Tailwind CSS-om* za modernizirano i responzivno korisničko sučelje. *JJWT API* bit će korišten za autentifikaciju i autorizaciju putem *JWT (JSON Web Token)* tehnologije, dok će *Lombok* i *ModelMapper* pojednostaviti razvojni proces eliminiranjem *boilerplate koda* i olakšavanjem mapiranja objekata. U trećem i četvrtom poglavlju opisana je implementacija programskog rješenja na *backendu* i na *frontendu*.

Rezultati ovog istraživanja mogli bi biti korisni za razvojne timove koji žele sami razviti web aplikaciju za praćenje projekata i projektnih timova te napraviti besplatnu i prilagođenu verziju koja će pratiti potrebe tima i članova unutar projekta. Rad također obuhvaća usporedbu ove aplikacije s već postojećim alatima za praćenje projekata i timova te će pokazati prednosti korištenja *Spring Framework-a* i *Spring Boot-a* u razvoju web aplikacije za praćenje projekata i projektnih timova.

2. Pregled postojećih rješenja

U ovom su poglavlju rada analizirana tri programska alata, točnije njihove funkcionalnosti i prednosti te je dana njihova međusobna usporedba. Istraženi alati su *Jira Software*, *Notion* i *ClickUp*.

Jira Software alat je koji se koristi u IT industriji za praćenje projekata i timova, ali se također korist i u drugim industrijama poput obrazovanja i marketinga. Njegova je glavna funkcionalnost upravljanje zadacima i problemima unutar projekta te praćenje napretka i izvještavanje. *Notion* je alat koji se koristi za organizaciju i praćenje projekata i timova te pruža fleksibilnu platformu za stvaranje baza podataka, dokumentacije i planiranja. *ClickUp* je također alat koji se koristi za upravljanje zadacima, planiranje i izvještavanje. Cilj je ovog ulomka pružiti pregled popularnih alata za praćenje projekata i projektnih timova te pokazati njihove glavne funkcionalnosti i prednosti.

2.1. Jira

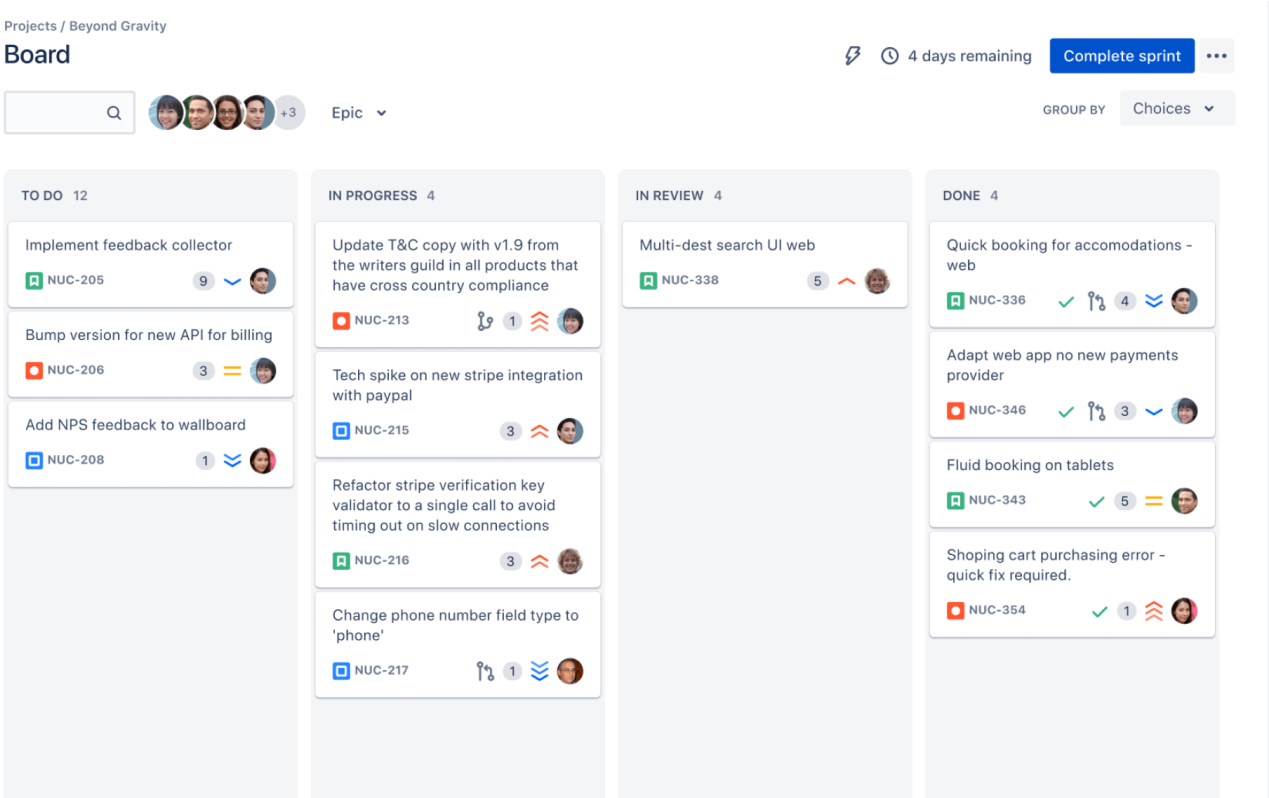
Jira Software je web-temeljen alat za upravljanje projektima i zadacima koji se često koristi u IT sektoru i razvoju *software-a*. Prema [1] alat je razvijen od strane *Atlassian*-a i dostupan je kao *cloud-based* i *self-hosted* opcija. *Jira Software* se koristi za planiranje, izvršavanje i analizu projekata te pomaže timovima da rade efikasnije i uspješnije. Ima različite funkcije koje pomažu u upravljanju projektima, uključujući kreiranje i upravljanje zadacima, sinkronizaciju s drugim alatima, izvještavanje i analizu te upravljanje radnom snagom. Ovaj alat se često koristi za *Agile software development* metodologiju, ali podržava i druge metodologije poput *Waterfall*. *Jira Software* sadrži različite alate za upravljanje zadacima, kao što su kreiranje i upravljanje zadacima, sinkronizaciju s drugim alatima, izvještavanje i analizu te upravljanje radnom snagom. Također, omogućava timovima da kreiraju i upravljaju projektnim ciljevima, vode evidenciju o radu i analiziraju projektne podatke. *Jira Software* se često koristi u IT sektoru, ali se također koristi i u drugim industrijama, kao što su financije, zdravstvo i marketinški sektor. To je jedan od najpopularnijih alata za upravljanje projektima u *software developmentu*, jer omogućava timovima efikasniji i uspješniji rad.

Isto tako *Jira* podržava integraciju s drugim alatima i aplikacijama, kao što su *Google Drive, Slack, Trello* i razni drugi. To omogućava timovima da koriste *Jira Software* zajedno s drugim alatima

koje već koriste u svom radnom okruženju. Prema [1] *Jira Software* također sadrži različite izvještaje i analitiku koja pomaže timovima pratiti napredak projekta i donositi informirane odluke. To omogućava timovima da bolje razumiju svoj rad i da pronađu moguća rješenja za probleme koji se javljaju tijekom projekta. *Jira* ima nekoliko značajki za sigurnost koje pomažu u zaštiti projekata i podataka od neovlaštenog pristupa:

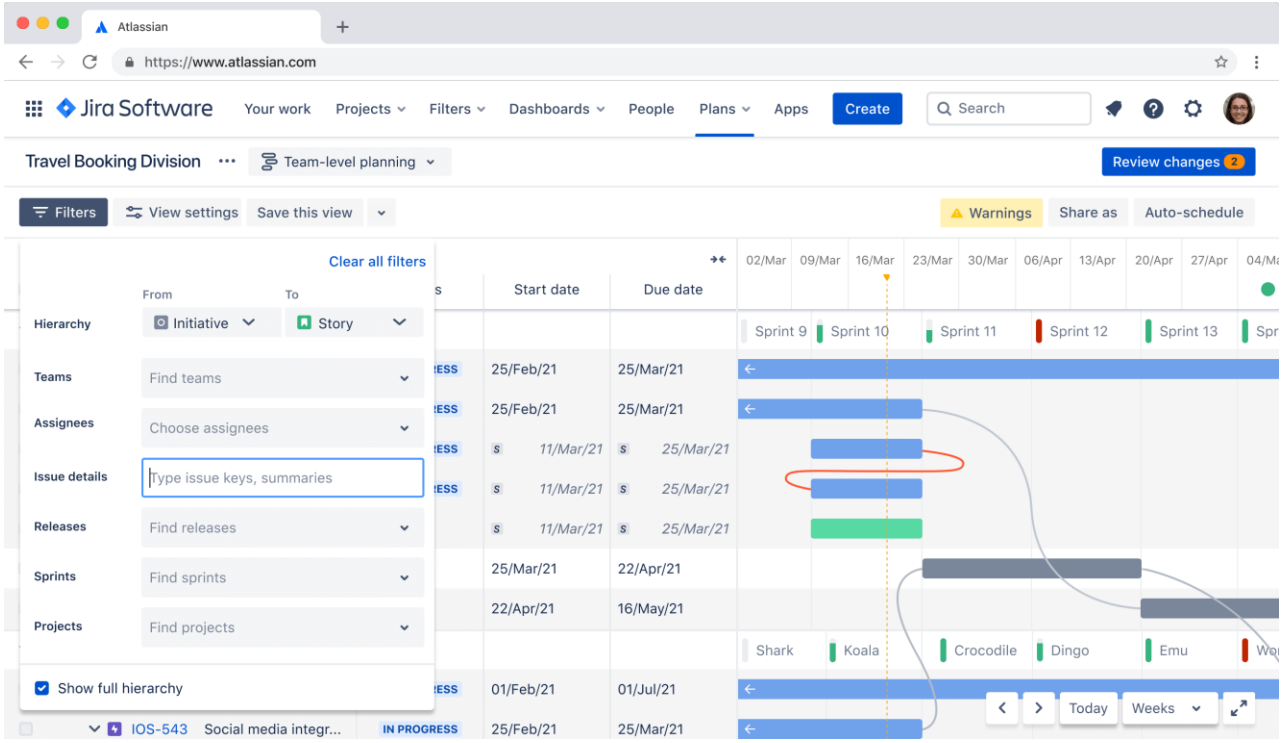
- Autentifikacija
- Autorizacija
- Enkripcija
- Dvostupanjska autentifikacija
- Sigurnosne radnje
- Sigurnosni incidenti

Prednost *Jira Software-a* u odnosu na druge alate jesu njegove robusne funkcionalnosti, integracije, praćenje projektnog napretka, sigurnost i privatnost te podrška za agilne metodologije rada.



Slika 2.1.: *Jira Software Board*

Jira Software Board (prikazano na slici broj 2.1.) korisničko je sučelje koje prikazuje projektne zadatke u obliku *kanban* ili *scrum* ploče. Zadaci su prikazani kao kartice koje se kreću kroz različite kolone na ploči, odgovarajući trenutnom statusu rada. Kolone se mogu prilagoditi prema potrebama projekta, a kartice sadrže detalje o zadatku, kao što su naziv, opis, datum zadnje promjene, dodijeljeni korisnik i druge. *Board* omogućava pregled rada, planiranje i raspodjelu zadataka te praćenje napretka rada.



Slika 2.2.*: Jira Software* sučelje

Jira Software Roadmap (prikazan na slici broj 2.2.) prikazuje planirane aktivnosti i ciljeve projekta u vidu kalendarskog prikaza. To može sadržavati više različitih planova, kao što su dugoročni ciljevi ili godišnji planovi. Svaki plan se sastoji od više faza, koje mogu sadržavati zadatke, ciljeve ili druge aktivnosti. *Roadmap* također može sadržavati informacije o prioritetima i resursima te omogućava pregled rada i praćenje napretka [1].

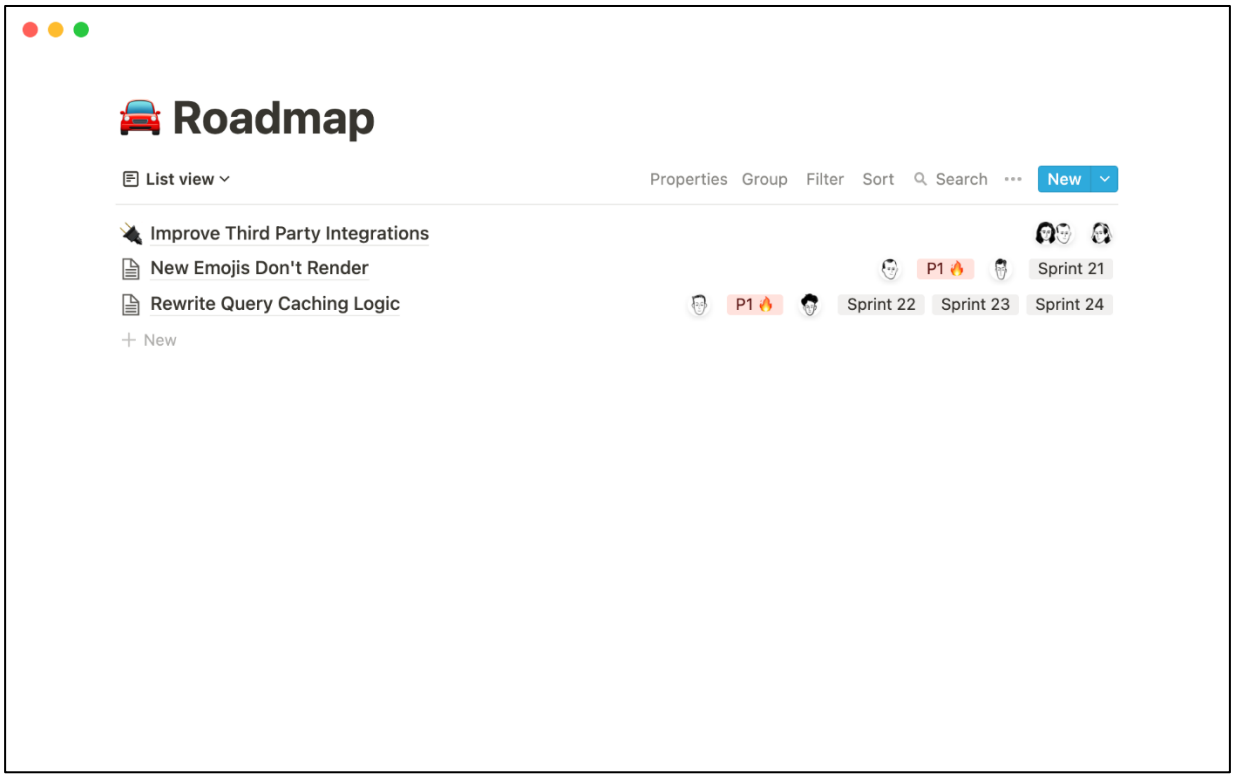
2.2. Notion

Notion Software (prikazan na slici broj 2.3.) alat je za produktivnost i organizaciju koji omogućava korisnicima da upravljaju i organiziraju svoje zadatke, bilješke, projekte i baze podataka na jednom mjestu. Prema [2] dostupan je kao aplikacija temeljena na web-u te kao aplikacija za

mobilne uređaje i desktop. *Notion* se može koristiti kao aplikacija za bilješke (eng. *notes application*), menadžer zadataka (eng. *task manager*), projekt menadžer (engl. *project manager*), a sve se to može integrirati u jednom mjestu. *Notion* omogućava kreiranje bilješki te organiziranje u različite tablice, liste ili kanban ploče. Također, omogućava kreiranje projekata, dijeljenje projekata, praćenje napretka te komunikaciju unutar tima. Osim toga, mogu se kreirati baze različitih podataka, od kontakta do projekata, članaka i više. *Notion* također podržava integraciju s drugim aplikacijama poput *Google Drive, Trello, Slack, Evernote* i drugih [2]

Notion je dobar alat za praćenje projekata zato što pruža fleksibilnost i prilagodljivost prilikom planiranja, praćenja i izvršavanja projekata.

Jira se glavnim dijelom koristi za razvoj softvera i IT timove za praćenje i upravljanje zadacima razvoja softvera, dok je *Notion* alat za produktivnost i organizaciju koji se može koristiti za širok raspon svrha kao što su zapis bilješki, upravljanje zadacima i upravljanje projektima od strane pojedinaca i timova.

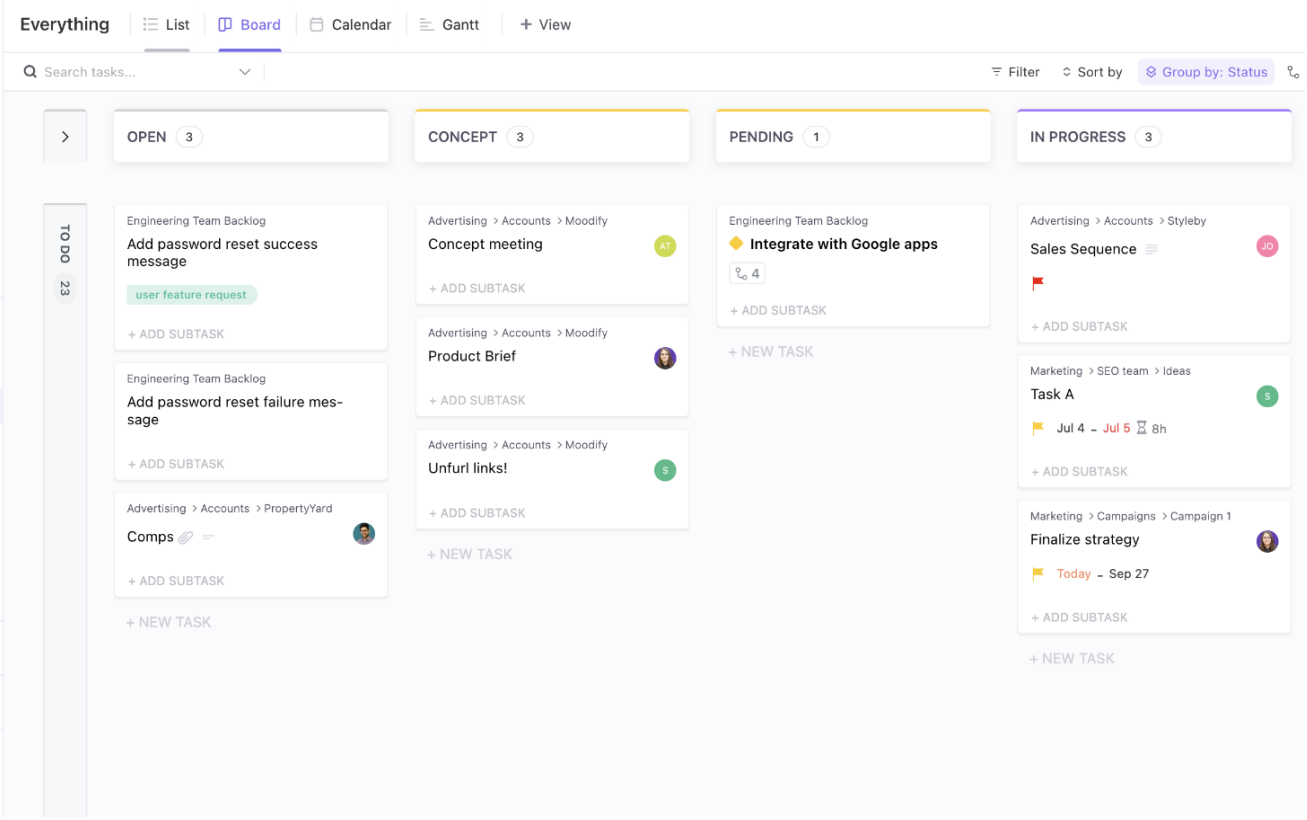


Slika 2.3.: *Notion Software Roadmap*

2.3. ClickUp

ClickUp je alat za upravljanje projektima i produktivnosti koji korisnicima omogućava da upravljaju svojim zadacima, projektima i članovima tima na jednom mjestu. *ClickUp* se može koristiti za planiranje, praćenje napretka i izvršavanje projekata te za komunikaciju unutar tima. Alat također podržava funkcionalnosti poput *Gantt* dijagrama, kalendara, podijeljenih listi zadataka i automatizaciju rada. Kao i drugi projekt management alati, *ClickUp*, isto kao i *Notion* softver, podržava integraciju s drugim aplikacijama poput *Google Drive, Trello, Slack, Evernote* i drugih. Na slici 2.4. prikazan je izgled *ClickUp boarda*.

ClickUp je dobar alat za praćenje projekata zato što pruža širok spektar funkcionalnosti koji pomažu korisnicima da planiraju, prate i efikasno izvršavaju projekte.



Slika 2.4.: *ClickUp Bord*

U usporedbi s *Jira Software-om*, *Jira* je više namijenjena za velike i kompleksne projekte, dok je *ClickUp* jednostavniji i prilagodljiviji alat za manje projekte. To omogućava korisnicima da kreiraju različite vrste zadataka, uključujući opise, ciljeve, datume isteka i druge detalje. Uz to, *ClickUp* podržava različite metodologije upravljanja projektima, poput *Scrum-a, Kanban-a* i drugih.

3. Korištene tehnologije

U ovom radu korištene su najnovije tehnologije kako bi se osigurala visoka razina funkcionalnosti i izvedbe. Programski jezik *Java*, koji je jedan od najpopularnijih i najčešće korištenih jezika za razvoj web aplikacija, korišten je kao glavni programski jezik za izradu aplikacije. Za rad s *back-endom* korišten je *Spring Framework* koji je kao i prethodno navedeno jedan od najpopularnijih okvira za razvoj *Java* aplikacija. *Spring Boot* i *Spring Data* komponente korištene su kao dodatni alati u radu sa *Spring Framework-om*. Uz korištenje ovih tehnologija, iskorištene su i druge komponente za izradu web aplikacije kako bi se osigurala optimalna funkcionalnost i izvođenje. Hibernate *ORM* korišten je za mapiranje objekata na relacijske tablice u *PostgreSQL* bazi podataka. Dodatno testiranje izvršeno je pomoću alata za automatizirano testiranje, *Postman*. Ovaj alat omogućava testiranje API-ja (engl. *Application Programming Interface*) i provjeru funkcionalnosti aplikacije prije njenog puštanja u produkciju.

Izrada preglednog dijela aplikacije odrađena je korištenjem *Reacta* zajedno s *Tailwind CSS-om*, modernim *frameworkom* za stiliziranje korisničkog sučelja. Za slanje *HTTP* zahtjeva između *frontenda* i *backenda* korišten je *Axios* što je omogućilo jednostavnu i efikasnu razmjenu podataka između komponenti aplikacije.

Autentifikacija i autorizacija unutar aplikacije osigurana je korištenjem *JJWT API-ja* koji omogućava sigurno rukovanje *JSON Web Tokenima (JWT)*. Ova kombinacija tehnologija omogućila je izradu preglednog i funkcionalnog sučelja za krajnjeg korisnika.

3.1. Programski jezik java

Java je objektno-orijentirani programski jezik koji se koristi za razvoj platformski neovisnih aplikacija. Razvijen je u Sun Microsystems-u početkom 90-ih pod vodstvom Jamesa Goslinga. Jedna od glavnih karakteristika *Java-e* je njegov objektno-orijentirani pristup što znači da se aplikacije razvijaju koristeći koncept klasa i objekata [3].

Klasa u programskom jeziku *Java* je predložak koji se koristi za stvaranje objekata u programu. Ona sadrži informacije o izgledu i funkcionalnosti objekta. Sastoji se od varijabli (znanih kao atributi) i metoda. Atributi su varijable koje opisuju objekt, a metode opisuju kako se objekt može ponašati. Klasa definira kalup za objekt i omogućava stvaranje više instanci objekta koji su svi istovjetni po atributima i metodama, ali imaju svoje vlastite vrijednosti atributa.

```
public class Razglednica {
    private String porukaRazglednice;
    private String;
    private String primateljRazglednice;

     public Razglednica (String porukaRazglednice, String
posiljateljRazglednice, String primateljRazglednice) {
        this. porukaRazglednice = porukaRazglednice;
        this. posiljateljRazglednice = posiljateljRazglednice;
        this. primateljRazglednice = primateljRazglednice;
    }

     public String getPorukaRazglednice() {
        return porukaRazglednice;
    }
     public String getPosiljateljRazglednice() {
        return posiljateljRazglednice;
    }

    public String getPrimateljRazglednice() {
        return primateljRazglednice;
    }
}
```

Slika 3.1.*: Klasa napisana u Java programskom jeziku s atributima, konstruktorom i metodama*

Klasa "Razglednica", koja je prikazana na slici 3.1., sadrži tri privatne varijable: *porukaRazglednice*, *posiljateljRazglednice* i *primateljRazglednice* te tri javne metode za dohvaćanje vrijednosti tih varijabli. Klasu se može proširiti dodatnim metodama za postavljanje varijabli, parametarske konstruktore te ostale metode koje se mogu pozivati nad metodama unutar klase, kao što je metoda *toString* koja bi vraćala objekt kao String.

Java programski jezik se temelji na tri ključna koncepta objektno orijentiranog pristupa, a to su enkapsulacija, nasljeđivanje i polimorfizam, prema [3].

Enkapsulacija je mehanizam za zaštitu podataka u objektno orijentiranom programiranju. Ona omogućava da se podaci unutar objekta čuvaju privatno što znači da se ne mogu pristupiti direktno izvan objekta. Umjesto toga, objekt definira metode koje omogućuju pristup i manipulaciju tih podataka što se zove *"getteri"* i *"setteri"*. Ova metoda omogućava da se podaci unutar objekta mijenjaju i pristupaju na siguran način što smanjuje mogućnost grešaka u *kodu*.

Nasljeđivanje je još jedan temeljni princip objektno orijentiranog programiranja koji omogućava novoj klasi da naslijedi svojstva i metode postojeće klase. Postojeća klasa se naziva osnovna klasa ili roditeljska klasa, dok se nova klasa naziva izvedena klasa ili djetetova klasa. Kroz nasljeđivanje, dijete klasa nasljeđuje atribute i ponašanja roditeljske klase(ovisno o pristupnom modifikatoru) te također može dodavati nove atribute i ponašanja. To omogućava ponovnu uporabu *koda*, smanjuje

količinu *koda* koji treba napisati i olakšava održavanje. Nasljeđivanje također omogućava hijerarhijsku vezu između klasa, gdje izvedena klasa može naslijediti osnovnu klasu koja može naslijediti još jednu osnovnu klasu, stvarajući lanac nasljeđivanja.

Polimorfizam se odnosi na sposobnost objekta da preuzme višestruke oblike. Drugim riječima, jedan objekt se može tretirati kao primjerak osnovne klase, sučelja ili izvedene klase. Polimorfizam omogućava objektima različitih tipova da se tretiraju kao objekti zajedničke osnovne klase što omogućava pisanje *koda* u više generički i ponovljiv način. Postoje dva glavna tipa polimorfizma:

- Statički polimorfizam (također poznat kao polimorfizam u trenutku kompilacije) koji se postiže preko preopterećenja funkcija i operatora
- Dinamički polimorfizam (također poznat kao polimorfizam u trenutku izvođenja) koji se postiže preko virtualnih funkcija i apstraktnih klasa.

Ukratko, polimorfizam je vrlo važan koncept objektno-orijentiranog programiranja koji omogućava jednoj funkciji ili metodi da radi s više tipova objekata, čime *kod* postaje fleksibilniji, čitljiviji i održiviji [3].

Ovaj programski jezik ima visoku razinu sigurnosti što ga čini idealnim za razvoj aplikacija koje rade sa osjetljivim podacima. Podrška za automatsko upravljanje memorijom također omogućava razvoj aplikacija koje su manje sklone greškama i koje se lakše održavaju. Podržava mrežno programiranje što omogućava razvoj aplikacija koje mogu komunicirati preko mreže i koje se mogu koristiti za različite vrste komunikacije, kao što su web servisi, web aplikacije i klijent-poslužitelj aplikacije.

Ovaj programski jezik ima veliku i aktivnu zajednicu koja doprinosi razvoju različitih biblioteka i alata koji olakšavaju razvoj aplikacija. To uključuje popularne *framework*-ove poput *Spring Framework-a, Hibernate-a* i *Apache Struts-a*. Često se koristi za razvoj *enterprise* aplikacija, web aplikacija, mobilnih aplikacija, igara i drugih aplikacija. To je vrlo popularan programski jezik koji se koristi u različitim industrijama, kao što su financijski sektor, telekomunikacije, trgovina i dr. *Java* je također jedan od jezika koji se koristi za razvoj *Android* aplikacija te videoigara pomoću dodatnih biblioteka kao što su LibGDX i LWJGL (*Lightweight Java Game Library*). U posljednjih nekoliko godina, *Java* je prešla na raspored izdavanja novih verzija svakih šest mjeseci. Najnovija verzija, *Java* 22, objavljena je u ožujku 2024. godine, ali pripada kategoriji *Feature* verzija, koje ne dobivaju dugoročnu podršku. Verzije s dugoročnom podrškom, poznate kao LTS verzije (*Long-*

Term Support), objavljuju se u intervalima od nekoliko godina. Dosadašnje LTS verzije uključuju *Java* 8, *Java* 11, *Java* 17 i najnoviju *Java* 21, koje su namijenjene okruženjima koja zahtijevaju stabilnost i kontinuirane sigurnosne nadogradnje kroz dulji vremenski period [4].

3.2. Spring razvojni okvir

Spring je *Java* aplikacijski okvir koji se koristi za razvoj softvera. On pruža skup alata i komponenti za pomoć u razvoju aplikacija, kao što su upravljanje zavisnostima, automatizirano testiranje, rad s bazama podataka i rad s web servisima. *Spring* se fokusira na pružanje modularnosti, skalabilnosti i performansi u aplikacijama te pruža podršku za različite arhitekture, poput mikroservisa i *cloud-native* arhitektura [5].

Spring Framework se sastoji od više komponenti koje se mogu koristiti prema potrebi. *Spring Core* osnovni je dio okvira koji pruža podršku za upravljanje zavisnostima i IoC (eng. *inversion of control*) princip. *Spring Web* komponente se koriste za razvoj web aplikacija, uključujući podršku za rad s web servisima i web MVC (engl. *Model–View–Controller*) arhitekturu. *Spring Data* se koristi za rad s bazama podataka i podršku za rad s različitim tehnologijama za perzistenciju podataka kao što su JPA, MongoDB i Redis [5].

Spring Security se koristi za sigurnosne zahtjeve aplikacije, a *Spring Batch* se koristi za automatizirane procese i rad sa velikim količinama podataka. *Spring Boot* je okvir koji se koristi za brz i jednostavan razvoj aplikacija na temelju *Spring Framework-a*.

Spring Framework također pruža podršku za različite tehnologije kao što su *JavaServer Faces (JSF), Portlet, Android i Grails*. *Spring Framework* se često koristi u razvoju *enterprise* aplikacija zbog svoje skalabilnosti, modularnosti i performansi [6].

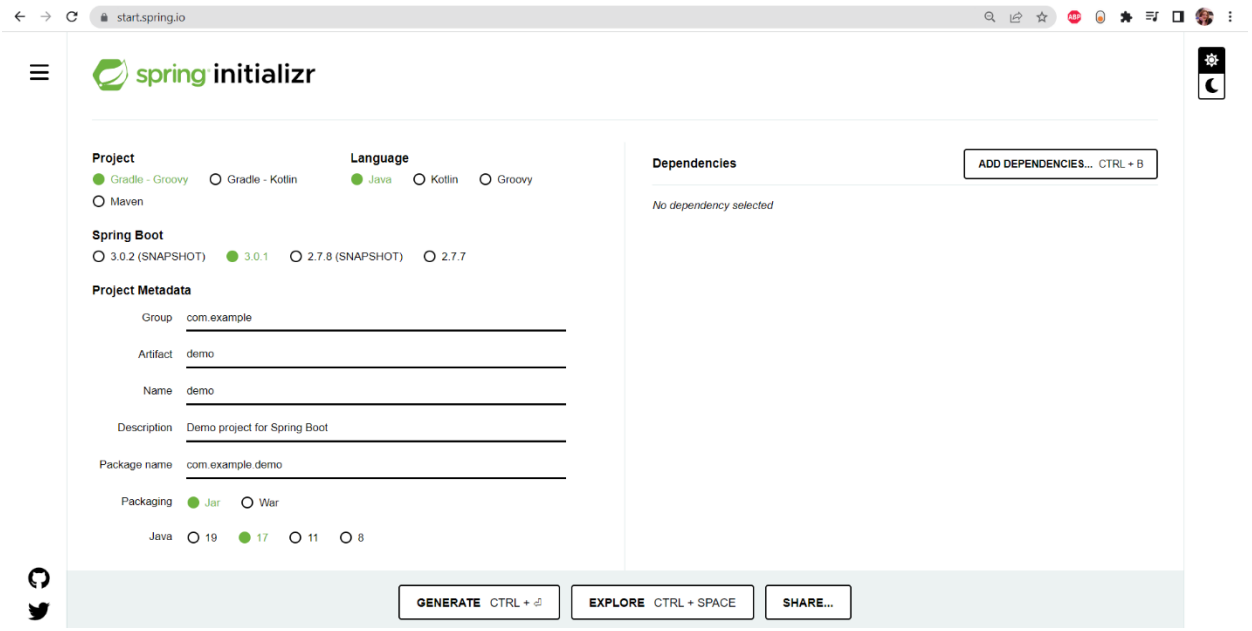
Spring Core je osnovna komponenta *Spring Framework-a* koja pruža podršku za upravljanje zavisnostima i IoC princip. Ovaj princip se koristi kako bi se objekti i njihove zavisnosti kontrolirali od strane *Spring Framework-a*, umjesto da se kontrola prebaci na objekte. To omogućava bolje održavanje i testiranje aplikacije te fleksibilnost u korištenju različitih tehnologija. *Spring Core* također pruža podršku za rad s različitim tehnologijama kao što su JDBC (eng. *Java Database Connectivity*) za rad s bazama podataka, ORM (eng. *Object-Relational Mapping*) za rad s objektima i bazama podataka, JMS (eng. *Java Message Service*) za rad s porukama i druge tehnologije. Osim toga, *Spring Core* pruža podršku za rad s oznakama (engl.

Annotation) koje se koriste za označavanje klasa, metoda i varijabli korištenih u aplikaciji. Ukratko, *Spring Core* je osnovna komponenta koja omogućava upravljanje zavisnostima, IoC i podršku za različite tehnologije u *Spring Framework-u*.

Spring Web je skup komponenti koje se koriste u *Spring Framework-u* za razvoj web aplikacija. To uključuje podršku za rad s web servisima i web MVC arhitekturu. Spring Web MVC je implemetacija MVC arhitekture koja se koristi za razvoj web aplikacija u *Spring Framework-u.* Ona omogućava razdvajanje logike aplikacije od prezentacije kroz korištenje kontrolera, modela i pogleda [5].

Spring Boot je okvir koji se koristi za brz i jednostavan razvoj aplikacija na temelju *Spring Framework-a*. Koristi se za automatizaciju procesa konfiguriranja i pokretanja aplikacije te smanjenje kompleksnosti konfiguracije. Pruža gotove konfiguracije za različite komponente *Spring Framework-a* što omogućava brži razvoj aplikacije i smanjenje kompleksnosti konfiguracije. Osim toga, *Spring Boot* pruža gotove konfiguracije za rad sa različitim tehnologijama, kao što su baze podataka, sigurnost, web servisi i drugo. Ovo omogućava razvoj aplikacije bez potrebe za ručnim konfiguriranjem svake komponente. Također omogućava brzo pokretanje i testiranje aplikacije što ga čini idealnim za razvoj brzih prototipova i PoC-ova [7].

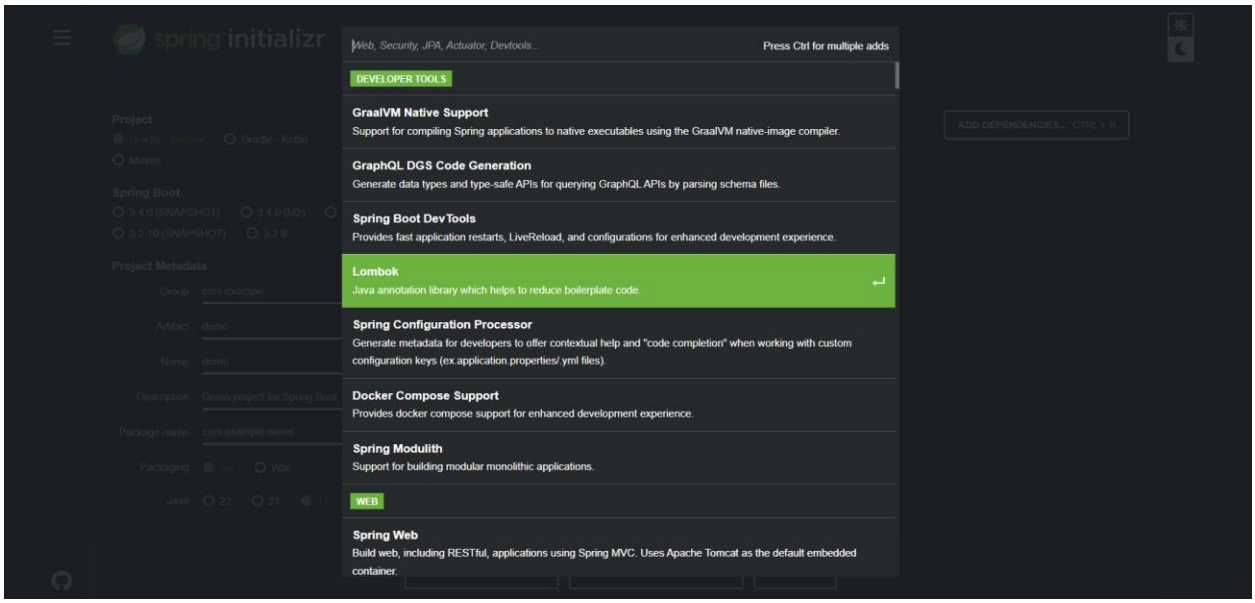
Spring Boot Initializr (prikazan na slici 3.2.) je web aplikacija koja omogućava razvojnim timovima da brzo generiraju novi *Spring Boot* projekt s potrebnim zavisnostima i konfiguracijama. Korisnik može odabrati željene zavisnosti i konfiguracije, kao što su odabir glavnog jezika, verzija *Spring Boot-a*, i druge. Nakon odabira, *Spring Boot Initializr* će generirati projekt u obliku ZIP arhive ili direktorija koji se može importirati u razvojno okruženje. Ovaj alat pruža jednostavnost i brzinu u kreiranju novog projekta što povećava produktivnost i ubrzava razvoj [7].



Slika 3.2.: *Spring Boot Inizializr*

Prilikom kreiranja novog projekta u *Spring Boot Initializr*. Moguće je odabrati nekoliko komponenti:

- Projekt- vrsta projekta koji želite kreirati, kao što su *Maven* ili *Gradle* projekt.
- Glavni jezik: glavni programski jezik koji će se koristiti u projektu, kao što su *Java, Kotlin* ili *Groovy*.
- Verzija *Spring Boot-a*- verzija *Spring Boot* okvira koja će se koristiti u projektu.
- Zavisnosti- različite zavisnosti koje će se koristiti u projektu, kao što su *Spring Web, Spring Data JPA, Spring Security* i druge.
- Konfiguracije- različite konfiguracije koje će se koristiti u projektu, kao što su konfiguracija za rad s bazama podataka, autentikaciju i autorizaciju, i druge.



Slika 3.3*: Spring Boot Inizializr: prikaz zavisnosti koje se mogu dodati u budući projekt*

Nakon odabira komponenti, *Spring Boot Initializr* će generirati projekt sa svim odabranim komponentama i zavisnostima te se na taj način olakšava i ubrzava proces kreiranja novog projekta (Slika 3.3.). Razvojnim timovima se omogućava da počnu s već postavljenim konfiguracijama i zavisnostima što smanjuje vrijeme potrebno za postavljanje okruženja i konfiguraciju projekta.

Spring Security je moćna i visoko podesiva komponenta koja se koristi za sigurnost web aplikacija temljenih na *Java-i*. Pruža kompletno sigurnosno rješenje za web. Izgrađena je na *Spring framework-u* što omogućava lako integriranje u postojeće aplikacije. Koristi se za autentifikaciju i autorizaciju korisnika, kontrolu pristupa zaštićenim dijelovima aplikacije, kao i zaštitu od napada poput CSRF(eng. *Cross-Site Request Forgery*) i SQL (eng. *Structured Query Language*) *injection*. Također, podržava rad s različitim autentikacijskim mehanizmima kao što su form-based, basic, *OAuth* i druge [6].

Spring Data je komponenta koja se koristi za povezivanje *Java* aplikacija s različitim vrstama baza podataka. Omogućava jednostavniji rad s bazama podataka, smanjujući količinu *koda* potrebnog za rad s bazama podataka. Temelji se na *Spring Framework-u* i dolazi s nizom projekata koji se fokusiraju na specifične vrste baza podataka (npr. *Spring Data JPA* za rad s bazama podataka relacijskog tipa, *Spring Data MongoDB* za rad s *NoSQL* bazama podataka).

Spring Batch je komponenta koja se koristi za automatizaciju serijskih poslova u *Java* aplikacijama. Ona omogućava razvojnim timovima lako kreiranje, pokretanje i praćenje velikih

količina podataka (*batch)* procesa. Temelji se na *Spring Framework-u* i dolazi s nizom alata za rad sa podacima kao što su čitanje i pisanje podataka, procesiranje podataka, i drugo.

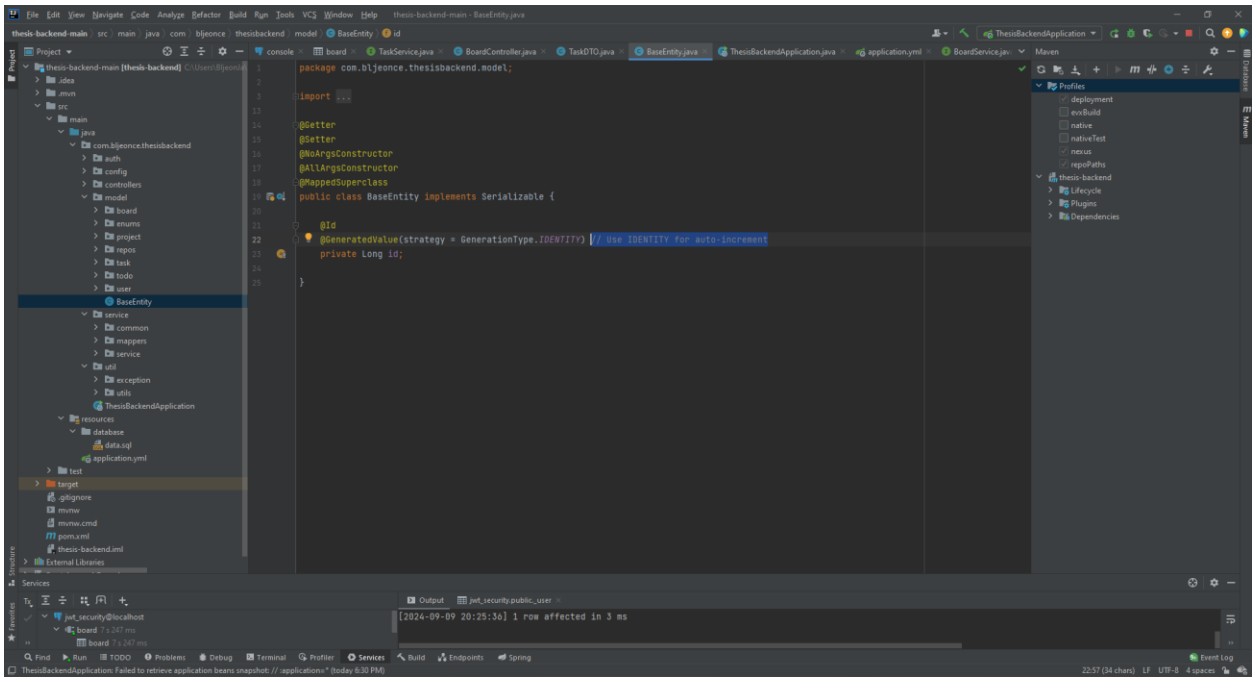
Isto tako, podržava različite vrste procesiranja podataka kao što su paralelno procesiranje, podrška za rad s različitim izvorima podataka, kao i mogućnost rada s različitim formatima podataka. Također, podržava rad s različitim vrstama poslova kao što su procesiranje transakcija, analiza podataka i drugo. *Spring Batch* je izvanredan alat za automatizaciju serijskih poslova i vrlo koristan za razvojne timove koji se suočavaju s velikim količinama podataka [6].

3.3. IntelliJ Idea

IntelliJ IDEA je moćno integrirano razvojno okruženje (IDE, engl. *Integrated Development Environment*) koje razvojni timovi širom svijeta često koriste. Omogućava automatizaciju različitih zadataka kao što su generiranje *koda*, refaktoriranje, otklanjanje pogreška (eng. *debugging*) i testiranje što povećava produktivnost i ubrzava razvoj. Sučelje je intuitivno i lako se koristi (Slika 3.4.), a podržava integraciju s različitim sustavima za verzioniranje kao što su *Git, SVN, Mercurial* i druge.

IntelliJ IDEA također ima razne alate za debugging i profiliranje što olakšava otkrivanje i rješavanje problema u *kodu*. Podržava različite programske jezike, uključujući *Java, Scala, Groovy, Kotlin, JavaScript, Python* i druge.

Fleksibilnost ovog alata omogućava razvojnim timovima da prilagode razvojno okruženje prema svojim potrebama. Na primjer, može se integrirati dodatak (engl. *plugin*) za specifičan alat za testiranje ili različite alate za upravljanje verzijama. Razni dodaci (engl. *plugin*) dizajnirani su kako bi se povećala produktivnost i efikasnost programera. To može značiti korištenje boljih navigacijskih alata, automatsko nadopunjavanje *koda* ili alata za refaktoriranje već postojećeg *koda*.



Slika 3.4.: *IntelliJ IDEA sučelje*

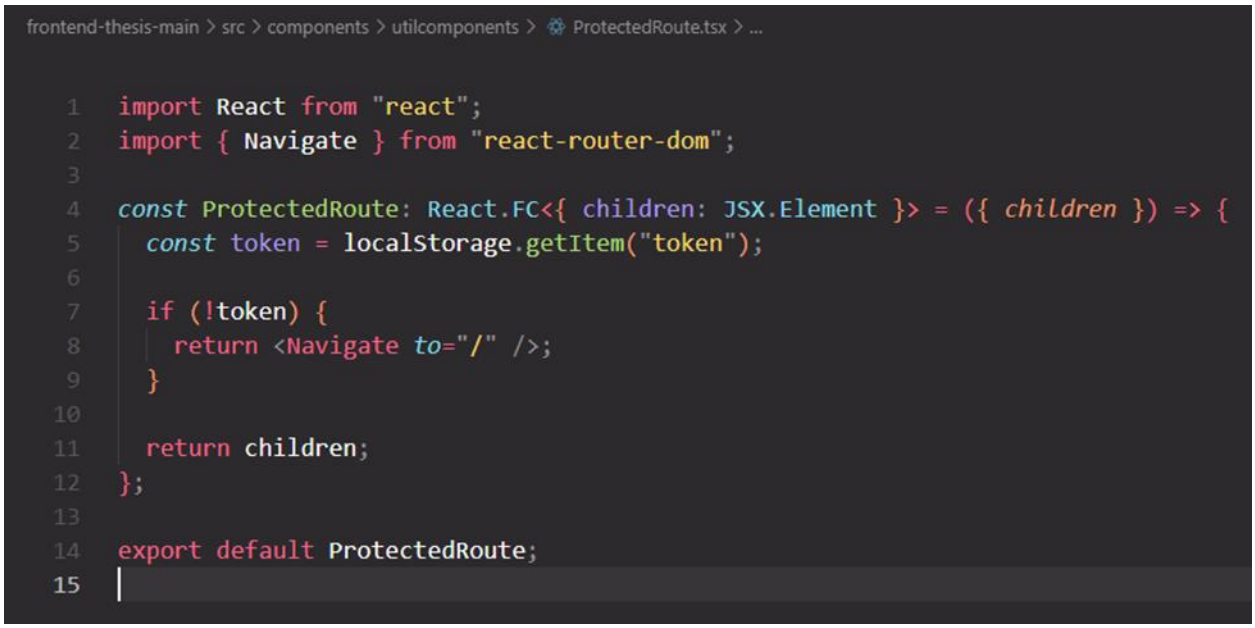
IntelliJ IDEA je izvanredan alat za razvoj softvera koji olakšava rad razvojnim timovima. Uključivanje ovog alata u razvojni proces može značajno poboljšati kvalitetu i brzinu proizvodnje softvera.

3.4. Ostale korištene tehnologije

U ovom poglavlju opisane su ostale tehnologije koje su korištene u razvoju ove aplikacije.

3.4.1. React

React je biblioteka za *JavaScript* otvorenog *koda* koja je prvenstveno dizajnirana za kreiranje korisničkih sučelja, s posebnim naglaskom na jednostranične aplikacije (*SPA, engl. Single Page Application*). *React* je postao jedan od vodećih alata za razvoj *web* aplikacija zahvaljujući svojoj prilagodljivosti, skalabilnosti i lakoći integracije u postojeće projekte. Ono što ga čini krajnje atraktivnim za programere je njegova sposobnost modularnog razvoja aplikacija putem komponentnog modela što omogućava jednostavnije održavanje i ponovno korištenje *koda*. Jedan od glavnih principa *Reacta* je komponentni pristup koji programerima omogućava stvaranje aplikacija podijeljenih na manje, nezavisne dijelove, poznate kao komponente. Primjer komponente koja se koristi u aplikaciji prikazana je na slici 3.5. [8].



Slika 3.5: *Primjer React komponente*

Svaka komponenta sadrži *HTML*, *CSS* i *JavaScript* logiku, čineći ih samostalnim i ponovljivim jedinicama unutar aplikacije. Ovakav način organizacije *koda* ne samo da pojednostavljuje razvoj već omogućava i lakše održavanje te nadogradnju aplikacija s novim funkcionalnostima. Komponente također mogu imati vlastito stanje (engl. *state*), koje upravlja njihovim ponašanjem te putem svojstava (*props*) komunicirati s drugim komponentama. Ovakva modularnost pruža veću fleksibilnost u dizajniranju kompleksnih aplikacija [8].

Jedna od najvažnijih inovacija koje *React* donosi u *web* razvoj je *Virtual DOM* (Virtualni *Document Object Model*). Umjesto izravnog rada s pravim *DOM-om* što može biti vrlo sporo kod velikih aplikacija, *React* koristi virtualnu verziju *DOM-a* koja služi kao privremena kopija. Kada dođe do promjene u aplikaciji, *React* prvo ažurira *Virtual DOM*, zatim izračuna minimalni broj izmjena koje su potrebne da se naprave u stvarnom *DOM-u*. Ovaj proces poznat kao usklađivanje(engl. *reconciliation*), omogućava brže ažuriranje korisničkog sučelja i poboljšava ukupne performanse aplikacije što rezultira boljim korisničkim iskustvom [9].

```javascript
import React, { useState } from 'react';

function Brojac() {
    // Definiranje stanja (state) s početnom vrijednosti 0
    const [broj, setBroj] = useState(0);

    // Funkcija za povećanje broja
    const povecajBroj = () => {
        setBroj(broj + 1);
    };

    console.log('Virtual DOM ažuriran - novi broj:', broj);

    return (
        <div>
            <h1>Trenutni broj: {broj}</h1>
            <button onClick={povecajBroj}>Povećaj broj</button>
        </div>
    );
}

export default Brojac;
```

Slika 3.6. *Prikaz načina rada sa Virtual DOM*

Na prethodnoj slici 3.6. prikazano je kako React ažurira Virtual DOM uspoređujući promjene sa stvarnim DOM-om. Ovaj pristup omogućava optimizirano i efikasno ažuriranje prikaza u aplikaciji, budući da se samo one promjene koje su zaista potrebne primjenjuju na stvarnom DOM-u. U ovom primjeru, promjena vrijednosti broja koja se događa nakon pritiska na gumb ažurira se prvo u Virtual DOM-u, a potom u stvarnom DOM-u, čime se osigurava visoka učinkovitost u radu s većim aplikacijama [8].

React također uvodi koncept jednosmjerne propusnosti podataka što znači da podaci teku u jednom smjeru – od roditeljske komponente prema njezinim podređenim komponentama. Jednosmjerna propusnost podataka u *Reactu* osigurava da podaci teku od roditeljskih prema podređenim komponentama što pojednostavljuje praćenje promjena i ponašanje aplikacije. Budući da se svi podaci i promjene kontroliraju s jednog centralnog mjesta, olakšava se kasnije održavanje *koda*.

Za aplikacije koje trebaju globalno upravljanje stanjem, gdje podaci moraju biti lako dostupni na različitim razinama komponenata, koriste se alati poput *Redux*-a ili *Context* API-ja. Ovi alati

omogućuju dijeljenje stanja među komponentama bez potrebe za slanjem podataka putem *props*-a kroz svaki nivo hijerarhije komponenti. To uvelike pojednostavljuje upravljanje stanjem u većim aplikacijama, omogućujući centralizirano praćenje i izmjene. Na taj način se izbjegava kompliciranо propuštanje podataka kroz više razina komponenata i olakšava održavanje aplikacije, a posebno kad se radi o velikim sustavima.

React Router je jedan od ključnih alata koji omogućava navigaciju unutar jednostraničnih aplikacija (SPA) razvijenih u *Reactu*. Njegova osnovna funkcija je omogućiti definiciju različitih URL-ova unutar aplikacije, čime se korisniku pruža osjećaj višestranične aplikacije bez potrebe za stalnim osvježavanjem stranice. Ova tehnologija omogućava glatku navigaciju između različitih dijelova aplikacije simulirajući tako korisničko iskustvo tradicionalnih višestraničnih web stranica unutar jednog dokumenta. *React Router* omogućava mapiranje svakog URL-a na određenu komponentu čime se prikazuje odgovarajući sadržaj unutar aplikacije na temelju korisnikovih akcija, a sve bez ponovnog učitavanja stranice [8].

React Router funkcionira na principu virtualne navigacije, gdje se promjena URL-a ne odnosi na učitavanje nove stranice, već na dinamičko ažuriranje prikaza unutar aplikacije. Ovakav pristup značajno ubrzava performanse aplikacije i pruža bolje korisničko iskustvo jer se prikazuju samo one komponente koje su potrebne za trenutni prikaz, bez nepotrebnog ponovnog učitavanja cijele stranice. Osim toga, *React Router* omogućava implementaciju ugniježđenih ruta gdje jedna ruta može imati pod-rute što omogućava dublju hijerarhiju prikaza i bolju organizaciju aplikacije. Primjer implementacije osnovne navigacije u *Reactu* pomoću *React Routera* uključuje definiranje glavnih ruta (*path*) i povezivanje tih ruta s odgovarajućim komponentama. To se postiže pomoću *Route* komponente koja mapira određeni URL na odgovarajuću komponentu aplikacije. Unutar aplikacije koristi se *Link* komponenta za kreiranje navigacijskih elemenata čime se omogućava kretanje između različitih URL-ova bez osvježavanja stranice [8].

Osim već navedenog, *React* je poznat i po svojoj deklarativnoj prirodi. Programeri definiraju kako bi korisničko sučelje trebalo izgledati u određenom stanju, dok se *React* brine o ažuriranju sučelja kada dođe do promjena. Ovaj pristup omogućava programerima potpuno koncentriranje na logiku aplikacije, dok *React* automatski upravlja ažuriranjem sučelja na temelju zadane logike.

Zbog ogromne zajednice koja aktivno radi na razvoju *Reacta*, programeri imaju pristup raznim alatima i bibliotekama koje dodatno olakšavaju razvoj. Na primjer, *React DevTools* omogućava programerima da jednostavno pregledavaju strukturu komponenata i njihovo stanje unutar preglednika što je izuzetno korisno za otklanjanje grešaka i analizu ponašanja aplikacije [9].

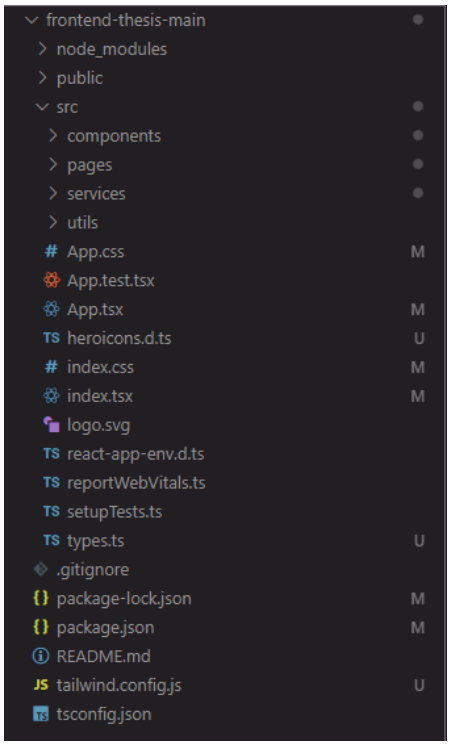
React je postao standard za razvoj modernih *web* aplikacija zbog svoje prilagodljivosti, brzine i sposobnosti modularne izgradnje aplikacija. Njegova jednostavnost korištenja, zajedno s naprednim tehnikama kao što su *Virtual DOM* i jednosmjerna propusnost podataka čini ga idealnim izborom za izradu skalabilnih i dinamičnih aplikacija. Alati poput *Tailwind CSS-a, Axios-a* i *React Router-a* omogućuju brz i efikasan razvoj estetski ugodnih i funkcionalnih korisničkih sučelja. Za izradu nove React aplikacije koristi se alat pod nazivom *Create React App*.

```
npx create-react-app naziv-aplikacije
```

Slika 3.7. *Naredba kreira osnovni predložak za aplikaciju s prilagođenim nazivom*

Naredba prikazana na slici 3.7. kreira osnovni predložak za aplikaciju, postavlja strukturu direktorija te instalira potrebne pakete poput *Reacta, Webpacka* i *Babel*-a. Rezultat je spreman projekt s prethodno definiranom konfiguracijom koja omogućava brz početak razvoja [9].

Nakon što je projekt generiran, osnovna struktura React aplikacije obično izgleda prema slici 3.8.



Slika 3.8. *Struktura projekta za diplomski i prikaz osnovne strukture React projekta*

Direktorij *node_modules*/ sadrži sve ovisnosti koje je projekt instalirao putem paketa, poput *React* biblioteke i drugih potrebnih modula. Ovaj direktorij se automatski kreira kada se ovisnosti

instaliraju koristeći npm (*Node Package Manager*). Svaka biblioteka ili alat koje projekt koristi nalazi se unutar ove mape, a budući da se ovisnosti često ažuriraju ovaj direktorij se izostavlja iz verzioniranja *koda* što je specificirano u datoteci *.gitignore*.

Direktorij *public*/ pohranjuje statičke datoteke koje aplikacija koristi. Najvažnija datoteka u ovom direktoriju je index.html, osnovni HTML dokument u koji *React* aplikacija 'ugrađuje' svoj sadržaj. Unutar ovog direktorija nalaze se i resursi poput slika, *favicon* ikona te drugih elemenata koji se ne mijenjaju tijekom rada aplikacije, već ostaju statični kroz cijeli proces izvršavanja aplikacije.

Direktorij src/ predstavlja srž aplikacije u kojem su sadržane sve ključne datoteke i kodovi koji omogućavaju rad *React* aplikacije. Tu se nalaze sve komponente, logika i funkcionalnosti aplikacije. Unutar src/ direktorija nalaze se različite podmape i datoteke koje služe specifičnim ulogama unutar projekta.

components/ direktorij pohranjuje sve React komponente koje čine osnovne gradivne blokove aplikacije. Svaka komponenta pokriva specifični dio korisničkog sučelja te tako omogućava modularni razvoj i višekratnu upotrebu *koda*. Direktorij *pages/* sadrži kompletne stranice povezane s određenim URL-ovima unutar aplikacije, dok se za navigaciju između tih stranica koristi alat *React Router*. Stranice se dinamički učitavaju bez potrebe za ponovnim osvježavanjem aplikacije čime se poboljšava korisničko iskustvo.

Osim komponenti i stranica, tu su i *services*/ datoteke koje upravljaju logikom komunikacije s vanjskim API-ima. One sadrže funkcije koje šalju zahtjeve prema poslužiteljima i dohvaćaju podatke, a zatim te podatke prosljeđuju u aplikaciju. Pomoćne funkcije koje su korisne na različitim mjestima u aplikaciji pohranjene su u *utils*/ direktoriju. Ove funkcije mogu obuhvaćati validacije, manipulacije podacima ili druge ponovljive operacije.

Glavna datoteka aplikacije, *App.tsx*, sadrži osnovnu logiku aplikacije i služi kao središte u kojem se renderiraju sve ostale komponente. Kroz ovu datoteku upravlja se ključnim funkcionalnostima aplikacije, poput navigacije i globalnog stanja. Uz nju, *index.tsx* je ulazna datoteka koja povezuje *React* aplikaciju s osnovnim index.html dokumentom te omogućava inicijalizaciju aplikacije unutar DOM-a.

.gitignore je datoteka koja specificira koje direktorije i datoteke *Git* treba ignorirati prilikom verzioniranja *koda*. Ova datoteka često isključuje *node_modules*/ direktorij kao i druge privremene ili konfiguracijske datoteke koje nisu potrebne za sam rad aplikacije. Time se izbjegava nepotrebno dodavanje velikih datoteka u verzijski sustav.

package.json je konfiguracijska datoteka koja sadrži ključne informacije o projektu uključujući popis ovisnosti koje projekt koristi te skripte za pokretanje ili izgradnju aplikacije. Svaka promjena u instaliranim paketima bilježi se u ovoj datoteci što omogućava drugim programerima da jednostavno instaliraju sve potrebne ovisnosti kada preuzmu projekt.

tsconfig.json je konfiguracijska datoteka specifična za *TypeScript* koja definira postavke za tipizaciju i pravila u projektu. Korištenje *TypeScripta* omogućava statičku tipizaciju u JavaScript *kodu* čime se smanjuje mogućnost grešaka, a aplikacija postaje stabilnija i lakša za održavanje.

Ova struktura direktorija unutar projekta omogućava vrlo djelotvornu organizaciju *React* projekata te čini razvoj aplikacije višesegmentnim, skalabilnim i jednostavnim za održavanje [8].

3.4.2. Typescript

TypeScript je nadogradnja nad jezikom *JavaScript* koja unosi statičke tipove podataka u razvoj aplikacija čime se omogućava veća sigurnost i pouzdanost u radu s podacima. Prema [10], *TypeScript* unosi strukturu i tipove u generalno dinamičan jezik kao što je *JavaScript*, pružajući programerima mogućnost da unaprijed definiraju tipove varijabli, funkcija i objekata.

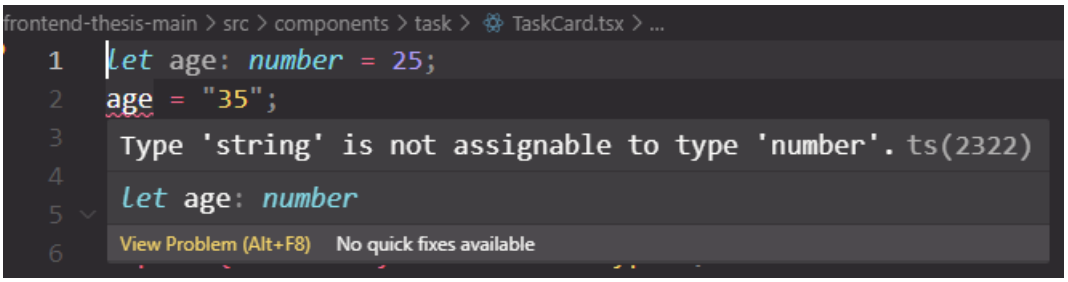
Statičko definiranje tipova objekata omogućava brže prepoznavanje grešaka i tijekom razvoja, umjesto da se problemi otkrivaju tek u vrijeme izvršavanja *koda*. *TypeScript* je postao ključan alat u razvoju velikih i složenih web aplikacija, posebno onih programera koji koriste razvojne okoline kao što su *Angular* ili *React*.

Na primjeru jednostavne *React* komponente koja je prikazana na slici broj 3.5., može se vidjeti kako *TypeScript* olakšava rad s podacima. Komponenta *TaskCard* koja je napisana u *TypeScriptu*, koristi jasno definirane tipove za svoje *props-e* što omogućava sigurnije i predvidljivije korištenje podataka unutar aplikacije [10].

Slika 3.9. *Primjeru jednostavne React komponente*

U primjeru na slici 3.9. na kojoj se prikazuje *kod* iz aplikacije ovog diplomskog rada, *TypeScript* koristi *interface* kako bi se definirali očekivani tipovi podataka za komponentu. *TaskCardProps* interface specificira da komponenta mora primiti dva *prop*-a: *task*, koji je tipa *TaskDTO*, i *onClick*, koji je funkcija što prima objekt tipa *TaskDTO* kao argument. Ovakvo definiranje tipova podataka pomaže u sprječavanju grešaka tijekom razvoja, jer se unaprijed definira kakvi podaci moraju biti proslijeđeni komponenti. Također, statičko definiranje tipova podataka omogućava alatima poput integriranih razvojnih okruženja (*IDE*) pružanje bolje povratne informacije u stvarnom vremenu, poput predviđanja tipova, automatskog dovršavanja i prijave pogrešaka.

Prednosti korištenja *TypeScripta* u *React* aplikacijama ne leže samo u sigurnosti kodiranja, već i u boljoj održivosti projekta na duži rok. Budući da *TypeScript* uvodi formalna pravila o vrstama podataka, znatno je lakše razumjeti logiku i strukturu *koda*, čak i kada na projektu radi veći tim programera ili kada je prošlo neko vrijeme od početka razvoja. Svaka promjena u tipu podatka odmah će biti detektirana od strane kompajlera što smanjuje mogućnost nepredviđenih grešaka u složenim sustavima. U praksi (Slika 3.10.) ovo znači da ako je varijabla definirana kao *number*, kompajler neće dopustiti da se toj varijabli dodijeli vrijednost tipa *string* ili bilo koji drugi nesukladni tip [10].



Slika 3.10. *Prikaz korištenja tipova podataka u Typescript-u*

Osim što pomaže u razvoju pouzdanog *koda*, *TypeScript* omogućava jednostavnu integraciju s postojećim *JavaScript* alatima i bibliotekama. Programeri koji već koriste JavaScript mogu lako uvesti *TypeScript* u svoj projekt, jer je kompatibilan sa svim modernim JavaScript značajkama. Nije potrebno potpuno napustiti postojeći *JavaScript kod* – *TypeScript* omogućava postupnu migraciju, čime se smanjuje rizik prilikom prelaska na statički tipizirani jezik. [10].

Uz sve navedeno, *TypeScript* u kombinaciji s *Reactom* omogućava razvoj složenih, interaktivnih web aplikacija koje su stabilne i lako održive. *TypeScript* pruža dodatnu sigurnost pri radu s komponentama, funkcijama i podatkovnim strukturama, a njegova fleksibilnost čini ga idealnim izborom za moderne aplikacije. Zbog svih ovih prednosti, *TypeScript* je danas nezaobilazni alat u razvoju složenih sustava temeljenih na *JavaScriptu* [9, 10].

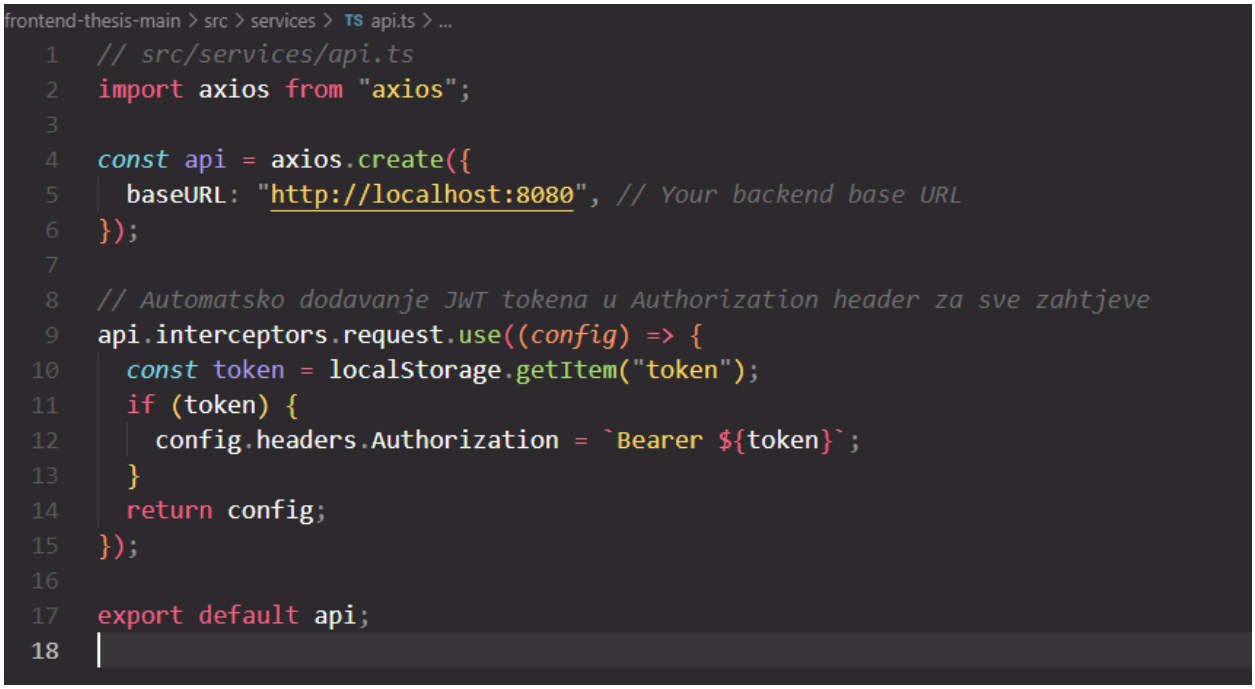
3.4.3. Axios

U web aplikacijama temeljenim na *Reactu*, *Axios* se često koristi za slanje HTTP zahtjeva prema *backendu*. Njegova fleksibilnost, jednostavna sintaksa i mogućnost rukovanja asinkronim operacijama čine ga idealnim izborom za rad s RESTful API-ima. *Axios* omogućava slanje različitih tipova zahtjeva poput GET, POST, PUT, i DELETE, a također podržava konfiguraciju zahtjeva, autentifikaciju i rukovanje pogreškama.

Axios se koristi za kreiranje prilagođene instance koja omogućava jednostavno slanje HTTP zahtjeva prema backendu. U datoteci api.ts kreiramo *Axios* instancu s definiranom osnovnom URL adresom (*baseURL*), koja predstavlja polaznu točku za sve zahtjeve prema *backendu*. Na ovaj se način ne mora svaki put navoditi punu URL adresu u pojedinačnim zahtjevima čime pojednostavljujemo rad s API-jem [11].

Osim toga, koriste se *Axios* interceptori kako bismo automatski dodali JWT u svaki zahtjev. Token se pohranjuje u *localStorage* nakon autentifikacije korisnika te se prilikom svakog sljedećeg

zahtjeva automatski pridružuje u *Authorization* zaglavlje (Sl. 3.11.). Ovaj mehanizam omogućava autentifikaciju korisnika bez potrebe za ručnim dodavanjem tokena u svaki zahtjev.



Slika 3.11. *Prikaz korištenja Axios biblioteke*

Koristi se funkcija *axios.create*() kako bi se kreirala instanca *Axiosa*. Ova instanca ima postavljen *baseURL* na lokalni server (*http://localhost:8080*) što znači da svi zahtjevi koji se pošalju koriste ovu adresu kao polazište. Ako se pošalje GET zahtjev na /*user*, stvarni URL će biti *http://localhost:8080/user*.

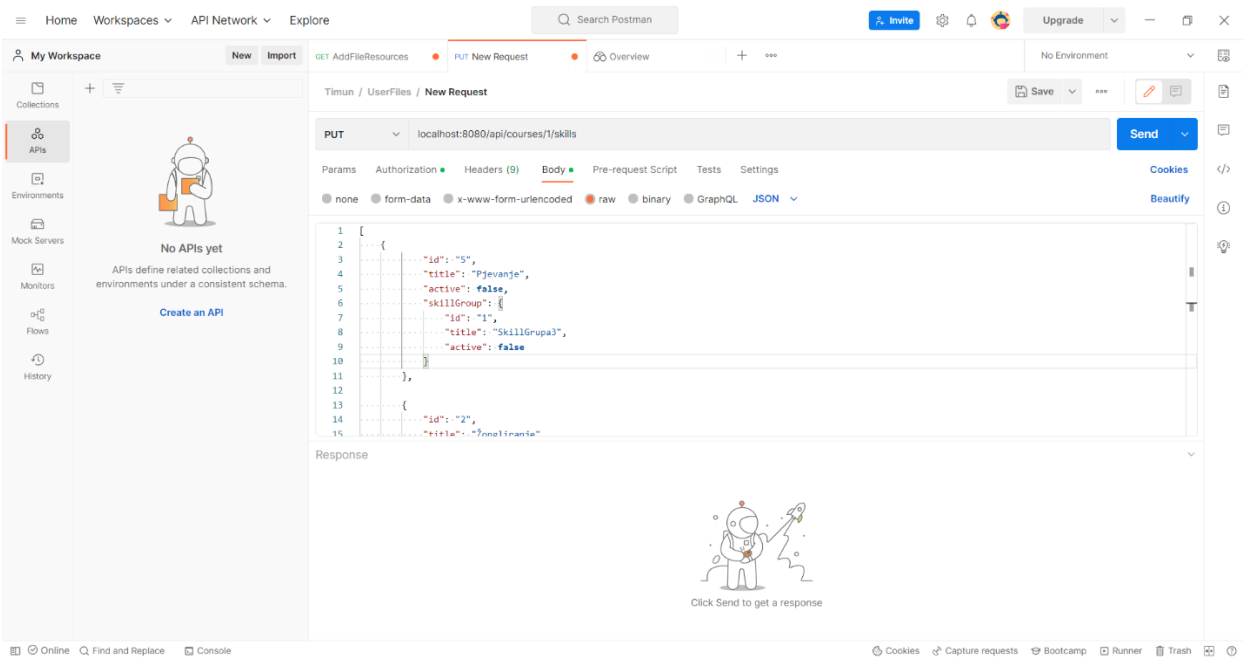
Axios nudi mogućnost postavljanja interceptora koji omogućuju presretanje zahtjeva prije nego što budu poslani. U ovom primjeru koristi se *request* interceptor kako bi se dohvatio *JWT token* iz *localStorage*-a i dodao u zaglavlje svakog zahtjeva. Ako token postoji dodaje se kao *Authorization* zaglavlje u formatu *Bearer <token>*.

Ovakav pristup omogućava automatiziranje svih zahtjeva prema *backendu* s aspekta autentifikacije, bez potrebe za ručnim dodavanjem tokena u svaki zahtjev. *Backend* poslužitelj zatim validira *JWT token* kako bi utvrdio je li korisnik ovlašten pristupiti određenim resursima. Umjesto *Axiosa*, moguće je koristiti integrirani *JavaScript API Fetch* koji također omogućava slanje HTTP zahtjeva prema API-ju, ali s manje mogućnosti prilagodbe. Druga opcija je *SuperAgent*, fleksibilna biblioteka za slanje HTTP zahtjeva koja pruža slične funkcionalnosti kao Axios, uključujući podršku za *promise* i automatsko rukovanje JSON podacima [8, 11].

3.4.4. Postman

Postman je moćan alat koji omogućava korisnicima da testiraju API-je kroz jednostavno i intuitivno sučelje olakšavajući razvoj i otklanjanje pogrešaka u komunikaciji s *backend* servisima. Osim slanja zahtjeva, pruža detaljan pregled odgovora uključujući statuse, zaglavlja i tijelo odgovora što omogućava precizno praćenje kako API funkcionira. Alat također omogućava dodavanje autentifikacije, kao što je JWT, *OAuth*, ili osnovna HTTP autentifikacija čime se omogućava testiranje zaštićenih ruta i API poziva s pristupnim pravima. Uz mogućnosti testiranja, *Postman* nudi funkcionalnosti za automatsku dokumentaciju API-ja što pomaže u dijeljenju resursa unutar tima [12].

Također omogućava kreiranje zbirki zahtjeva (engl. *collections*) što programerima omogućava grupiranje povezanih API poziva, čime je olakšana organizacija i ponavljanje testova tijekom razvoja.



Slika 3.12.: *Postman sučelje*

Glavne komponente *Postman*-a su:

– Intuitivno sučelje (prikazano na slici 3.12.) koje korisnicima omogućava lako slanje zahtjeva i pregled odgovora.

- Upravljanje zahtjevima: omogućava korisnicima da spremaju, organiziraju i automatiziraju zahtjeve što povećava produktivnost.
- Testiranje: pruža mogućnost automatizacije testova što omogućava razvojnim timovima da brzo i lako testiraju funkcionalnosti API-ja.
- Dokumentacija: omogućava korisnicima da generiraju dokumentaciju za API što pomaže u razumijevanju i upotrebi API-ja.
- Integracije: također pruža mogućnost integracije s drugim alatima kao što su *Jenkins*, *Travis* CI i druge što povećava produktivnost i pomaže u automatizaciji rada.
- Podrška za različite autentifikacijske mehanizme: podržava različite autentifikacijske mehanizme kao što su *Basic Auth, OAuth, AWS Signature* i druge.
- Preuzimanje i dijeljenje kolekcija: omogućava korisnicima da preuzmu i dijele kolekcije zahtjeva s drugim korisnicima što olakšava rad u timovima.
- Podrška za različite protokole: podržava različite protokole poput HTTP (engl. *Hypertext Transfer Protocol*), HTTPS (engl. *Hypertext Transfer Protocol Secure*), SOAP (engl. *Simple Object Access Protocol*) i druge što olakšava rad s različitim vrstama API-ja.

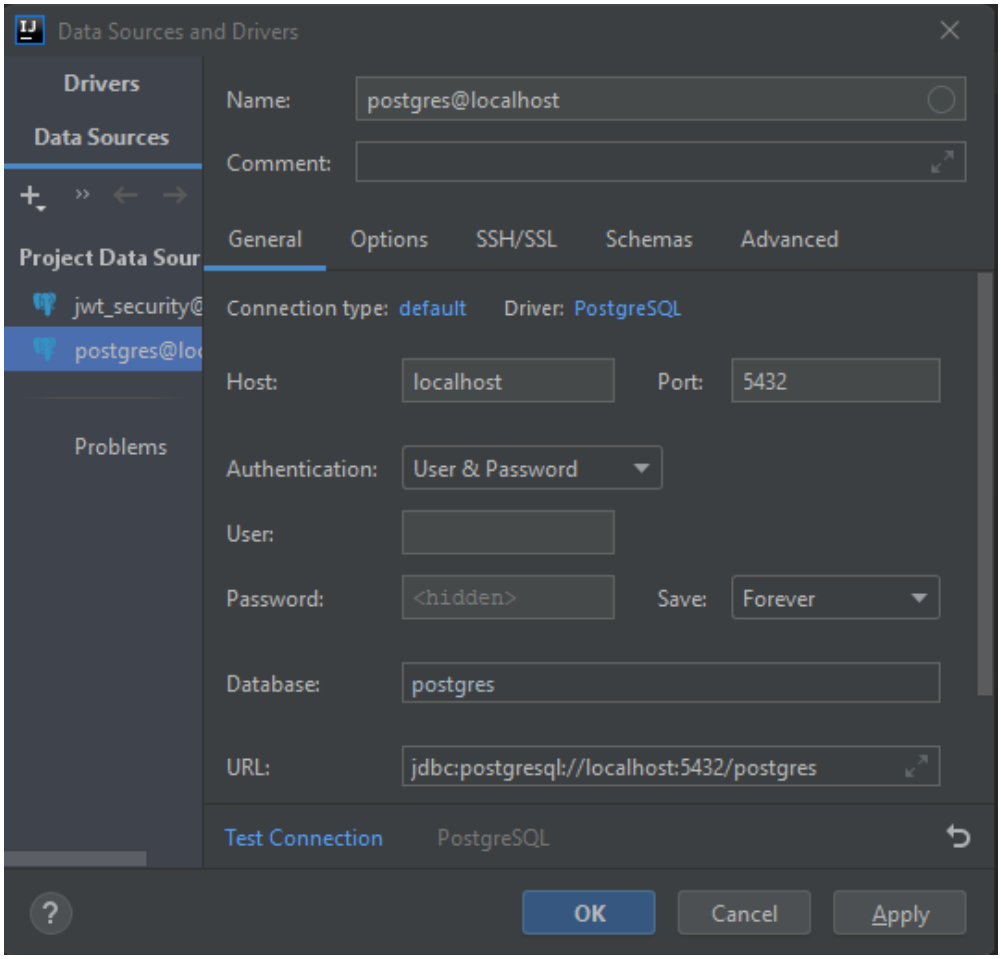
3.4.5. PostgreSql

PostgreSQL je *open source* objektno-relacijski sustav upravljanja bazama podataka RDBMS (eng. *Relational Database Management System*) podržava SQL jezik za rad s bazom podataka. Podržava različite vrste podataka, kao što su brojevi, *stringovi*, datumi i slike te pruža mogućnost rada sa spremljenim procedurama, *trigerima* i indeksima. Često se koristi u komercijalnim i poslovnim aplikacijama te u razvoju softvera za analizu podataka i geografske informacije [13].

Spring Framework pruža podršku za rad s *PostgreSQL-om* kroz tehnologiju objektno-relacijskog mapiranja (ORM), kao što je *Hibernate*, koja omogućava razvojnim timovima da rad s bazom podataka putem *Java* objekata, umjesto korištenja SQL naredbi. Na ovakav se način povećava produktivnost programera i smanjuje vjerojatnost grešaka olakšavajući rad s bazom podataka. *Spring Framework* također pruža podršku za rad s različitim vrstama transakcija i omogućava konfiguriranje pristupa bazi podataka.

U nastavku je prikazan postupak konfiguracije veze s PostgreSQL bazom podataka unutar integriranog razvojnog okruženja (IDE) poput *JetBrains DataGrip*-a ili *IntelliJ* IDEA. U ovom

dijelu diplomskog rada, objašnjava se kako pravilno konfigurirati izvor podataka kako bi aplikacija mogla pristupiti *PostgreSQL* bazi podataka lokalno [13].



Slika 3.13.: *Veza s bazom podataka*

Veza s bazom podataka (Sl. 3.13.) uspostavlja se putem sljedećih parametara:

- *Host* je postavljen na *localhost* što označava pokretanje baze podataka na lokalnom računalu.
- *Port* je definiran kao 5432 što je standardni *port* za PostgreSQL, osim ako nije drugačije specijalno konfiguriran.
- *Authentication* je postavljen na *User & Password*, čime se omogućava unos imena korisnika i lozinke za autentifikaciju prilikom povezivanja s bazom.
- *Database* je postavljen na *postgres* što je zadana baza koja dolazi s instalacijom PostgreSQL-a.

– JDBC URL (Java Database Connectivity) generiran je u obliku: *jdbc:postgresql://localhost:5432/postgres*, gdje su navedeni protokol (*postgresql*), *host* (*localhost*), *port* (5432) i naziv baze podataka (*postgres*).

Ovakva konfiguracija omogućava aplikaciji razvijenoj u *Javi* putem JDBC API-a povezivanje s *PostgreSQL* bazom podataka i izvođenje operacija nad njom. Nakon unosa potrebnih podataka za autentifikaciju i povezivanje moguće je testirati vezu kako bi se osiguralo da je sve ispravno konfigurirano. Ovakav postupak čini ključan korak u razvoju aplikacija koje koriste baze podataka jer omogućava pristup podacima i njihovom upravljanju kroz aplikacijski sloj. [13].

3.4.6. Hibernate

Hibernate je *open source* objektno-relacijski mapiranje (ORM) okvir za *Java*. Koristi se za mapiranje *Java* objekata na relacijske tablice baze podataka i obrnuto. Glavni cilj *Hibernate-a* je pružiti učinkovit način za pristup i rad s podacima u bazi podataka. To se postiže automatizacijom rada s podacima i smanjenjem potrebe za ručnim pisanjem SQL upita.

Hibernate također podržava rad s različitim vrstama transakcija i rad s radnim sesijama. On također podržava različite vrste baza podataka i omogućava lako prebacivanje između različitih baza podataka bez izmjene *koda*. To je jedna od najpopularnijih tehnologija za ORM u *Java* svijetu. *Spring Framework* pruža podršku za *Hibernate* kroz njegov ORM modul što omogućava lako integriranje *Hibernate*-a s ostatkom *Spring* sustava.

Da bi se koristio *Hibernate* u *Spring* aplikaciji, potrebno je konfigurirati *bean*-ove za izvor podataka, radnu tvornicu *sesija* i upravitelj transakcija u *Spring* konfiguracijskoj datoteci. U projekt ga uključujemo preko *pom file*-a u projektu (Sl. 3.14.) [14].

Jednom konfiguriran, *Hibernate SessionFactory* se može koristiti za stvaranje *Hibernate* sesija koje se mogu koristiti za interakciju s bazom podataka i izvršavanje operacija CRUD. *Spring Framework* također pruža podršku za deklarativno upravljanje transakcijama što omogućava lako upravljanje transakcijama pri radu s *Hibernate*-om i bazom podataka. Osim toga, *Spring* pruža podršku za JPA (engl. *Java Persistence API*) koji je specifikacija za ORM, *Hibernate* je jedna od implementacije JPA-a.

```
<dependency> <groupId>org.hibernate</groupId>
<artifactId>hibernate-core</artifactId>
<version>5.6.15.Final</version> </dependency>
```

Slika 3.14.: *Integriranje Hibernate-a u projekt*

4. Implementacija programskog rješenja

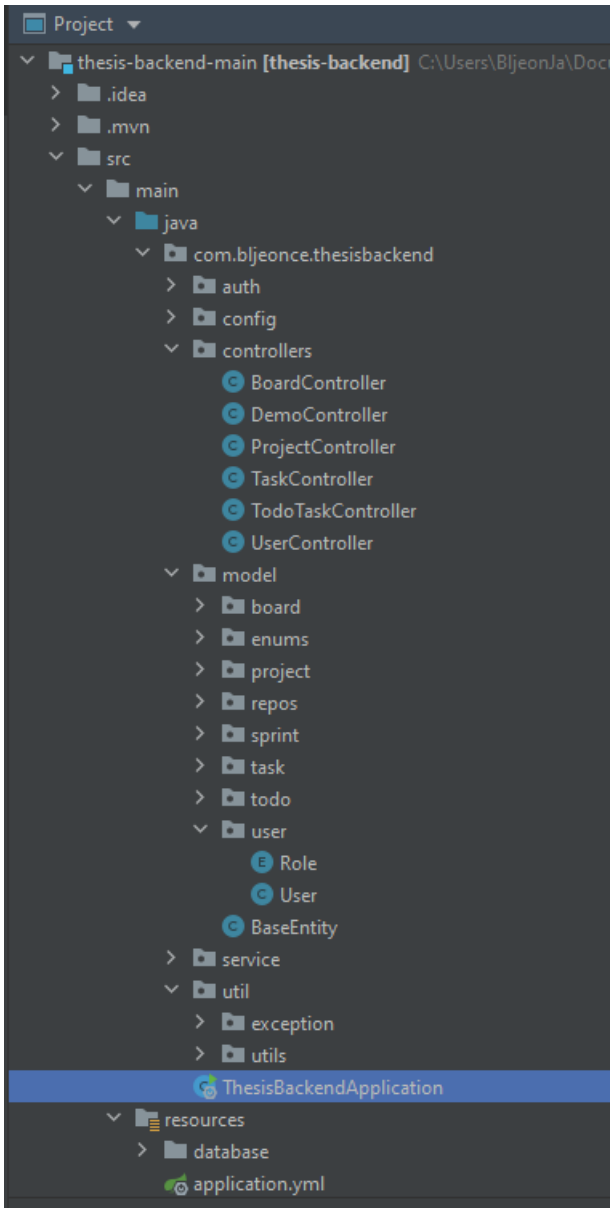
U ovom poglavlju se detaljno razmatraju tehnologije korištene u implementaciji programskog rješenja. Isto tako opisuje se kako su te tehnologije integrirane u arhitekturu sustava radi optimizacije performansi i sigurnosti.

4.1. Opis aplikacije

Implementirana aplikacija je web sustav za upravljanje projektima i projektnim timovima, razvijena s ciljem pružanja besplatne, prilagodljive alternative komercijalnim rješenjima. Temelji se na korištenju *Spring Framework*-a i *Spring Boot*-a za *backend* što omogućava jednostavnu implementaciju web servisa i interakciju s *PostgreSQL* bazom podataka putem *Hibernate* ORM-a. Osnovne funkcionalnosti poput CRUD operacija nad projektima i timovima omogućene su kroz RESTful API, a testiranje je provedeno putem *Postman*-a.

Frontend dio aplikacije razvijen je korištenjem *React*-a i *Tailwind* CSS-a čime je osigurano responzivno i intuitivno korisničko sučelje. Za komunikaciju između *frontenda* i *backenda* koristi se *Axios* dok je sigurnost aplikacije osigurana implementacijom JWT autentifikacije putem biblioteke JJWT. Nadalje, *Lombok* i *ModelMapper* pojednostavljuju razvoj eliminiranjem ponavljajućeg *koda* i olakšavanjem mapiranja podataka. Aplikacija omogućava jednostavno praćenje projekata, upravljanje zadacima i statusima članova tima, čime se poboljšava cjelokupna učinkovitost i produktivnost cijelog tima.

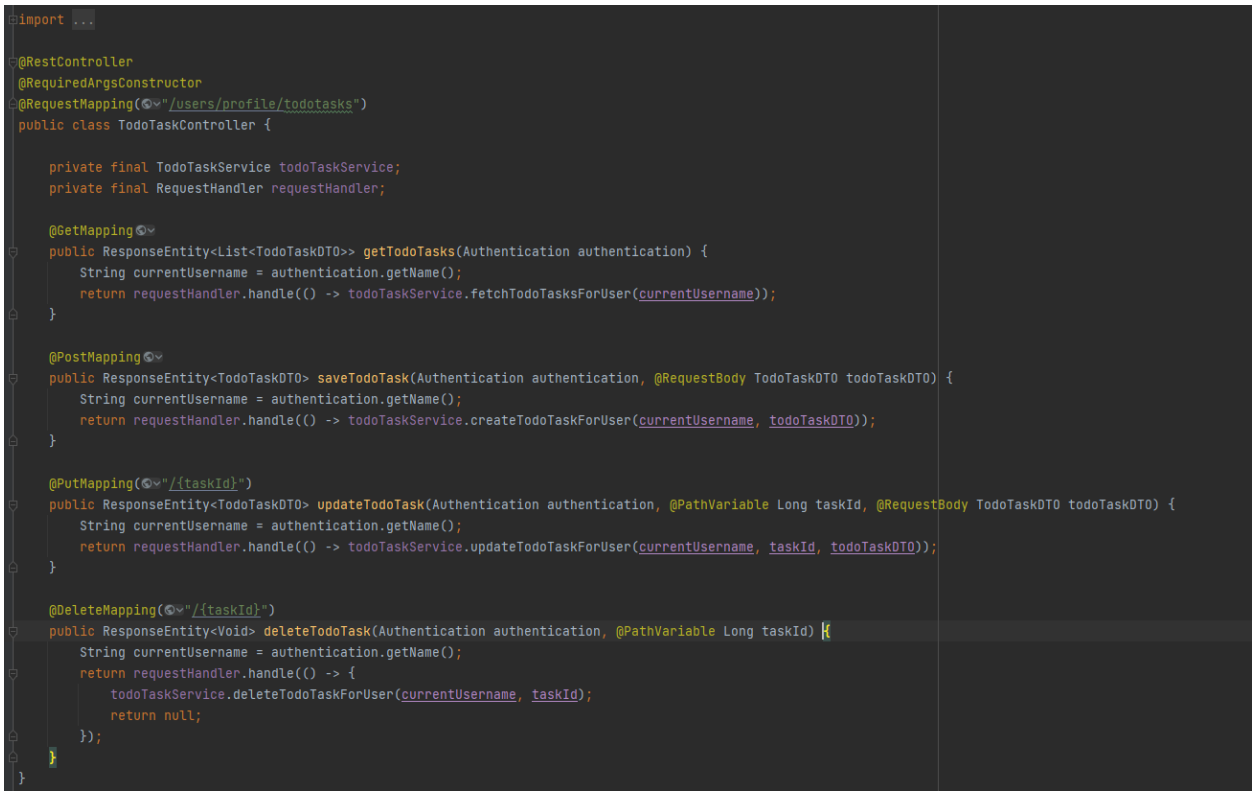
Backend dio aplikacije strukturiran je koristeći *Spring Boot framework* koji omogućava efikasan razvoj i upravljanje poslovnom logikom. Glavna arhitektura aplikacije sastoji se od više slojeva, od kojih svaki ima jasno definiranu funkciju i odgovornosti. Ova modularna organizacija omogućava jednostavno održavanje i proširenje aplikacije, a koristi se u većini suvremenih *Java* aplikacija temeljenih na *Spring frameworku*. Na slici 4.1. prikazana je cjelokupna struktura projekta na *backend* dijelu.



Slika 4.1.: *Struktura na backend dijelu implementacije projekta*

U direktoriju *src/main/java*, glavni paket aplikacije je *com.bljeonce.thesisbackend* koji sadrži više paketa strukturiranih prema funkcionalnostima. Paket *controllers* sadrži kontrolere odgovorne za rukovanje HTTP zahtjevima. Svaki *controller* upravlja specifičnim resursom unutar aplikacije i omogućava pristup CRUD operacijama. Primjeri uključuju *ProjectController* za upravljanje projektima i *UserController* za upravljanje korisnicima, *TodoTaskController* za upravljanje *Todo* zadacima (Sl. 4.2.). Ovi kontroleri koriste RESTful pristup što znači da svaki resurs ima svoj API *endpoint* i odgovarajuće metode za dohvat, kreiranje, ažuriranje ili brisanje podataka. U *kodu* na slici 4.2. koristi se *Lombokova* anotacija *@RequiredArgsConstructor* koja automatski generira konstruktor za sva finalna polja klase. U ovom slučaju za *todoTaskService* i *requestHandler*.

Anotacije poput @Getter i @Setter smanjuju potrebu za ručnim pisanjem kodova za pristupanje poljima unutar klasa. Također, anotacija *@Builder* omogućava kreiranje objekata uz upotrebu graditelja (*builder pattern*) što olakšava rad s kompleksnim konstruktorima.. Na ovakav način se smanjuje količina *boilerplate* koda što olakšava održavanje i čistoću same klase [15].



```
import ...

@RestController
@RequiredArgsConstructor
@RequestMapping("/users/profile/todotasks")
public class TodoTaskController {

    private final TodoTaskService todoTaskService;
    private final RequestHandler requestHandler;

    @GetMapping
    public ResponseEntity<List<TodoTaskDTO>> getTodoTasks(Authentication authentication) {
        String currentUsername = authentication.getName();
        return requestHandler.handle(() -> todoTaskService.fetchTodoTasksForUser(currentUsername));
    }

    @PostMapping
    public ResponseEntity<TodoTaskDTO> saveTodoTask(Authentication authentication, @RequestBody TodoTaskDTO todoTaskDTO) {
        String currentUsername = authentication.getName();
        return requestHandler.handle(() -> todoTaskService.createTodoTaskForUser(currentUsername, todoTaskDTO));
    }

    @PutMapping("/{taskId}")
    public ResponseEntity<TodoTaskDTO> updateTodoTask(Authentication authentication, @PathVariable Long taskId, @RequestBody TodoTaskDTO todoTaskDTO) {
        String currentUsername = authentication.getName();
        return requestHandler.handle(() -> todoTaskService.updateTodoTaskForUser(currentUsername, taskId, todoTaskDTO));
    }

    @DeleteMapping("/{taskId}")
    public ResponseEntity<Void> deleteTodoTask(Authentication authentication, @PathVariable Long taskId) {
        String currentUsername = authentication.getName();
        return requestHandler.handle(() -> {
            todoTaskService.deleteTodoTaskForUser(currentUsername, taskId);
            return null;
        });
    }
}
```

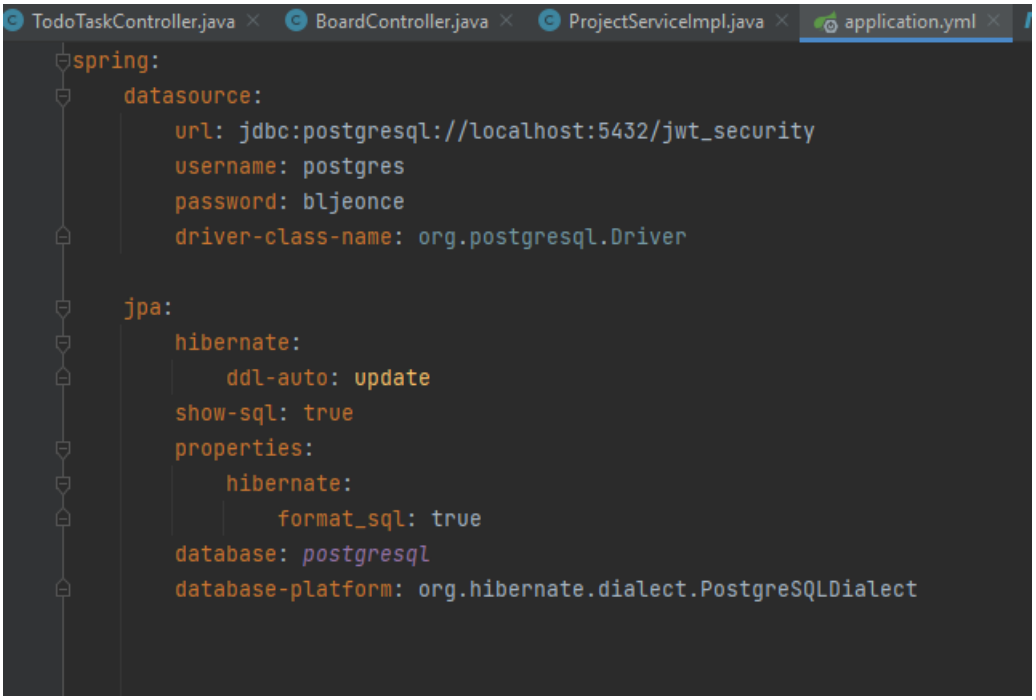
Slika 4.2.: *Primjer jednog controllera iz implementirane web aplikacije*

Paket *model* definira entitete koji predstavljaju podatke unutar aplikacije. Svaki je entitet preslikan u tablicu u bazi podataka koristeći *Hibernate* ORM. Entitet *User* predstavlja korisnika sustava s pripadajućim atributima poput korisničkog imena i e-mail adrese dok entitet *Project* modelira projekte u sustavu. Svi entiteti nasljeđuju klasu *BaseEntity* koja daje zajedničke atribute kao što je jedinstveni identifikator čime se omogućava dosljednost u radu s podacima i lakša organizacija te kreiranje novih entiteta.

Sloj *service* odgovoran je za poslovnu logiku aplikacije. Ovdje se nalaze servisi koji upravljaju složenim operacijama i osiguravaju koheziju između kontrolera i sloja za pristup podacima. Na primjer, *UserService* rukuje svim poslovnim pravilima vezanim uz korisnike dok *ProjectService* upravlja logikom vezanom uz projekte.

Za rješavanje iznimaka i pomoćne funkcije koristi se paket *util* koji sadrži klase za hvatanje i obrađivanje iznimki što omogućava robustan sustav za upravljanje greškama. Konfiguracija

aplikacije pohranjena je u datoteci *application.yml* koja je prikazana na slici broj 4.3. unutar direktorija *resources*, koja sadrži ključne postavke poput podataka o bazi podataka, sigurnosnih protokola te konfiguracija specifičnih za rad *Spring* aplikacije [6].



Slika 4.3. *Prikaz application.yml file-a*

Ova strukturirana arhitektura osigurava efikasnu i skalabilnu aplikaciju omogućujući razvojnom timu da lako održava i proširuje funkcionalnosti.

4.1.1. Entiteti aplikacije i repozitoriji

U *backend* dijelu aplikacije koristi se arhitektura koja se temelji na *Spring Boot frameworku*, pri čemu su svi ključni entiteti i repozitoriji jasno definirani i strukturirani kako bi se omogućilo jednostavno upravljanje podacima i implementacija poslovne logike. Svaki entitet unutar sustava predstavlja specifični objekt koji je preslikan na tablice u relacijskoj bazi podataka koristeći *Hibernate* ORM. Na taj način omogućena je potpuna integracija između aplikacijskog sloja i baze podataka, a *Spring Data* JPA nudi mogućnost jednostavnog upravljanja podacima putem repozitorija, bez potrebe za ručnim pisanjem SQL upita.

Entiteti u ovoj aplikaciji predstavljaju osnovne komponente sustava za upravljanje projektima. Primjerice, *Board* entitet predstavlja "ploču" unutar sustava (Sl. 4.4.), koja služi za organizaciju zadataka u kontekstu upravljanja projektima dok Project entitet sadrži sve relevantne informacije

o pojedinom projektu, uključujući naziv, opis i povezanost s korisnicima i zadacima. *Sprint* entitet koristi se za praćenje radnih ciklusa unutar *Agile* metodologije, a *Task* entitet omogućava definiranje pojedinačnih zadataka koji su dodijeljeni korisnicima ili timovima unutar projekta. Svaki od tih entiteta definira se atributima koji opisuju podatke koje sadrži, a njihovi međusobni odnosi osiguravaju povezivanje svih aspekata upravljanja projektima, od korisnika do zadataka i sprintova.

```
public class Board extends BaseEntity {

    @Column(name = "name", unique = true)
    private String name;

    @Column(name = "description")
    private String description;

    @Column(name = "creationDate")
    private LocalDate creationDate;

    @Column(name = "lastUpdated")
    private LocalDate lastUpdated;

    @Enumerated(EnumType.STRING)
    @Column(name = "visibility")
    private Visibility visibility;

    @ManyToMany(cascade = {CascadeType.PERSIST, CascadeType.MERGE})
    @JoinTable(
            name = "board_users",
            joinColumns = @JoinColumn(name = "board_id"),
            inverseJoinColumns = @JoinColumn(name = "user_id")
    )
    private List<User> users;

    @OneToMany(mappedBy = "board", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Task> tasks;

    @{...}
    private Project project;

    @OneToMany(mappedBy = "board", cascade = CascadeType.ALL)
    private List<Sprint> sprints;

}
```

Slika 4.4.: *Primjer entiteta, Board, iz implementirane aplikacije*

Za korisničke podatke, *User* entitet definira sve informacije vezane za korisnike sustava, poput korisničkog imena, lozinke i uloge (Sl. 4.6.).

Entitet *User* koristi polje *id* koji je jedinstveni identifikator korisnika unutar baze podataka. Identifikator je automatski generiran što osigurava jedinstvenost svakog korisnika. Osim toga,

pohranjuju se i osnovni podaci kao što su ime (engl. *firstName*), prezime (engl. *lastName*), e-mail adresa i lozinka. E-mail adresa korisnika služi kao njegovo korisničko ime te mora biti jedinstvena unutar sustava što se osigurava korištenjem „*@Column(unique = true)*“ anotacije. Valjanost e-mail adrese dodatno se provjerava pomoću anotacije *@Email* dok se za lozinku primjenjuje validacija kojom se osigurava minimalna duljina od sedam znakova. Korištenje ovakvih pravila validacije omogućava veću sigurnost i konzistentnost unutar sustava.

Entitet *User* je proširen sa sučeljem *UserDetails*. Ovo sučelje pruža osnovne informacije o korisnicima koji su potrebni za autentifikaciju i autorizaciju unutar sustava. Glavna svrha *UserDetails* sučelja je definiranje pravila o tome koje korisničke podatke treba osigurati sustavu za autentifikaciju, poput korisničkog imena, lozinke i korisničkih prava (autoriteta). Ono što je važno napomenuti je da ovo sučelje nije izravno odgovorno za sigurnosne mehanizme, već služi kao objekt koji pohranjuje korisničke podatke, a ti podaci se kasnije *kapsuliraju* u objektu *Authentication*, koji koristi *Spring Security* za provođenje autentifikacije i autorizacije. Na primjer, kada korisnik unese svoje podatke za prijavu, ti se podaci prosljeđuju kroz objekte koji implementiraju ovo sučelje, a zatim se koriste za provjeru autentičnosti [6].

```java
/**
 * Provides core user information.
 *
 * <p>
 * Implementations are not used directly by Spring Security for security purposes. They
 * simply store user information which is later encapsulated into {@link Authentication}
 * objects. This allows non-security related user information (such as email addresses,
 * telephone numbers etc) to be stored in a convenient location.
 * <p>
 * Concrete implementations must take particular care to ensure the non-null contract
 * detailed for each method is enforced. See
 * {@link org.springframework.security.core.userdetails.User} for a reference
 * implementation (which you might like to extend or use in your code).
 *
 * @author Ben Alex
 * @see UserDetailsService
 * @see UserCache
 */
public interface UserDetails extends Serializable {

    Returns the authorities granted to the user. Cannot return null.
    Returns: the authorities, sorted by natural key (never null)
    Collection<? extends GrantedAuthority> getAuthorities();

    Returns the password used to authenticate the user.
    Returns: the password
    String getPassword();

    Returns the username used to authenticate the user. Cannot return null.
    Returns: the username (never null)
    String getUsername();
```

Slika 4.5.: *Primjer entiteta, Board, iz implementirane aplikacije*

Komentar u *kodu*, prikazan na slici 4.5., također ističe da je bitno da svaka konkretna implementacija ovog sučelja strogo slijedi pravila i osigurava da povratne vrijednosti metoda ne budu *null* jer to može uzrokovati sigurnosne propuste ili probleme u sustavu. Kao primjer reference, spominje se klasa *User* iz paketa *org.springframework.security.core.userdetails.User*, koja je često korištena implementacija ovog sučelja i može poslužiti kao baza za prilagodbu ili proširenje u aplikacijama [6].

```
@Table(name = "users")
public class User implements UserDetails {
    @SequenceGenerator(
            name = "users_sequence",
            sequenceName = "users_sequence",
            allocationSize = 1
    )
    @Id
    @GeneratedValue
    private Long id;
    @Column(name = "first_name")
    private String firstName;
    @Column(name = "last_name")
    private String lastName;
    //USERNAME KORISNIKA
    @NotNull(message = "Email cannot be empty")
    @Email(message = "Please enter a valid email address")
    @Column(name = "email", unique = true)
    private String email;
    @NotNull(message = "Password cannot be empty")
    @Length(min = 7, message = "Password should be at least 7 characters long")
    @Column(name = "password")
    private String password;
    @Enumerated(EnumType.STRING)
    @Column(name = "role")
    private Role role;
    @OneToMany(mappedBy = "assignedUser", fetch = FetchType.LAZY)
    private List<TodoTask> todoTasks;
    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true, mappedBy = "assignedTo")
    private List<Task> tasksAssigned;
    @OneToMany(mappedBy = "assignee", fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    private List<Task> tasksCreated;
    @ManyToMany
    @JoinTable(
            name = "projects_users",  // Using 'projects_users' to match
            joinColumns = @JoinColumn(name = "user_id"),
            inverseJoinColumns = @JoinColumn(name = "project_id")
    )
    private List<Project> projects;
```

Slika 4.6.: *Primjer entiteta, Board, iz implementirane aplikacije*

Jedan od kljunih dijelova ovog entiteta je polje uloga (engl. *role*), koje definira korisničke uloge unutar sustava. Svaki korisnik može imati samo jednu ulogu, a uloge su definirane u obliku *enum* tipa kako bi se olakšalo upravljanje korisničkim pravima. Primjerice, uloge mogu uključivati administratora, projektnog menadžera ili člana tima, pri čemu svaka uloga ima specifična prava pristupa. U slučaju implementirane aplikacije postoje uloge korisnika i administratora [6].

```
@Table(name = "tasks")
public class Task extends BaseEntity {

    @Column(name = "name")
    private String name;

    @Column(name = "description")
    private String description;

    @Column(name = "deadline")
    private LocalDate deadline;

    @Column(name = "status")
    @Enumerated(EnumType.STRING)
    private TaskStatus status;

    @ManyToOne
    @JoinColumn(name = "assigned_id")
    private User assignedTo;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "assignee_id")
    private User assignee;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "board_id", nullable = false)
    private Board board;

    @ManyToOne
    @JoinColumn(name = "project_id")
    private Project project;

    //ako je prazno onda je task u backlogu
    @ManyToOne
    @JoinColumn(name = "sprint_id", nullable = false)
    private Sprint sprint;
}
```

Slika 4.7.: *Primjer entiteta, Task, iz implementirane aplikacije*

Entitet *User* povezan je s drugim entitetima u aplikaciji kroz različite veze. Jedna od tih veza je s entitetom *Task* (Sl. 4.7.). Korisnici mogu biti odgovorni za zadatke unutar projekta te se ta veza

prikazuje na polju *tasks*. Korisnik ima svoje privatne bilješke, a ova odgovornost se preslikava kroz polje *todoTasks*, koje sadrži listu zadataka dodijeljenih korisniku i samo su povezani s tim korisnikom. Također, korisnik može biti odgovoran za kreiranje zadataka pa je to pohranjeno u polju *tasksCreated* dok polje *tasksAssigned* prati zadatke koji su dodijeljeni korisniku kao izvršitelju. Ovi odnosi omogućuju praćenje aktivnosti korisnika unutar projekata i zadataka te osiguravaju preglednost i transparentnost tijekom rada.

Posebno zanimljiv aspekt entiteta *User* jest veza s projektima koja je definirana kao *ManyToMany* odnos. Korisnici mogu biti povezani s više projekata, a svaki projekt može uključivati više korisnika (Sl. 4.8.). Ova veza omogućava fleksibilnost u upravljanju projektima jer različiti korisnici mogu imati pristup istim projektima, ali s različitim pravima, ovisno o njihovoj ulozi. Takva arhitektura omogućava korisnicima da rade na više projekata u isto vrijeme te ovisno o ulozi potpuno prate status svih projekata.

```
@ManyToMany(cascade = {CascadeType.PERSIST, CascadeType.MERGE})
@JoinTable(
        name = "projects_users",
        joinColumns = @JoinColumn(name = "project_id"),
        inverseJoinColumns = @JoinColumn(name = "user_id")
)
private List<User> users;
```

Slika 4.8.: *Prikaz primjene anotacija koje omogućuju povezanost projekta s više korisnika*

Repozitoriji unutar aplikacije predstavljaju sloj koji olakšava pristup podacima u bazi podataka. Svaki entitet ima odgovarajući repozitorij koji omogućava izvršavanje osnovnih operacija poput dohvaćanja, kreiranja, ažuriranja i brisanja podataka. Na primjer, *ProjectRepository* omogućava upravljanje podacima o projektima dok *TaskRepository* služi za operacije vezane uz zadatke. Korištenje *JpaRepository* sučelja iz *Spring Data* JPA omogućava automatsko kreiranje metoda za ove operacije čime se ubrzava razvojni proces te smanjuje količina *koda* potrebnog za upravljanje bazom podataka.

Primjer kako *JpaRepository* generira upite jest kroz prepoznavanje ključnih riječi unutar imena metode. Na primjer, kada se definira metoda poput *findByBoardId*(*Long boardId*), *Spring* automatski generira SELECT upit koji će dohvatiti sve zapise iz baze na temelju atributa boardId. Ovaj mehanizam naziva se *query derivation,* gdje *Spring* generira upit koristeći fragmente metode kao kriterije.

Ključne riječi poput "*find*", "*By*", "*And*", "*Or*" omogućavaju *Springu* prepoznavanje atributa nad kojima se pretražuje. Nadalje, *Spring Data* JPA podržava metode za paginaciju i sortiranje pri čemu koristi parametre kao što su *Pageable* i *Sort* kako bi automatski dodao uvjete za ograničavanje broja rezultata ili definirao redoslijed sortiranja. Na taj način *Spring* olakšava i ubrzava razvoj smanjujući potrebu za ručnim pisanjem SQL upita, a programeri se mogu fokusirati na poslovnu logiku aplikacije.

JPA (*Java Persistence* API) repozitoriji u *Springu* omogućuju jednostavno upravljanje podacima u bazi bez potrebe za ručnim pisanjem SQL upita. Korištenjem sučelja *JpaRepository* nasljeđuju se osnovne CRUD operacije, što ubrzava razvoj i pojednostavljuje rad s bazom podataka [17].

```java
import java.util.List;

public interface TaskRepository extends JpaRepository<Task, Long> {
    List<Task> findByBoardId(Long boardId);
    List<Task> findByAssignedToId(Long userId);
    List<Task> findByBoardIdAndAssignedToId(Long boardId, Long userId);
    Page<Task> findByAssignedTo(User user, Pageable pageable);
    List<Task> findAllByProject(Project project);
    List<Task> findByBoardIdAndSprintId(Long boardId, Long sprintId);
    List<Task> findByBoardIdAndSprintIsNull(Long boardId);
}
```

Slika 4.9.: *Prikaz implementacije repozitorija za upravljanje entitetom Task*

Na slici 4.9. koja prikazuje *TaskRepository* vidljivo je kako se JPA koristi za kreiranje specifičnih upita na temelju metoda. Repozitorij sadrži metode poput *findByBoardId*(*Long boardId*) i *findByAssignedToId*(*Long userId*) koje omogućuju dohvaćanje zadataka temeljem *board* ID-a ili ID-a korisnika. Nadalje, metoda *findByBoardIdAndAssignedToId*(*Long boardId*, *Long userId*) kombinira oba parametra za preciznije pretrage dok *findByAssignedTo*(*User user*, *Pageable pageable*) omogućava paginirano dohvaćanje zadataka.

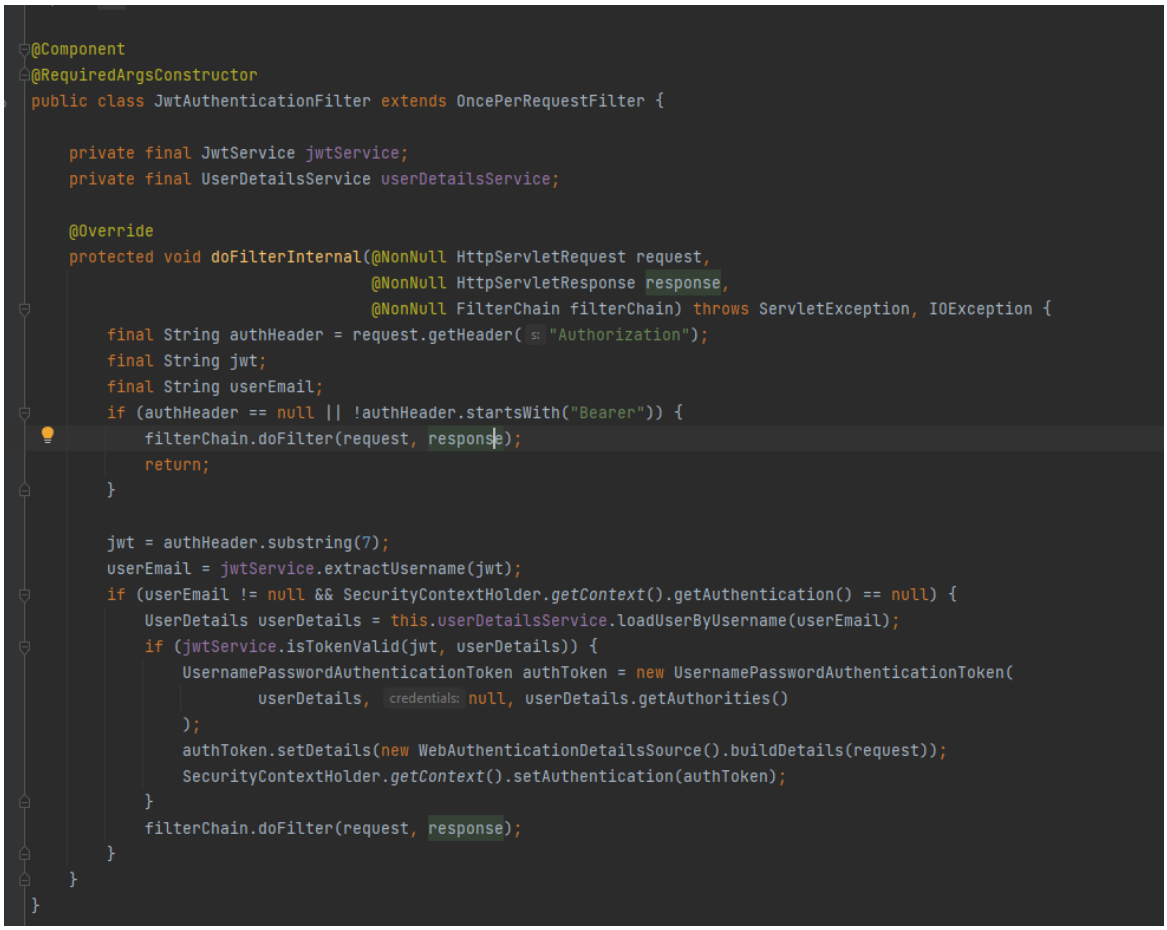
JPA repozitoriji eliminiraju potrebu za pisanjem specifičnih SQL upita jer *Spring* automatski generira odgovarajuće upite na temelju naziva metoda. Ova apstrakcija olakšava upravljanje složenijim aplikacijama te čini rad s bazama podataka bržim i učinkovitijim [17].

4.1.2. Autentifikacija

Autentifikacija pomoću *JWT*-a postala je standard u modernim web aplikacijama, posebno zbog svoje sposobnosti da osigura *stateless* autentifikaciju, što znači da sustav ne mora pohranjivati korisničke sesije na serveru. JWT omogućava sigurno prosljeđivanje korisničkih podataka putem tokena koji se generira pri prijavi, a zatim koristi u svakom zahtjevu prema zaštićenim dijelovima aplikacije. Ovaj pristup omogućava jednostavniju skalabilnost aplikacije i smanjuje opterećenje na serveru jer ne postoji potreba za upravljanjem sesijama. Unutar aplikacije koja koristi *Spring Security* za autentifikaciju, JWT se generira i validira kako bi se osigurala pravilna autentifikacija korisnika. [6, 18]

Korištenjem klase *JwtService*, aplikacija može generirati token za autentificiranog korisnika koji sadrži osnovne podatke kao što su korisničko ime te potpis za osiguranje integriteta tokena. Token se zatim vraća klijentu kao odgovor na uspješnu prijavu ili registraciju. Klijent taj token pohranjuje (obično u lokalnu pohranu preglednika) i šalje ga u svakom sljedećem zahtjevu putem zaglavlja "*Authorization*", omogućujući korisniku pristup zaštićenim resursima bez potrebe za ponovnom prijavom. [18].

Svaki dolazni HTTP zahtjev filtrira se kroz *JwtAuthenticationFilter* (Sl.4.10.), koji prepoznaje JWT u zaglavlju i provjerava njegovu valjanost koristeći *JwtService*. Ako je token važeći, korisničke vjerodajnice se automatski dodaju u *SecurityContext* čime se omogućava autorizacija korisnika za pristup određenim dijelovima sustava. Ovaj proces provjere valjanosti tokena osigurava da samo ovlašteni korisnici imaju pristup zaštićenim resursima. U slučaju da token nije važeći ili je istekao, sustav odbija pristup i traži ponovno autentificiranje [18].



```java
@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private final JwtService jwtService;
    private final UserDetailsService userDetailsService;

    @Override
    protected void doFilterInternal(@NonNull HttpServletRequest request,
                                    @NonNull HttpServletResponse response,
                                    @NonNull FilterChain filterChain) throws ServletException, IOException {
        final String authHeader = request.getHeader( s: "Authorization");
        final String jwt;
        final String userEmail;
        if (authHeader == null || !authHeader.startsWith("Bearer")) {
            filterChain.doFilter(request, response);
            return;
        }

        jwt = authHeader.substring(7);
        userEmail = jwtService.extractUsername(jwt);
        if (userEmail != null && SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails = this.userDetailsService.loadUserByUsername(userEmail);
            if (jwtService.isTokenValid(jwt, userDetails)) {
                UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(
                        userDetails,  credentials: null, userDetails.getAuthorities()
                );
                authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
            filterChain.doFilter(request, response);
        }
    }
}
```

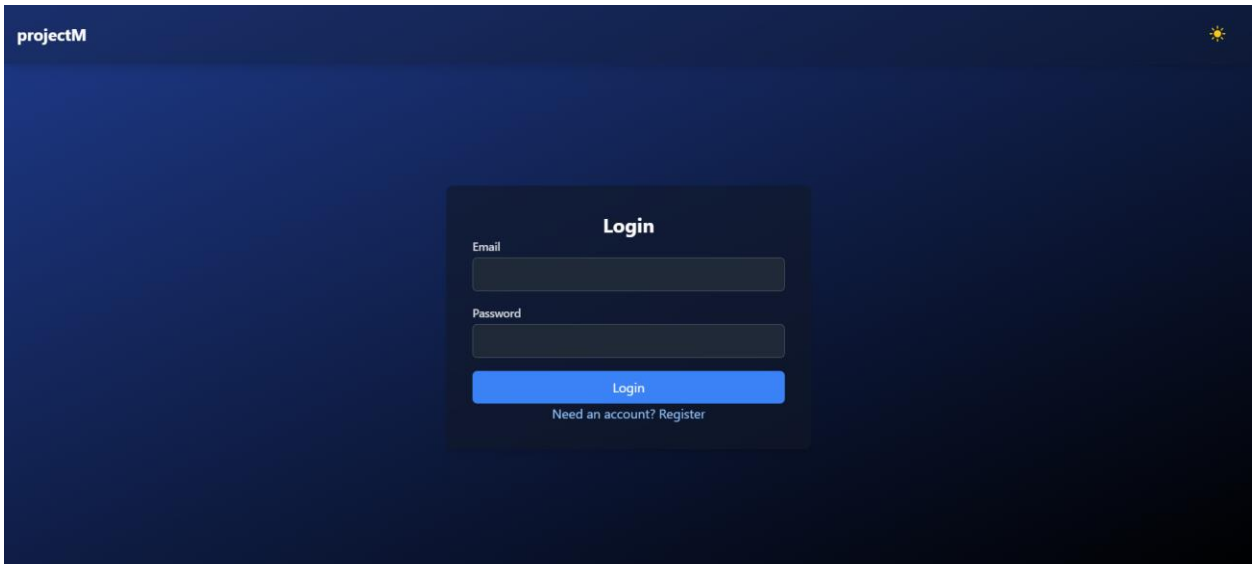
Slika 4.10.: *Prikaz klase JwtAuthenticationFilter koja se koristi u procesu autentifikacije*

Autentifikacija putem JWT-a u ovoj aplikaciji oslanja se na *stateless* arhitekturu što znači kako se ne koriste sesije za praćenje korisničkog stanja. Umjesto toga, svaki zahtjev nosi svoj vlastiti token koji server koristi za identifikaciju i provjeru korisnika. Ovakav način autentifikacije pogodan je za sustave s velikim brojem korisnika ili distribuirane sustave jer se svi podaci potrebni za autentifikaciju prenose unutar tokena, bez potrebe za održavanjem stanja na strani servera. Dodatna prednost ovakvog pristupa jest sigurnost, budući da su svi osjetljivi podaci šifrirani i ne mogu se mijenjati bez otkrivanja ključa za potpisivanje tokena. *JwtService* osigurava da su svi tokeni potpisani sigurnim kriptografskim ključem kako bi se spriječila bilo kakva manipulacija podacima unutar tokena. Ovaj ključ koristi se za validaciju tokena i njegovu sigurnost što je ključno za sprječavanje napada poput "*token hijacking*-a" ili neovlaštene manipulacije korisničkim podacima. Time se osigurava visoka razina sigurnosti u aplikaciji dok *JwtAuthenticationFilter* i ostale sigurnosne postavke osiguravaju da je pristup zaštićenim resursima omogućen samo pravovaljano autentificiranim korisnicima [18].

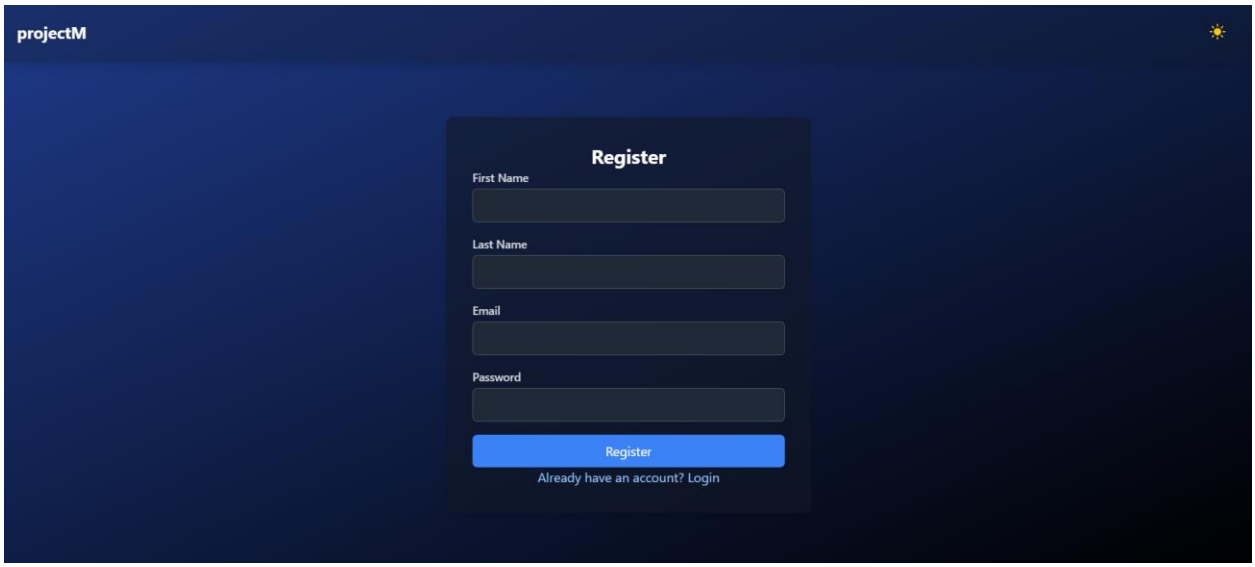
4.2. Prikaz klijentskog dijela aplikacije

U ovom poglavlju ukratko je opisan pregledni dio aplikacije pomoću kojeg se mogu pozivati metode sa *backend* dijela i upravljati podacima te imati njihov pregled.

Slika 4.11. predstavlja login formu aplikacije koja omogućava autentifikaciju korisnika. U gornjem lijevom kutu vidljiv je naziv aplikacije "projectM" dok središnji dio zaslona sadrži formu za unos korisničkih podataka odnosno email adrese i lozinke. Nakon unosa potrebnih podataka korisnik klikom na gumb "*Login*" šalje zahtjev za autentifikaciju pri čemu aplikacija provjerava ispravnost vjerodajnica i dodjeljuje JWT token koji će se koristiti za daljnje sesije. Ukoliko korisnik još uvijek nema kreiran korisnički račun postoji mogućnost registracije putem linka "*Need an account*? *Register*" (Sl. 4.12.). Ovaj proces je ključan za zaštitu korisničkih podataka te osigurava siguran i jednostavan način autentifikacije.

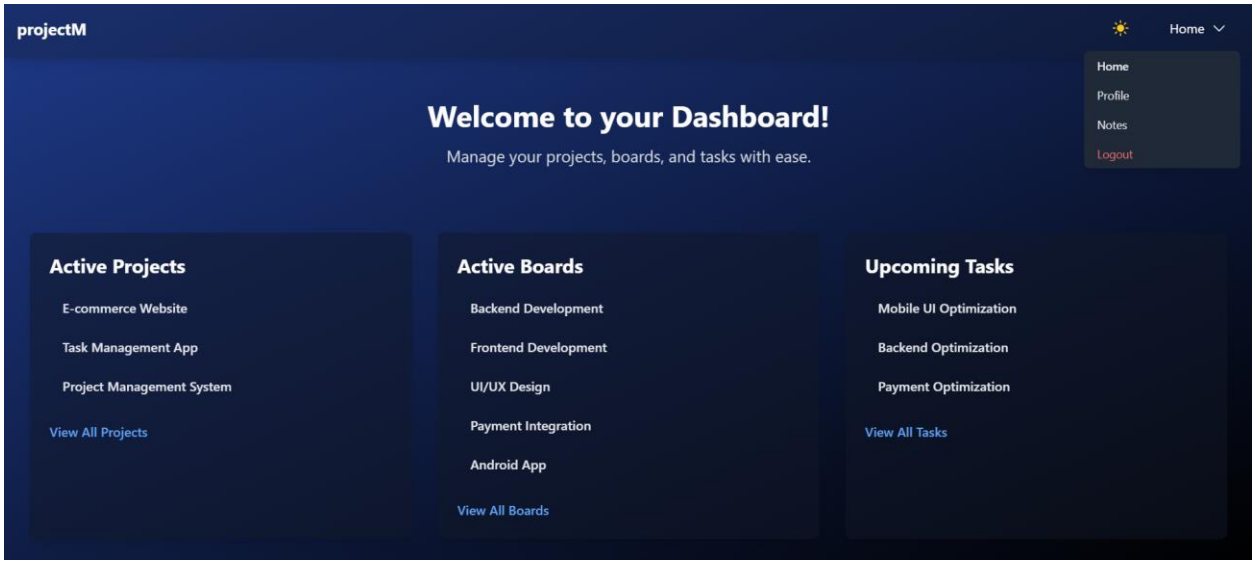


Slika 4.11.: *Prikaz login forme aplikacije*



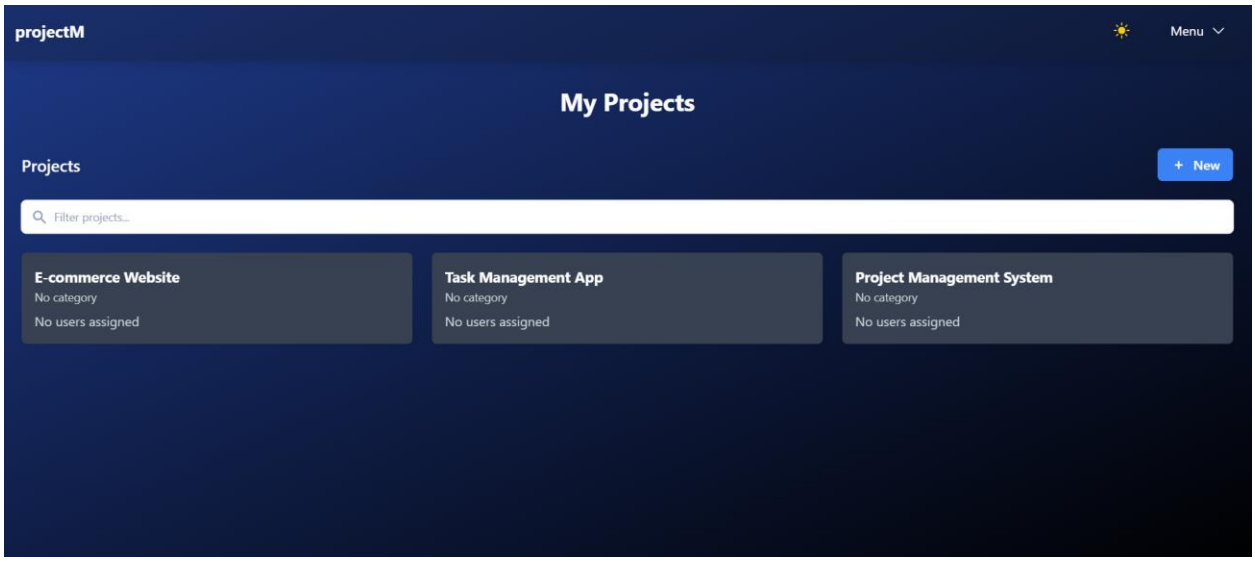
Slika 4.12.: *Prikaz forme za registriranje*

Korisniku se nakon prijave prikazuje nadzorni panel s pregledom projekata, ploča i zadataka (Sl. 4.13.). Lijevo su prikazani aktivni projekti, u sredini aktivne ploče, a desno nadolazeći zadaci. Gornji desni izbornik omogućava pristup korisničkim postavkama i odjavi. Ovakav prikaz omogućava brzu navigaciju i učinkovito upravljanje projektima.



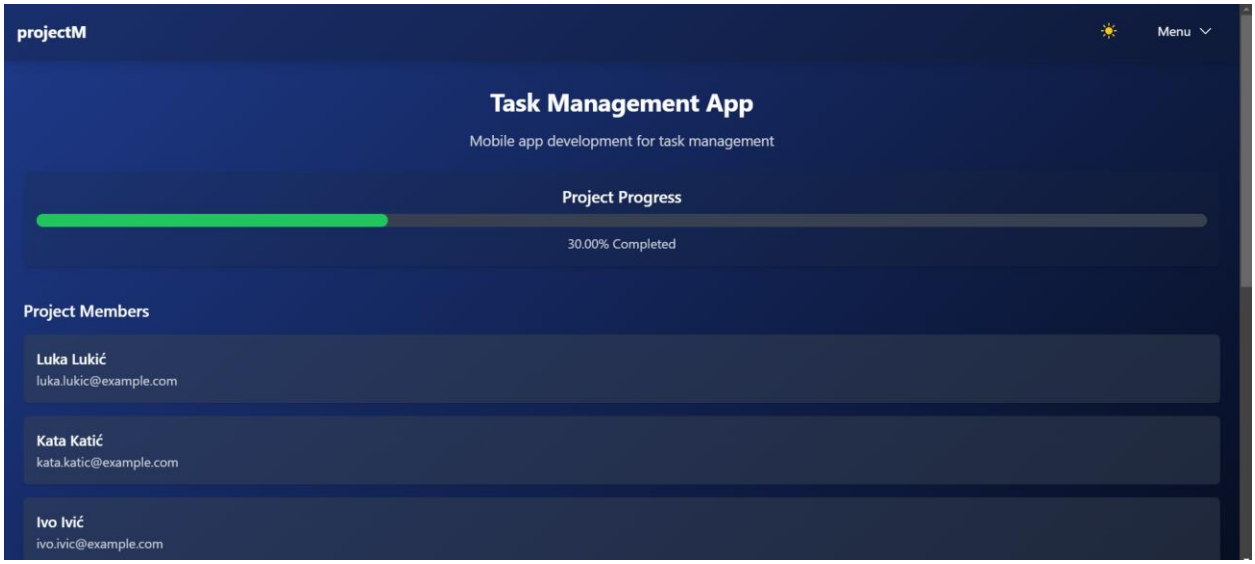
Slika 4.13.: *Prikaz glavnog sučelja nakog uspješnog autentificiranja korisnika*

Korisnik može vidjeti svoje projekte s opcijom filtriranja i pregledom kartica za svaki projekt (Sl. 4.14.). Svaka kartica sadrži osnovne informacije o projektu poput naziva i kategorije. Desno se nalazi gumb "*+New*" za dodavanje novih projekata. Ovaj prikaz omogućava jednostavnu organizaciju i kreiranje novih radnih projekata te njihovo filtriranje.



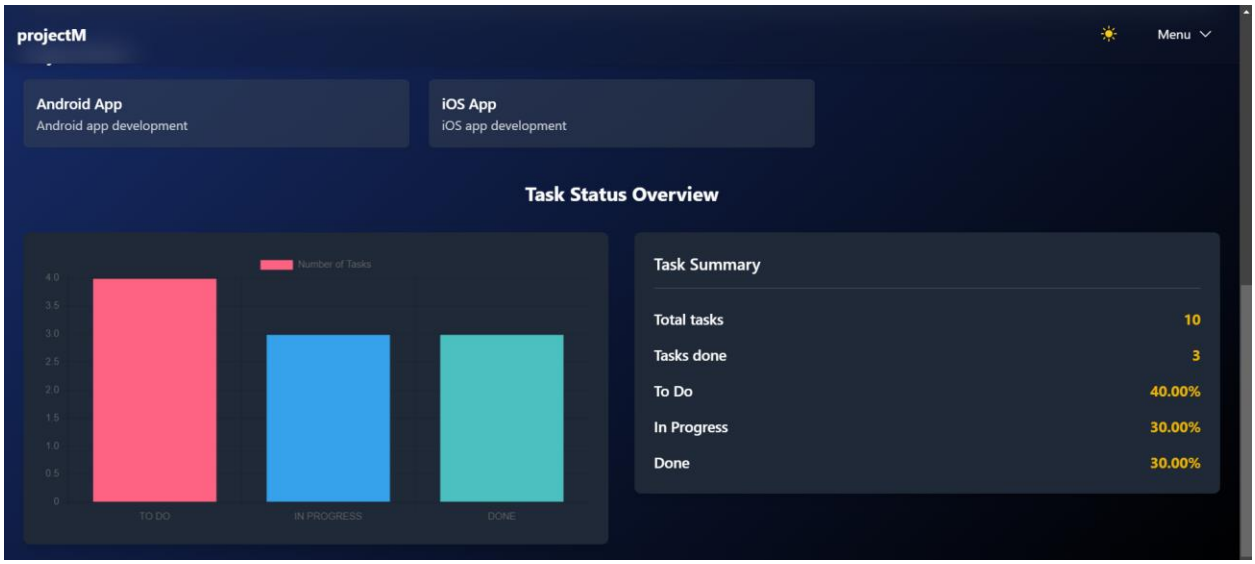
Slika 4.14.: *Sučelje koje prikazuje projekte u kojima je korisnik uključen*

Klikom na određeni projekt otvara se sučelje koje pokazuje detalje odabranog projekta. Na slici 4.15., prikazana je traka napretka projekta s trenutnom dovršenosti od 30%. Ispod su navedeni članovi projekta s njihovim email adresama. Ovakav prikaz omogućava praćenje napretka projekta i transparentnost u raspodjeli zadataka. Pruža brz uvid u odgovornosti članova tima.



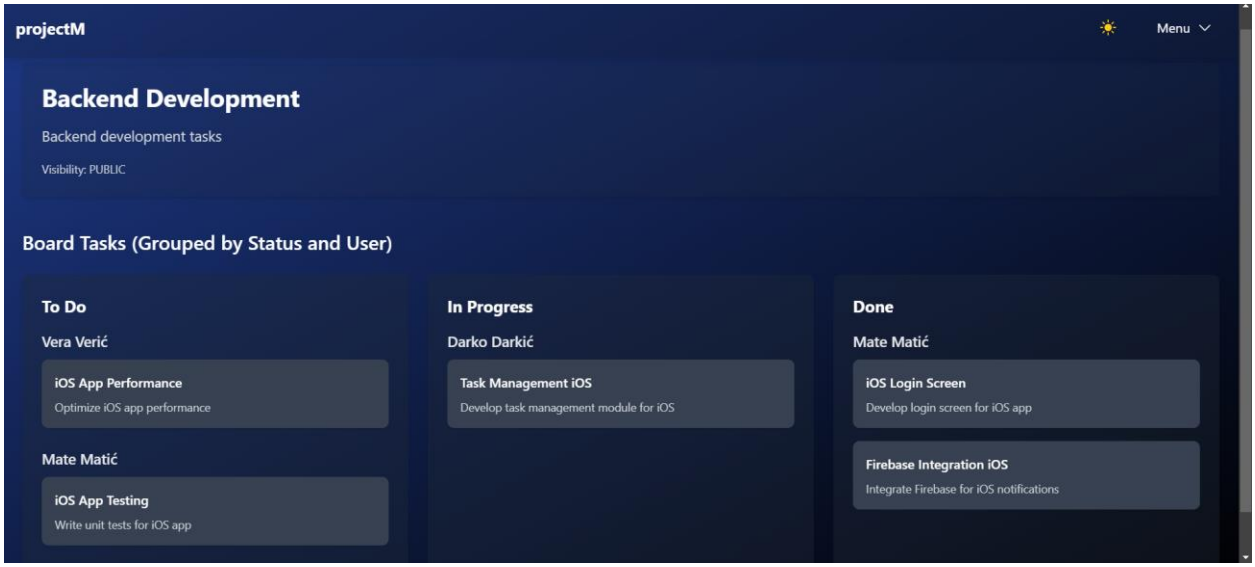
Slika 4.15.: *Sučelje koje prikazuje detalje o određenom projektu*

Implementiran je stupčasti grafikon s brojem zadataka po fazama (Sl. 4.16.): *TO DO*, *IN PROGRESS* i *DONE*. Desno se nalazi sažetak s ukupnim brojem zadataka i njihovom raspodjelom. Ovakav prikaz olakšava praćenje napretka projekta i statusa zadataka. Koristan je za bolje upravljanje radnim procesima.



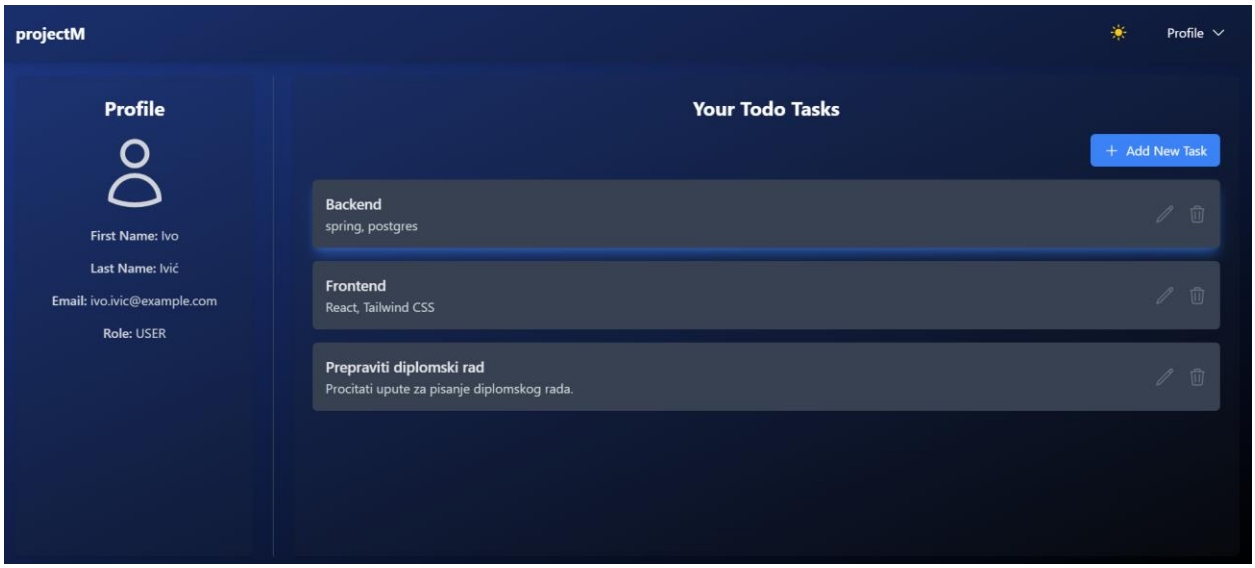
Slika 4.16.: *Sučelje koje prikazuje detalje o određenom projektu*

Zadaci unutar ploče "*Backend Development*" organizirani su prema statusima *TO DO*, *IN PROGRESS* i *DONE*. (Sl. 4.17.) Svaka kartica prikazuje ime zadatka i odgovornu osobu. Ovakav prikaz omogućava korisnicima praćenje statusa i odgovornosti za zadatke. Pruža se jasna slika trenutnog stanja ploče i zadataka.



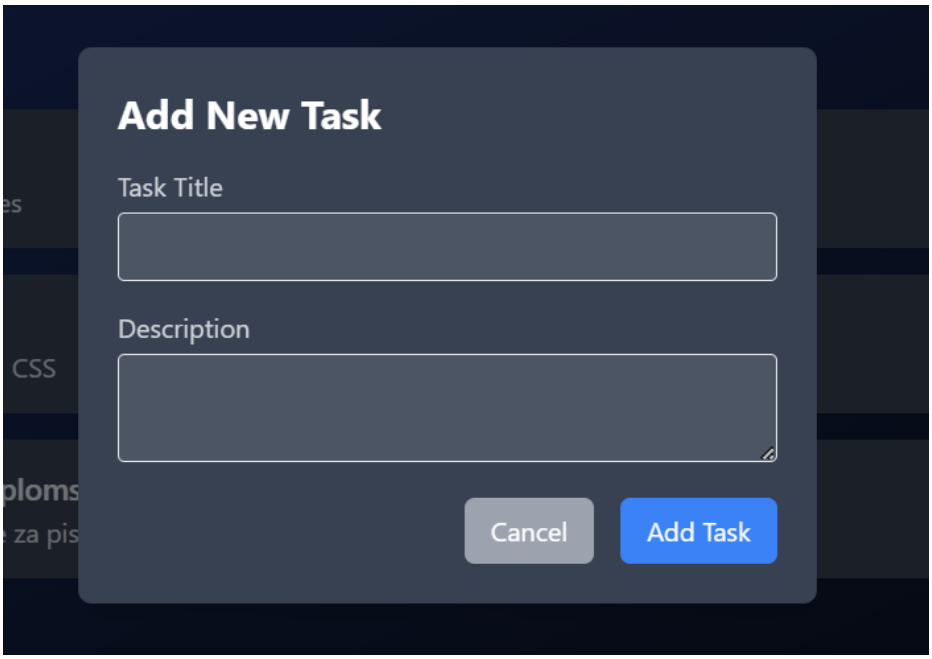
Slika 4.17.: *Prikaz jednog Boarda unutar nekog projekta sa drag-and-drop komponentama*

Na slici 4.18. prikazane su osnovne informacije o korisniku i popis dodijeljenih *todo* zadataka. Zadaci imaju mogućnost uređivanja ili brisanja, a novi se *todo* zadaci mogu dodati putem gumba "*+Add New Task*". Ovaj prikaz omogućava upravljanje osobnim zadacima i pregled korisničkih informacija. Korisniku nudi jednostavan način organizacije privatnih *todo* zadataka.



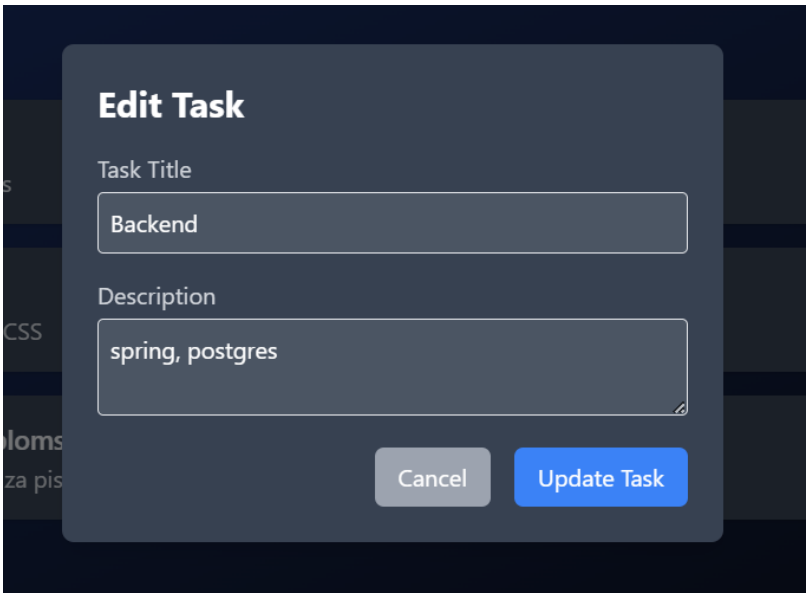
Slika 4.18.: *Korisnikov profil*

Korisnik ima mogućnost dodavanja novog zadatka unosom naziva zadatka (*Task Title*) i opisa (*Description*), a postupak se potvrđuje klikom na gumb "*Add Task*" (Sl. 4.19.). Ova funkcionalnost omogućava brzo i jednostavno kreiranje zadataka što je ključno za učinkovitu organizaciju i upravljanje projektima.



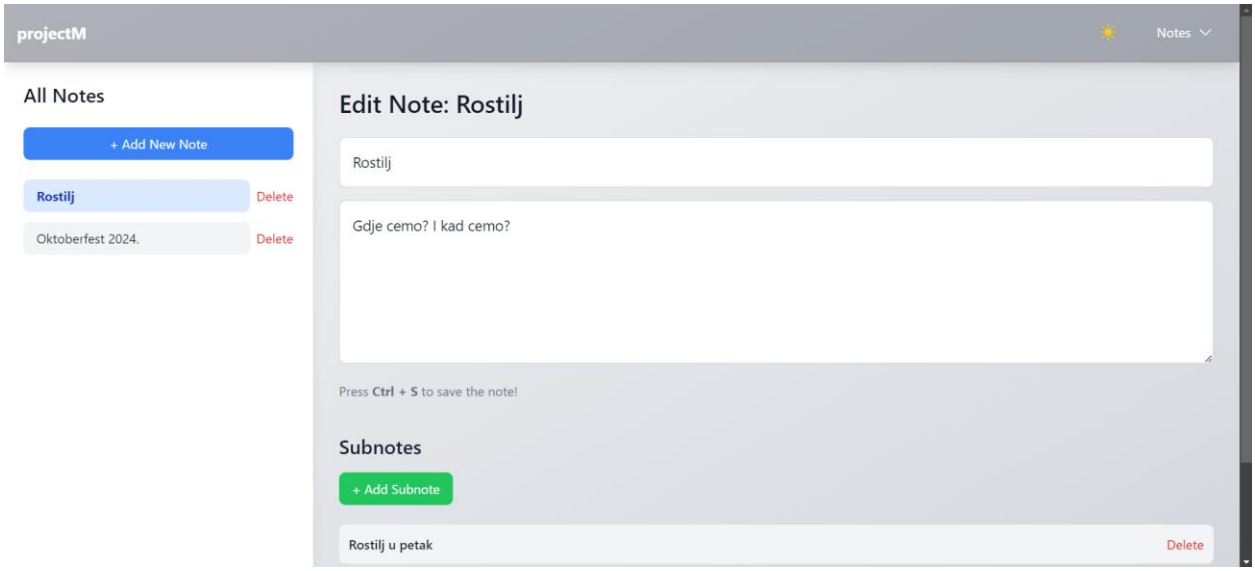
Slika 4.19.: *Dodavanje novog todo taska*

Korisnik može urediti postojeći zadatak promjenom njegovog naziva i opisa, nakon čega se promjene potvrđuju klikom na gumb "*Update Task*" (Sl. 4.20.). Ovaj proces omogućava fleksibilno prilagođavanje zadataka u stvarnom vremenu što doprinosi boljem praćenju i upravljanju radnim procesima te održavanju točnih informacija o zadacima unutar sustava.



Slika 4.20.: Uređivanje korisnikovog todo taska

Na slici 4.21. prikazan je izgled generalnog *notesa* za sve članove unutar jedne tvrtke. Korisnici mogu dodavati bilješke i uređivati već postojeće bilješke te je sve na razini cijele tvrtke.



Slika 4.21.: *Generalni notes za sve korisnike unutar firme*

5. Zaključak

U ovom diplomskom radu obrađena je izrada web aplikacije za upravljanje projektima i projektnim timovima s posebnim naglaskom na korištenje modernih tehnologija kao što su Spring Framework, Hibernate, PostgreSQL i React. Cilj je bio kreirati besplatnu, prilagodljivu alternativu postojećim komercijalnim alatima poput Jira, Notion i ClickUp, koja je usmjerena na potrebe projektnog tima i pojedinačnih članova.

Kroz implementaciju su prikazane osnovne funkcionalnosti aplikacije, uključujući praćenje projekata, zadataka i statusa članova tima. Korištenje Spring Boot-a za backend te React-a za frontend omogućilo je razvoj aplikacije koja je modularna, lako proširiva i jednostavna za održavanje. Primjena tehnologija kao što su JWT za autentifikaciju i Axios za komunikaciju između klijenta i poslužitelja osigurala je visoku razinu sigurnosti i efikasnosti aplikacije.

Usporedbom razvijene aplikacije s popularnim alatima na tržištu istaknute su prednosti korištenja besplatnog, prilagodljivog rješenja koje odgovara specifičnim potrebama korisnika. Implementacijom jednostavnog i preglednog korisničkog sučelja te podrškom za upravljanje projektima, aplikacija omogućava poboljšanje produktivnosti i efikasnosti rada timova. Ovaj rad predstavlja temelj za daljnji razvoj i proširenje funkcionalnosti, uz mogućnost integracije s dodatnim tehnologijama i alatima.

Literatura

[1] “Jira Software Support Documentation“ [online], Atlassian, dostupno na: https://confluence.atlassian.com/jira [posjećeno 2. kolovoza 2024.].

[2] „Notion Docs „[online], Notion, dostupno na: https://www.notion.so/product/docs [posjećeno 2. kolovoza 2024

[3] B. Eckel, "Thinking in Java", 4th Edition, Prentice Hall, 2006.

[4] B. Evans, C. Glover, "Java Persistence with Hibernate", Second Edition, Manning Publications, 2015.

[5] C. Walls, "Spring in Action", 5th Edition, Manning Publications, 2018.

[6] "Spring Framework Documentation" [online], Spring, dostupno na: https://spring.io/projects/spring-framework [posjećeno 3. kolovoza 2024.]

[7] "Spring Boot Documentation" [online], Spring, dostupno na: https://spring.io/projects/spring-boot [posjećeno 3. kolovoza 2024.]

[8] "React Documentation" [online], React, dostupno na: https://reactjs.org/ [posjećeno 5. kolovoza 2024.]

[9] M. Warcholinski, "10 Famous React Apps: Examples (2022): Why do Internet giants choose React apps?" [online], Brainhub, dostupno na: https://brainhub.eu/library/famous-apps-using-reactjs [posjećeno 10. kolovoza 2024.]

[10] "Typescript Documentation" [online], TypeScript, dostupno na: https://www.typescriptlang.org/docs/ [posjećeno 10. kolovoza 2024.].

[11] "Axios Documentation" [online], Axios, dostupno na: https://axios-http.com/docs/intro [posjećeno 19. kolovoza 2024.]

[12] "Postman Documentation" [online], Postman, dostupno na: https://www.postman.com/ [posjećeno 19. kolovoza 2024.]

[13] "PostgreSQL Documentation" [online], PostgreSQL, dostupno na: https://www.postgresql.org/docs/ [posjećeno 26. kolovoza 2024..]

[14] "Hibernate ORM Documentation" [online], Hibernate, dostupno na: https://hibernate.org/orm/documentation/ [posjećeno 26. kolovoza 2024.]

[15] "Lombok Documentation" [online], Project Lombok, dostupno na: https://projectlombok.org/ [posjećeno 26. kolovoza 2024.]

[16] "ModelMapper Documentation" [online], ModelMapper, dostupno na: http://modelmapper.org/ [posjećeno 26. kolovoza 2024.]

[17] "SpringJPA Documentation" [online], SpringJPA, dostupno na: https://docs.spring.io/spring-data/jpa/reference/index.html [posjećeno 26. kolovoza 2024.]

[18] "JWT (JSON Web Token) Documentation" [online], JJWT, dostupno na: https://github.com/jwtk/jjwt [posjećeno 26. kolovoza 2024.]

Sažetak

Ovaj rad prikazuje razvoj web aplikacije za upravljanje projektima i projektnim timovima koristeći Spring Boot za backend i React za frontend. Cilj je bio kreirati besplatnu i prilagodljivu alternativu postojećim alatima poput Jira, Notion i ClickUp, koja bi omogućila timovima efikasnije praćenje projekata, zadataka i napretka članova tima. Kroz rad su detaljno opisane korištene tehnologije, uključujući PostgreSQL, Hibernate, JWT za autentifikaciju te Axios za komunikaciju između frontenda i backenda. Aplikacija pruža osnovne funkcionalnosti poput CRUD operacija, autentifikacije korisnika i upravljanja projektima te je prilagodljiva potrebama korisnika. Zaključno, rad pruža temelj za daljnji razvoj i proširenje aplikacije, kao i usporedbu s postojećim komercijalnim rješenjima.

Ključne riječi: Axios, Hibernate, OAuth, projekt menadžment, ReactJS, Spring Boot, Spring Framework, TailwindCSS, Web aplikacija

Abstract

This thesis presents the development of a web application for project and team management using Spring Boot for the backend and React for the frontend. The goal was to create a free and customizable alternative to existing tools such as Jira, Notion, and ClickUp, enabling teams to more efficiently track projects, tasks, and team member progress. The thesis provides a detailed description of the technologies used, including PostgreSQL, Hibernate, JWT for authentication, and Axios for communication between the frontend and backend. The application offers core functionalities such as CRUD operations, user authentication, and project management, and is adaptable to user needs. In conclusion, the work provides a foundation for further development and expansion of the application, as well as a comparison with existing commercial solutions.

Keywords: Axios, Hibernate, OAuth, project management, ReactJS, Spring Boot, Spring Framework, TailwindCSS, Web application.

Prilozi

Prilog 1. Diplomski rad u datoteci docx.

Prilog 2. Diplomski rad u datoteci pdf.

Prilog 3. Github repozitoriji programskog *koda* aplikacije za *backend* i *frontend* dijelove:
https://github.com/bbj97/diplomski_frontend.git
https://github.com/bbj97/diplomski_backend.git