SUSTAV ZA UPRAVLJANJE IZDANJIMA I ZNAČAJKAMA APLIKACIJE

Šolić, Ante

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni prijediplomski studij Računarstvo

Ante Šolić

Z A V R Š N I R A D

Sustav za upravljanje izdanjima i značajkama aplikacije

Split, rujan 2024.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni prijediplomski studij Računarstvo

Predmet: Programiranje u Javi

Z A V R Š N I R A D

Kandidat: Ante Šolić

Naslov rada: Sustav za upravljanje izdanjima i značajkama aplikacije

Mentor: viši predavač Josip Vrlić, dipl. ing. rač.

Split, rujan 2024.

Sadržaj

Sažetak

Cilj završnog rada je razvoj web aplikacije za upravljanje izdanjima i značajkama kojom će se moći bolje organizirati razvoj softvera. Aplikacija je namijenjena upravljanju izdanjima *cloud-based* web aplikacija i sastoji se od *frontenda*, *backenda* te pripadajuće biblioteke za *feature flag*. Ključne značajke aplikacije uključuju sustav za registraciju i autentikaciju korisnika, upravljanje projektima prema korisničkim ulogama, te praćenje i upravljanje datumima izdanja, statusima izdanja i značajkama povezanima s izdanjem. Aplikacija također omogućuje upravljanje klijentima, uključujući prilagodbu dostupnih značajki ovisno o potrebama klijenta. Integracija svih ovih komponenti osigurava potpunu kontrolu nad razvojnim procesom softvera, od najmanjih značajki do cjelokupnih projekata. Za razvoj aplikacije korištene su tehnologije Java, React, Spring Boot, PostgreSQL, Flyway.

Ključne riječi: biblioteka, izdanje, klijent, projekt, značajka

Summary

Release and feature management system

The aim of this thesis is the development of a web application for release and feature management, designed to better organize software development. The application is intended for managing releases of cloud-based web applications and consists of a frontend, backend, and a library for feature flag. The key features of the application include a user registration and authentication system, project management based on user roles, and tracking and managing release dates, release statuses, and features associated with releases. The application also enables client management, including the customization of available features according to client needs. The integration of all these components ensures complete control over the software development process, from the smallest features to entire projects. The technologies used for the development of the application include Java, React, Spring Boot, PostgreSQL, and Flyway.

Keywords: client, feature, library, project, release

1. Uvod

Učinkovito upravljanje softverskim izdanjima ključno je za osiguravanje nesmetanih implementacija, minimiziranje zastoja i održavanje visoke kvalitete softverskih proizvoda. Sustav za upravljanje izdanjima i značajkama, koji uključuje planiranje, raspoređivanje i kontrolu softverskih verzija, igra važnu ulogu u uspjehu softverskih projekata. Kako se složenost softverskih sustava povećava, raste i potreba za naprednim alatima koji mogu učinkovito upravljati svim dijelovima razvoja i implementacije softvera. U ovom kontekstu, web aplikacije za upravljanje izdanjima i značajkama postaju nezamjenjiv alat za razvojne timove, omogućujući im da optimiziraju procese, smanje rizike i osiguraju pravovremenu isporuku softverskih rješenja.

Ovaj završni rad predstavlja razvoj web aplikacije osmišljene za rješavanje izazova upravljanja softverskim izdanjima. Aplikacija, razvijena korištenjem modernih web tehnologija kao što su React za *frontend*, Java i Spring Boot za *backend*, PostgreSQL za bazu podataka i Flyway za migracije baza podataka, nudi niz značajki usmjerenih na pojednostavljenje procesa izdavanja softvera za *cloud-based* web aplikacije. Primarni cilj ove aplikacije je pružiti platformu koja poboljšava organizaciju i učinkovitost timova za razvoj softvera automatiziranjem i pojednostavljenjem upravljanja projektima, izdanjima, značajkama, klijentima i korisnicima.

Jedan od temeljnih aspekata aplikacije je implementacija autentikacije korisnika. Koristeći Spring Security, aplikacija osigurava da se korisnici mogu sigurno registrirati i prijaviti u sustav. Ova značajka je ključna za održavanje integriteta i sigurnosti aplikacije, osiguravajući da samo ovlašteno osoblje ima pristup određenim funkcionalnostima temeljenim na njihovim ulogama. Uspostavljanjem mehanizma autentikacije, aplikacija postavlja temelje za siguran i kontroliran pristup svim ostalim značajkama.

Učinkovito upravljanje softverskim projektima ključno je za koordinaciju različitih timova i osiguravanje da su svi zadaci usklađeni. Modul za upravljanje projektima u aplikaciji omogućava voditeljima projekata stvaranje, uređivanje i brisanje projekata. Dodatno, aplikacija nudi funkcionalnosti za pretraživanje, filtriranje i sortiranje projekata, što voditeljima projekata omogućava učinkovito organiziranje i upravljanje projektima. Ovaj

modul služi kao središnje mjesto iz kojeg se koordiniraju sve sljedeće aktivnosti upravljanja izdanjima i značajkama.

U središtu aplikacije nalazi se modul za upravljanje izdanjima, koji je posebno dizajniran za olakšavanje stvaranja i upravljanja softverskim izdanjima. Voditelji izdanja imaju na raspolaganju alate za kreiranje, uređivanje i brisanje izdanja povezanih s različitim projektima. Mogućnost pretraživanja, filtriranja i sortiranja izdanja omogućava učinkovito praćenje i upravljanje cjelokupnim procesom izdanja. Nadalje, aplikacija uvodi funkcionalnost koja omogućava uključivanje ili isključivanje značajki na temelju datuma izdanja, pružajući dinamičan pristup upravljanju softverskim značajkama u odnosu na njihove rasporede izdanja. Ova sposobnost posebno je korisna u okruženjima gdje je potrebno postupno uvođenje značajki ili njihovo selektivno omogućavanje za specifične korisnike ili grupe.

Osim upravljanja izdanjima, aplikacija uključuje i modul za upravljanje značajkama. Ovaj modul omogućava razvojnim inženjerima dodavanje, uređivanje i upravljanje značajkama unutar aplikacije. Ključna funkcionalnost ovog modula je mogućnost uključivanja ili isključivanja značajki, čime se pruža fleksibilnost u načinu na koji se značajke distribuiraju krajnjim korisnicima. Aplikacija podržava različite vrste omogućavanja značajki, kao što su omogućavanje značajki za određene ili sve korisnike.

Modul za upravljanje klijentima omogućava administratorima dodjeljivanje ili uklanjanje klijenata iz određenih značajki, pružajući mehanizam za kontrolu pristupa značajkama na razini pojedinog klijenta. Administrator ima i mogućnost pregleda svih korisničkih računa te njihovog uređivanja i brisanja. Aplikacija također uključuje funkcionalnosti za pretraživanje, filtriranje i sortiranje korisničkih računa, čime se administratorima olakšava upravljanje velikim brojem korisnika na učinkovit način. Integracijom upravljanja klijentima s upravljanjem značajkama, aplikacija pruža sveobuhvatno rješenje za upravljanje specifičnim zahtjevima klijenata i uvođenjem značajki.

U idućim poglavljima opisane su korištene tehnologije kao što su Java, Spring Boot, React, PostgreSQL i Flyway, aplikacija i njene značajke, baza podataka sa slikom modela, opisom entiteta te načinom migracije, implementacija gdje su opisane najvažnije klase te primjeri kôda. Također je opisan način postavljanja aplikacije na čistom računalu ili poslužitelju nakon čega slijedi zaključak završnog rada.

2. Korištene tehnologije

2.1. Java

Java je jedan od najpopularnijih programskih jezika na svijetu, prvi put predstavljen 1995. godine, a danas se koristi za izradu različitih vrsta softvera od web aplikacija do mobilnih aplikacija.

Jedna od ključnih prednosti Jave je njezina platformska neovisnost. To znači da se Java programi mogu pokretati na bilo kojem uređaju ili operativnom sustavu koji podržava JVM (Java Virtual Machine). Kad se Java program prevodi (engl. *compile*), ne pretvara se direktno u strojni kôd (engl. *Machine code*), već u bajtkôd (engl. *bytecode*), koji je neutralan prema platformi. JVM zatim čita i izvršava taj bajtkôd, što omogućava da isti program radi na različitim operativnim sustavima bez potrebe za ponovnim prevođenjem.

JIT (Just-In-Time) prevođenje je optimizacija koja se koristi u JVM-u. Umjesto da se svaki bajtkôd prevodi u vrijeme izvršavanja, JIT prevodilac (engl. *compiler*) analizira izvršavanje programa i dinamički prevodi često korištene dijelove kôda u strojni kôd. To značajno povećava performanse Java programa.

Java je također objektno-orijentirani jezik, što znači da se programi temelje na objektima i klasama. Ovaj pristup olakšava organizaciju kôda i omogućuje ponovnu uporabu komponenti, što smanjuje vrijeme razvoja i održavanja softvera. Programeri mogu koristit postojeće biblioteke klasa koje pružaju razne funkcionalnosti, od jednostavnih operacija kao što su matematički izračuni, do složenih kao što je upravljanje bazama podataka.

2.2. Spring Boot

Spring Boot je razvojni okvir unutar Spring ekosustava koji je dizajniran za pojednostavljenje procesa razvoja i implementacije aplikacija na temelju Java programskog jezika. Njegov glavni cilj je omogućiti programerima da brzo i lako započnu s razvojem aplikacija, eliminirajući pritom potrebu za konfiguracijom koja je često kompleksna i dugotrajna u standardnom razvojnom okviru Spring.

Spring Boot automatski konfigurira aplikaciju na temelju komponenti prisutnih na klasnoj putanji (engl. *Classpath*). Ovo značajno smanjuje potrebu za ručnom

konfiguracijom, omogućujući programerima da se fokusiraju na pisanje kôda umjesto na podešavanje konfiguracijskih datoteka.

Spring Boot uvodi koncept *starter* paketa koji uključuju sve potrebne ovisnosti za određeni tip aplikacije. Na primjer, `spring-boot-starter-web` uključuje sve potrebne ovisnosti za razvoj web aplikacija, poput Spring MVC-a, ugrađenog Tomcat poslužitelja i Jacksona za rad s JSON podacima. Ovaj pristup pojednostavljuje upravljanje ovisnostima i osigurava konzistentnost među njima.

Spring Boot dolazi s ugrađenim web poslužiteljima kao što su Tomcat, Jetty i Undertow. To znači da se aplikacija može pokrenuti kao samostalna Java aplikacija s jednostavnom `java -jar` naredbom, bez potrebe za instalacijom vanjskog web poslužitelja. Ova značajka omogućava brzu implementaciju aplikacija, smanjujući kompleksnost okruženja za implementaciju.

Spring Boot omogućava konfiguraciju aplikacija putem datoteka `application.properties` ili `application.yml` ovisno o tome koristi li se `maven` ili `gradle`. Ove datoteke omogućavaju jednostavno postavljanje konfiguracijskih parametara kao što su postavke baze podataka, sigurnosne opcije i vanjske API konekcije. Fleksibilnost ovih datoteka omogućava brze promjene u ponašanju aplikacije bez potrebe za rekonstrukcijom ili ponovnim pokretanjem aplikacije.

Jedna od ključnih prednosti Spring Boota je njegova sposobnost da ubrza proces razvoja aplikacija. Razvojni proces obično započinje izborom odgovarajućeg *starter* paketa ovisno o vrsti aplikacije koja se razvija (npr. web aplikacija, aplikacija s RESTful API-jem, aplikacija s pristupom bazi podataka itd.). Nakon početnog postavljanja, programeri mogu odmah započeti s implementacijom poslovne logike, budući da je većina infrastrukture automatski konfigurirana.

Spring Boot predstavlja ključni alat u modernom Java razvoju, omogućujući brzi razvoj, jednostavnu konfiguraciju i visoku učinkovitost aplikacija. Svojom sposobnošću da integrira različite tehnologije i pojednostavi složene procese, Spring Boot je postao jedan od najpopularnijih razvojnih okvira za razvoj raznovrsnih aplikacija, od malih uslužnih aplikacija do velikih poduzetničkih sustava. U kontekstu ovog završnog rada, Spring Boot

je bio ključan u postizanju efikasnosti i organiziranosti cijelog sustava upravljanja softverskim izdanjima.

2.3. React

React je JavaScript biblioteka otvorenog kôda razvijena od strane Facebooka, koja je postala jedan od najpopularnijih alata za izgradnju korisničkih sučelja (engl. *User interface*) na webu. Glavni cilj Reacta je omogućiti razvoj dinamičkih i responzivnih korisničkih sučelja na učinkovit način. Njegova snaga leži u komponentnom pristupu, koji omogućava razdvajanje aplikacije na male, ponovno iskoristive dijelove, što olakšava razvoj, održavanje i skaliranje web aplikacija.

React omogućava razdvajanje korisničkog sučelja na male, neovisne komponente. Svaka komponenta je samostalna jedinica koja može imati vlastitu logiku i stanje, a može se ponovno koristiti unutar različitih dijelova aplikacije. Komponentni model olakšava razvoj složenih aplikacija jer omogućava modularnost i jednostavnije testiranje pojedinačnih dijelova aplikacije.

React koristi koncept Virtualnog DOM-a (Virtual Document Object Model), koji je efikasna replika stvarnog DOM-a u memoriji. Kada se stanje aplikacije promijeni, React prvo ažurira Virtual DOM, a zatim uspoređuje promjene s pravim DOM-om (proces poznat kao "*reconciliation*"). Ovaj pristup značajno poboljšava performanse jer minimizira broj operacija nad stvarnim DOM-om, koje su često skupe u smislu performansi.

React koristi jednosmjerni tok podataka (engl. *One-way data binding*), što znači da podaci teku u jednom smjeru, od roditeljskih komponenti prema dječjim. Ovaj pristup osigurava bolju kontrolu nad podacima i smanjuje mogućnost pogrešaka koje su često prisutne u dvosmjernim data-binding modelima.

React koristi JSX, sintaksu koja omogućava pisanje HTML-a unutar JavaScript kôda. JSX kombinira snagu JavaScripta s fleksibilnošću HTML-a, omogućujući programerima da lako kreiraju komponente s bogatom logikom i strukturama. Iako nije obavezan, JSX je široko prihvaćen zbog svoje sposobnosti da poboljša čitljivost i održavanje kôda.

React se može koristiti ne samo za razvoj web aplikacija već i za izgradnju mobilnih aplikacija kroz React Native. Ova univerzalnost omogućava programerima da koriste iste

principe i kôdnu bazu za razvoj aplikacija na različitim platformama, čime se smanjuju troškovi razvoja i vrijeme potrebno za izgradnju proizvoda.

React je dizajniran da bude fleksibilan i jednostavan za integraciju u različite dijelove web aplikacija, bilo da se koristi za izgradnju cijelog korisničkog sučelja ili samo za pojedine dijelove aplikacije. React aplikacije obično se razvijaju kroz kreiranje niza malih, samostalnih komponenti koje se međusobno kombiniraju kako bi se kreirala kompleksnija struktura.

U okviru ovog završnog rada, React je korišten za izradu korisničkog sučelja aplikacije za upravljanje softverskim izdanjima. React omogućava razvoj dinamičkih i responzivnih sučelja koja olakšavaju upravljanje projektima, izdanjima, značajkama, klijentima i korisnicima. Komponentni pristup Reacta omogućio je stvaranje modularnog korisničkog sučelja koje se lako može proširiti i prilagoditi novim zahtjevima.

Kroz korištenje Reacta, sučelje aplikacije postalo je intuitivno i lako za korištenje, pružajući korisnicima jasno strukturirane i lako dostupne funkcionalnosti. Integracija Reacta s *backend* sustavom razvijenim u Spring Bootu omogućila je učinkovitu komunikaciju između *frontenda* i *backenda*, čime je postignuta visoka razina interaktivnosti i brzine odziva aplikacije.

React se postavio kao jedan od vodećih alata za razvoj modernih web aplikacija, nudeći moćne značajke i fleksibilnost koja je potrebna za izgradnju responzivnih i proširivih korisničkih sučelja. Njegova sposobnost da pojednostavi razvoj kroz komponentni model, uz poboljšanje performansi korištenjem Virtual DOM-a, čini ga idealnim izborom za projekte koji zahtijevaju dinamičnost i brzinu. U kontekstu ovog završnog rada, React je bio ključan u izgradnji korisničkog sučelja koje omogućava efikasno upravljanje softverskim izdanjima i drugim bitnim aspektima aplikacije.

2.4. PostgreSQL

PostgreSQL je sustav otvorenog koda (engl. *Open-Source*) za upravljanje relacijskim bazama podataka (RDBMS) koji je poznat po svojoj stabilnosti, fleksibilnosti i skalabilnosti. Razvijen je s ciljem podržavanja naprednih karakteristika kao što su složeni upiti, integritet podataka, podrška za različite tipove podataka i transakcije. PostgreSQL je postao jedan od najpopularnijih izbora za razvojne inženjere i administratore baza podataka zbog svoje

sposobnosti da pouzdano rukuje velikim količinama podataka i kompleksnim aplikacijskim potrebama.

PostgreSQL implementira puni SQL standard (ANSI-SQL), što omogućava složene upite, uključujući CTE (Common Table Expressions), podupite, transakcije, JOIN operacije i slično. Ova podrška omogućava programerima da kreiraju detaljne i optimizirane upite koji mogu manipulirati podacima na složen način.

PostgreSQL podržava širok raspon tipova podataka, uključujući standardne tipove (INTEGER, VARCHAR, DATE), kao i napredne tipove kao što su JSON, HSTORE, XML i ARRAY. To omogućava aplikacijama da pohranjuju raznolike podatke i upravljaju njima na način koji odgovara specifičnim poslovnim potrebama.

PostgreSQL pruža mehanizme za osiguranje integriteta podataka, uključujući primarne i strane ključeve, jedinstvene indekse, NOT NULL ograničenja i CHECK ograničenja. Ovi mehanizmi osiguravaju dosljednost i točnost podataka u bazi, što je ključno za aplikacije koje zahtijevaju visoku razinu pouzdanosti.

PostgreSQL je često prvi izbor za *backend* sustave u razvoju web aplikacija. Njegova sposobnost da se lako integrira s različitim aplikacijskim okvirima, uključujući Spring Boot, omogućava jednostavno upravljanje podacima unutar složenih sustava. Razvoj aplikacija s PostgreSQLom obično započinje dizajnom sheme baze podataka, definirajući tablice, indekse, odnose i ograničenja koja su ključna za aplikaciju.

Jedna od značajnih prednosti PostgreSQLa u razvoju aplikacija je njegova podrška za JSON tip podataka, koji omogućava pohranu nestrukturiranih podataka unutar relacijskih tablica. Ovo je posebno korisno za aplikacije koje rade s dinamičkim ili nestandardiziranim podacima, omogućavajući fleksibilno rukovanje i brzu analizu podataka.

U okviru ovog završnog rada, PostgreSQL je korišten kao glavni sustav za upravljanje bazom podataka aplikacije za upravljanje softverskim izdanjima. Baza podataka pohranjuje sve bitne podatke o projektima, izdanjima, značajkama, korisnicima i klijentima. PostgreSQL je osigurao stabilnu platformu za pohranu i pristup podacima, omogućavajući složene upite i brzu obradu podataka potrebnih za funkcionalnost aplikacije.

Jedna od ključnih prednosti korištenja PostgreSQLa u ovom projektu bila je njegova mogućnost rukovanja transakcijama i osiguranje integriteta podataka, što je bilo ključno za točnost i pouzdanost sustava. Korištenje alata poput Flywaya za migraciju baza podataka omogućilo je jednostavno upravljanje promjenama u shemi baze podataka tijekom cijelog životnog ciklusa aplikacije.

PostgreSQL se dokazao kao moćan i fleksibilan sustav za upravljanje relacijskim bazama podataka, koji može zadovoljiti širok raspon potreba modernih aplikacija. Njegova stabilnost, te podrška za složene podatkovne strukture i operacije čine ga idealnim izborom za razvojne timove koji žele izgraditi pouzdane i proširive sustave. U kontekstu ovog završnog rada, PostgreSQL je bio ključan element u osiguravanju učinkovite i sigurne pohrane podataka, te je omogućio aplikaciji da pruži sve potrebne funkcionalnosti uz visoke performanse.

2.5. Flyway

Flyway je popularan alat otvorenog kôda za upravljanje migracijama baza podataka. Njegova glavna svrha je olakšati razvojnim timovima upravljanje promjenama u shemi baze podataka na konzistentan i kontroliran način. Flyway omogućava automatizaciju procesa migracije baze podataka, čime se smanjuje mogućnost pogrešaka i osigurava da sve instance baze podataka unutar različitih okruženja (razvoj, testiranje, produkcija) budu u skladu s najnovijom verzijom aplikacije.

Flyway koristi numeriranje verzija migracijskih skripti koje omogućava timovima praćenje i kontrolu nad promjenama sheme baze podataka. Svaka migracija je numerirana i izvršava se redoslijedom verzija, čime se osigurava da su promjene primijenjene u ispravnom slijedu.

Flyway je izuzetno jednostavan za upotrebu, bez potrebe za dodatnim konfiguracijama. Razvojni timovi mogu brzo početi s implementacijom migracija jednostavnim dodavanjem SQL ili Java skripti u predviđeni direktorij. Flyway podržava širok raspon relacijskih baza podataka, uključujući PostgreSQL, MySQL, Oracle, SQL Server, i mnoge druge. Ova svestranost čini ga pogodnim za upotrebu u različitim okruženjima i s različitim sustavima baza podataka.

Flyway omogućava povratak na prethodnu verziju baze podataka u slučaju da se pojave problemi s novim migracijama. Ova značajka je kritična za osiguranje stabilnosti aplikacije tijekom nadogradnji. Flyway omogućava višekorisničko upravljanje migracijama, gdje više članova tima može dodavati, ažurirati ili poništavati migracije bez rizika od konflikata. Svaka promjena je jasno dokumentirana i verzionirana, što olakšava praćenje i reviziju promjena.

U okviru ovog završnog rada, Flyway je korišten za upravljanje migracijama baze podataka PostgreSQL. Kako se aplikacija razvijala i kako su se zahtjevi mijenjali, bilo je potrebno redovito ažurirati shemu baze podataka kako bi podržala nove funkcionalnosti.

Flyway je omogućio jednostavno dodavanje novih migracija i automatsko primjenjivanje tih migracija na različitim instancama baze podataka. Ovaj pristup osigurao je da sve baze podataka ostanu u konzistentnom stanju bez potrebe za ručnim ažuriranjima ili složenim skriptama za migraciju.

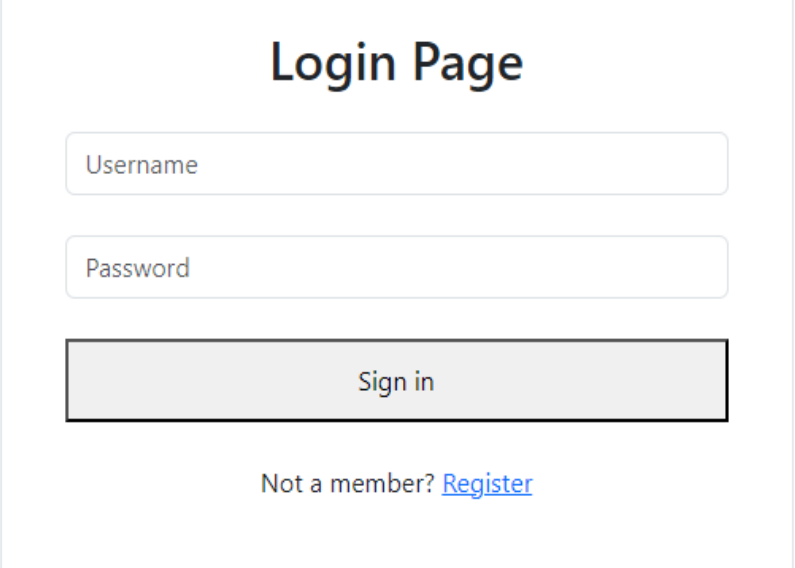
Korištenje Flywaya također je smanjilo rizik od ljudske pogreške prilikom implementacije promjena u bazi podataka, jer je cijeli proces migracije bio automatiziran i strogo kontroliran kroz verzioniranje.

Flyway je ključan alat za upravljanje migracijama baza podataka u modernom razvoju aplikacija. Njegova jednostavnost korištenja, zajedno s moćnim značajkama kao što su upravljanje verzijama, podrška za vraćanje na prethodnu verziju (engl. *Rollback*), i integracija s CI/CD alatima, čini ga neizostavnim dijelom svakog ozbiljnog projektnog razvoja. U kontekstu ovog završnog rada, Flyway je igrao važnu ulogu u osiguravanju dosljednosti i stabilnosti baze podataka kroz cijeli razvojni ciklus aplikacije, omogućavajući da se fokus prebaci na razvoj novih funkcionalnosti bez brige o ručnom upravljanju promjenama u bazi podataka.

3. Opis Aplikacije

3.1 Registracija i prijava korisnika

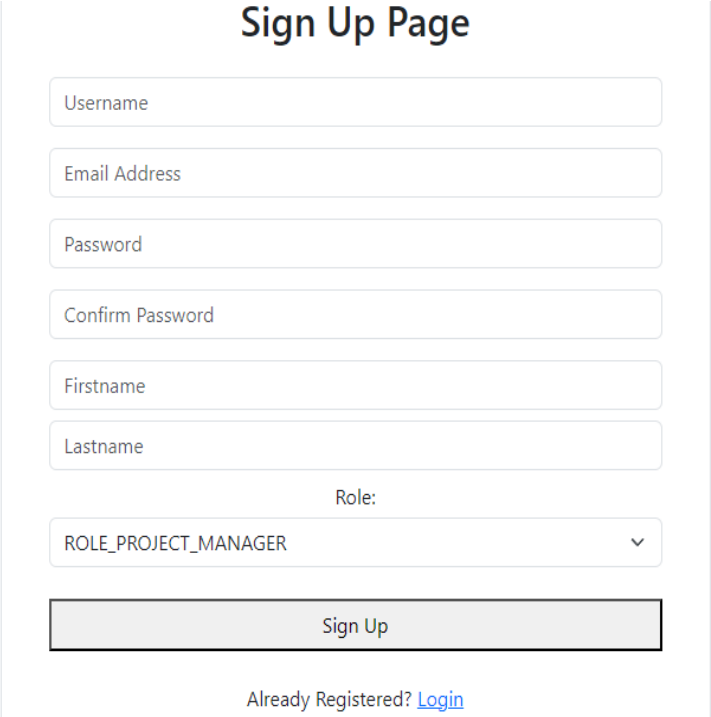
Kada korisnik prvi put pristupi aplikaciji, prikazuje mu se opcija za prijavu u sustav. Prijava je u potpunosti sigurna zbog korištenja Spring Security sustava. Ako korisnik nema važeći korisnički račun isti može napraviti klikom na link za registraciju.



Slika 1: Sučelje za prijavu

Ulaskom na link za registraciju korisniku se prikazuje sučelje za registraciju korisnika prikazano na Slici 2. Unutar samog sučelja korisnik može izabrati svoje korisničko ime, adresu elektroničke pošte koju koristi, svoje ime i prezime, unijeti i potvrditi svoju lozinku te odabrati ulogu. Korisničko ime i adresa elektroničke pošte korisnika moraju biti jedinstveni. Ne postoji mogućnost da dva korisnika imaju isto korisničko ime i adresu elektroničke pošte. Uspješnom registracijom korisnik se dodaje u bazu podataka te se preusmjerava natrag na sučelje za prijavu (Slika 1.).

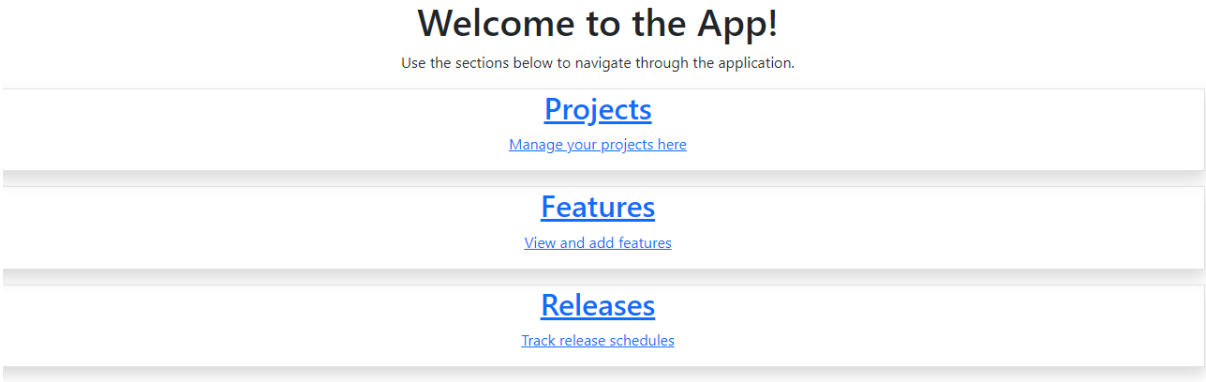
Korisnik može imati jednu od četiri uloge. Prva od njih je uloga administratora koja omogućava korisniku pristup cijelom sustavu, što uključuje pristup projektima, izdanjima i značajkama, kao i korisnicima sustava i klijentima. Slijedi uloga voditelja projekta (engl. *Project manager*), koji može dodjeljivati korisnike projektima, dodavati i uređivati projekte, te dodjeljivati značajke projekata klijentima. Uloga voditelja izdanja (engl. *Release manager*) uključuje dodavanje i uređivanje izdanja, dok razvojni programer (engl. *Developer*) ima pristup značajkama i njihovim statusima.



Slika 2: Sučelje za registraciju

3.2. Upravljanje korisnicima

Uspješnom prijavom korisniku se otvara početna stranica (engl. *Homepage*) (Slika 3.). Na početnoj stranici korisniku je ponuđeno nekoliko opcija u vidu kartica za navigaciju sustavima aplikacije.



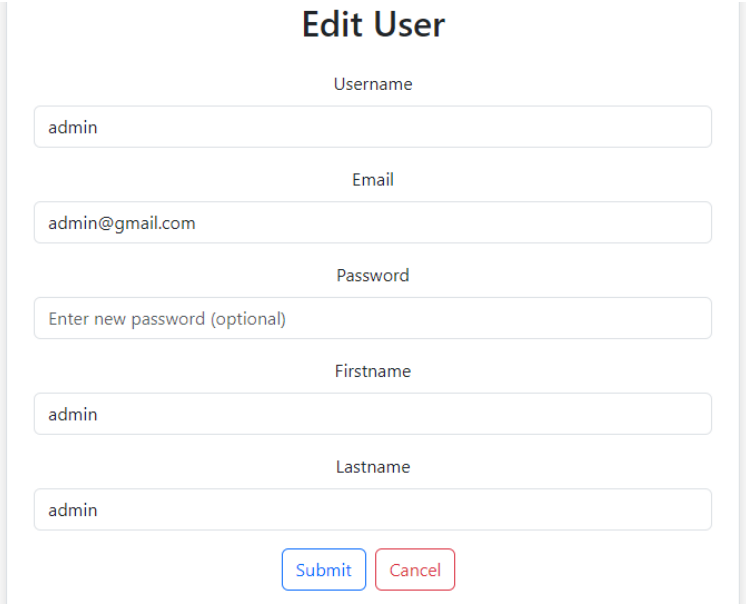
Slika 3: Početna stranica

Cijelim gornjim dijelom ekrana prostire se navigacijska traka unutar koje se nalaze kratice za navigaciju aplikacijom kao i tipke za uređivanje korisničkog računa i odjavu (Slika 4.). Navigacijska traka ima različite tipke, ovisno o ulozi korisnika koji je trenutno prijavljen, pa na primjer korisnik uloge administrator može pristupiti cijelom sustavu dok netko tko ima

ulogu razvojnog programera može pristupiti samo onim dijelovima sustava koji su nužni za njegov rad.

Slika 4: Navigacijska traka

Pritiskom na tipku „*Hi {{username}}*“ korisnik ulazi u sučelje u kojem može uređivati osobne podatke. Unutar sučelja korisnik može mijenjati svoje korisničko ime, adresu elektroničke pošte, lozinku te ime i prezime (Slika 5.). Korisnik ovdje ne može mijenjati svoju ulogu. Razlog toga je dvostruk. Prvi je taj što ulogu može mijenjati samo onaj korisnik čija je uloga administrator, a drugi razlog je što se uloga korisnika mijenja unutar sučelja za projekte.

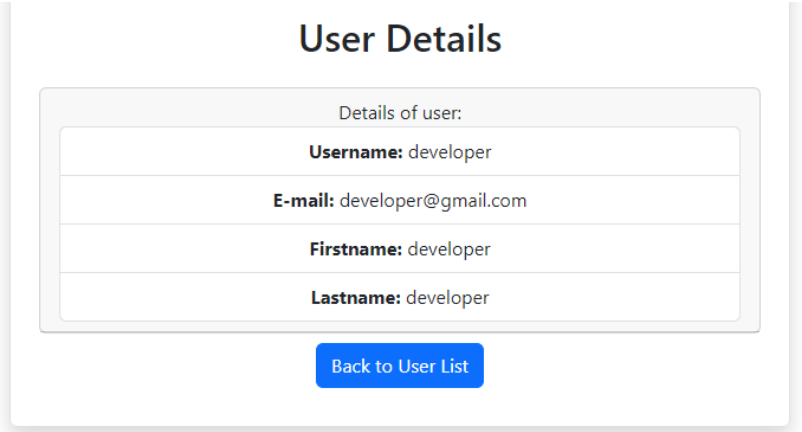


Slika 5: Sučelje za uređivanje korisnika

Ako je u sustav prijavljen korisnik uloge administrator unutar navigacijske trake ponuđena mu je tipka „*Users*“. Pritiskom na naveden tipku korisnik pristupa listi korisnika aplikacije (Slika 6.).

Slika 6: Sučelje liste korisnika

Unutar liste mogu se vidjeti korisničko ime, adresa elektroničke pošte te ime i prezime svakog korisnika. Iznad tablice postoji mogućnost pretraživanja korisnika po njihovom korisničkom imenu, te opcija sortiranja korisnika po određenim parametrima kao i broj korisnika koji će se prikazivati po stranici tablice. Ispod tablice se nalaze dvije tipke za navigaciju paginiranim stranicama. U zadnjem stupcu tablice korisnik ima tri tipke kojima može upravljati korisnicima. Prva od tih tipki je ona pod nazivom „*View*“. Pritiskom na nju ulazi se u detaljniji opis svakog korisnika (Slika 7.).

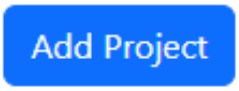


Slika 7: Prikaz detalja o korisniku

U sredini se nalazi tipka pod nazivom „*edit*“. Pritiskom na navedenu tipku ulazi se u sučelje za uređivanje podataka korisnika. To sučelje je isto ono koje korisnik koristi za uređivanje vlastitih podataka (Slika 5.) , ali u ovom slučaju se uređuju podaci korisnika pored čijeg imena se tipka nalazila. Posljednja tipka je ona pod nazivom „*Delete*“. Njenim pritiskom brišu se podaci korisnika iz baze podataka te se korisnik više ne može prijaviti u sustav.

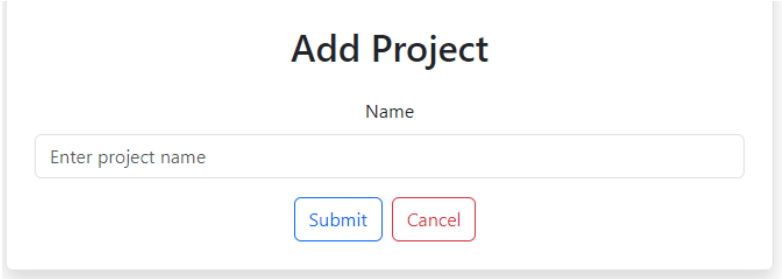
3.3. Upravljanje projektima

Pristupiti listi projekata korisnik može učiniti na dva načina, preko poveznice na početnoj stranici ili preko tipki na navigacijskoj traci. Ovisno o ulozi korisnika na sučelju za projekte bit će omogućene različite mogućnosti. Prijavljeni korisnik uloge administrator ili voditelj projekta može dodati novi projekt u sustav s pomoću tipke na vrhu stranice projekata (Slika 8.).



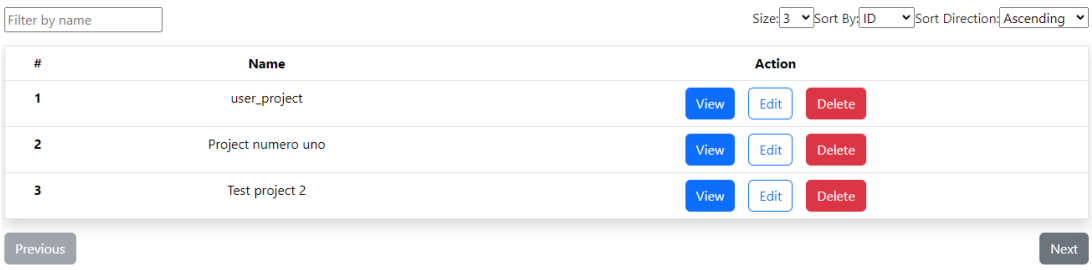
Slika 8: Dodavanje projekta

Pritiskom na navedenu tipku otvara se sučelje za stvaranje novog projekta (Slika 9.). Unutar navedenog sučelja unosi se ime projekta te se on dodaje u sustav.



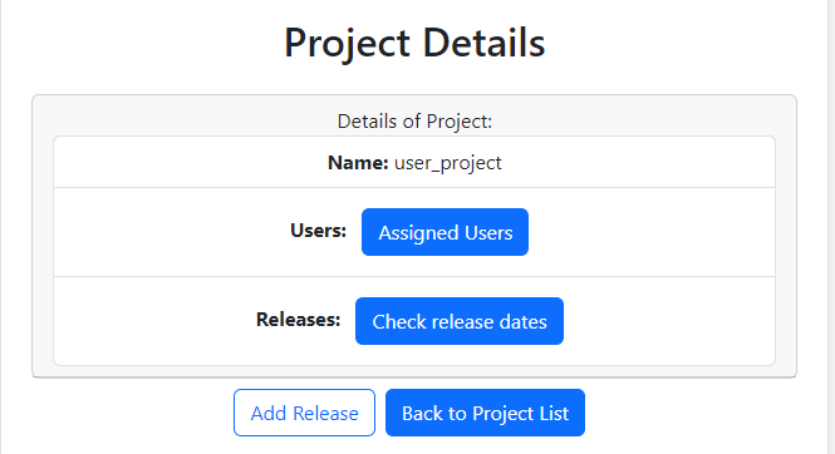
Slika 9: Sučelje za dodavanje projekta

Samo sučelje liste projekata je nalik onom liste korisnika. Korisničko sučelje prikazuje tablicu s podacima projekata iznad koje se nalaze alati za filtriranje i sortiranje projekata, te ispod koje se nalaze tipke za kretanje po stranicama paginirane liste (Slika 10.). Korisnicima uloge administrator prikazuje se lista svih projekata u sustavu, dok ostale uloge imaju prikaz onih projekata kojima su dodijeljeni. U zadnjem stupcu tablice se nalaze mogućnosti koje korisnik ovisno o svojoj ulozi ima.



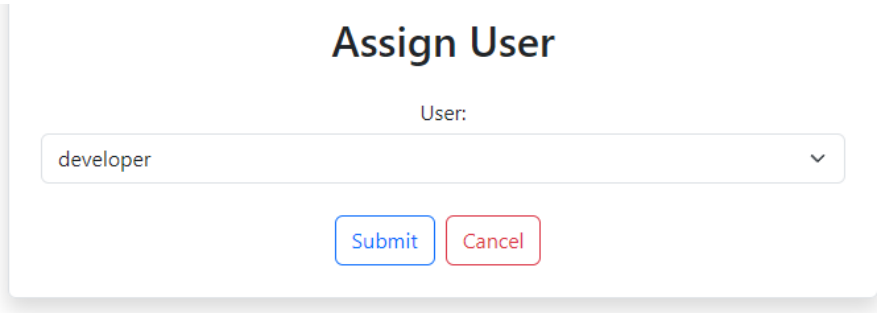
Slika 10: Prikaz liste projekata

Prva od njih i dostupna svim ulogama je mogućnost pregleda detalja projekta preko tipke „*View*“. Pritiskom na navedenu tipku otvara se sučelje koje će ovisno o ulozi korisnika nuditi različite mogućnosti (Slika 11.). U tom sučelju osim što je prikazano ime projekta, prikazano je i nekoliko dodatnih mogućnosti koje ovise o ulozi korisnika. Tako korisnik koji ima ulogu voditelja izdanja ima mogućnost za navedeni projekt dodati novo izdanje. Korisnik uloge voditelja projekta može pristupiti svim korisnicima povezanima s navedenim projektom dok sve uloge mogu pristupiti izdanjima vezanima za taj projekt.



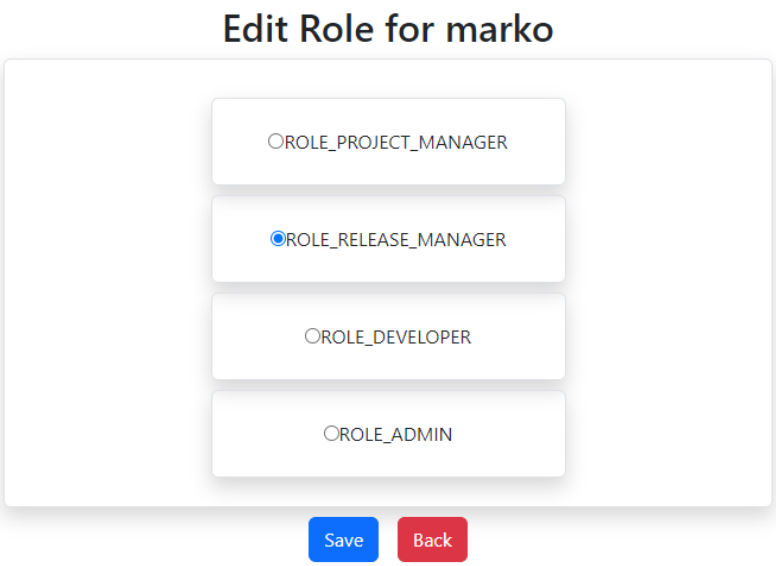
Slika 11: Detaljan prikaz projekta

Ulaskom u sučelje „*Assigned Users*“ korisnik ima uvid u sve one korisnike koji su dodijeljeni projektu. Na vrhu se nalazi tipka za dodjeljivanje novog korisnika projektu. Sučelje omogućava dodjeljivanje korisnika koji još nisu dodijeljeni projektu.(Slika 12.).



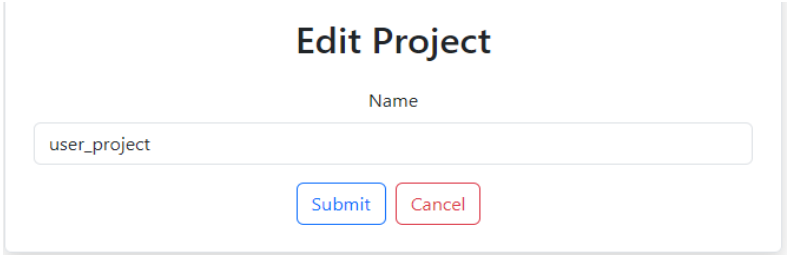
Slika 12: Dodjela korisnika projektu

Tablica koja sadrži popis korisnika vezanih za projekt iznad sebe ima mogućnosti filtriranja i sortiranja, dok se ispod nalaze tipke za navigaciju popisa korisnika. U zadnjem stupcu nalaze se tri mogućnosti s korisnicima vezanima za projekt. Prva od njih je pregled detalja korisnika, druga je mogućnost promjene uloge korisnika (Slika 13.), dok treća omogućava da se korisnika ukloni iz popisa korisnika dodijeljenih projektu.



Slika 13: Promjena uloge korisnika

Ostale dvije mogućnosti unutar sučelja popisa projekata su uređivanje i brisanje projekata. Pritiskom na tipku „*Edit*“ ulazi se u sučelje unutar kojeg se uređuju podaci o projektu (Slika 14.).

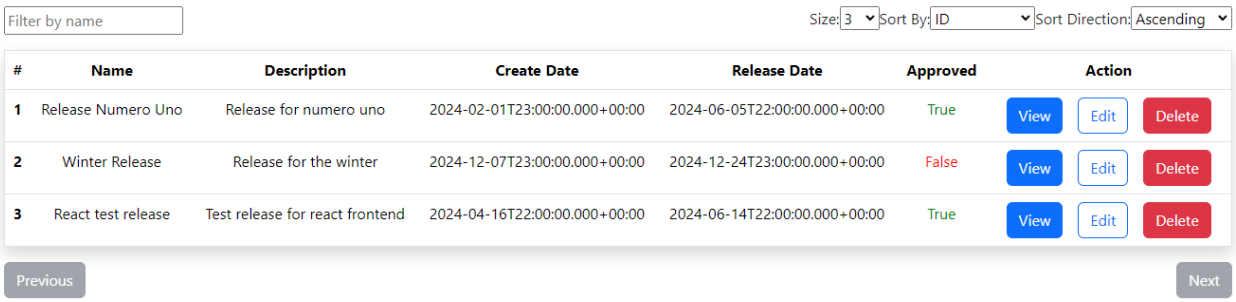


Slika 14: Uređivanje projekta

Posljednja od tri mogućnosti kojoj imaju pristup korisnici uloga administrator i voditelj projekta je mogućnost „*Delete*“. Pritiskom na nju briše se projekt iz baze podataka.

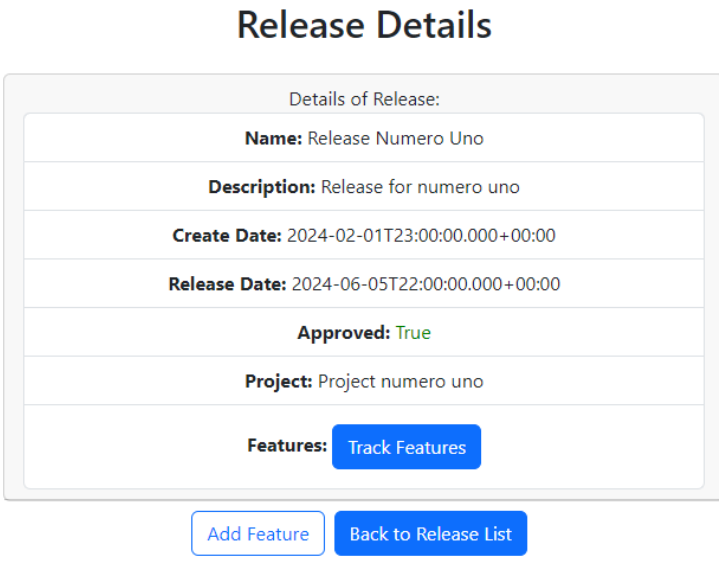
3.4. Upravljanje izdanjima

Izdanjima se pristupa kao i kod projekata preko početne stranice ili preko navigacijske trake (Slika 15.). Ulaskom u sučelje prikazuje se lista izdanja koja se mogu sortirati, filtrirati te pregledavati stranicu po stranicu. Ovisno o ulozi korisnika prikazana će biti ili sva izdanja (Administrator) ili samo ona izdanja koju su povezana s projektima za koje je korisnik vezan. Pretposljednji stupac prikazuje je li projekt odobren od strane voditelja izdanja, dok se u posljednjem stupcu nalaze tri mogućnosti koje su također dostupne ovisno o ulozi korisnika.



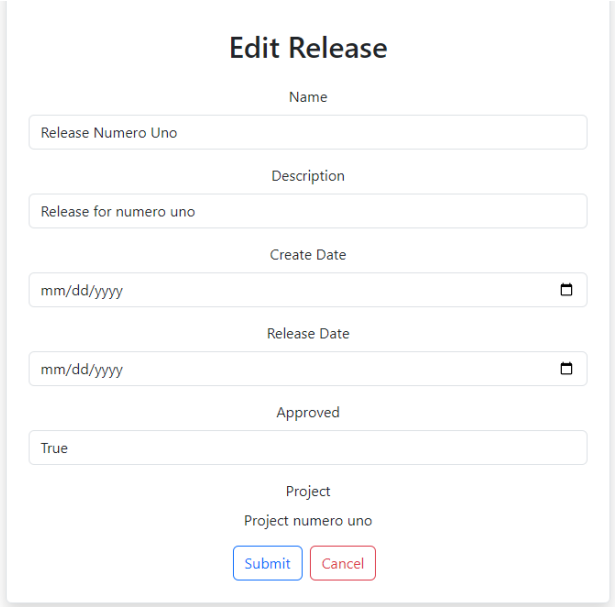
Slika 15: Popis izdanja

Pritiskom na tipku „*View*“ ulazi se na pregled detalja projekta (Slika 16.). Unutar tog sučelja može se vidjeti kojem je projektu izdanje dodijeljeno kao i mogućnost praćenja svih značajki koje su vezane za to izdanje. Ako je prijavljeni korisnik u ulozi razvojnog programera također mu je dostupna mogućnost dodavanja nove značajke za to izdanje.



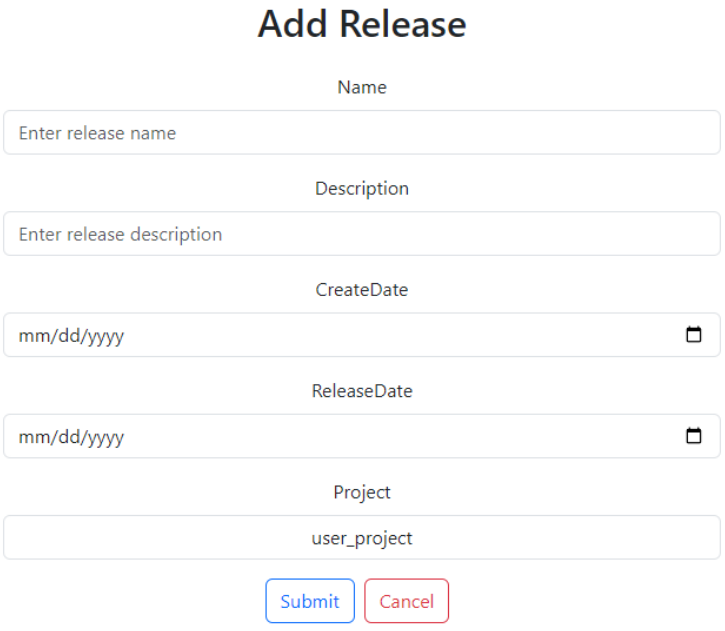
Slika 16: Detalji izdanja

Pritiskom na tipku „*Edit*" ulazi se na sučelje za uređivanje detalja izdanja (Slika 17.). Unutar tog sučelja mogu se mijenjati razne informacije kao što su ime izdanja, opis izdanja kao i datumi stvaranja i izlaska te je li izdanje spremno za izlazak. Posljednja od opcija se nalazi pod nazivom „*Delete*“ i omogućuje brisanje izdanja iz baze podataka.



Slika 17: Uređivanje izdanja

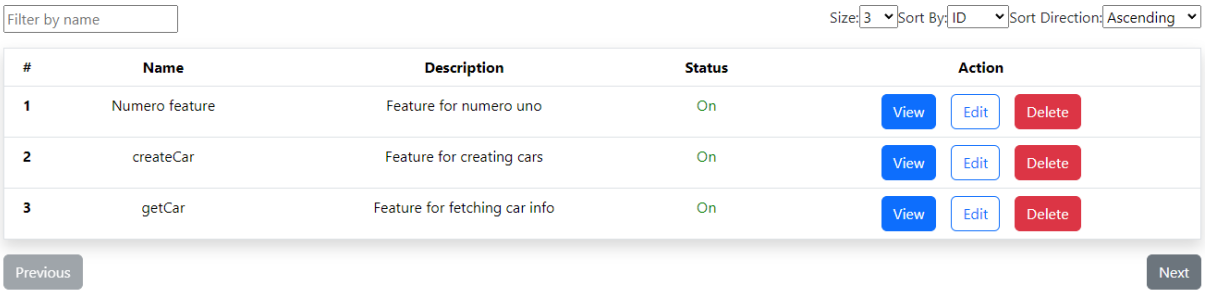
Sučelje za dodavanje novih izdanja nalazi se unutar sučelja detaljnog pregleda projekta. Pritiskom na tipku „*Add Release*“ kojemu pristup ima korisnik uloge voditelj izdanja otvara se sučelje za dodavanje novog izdanja gdje se izdanju zadaju ime, opis, datum stvaranja i datum izlaska prikazano na Slici 18.



Slika 18: Dodavanje novog izdanja

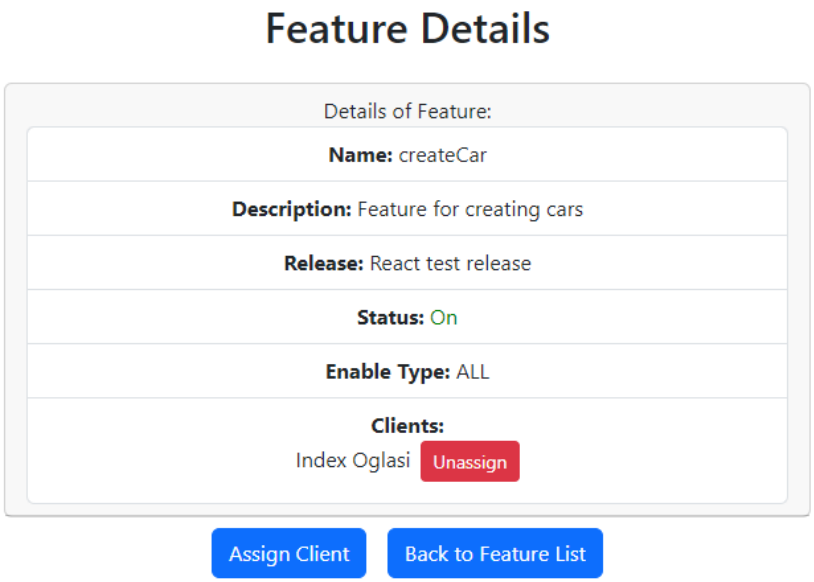
3.5. Upravljanje značajkama

Značajkama se upravlja preko sučelja za značajke (Slika 19.). Samo sučelje je kao i kod prethodnih primjera tablica koja ima opcije filtriranja i sortiranja kao i tipke za listanje stranica. U trećem stupcu se nalazi informacija o tome je li značajka uključena. Više detalja o samoj značajki može se vidjeti pritiskom na tipku „*View*“.



Slika 19: Lista značajki

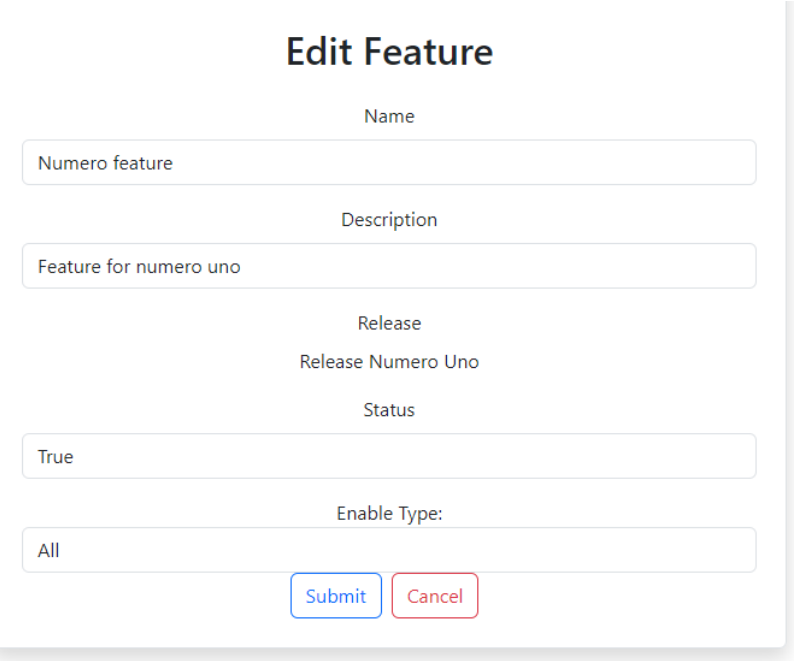
Unutar sučelja za detaljan pregled značajki nalaze se uobičajene informacije o značajki kao i neke nove kao što su „*Enable Type*“ koji govori kojim će klijentima značajka biti dostupna (Slika 20.). Postoje dvije varijante, prva kada je „*Enable Type*“ omogućen za sve klijente, a druga kada je „*Enable Type*“ postavljen u opciju „*Per Account*“. Razlika između te dvije opcije je ta što će sustav ovisno o odabranoj opciji gledati koji su klijenti vezani za značajku. Kada je „*Enable Type*“ postavljen u opciju „*ALL*“ tada će sustav neovisno o klijentima vezanima za značajku istu omogućiti za sve, dok će u suprotnom značajka biti dostupna samo za one klijente koji su vezani za značajku.



Slika 20: Detalji značajke

Ispod informacije „*Enable Type*“ nalazi se popis klijenata koji su vezani za značajku. Pored svakog klijenta se također nalazi mogućnost uklanjanja klijenta s popisa vezanih klijenata. Ta mogućnost je dostupna samo onim korisnicima koji imaju ulogu administrator ili voditelj projekta.

Osim mogućnosti pregleda detalja određene značajke sučelje omogućava još dvije mogućnosti. Jedna od njih se nalazi iza tipke „*Edit*“ te otvara sučelje za uređenje značajke (Slika 21.). Unutar tog sučelja može se promijeniti ime i opis značajke kao i status i „*Enable Type*“. Status je vezan za datum izlaska izdanja. U trenutku kad izdanje dođe do datuma izlaska, ako je to isto izdanje potvrđeno kao spremno, sve značajke koje su povezane s njim prebacit će se iz statusa „*OFF*“ u status „*ON*“. Naravno ako je značajka spremna prije nego dođe datum izlaska izdanja ovim putem se ona može ručno uključiti.



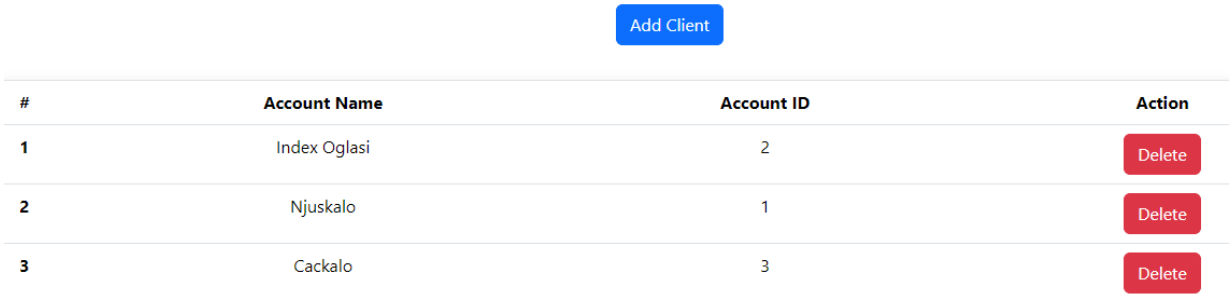
Slika 21: Uređenje značajke

Posljednja tipka u sučelju popisa svih značajki je tipka „*Delete*“. Njenim pritiskom se značajka briše iz baze podataka.

3.6. Upravljanje klijentima

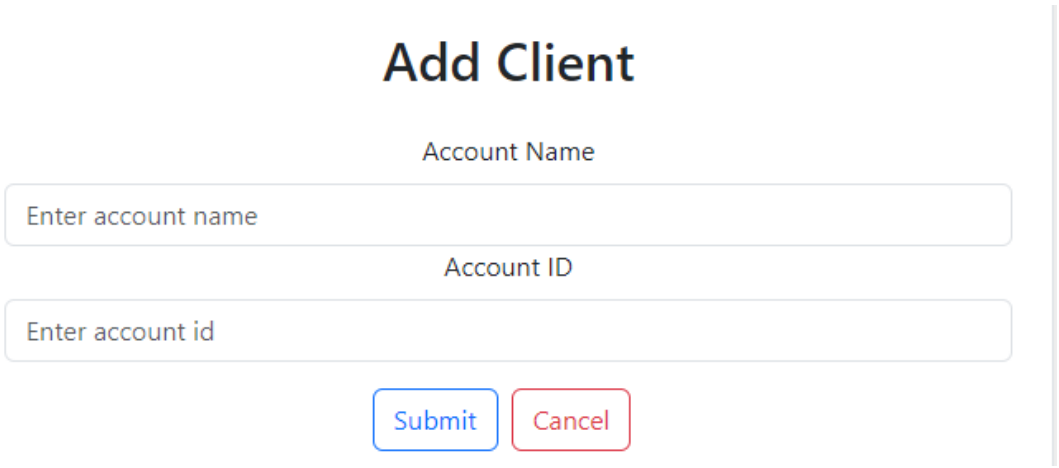
Klijent u kontekstu ove aplikacije predstavlja tvrtku koja koristi *cloud-based* web aplikaciju. To znači da više različitih klijenata može nuditi pristup istoj aplikaciji. Ovisno o klijentu ta aplikacija može imati različite značajke koje će biti omogućene. Na primjer klijenti pod nazivom „Njuškalo“ i „Index Oglasi“ će koristiti istu *cloud-based* web aplikaciju, ali značajke koje oni nude krajnjem korisniku ne moraju biti iste. Tako će na primjer korisnik koji koristi usluge „Njuškala“ imati pristup oglasima za brodove, dok korisnik koji koristi „Index Oglase“ pristup toj značajki neće imati.

Klijentima mogu upravljati samo korisnici uloge administrator. Ulaskom u sučelje za upravljanje klijentima prikazuje se tablica svih klijenata iznad koje se nalazi tipka za dodavanje novih klijenata. U trećem stupcu tablice nalazi se stupac pod nazivom „*Account ID*“. To nije primarni ključ koji se nalazi u tablici „*Clients*“ baze podataka već poseban stupac čija se vrijednost zadaje pri kreiranju novog klijenta. Ta vrijednost mora biti jedinstvena, a njena namjena je kada vanjska aplikacija provjerava je li značajka za nju dostupna da se usporedi primljeni ID s „*Account ID*“ iz baze podataka. U posljednjem stupcu nalazi se mogućnost brisanja klijenta iz baze podataka (Slika 22.).



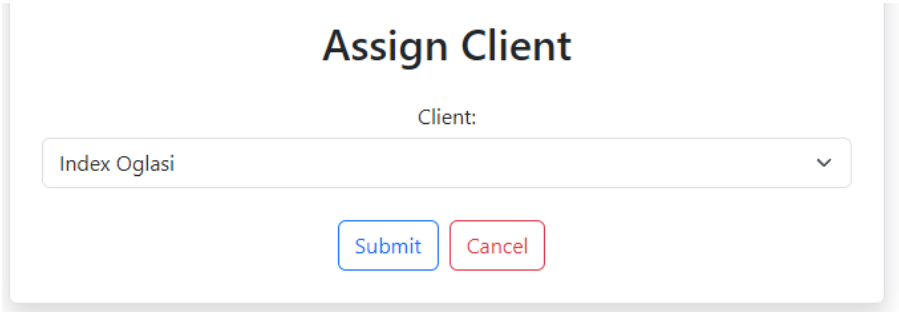
Slika 22: Popis klijenata

Unutar sučelja za dodavanje novog klijenta kojem se pristupa pritiskom na tipku „*Add Client*“ klijentu se zadaje naziv te identifikacijski broj (Slika 23.).



Slika 23: Dodavanje novog klijenta

Posljednji dio upravljanja klijentima nalazi se unutar sučelja za detaljni pregled značajke. Pritiskom na tipku „*Assign Client*“ ulazi se u sučelje za dodavanje klijenta određenoj značajki (Slika 24.). Pristup tom sučelju imaju samo korisnici uloga administrator i voditelj projekta.



Slika 24: Dodjela klijenta značajki

3.7. Feature flag

Feature flag je tehnika u razvoju softvera koja omogućuje aktiviranje ili deaktiviranje određenih funkcionalnosti aplikacije. Ponaša se kao prekidač koji može uključiti ili isključiti određenu značajku aplikacije u stvarnom vremenu.

Feature flagovi se koriste kako bi se moglo kontrolirati izdanje novih značajki, što omogućava testiranje na manjem skupu klijenata prije nego što značajka bude dostupna svima. U slučaju da neka nova značajka uzrokuje probleme, može se brzo deaktivirati. Još jedna prednost je ta što različiti timovi mogu raditi na različitim značajkama neovisno, a zatim ih aktivirati kada su spremne.

U sklopu ovog projekta *feature flag* je zamišljen kao alat kojim se može provoditi kontrola pristupa značajkama, ovisno o postavljenim parametrima za tu značajku. Svaka od značajki unutar baze podataka ima dva atributa o kojima ovisi hoće li određeni klijent istoj imati pristup. Prvi od njih je `Enable Type` čija vrijednost može biti „*ALL*“ ili „*PER ACCOUNT*“. Mijenjanjem između njih određena značajka će biti dostupna svima, ili samo onim klijentima koji su dodijeljeni toj značajki. Drugi atribut je `status`. On također ima dvije vrijednosti. Kada je postavljen u vrijednost „*OFF*“ nitko nema pristup značajki, a kada je postavljen u vrijednost „*ON*“ pristup značajki će ovisiti o vrijednosti atributa `Enable Type`.

S obzirom na to da je svaka značajka vezana uz izdanje projekta, atribut `status` ovisi i o datumu izlaska izdanja. U trenutku kada datum izlaska izdanja prođe, i to isto izdanje je potvrđeno kao spremno za izlazak, sve značajke povezane s tim izdanjem mijenjaju svoj `status` iz vrijednosti „*OFF*“ u vrijednost „*ON*“. Naravno korisnici s odgovarajućom ulogom mogu istu tu značajku neovisno o datumu izlaska izdanja uključivati i isključivati. To znači da tijekom razvoja aplikacije neke značajke krajnjem korisniku mogu biti omogućene prije drugih.

Kako bi *feature flag* funkcionirao u praksi bilo je potrebno napraviti biblioteku (engl. *library*). Ta biblioteka sadrži metodu koja će primati zahtjeve klijenata te provjeriti je li značajka koju su ti klijenti zatražili dostupna na temelju njihovog identifikacijskog broja te naziva zatražene značajke.

Kako bi klijentske aplikacije mogle pristupiti statusu zatraženih značajki, unutar `pom.xml` datoteke moraju imati uvezenu (engl. *imported*) biblioteku Feature flag (Ispis 1.).

```
<dependency>
      <groupId>com.featureflag</groupId>
      <artifactId>Feature-Flag-Library</artifactId>
      <version>0.0.1-SNAPSHOT</version>
</dependency>
```

Ispis 1: Uvoz biblioteke Feature flag

Biblioteci se šalju dvije informacije, naziv značajke te identifikacijski broj klijenta (Ispis 2.). Biblioteka zatim, koristeći HTTP zahtjeve komunicira s *backend* servisom koji sadrži konfiguraciju značajki. Na temelju konfiguracije, biblioteka vraća `boolean` vrijednost koja označava je li značajka omogućena za klijenta koji je poslao upit.

```
@GetMapping("/")
public List<Car> getAllCars(@RequestParam String accountId) {
      if (featureService.isFeatureEnabled("getAllCars",
accountId).isPresent()) {
            boolean isEnabled =
featureService.isFeatureEnabled("getAllCars", accountId).get();
      if (isEnabled) {
                return carService.getAllCars();
      } else {
        throw new RuntimeException("feature not available!");
      } else {
             throw new RuntimeException("failed to determine
      feature state");}
    }
```

Ispis 2: Primjer klijentske aplikacije

4. Baza podataka

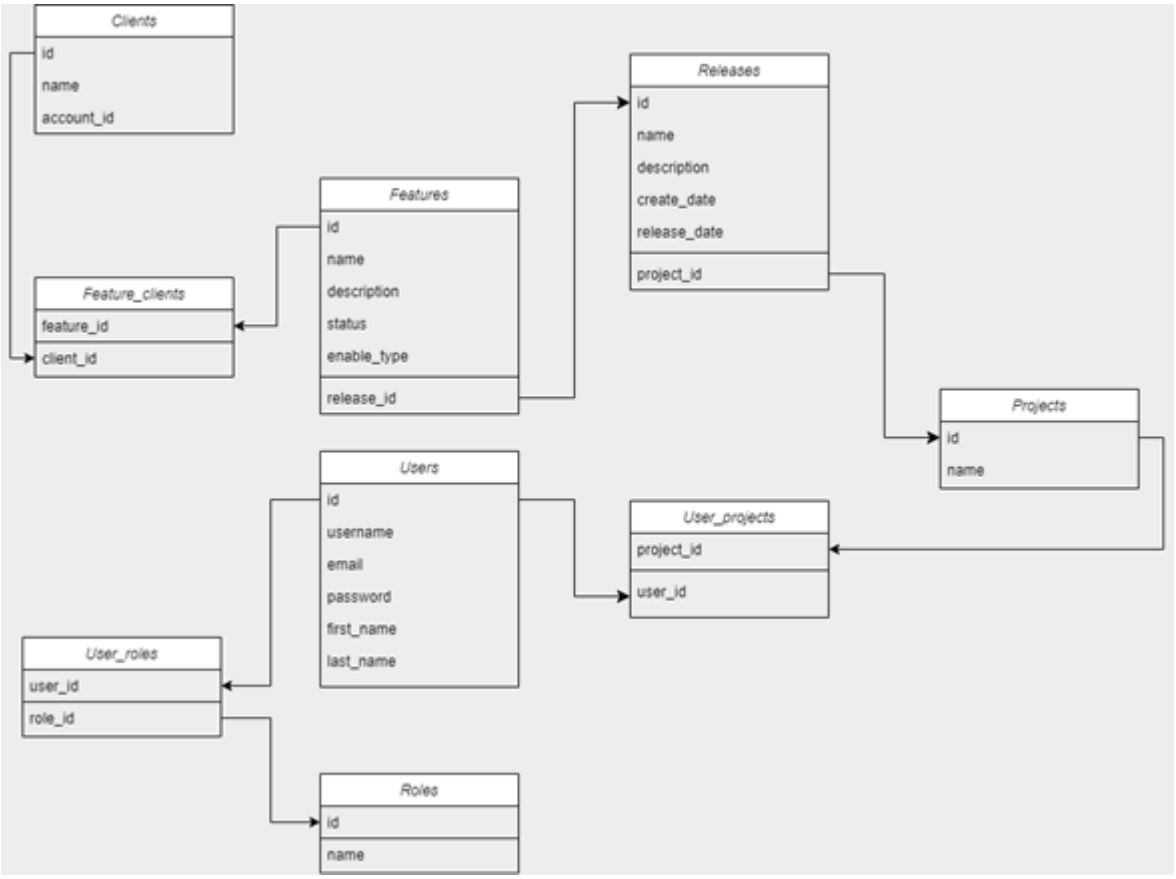
4.1. Opis baze podataka

Baza podataka predstavlja temelj aplikacije za upravljanje izdanjima i značajkama (Slika 25.). Ona organizira, pohranjuje i upravlja podacima. Baza služi kao temelj za praćenje i upravljanje projektima, klijentima, značajkama, izdanjima i korisnicima. Pažljivo je dizajnirana kako bi se osigurala efikasna pohrana, dohvat i ažuriranje podataka.

Glavni elementi baze podataka su entiteti. U slučaju ove baze podataka glavni entiteti su: klijenti, izdanja, značajke, korisnici, projekti i uloge. Svaki od entiteta ima svoje atribute koji opisuju njegova svojstva.

Entiteti su međusobno povezani različitim vrstama relacija. Najčešće vrste relacija u ovoj bazi podataka su jedan na više (engl. *One-to-many*) i više na više (engl. *Many-to-many*) relacije. Relacija jedan na više znači da jedan entitet može biti povezan s više drugih entiteta, ali obrnuto ne mora vrijediti. Na primjer, jedan projekt može imati više izdanja, ali jedno izdanje pripada samo jednom projektu. S druge strane relacija više na više znači da jedan entitet može biti povezan s više drugih entiteta, i obrnuto. Na primjer, jedna značajka može biti povezana s više klijenata, i obrnuto.

Pomoćne tablice sadrže kombinaciju primarnih ključeva iz dvaju ili više povezanih tablica. Na primjer, tablica `Feature_clients` sadrži kombinaciju primarnih ključeva iz tablica `Features` i `Clients`, što omogućava praćenje kojoj značajki pripada koji klijent.



Slika 25: Model baze podataka

Kako bi se Spring Boot aplikacija mogla povezati s PostgreSQL bazom podataka unutar `application.properties` datoteke moraju se unijeti određeni podaci (Ispis 3.). Unutar navedene datoteke treba se navesti protokol preko kojeg se povezuje na bazu. U našem slučaju to će biti `jdbc:postgresql://`. Nakon toga slijedi oznaka `localhost` koja označava da je baza pohranjena na lokalnom računalu. Kad bi se baza nalazila na udaljenom poslužitelju ovdje bi bila upisana IP adresa ili domena tog poslužitelja. Broj „5432“ označava standardni port PostgreSQL baze podataka, dok je `releaseManagement` naziv baze na koju se aplikacija povezuje.

Kod `spring-datasource-username` i `spring-datasource-password` upisani su korisničko ime i lozinka koju PostgreSQL koristi na lokalnom računalu. Posljednje dvije linije su `org.postgresql.Driver` koji omogućava Java aplikaciji povezivanje s PostgreSQL bazom, te `spring.jpa.hibernate.ddl-`

auto=none koji označava kako Hibernate neće automatski mijenjati bazu podataka već će se za to koristiti Flyway migracije.

```
spring.datasource.url=jdbc:postgresql://localhost:5432/releaseMa
nagement

spring.datasource.username=postgres

spring.datasource.password=1234

spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.hibernate.ddl-auto=none
```

Ispis 3: application.properties

4.2. Entiteti baze podataka

Baza podataka aplikacije za upravljanje izdanjima i značajkama sastoji se od više entiteta. Svaki entitet predstavlja ključnu komponentu sustava, definirajući kako se podaci pohranjuju, povezuju i koriste unutar aplikacije. Razumijevanje tih entiteta i njihovih atributa ključno je za shvaćanje načina na koji aplikacija upravlja kompleksnim odnosima između različitih elemenata projekta.

Entitet Clients predstavlja pojedinačne klijente koji su uključeni u projekt. Klijenti su ključni jer oni često imaju specifične zahtjeve i potrebe koje moraju biti zadovoljene kroz razvoj i isporuku softverskih funkcionalnosti. Atributi ovog entiteta uključuju id, name i account_id. Id je jedinstveni identifikator svakog klijenta. Ovaj atribut omogućava jednoznačno razlikovanje klijenata unutar sustava. Name je ime klijenta koje pomaže pri identifikaciji i upravljanju klijentima unutar aplikacije. Account_id je identifikator računa povezanog s klijentom. Služi kako bi vanjska aplikacija znala ima li klijent pristup određenoj značajki.

Entitet Feature_clients povezuje tablice Features i Clients, te opisuje koje su značajke dostupne kojim klijentima. Atributi uključuju dva strana ključa, feature_id i client_id koji omogućuje vezu više na više.

Entitet Features predstavlja pojedinačne značajke izdanja. Atribut Id je jedinstveni identifikator značajke. Name je naziv značajke, što olakšava prepoznavanje i upravljanje funkcionalnostima unutar projekta. Description je opis značajke koji pruža

detalje o njenoj svrsi. `Status` značajke omogućava kontrolu nad time koje su značajke dostupne korisnicima. `Enable_type` je vrsta omogućavanja značajke, gdje ona može biti omogućena po računu ili za sve klijente. `Release_id` je identifikator izdanja s kojim je značajka povezana.

Entitet `Releases` opisuje različita izdanja projekta. U softverskom razvoju, upravljanje izdanjima je kritično za osiguranje kvalitete i pravovremenosti isporuke proizvoda. Atribut `Id` predstavlja jedinstveni identifikator svakog izdanja. `Name` je naziv izdanja koji omogućava lako prepoznavanje unutar sučelja. `Description` je opis izdanja koji pruža detaljne informacije o istom. `Create_date` i `Release_date` daju informaciju o datumu kreiranja i datumu izdanja. Oni pomažu u praćenju napretka i povijesti razvoja. `Project_id` je identifikator projekta s kojim je izdanje povezano, što omogućava praćenje verzija unutar određenih projekata.

Entitet `Projects` predstavlja pojedinačne projekte koje sustav prati. Projekti su središnji dio aplikacije, jer svaki projekt sadrži više izdanja, značajki i korisnika koji na njemu rade. Atributi uključuju `id` kao primarni ključ te `name`, koji predstavlja naziv projekta te je ključan za identifikaciju istog u sučelju aplikacije.

Entitet `User_projects` povezuje tablice `Users` i `Projects`, te opisuje koji korisnici imaju pristup kojim projektima. Sastoji se od dva strana ključa, `project_id` i `user_id` koji omogućavaju upravljanje pristupa projektima samo onim korisnicima koji su tim projektima dodijeljeni.

Entitet `Users` opisuje korisnike koji imaju pristup aplikaciji. Uloga ovog entiteta je sigurnost sustava kao i upravljanje korisničkim računima i pravima pristupa. Atribut `Id` predstavlja primarni ključ entiteta, `username` predstavlja korisničko ime, `email` je adresa elektroničke pošte korisnika, `password` je enkriptirana lozinka, dok `first_name` i `last_name` predstavljaju ime i prezime korisnika.

Entitet `User_roles` povezuje tablice `Users` i `Roles`, te opisuje koje uloge imaju pojedini korisnici. Ovaj entitet je ključan za upravljanje pravima pristupa i odgovornostima unutar sustava. Sastoji se od dva strana ključa, `user_id i role_id`.

Entitet `Roles` opisuje različite uloge koje korisnici mogu imati unutar sustava. Uloge definiraju prava pristupa i odgovornosti korisnika, što je ključno za sigurnost i upravljanje sustavom. Atributi uključuju `id` kao primarni ključ entiteta, te `name` kao naziv uloge što pomaže pri upravljanju ulogama unutar sustava.

4.3. Flyway migracije

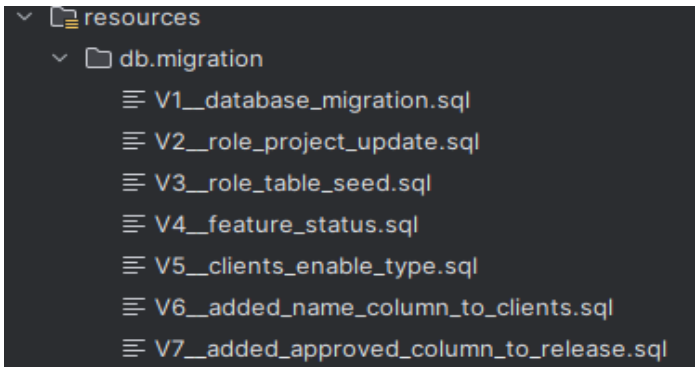
Flyway je popularan alat za upravljanje verzijama baze podataka. Omogućuje jednostavno praćenje, primjenu, i upravljanje promjenama baze podataka kroz migracijske skripte (Ispis 5.). Kada koristite Flyway unutar Spring Boot aplikacije, proces migracije baze podataka je automatiziran i integriran unutar aplikacije.

Kako bi koristili Flyway unutar projekta potrebno je dodati ovisnost unutar `pom.xml` datoteke ako se koristi `maven` (Ispis 4.).

```
<dependency>

      <groupId>org.flywaydb</groupId>

      <artifactId>flyway-core</artifactId>

</dependency>
```

Ispis 4: Ovisnost unutar `pom.xml`

Kako bi se kreirala migracijska skripta bilo je potrebno unutar `src/main/resources/db/migration` direktorija dodati datoteku koja mora imati točno određen naziv (Slika 26.). Naziv mora počinjati s verzijom migracije. Ako je to prva migracijska skripta za projekt datoteka će početi s nazivom „V1“. Nakon toga slijedi dvostruka donja crta iza koje slijedi opis migracije te vrsta datoteke koja će u slučaju migracija uvijek završavati s nastavkom „.sql“.



Slika 26: Direktorij s migracijama

Spring Boot automatski pokreće Flyway migracije prilikom pokretanja aplikacije. Kada se aplikacija pokrene, Flyway će provjeriti bazu podataka i primijeniti sve migracije koje još nisu primijenjene.

```
DROP TABLE IF EXISTS User_project_roles;

DROP TABLE IF EXISTS User_roles;

CREATE TABLE IF NOT EXISTS User_projects (

    user_id UUID REFERENCES Users(id) ON DELETE CASCADE,

    project_id UUID REFERENCES Projects(id) ON DELETE CASCADE,

    PRIMARY KEY (user_id, project_id)

);

CREATE TABLE Roles(

    id UUID PRIMARY KEY,

    name VARCHAR(100) NOT NULL

);

CREATE TABLE IF NOT EXISTS User_roles (

    user_id UUID REFERENCES Users(id) ON DELETE CASCADE,

    role_id UUID REFERENCES Roles(id) ON DELETE CASCADE,

    PRIMARY KEY (user_id, role_id)

);
```

Ispis 5: Primjer migracijske skripte

Nakon što se aplikacija pokrene, rezultati se mogu provjeriti unutar baze podataka. Flyway stvara tablicu pod nazivom `flyway_schema_history`, gdje pohranjuje sve informacije o provedenim migracijama.

5. Implementacija

5.1. Prijava i registracija

Unutar direktorija `controllers` u aplikaciji nalazi se datoteka pod nazivom `AuthController`. Unutar navedene datoteke se nalaze *endpointi* za registraciju i prijavu korisnika. Autentikacija je implementirana koristeći JWT (JSON Web Token).

Prvi ključni aspekt ovog kôda je proces registracije korisnika, realiziran putem metode označene s `@PostMapping(„signup")`. Ova metoda omogućuje novim korisnicima da se registriraju u sustav. Nakon što korisnik pošalje zahtjev za registraciju s podacima poput korisničkog imena, adrese elektroničke pošte i lozinke, aplikacija obrađuje ove podatke s pomoću metode `userService.registerUser`. Ova metoda obavlja sve potrebne provjere i pohranjuje podatke o korisniku u bazu podataka.

```
public static String generateToken(Authentication auth) {
    Collection<? extends GrantedAuthority> authorities =
auth.getAuthorities();
    String roles = populateAuthorities(authorities);
    @SuppressWarnings("deprecation")
    String jwt = Jwts.builder()
            .setIssuedAt(new Date())
            .setExpiration(new Date(new
Date().getTime()+86400000))
            .claim("username", auth.getName())
            .claim( "authorities",roles)
            .signWith(key)
            .compact();
    System.out.println("Token for parsing in JwtProvider: " +
jwt);
    return jwt;
}
```

Ispis 6: Metoda koja generira JWT

Jednom kada je korisnik uspješno registriran, aplikacija kreira objekt `Authentication` koristeći unesene podatke. Ovaj objekt predstavlja identitet korisnika i koristi se za autentikaciju unutar aplikacije. Kroz `SecuirtyContextHolder`, koji je

središnji mehanizam Spring Securityja, ova autentikacija se postavlja kao trenutni sigurnosni kontekst za korisnika. Ova faza osigurava da je korisnik odmah nakon registracije autenticiran i spreman za interakciju s ostalim dijelovima aplikacije.

Nakon autentikacije, generira se JWT s pomoću metode `generateToken(authentication)` što se može vidjeti u Ispisu 6. JWT je ključan element jer omogućuje da aplikacija prepozna autenticiranog korisnika bez potrebe za ponovnim unosom podataka. JWT se vraća korisniku s porukom o uspješnoj registraciji, čime se završava proces registracije.

Drugi dio ovog sigurnosnog sustava odnosi se na prijavu korisnika, implementiran kroz metodu `@PostMapping(„signin")`. Prijava je proces putem kojeg postojeći korisnici unose svoje podatke kako bi pristupili aplikaciji. Ova metoda prima `userDto` objekt s korisničkim imenom i lozinkom te poziva metodu `authenticate` kako bi se provjerila vjerodostojnost podataka.

Unutar metode `authenticate`, aplikacija koristi `loadUserByUsername` za učitavanje podataka o korisniku iz baze podataka. Ako uneseno korisničko ime ne postoji ili ako lozinka ne odgovara, baca se iznimka `BadCredentialsException`. Ona označava neuspjelu prijavu u sustav. U suprotnom se stvara JWT s pomoću metode `UsernamePasswordAuthenticationToken`, koji predstavlja uspješno prijavljenog korisnika i sadrži sve potrebne informacije o njegovim ovlastima.

Kao i kod registracije, nakon uspješne prijave generira se JWT, koji se šalje korisniku kao potvrda uspješne prijave. JWT omogućava korisniku da pristupi zaštićenim dijelovima aplikacije bez potrebe za ponovnim unosom podataka, sve dok je isti važeći.

5.2. CRUD operacije

CRUD (Create, Read, Update, Delete) akronim predstavlja skup temeljnih operacija koje se koriste za upravljanje podacima u većini aplikacija, posebice onih koje se temelje na bazama podataka. Ove operacije omogućuju stvaranje novih zapisa, čitanje postojećih, ažuriranje postojećih i brisanje nepotrebnih. U kontekstu web razvoja, CRUD operacije se često implementiraju putem RESTful API-ja, koji omogućavaju klijentskim aplikacijama da

komuniciraju sa poslužiteljom koristeći standardne HTTP metode (GET, POST, PUT, DELETE).

```
@PostMapping("/create")
@PreAuthorize("hasAnyRole('ROLE_DEVELOPER',
'ROLE_ADMIN')")
public ResponseEntity<String>
createFeature(@RequestBody FeatureDto featureDto){
    featureService.createFeature(featureDto);
    return new ResponseEntity<>("Success",
HttpStatus.OK);
}
```

Ispis 7: Izrada značajke u kontroleru

Kada korisnik želi stvoriti novu značajku, šalje HTTP POST zahtjev na odgovarajuću putanju. U tijelu zahtjeva se nalazi prikaz nove značajke. Poslužitelj tada provjeri autentičnost korisnika i, ako je ovlašten, pozove servisni sloj koji je zadužen za pohranu podataka u bazu (Ispis 7.). Nakon uspješnog stvaranja, poslužitelj vraća HTTP status 201 koji označava uspješno stvaranje novog zapisa u bazi.

Za dohvat jednog ili više zapisa, korisnik šalje HTTP GET zahtjev. Ako se želi dohvatiti samo jedan zapis iz baze podataka, u URL-u se navodi jedinstveni identifikator (Ispis 8.) tog zapisa. Za dohvat više zapisa, koristi se URL koji predstavlja kolekciju svih zapisa. Poslužitelj će vratiti status 200 (OK) s traženim podacima.

```
@GetMapping("/{id}")
public Release getRelease(@PathVariable UUID id)
throws ReleaseNotFoundException {
    return releaseService.findReleaseById(id);
}
```

Ispis 8: Primjer HTTP GET zahtjeva

Kada je potrebno izmijeniti postojeći zapis, korisnik šalje HTTP PUT zahtjev na URL koji identificira taj zapis. U tijelu zahtjeva se nalaze izmijenjeni podaci. Poslužitelj provjeri autentičnost korisnika i, ako je ovlašten, pozove servisni sloj koji će ažurirati

podatke u bazi (Ispis 9.). Nakon uspješnog ažuriranja, poslužitelj vraća HTTP status 200 (OK) s ažuriranim podacima.

```
public Project updateProject(ProjectDto
updatedProjectDto, UUID projectId){
    Project project =
projectRepository.findById(projectId).get();
    project.setName(updatedProjectDto.getName());
    return projectRepository.save(project);
}
```

Ispis 9: Ažuriranje u servisnom sloju

Za brisanje zapisa, korisnik šalje HTTP DELETE zahtjev na URL koji identificira taj zapis. Poslužitelj provjeri autentičnost korisnika i, ako je ovlašten, pozove servisni sloj koji će obrisati podatke iz baze. Nakon uspješnog brisanja, poslužitelj obično vraća HTTP status 204 (No Content).

5.3. Biblioteka Feature flag

Implementacija *feature flaga* sastoji se od nekoliko različitih dijelova koji međusobno čine cjelinu. Unutar direktorija `controllers` aplikacije za upravljanje izdanjima i značajkama nalazi se datoteka `FeatureControllers`. Jedna od metoda koja se u toj datoteci nalazi je metoda pod nazivom `isFeatureEnabled (`Ispis 10.`)`. Ovu metodu preko biblioteke Feature flag poziva klijent kada želi dobiti status određene značajke.

```
@GetMapping("/enabled/{featureName}/{accountId}")

    public ResponseEntity<Boolean>
isFeatureEnabled(@PathVariable String featureName,
@PathVariable String accountId) throws
FeatureNotFoundException{

     var isEnabled =
featureService.isFeatureEnabled(featureName, accountId);

        return ResponseEntity.ok(isEnabled); }
```

Ispis 10: *Endpoint* za provjeru značajke

U trenutku kada se taj *endpoint* pozove šalje se upit na servisni sloj aplikacije gdje metoda pod istim nazivom kao i ona u kontroleru, provjerava status značajke te za istu vraća `boolean` vrijednost.

S obzirom na to da značajka osim `statusa` ima i `enable type`, servisni sloj će prema imenu značajke prvo provjeriti njen `status`, a zatim prema `enable typeu` i to je li značajka dostupna onom klijentu koji šalje upit (Ispis 11.).

```
private boolean isFeatureEnabledForAccount(Feature
feature,String accountId){

        return feature.getClients().stream().anyMatch(client ->
client.getAccountId().equals(accountId));

    }
```

Ispis 11: Provjera značajke po klijentu

Kako bi se automatizirao proces razvoja značajki, svaka značajka povezana je s izdanjem. Kao i značajka, izdanje ima svoj `status` koji korisnici s odgovarajućim ulogama

```
@Scheduled(fixedRate = 40000)
public void checkReleaseDate(){
    var today = new Date();
    var releases = releaseRepository.findAll();
    List<Feature> featuresToUpdate = new ArrayList<>();
    for(Release release: releases){
        if (release.getReleaseDate().before(today) &&
release.getApproved()){
 System.out.println("Feature to update: " + release.getName());
       var features = featureRepository.findByRelease(release);
            featuresToUpdate.addAll(features); }
    }
    if (!featuresToUpdate.isEmpty()) {
        for (Feature feature : featuresToUpdate) {
            feature.setStatus(true); }
        featureRepository.saveAll(featuresToUpdate); }
}
```

Ispis 12: Provjera datuma izlaska izdanja

mogu mijenjati. `Status` kod izdanja predstavlja je li to izdanje spremno za svoj datum izlaska. U trenutku kada datum izlaska izdanja prođe, ako je izdanje preko `statusa` postavljeno kao spremno, svaka značajka povezana s tim izdanjem bit će omogućena (Ispis 12.).

Biblioteka Feature flag sadrži metodu `isFeatureEnabled()` koja će primati zahtjeve klijenata te provjeriti je li značajka koju su ti klijenti zatražili dostupna na temelju njihovog identifikacijskog broja te naziva zatražene značajke (Ispis 13.). Također sadrži i konstruktor u kojem se nalazi parametar `restTemplate` koji se koristi pri izvođenju HTTP zahtjeva prema vanjskom API-ju.

```
public FeatureService(RestTemplate restTemplate,
@Value("${feature.api.url:http://localhost:8080}") String
featureApiUrl, @Value("/feature") String featurePathSegment) {
    this.restTemplate = restTemplate;
    this.featureApiUrl = featureApiUrl;
    this.featurePathSegment = featurePathSegment;
}

public Optional<Boolean> isFeatureEnabled(String featureName,
String accountId) {
    String url = featureApiUrl + featurePathSegment + "/" +
featureName + "/" + accountId;
    return Optional.ofNullable(restTemplate.getForObject(url,
Boolean.class));
}
```

Ispis 13: Biblioteka Feature flag

5.4. React frontend

U ovom projektu za zadatke korisničkog sučelja korištena je React biblioteka. React na *frontendu* komunicira sa Spring Boot *backendom* putem HTTP zahtjeva, koristeći biblioteku `Axios`. Spring Boot pruža RESTful API-e koji omogućuju aplikaciji da dobiva, kreira, uređuje i briše podatke na poslužitelju. Ovo razdvajanje između *backenda* i *frontenda* omogućuje veću fleksibilnost i skalabilnost, jer se svaki dio aplikacije može razvijati i testirati neovisno. Ovakav pristup također omogućuje *frontend* aplikaciji da ostane lagana

(engl. *lightweight*) i responzivna, dok *backend* obrađuje poslovnu logiku i interakcije s bazom podataka.

Svaka komponenta *frontenda* koristi `React hooks`. Dva najvažnija su `useState` i `useEffect`, koji upravljaju stanjem i životnim ciklusom komponente. Svaki puta kada se pristupi nekoj od React komponenti prvo se provjerava je li korisnik autenticiran pomoću JWT-a koji se sprema u `localStorage` nakon prijave (Ispis 14.) . Ako JWT postoji, koristi se za postavljanje zaglavlja autorizacije za sve buduće HTTP zahtjeve. Također, JWT se dekodira kako bi se odredile korisničke uloge, što omogućuje aplikaciji da prilagodi prikaz i funkcionalnosti ovisno o ulozi.

```
useEffect(()=>{
        const token = localStorage.getItem('jwtToken');
        console.log('Token from local storage:', token);
        if (token) {
            axios.defaults.headers.common['Authorization'] =
`Bearer ${token}`;
            console.log('Authorization header set:',
axios.defaults.headers.common['Authorization']);
        } else {
            setIsAuthenticated(false);
        }
        loadRelease();
    },[]);
```

Ispis 14: Primjer *frontend* provjere

Jedan od primjera gdje je prikaz prilagođen ovisno o ulozi korisnika može se pronaći unutar komponente `ReleaseList.js`. Pristupom komponenti unutar `useEffecta` se provjerava uloga korisnika te se na temelju toga vrši poziv na različit API (Ispis 15.). Ako je korisnik administrator, komponenta učitava sve dostupne podatke o izdanjima preko API poziva `/release/find/all`. Za korisnike s drugim ulogama, učitavaju se samo ona izdanja koja su dodijeljena trenutnom korisniku, koristeći API poziv na `/release/find/assigned/{userId}`.

```
useEffect(() => {

        if (!isAuthenticated) {

            navigate('/login');

            return;

        }

        if (isAdmin) {

            loadReleases();

        } else if (isReleaseManager || isProjectManager ||
isDeveloper) {

            loadAssignedReleases(page);

        }

    }, [page, size, sortBy, sortDir, filter, isAdmin,
isReleaseManager, isProjectManager, isDeveloper,
isAuthenticated]);
```

Ispis 15: Učitavanje podataka prema ulozi

Komponente koje su zadužene za prikazivanje popisa zapisa iz baze podataka omogućuju korisniku da filtrira, sortira i paginira isti popis. Filtriranje omogućava korisnicima pretragu popisa po raznim parametrima, dok sortiranje omogućuje prikaz po raznim kriterijima. Uzme li se za primjer lista izdanja, ista će se moći pretraživati po nazivu, a sortirati prema različitim kriterijima kao što su ID, naziv, datum kreiranja i datum izdanja. Paginacija omogućuje korisnicima da prelistavaju izdanja u manjim dijelovima, što je posebno korisno kada se radi s velikim količinama podataka.

6. Instalacija aplikacije

Kako bi se uspješno instalirala i pokrenula aplikacija na svježem računalu ili poslužitelju, potrebno je proći nekoliko koraka. Ovi koraci uključuju instalaciju osnovnih alata i okruženja, konfiguraciju potrebnih servisa, te preuzimanje i postavljanje same aplikacije. U nastavku su detaljno opisani svi potrebni koraci.

Aplikacija je razvijena u Java programskom jeziku koristeći Spring Boot okruženje, što znači da je potrebno instalirati JDK (Java Development Kit). JDK se može preuzeti preko službene stranice. Preuzima se najnovije izdanje JDK-a koje je kompatibilno s korisnikovim operativnim sustavom. Tijekom instalacije treba pratiti upute na ekranu kako bi instalacija bila uspješna. Nakon instalacije, provjera se vrši otvaranjem terminala (CMD ili PowerShell) i unosom naredbe `java -version`. Ova naredba bi trebala prikazati izdanje Jave koje se upravo instaliralo.

Maven je alat za upravljanje projektima i njegovim zavisnostima. Koristi se za izgradnju i upravljanje Java projektima. S Apache Maven službene stranice treba preuzeti binarni arhivski paket. Preuzeti paket treba raspakirati te postaviti stazu do `bin` direktorija u `PATH` sistemsku varijablu. Instalacija se provjerava otvaranjem terminala i unosom naredbe `mvn -v`.

Aplikacija koristi PostgreSQL kao sustav za upravljanje bazom podataka. Sa službene PostgreSQL stranice je potrebno preuzeti instalacijski paket za kompatibilni operativni sustav. Instalacija se vrši prema uputama na ekranu. Tijekom instalacije potrebno je postaviti korisničko ime i lozinku za korisnika. Ako se želi pokrenuti aplikacija bez ikakvih promjena u kôdu, korisničko ime neka bude „postgres“, a lozinka „1234“. Nakon instalacije, unutar programa pgAdmin treba stvoriti bazu podataka s nazivom „releaseManagement“.

Budući da će pri instalaciji biblioteke Feature flag biti potreban IDE (Integrated Development Environment) na računalo treba instalirati IntelliJ IDEA. Preko službene stranice se preuzima izdanje koje odgovara korisnikovom operativnom sustavu.

Kako bi se pristupilo i pokrenulo frontend potrebno je preuzeti i Visual Studio Code. Nakon preuzimanja instalacijskog paketa sa službene stranice, isti se pokreće uz pratnju uputa na ekranu.

Imajući na umu da aplikacija za *frontend* koristi razvojni okvir React unutar Visual Studio Codea bit će potrebno instalirati Node.js. Node.js je JavaScript okolina (engl. *environment*) za izvršavanje (engl. *Runtime*) koje omogućuje pokretanje JavaScript kôda izvan preglednika. S Node.js dolazi i „npm“ (Node Package Manager) koji se koristi za upravljanje paketima, uključujući React.

Sa službene stranice potrebno je preuzeti LTS (Long Term Support) izdanje koje je preporučeno za većinu korisnika. Nakon preuzimanja treba pokrenuti paket te pratiti upute za instalaciju na ekranu. Nakon instalacije, otvaranjem terminala i upisivanjem naredbi `node -v` i `npm -v` provjerava se uspješnost instalacije.

Nakon što su preuzeti svi potrebni alati aplikacija se može preuzeti na dva načina. Prvi je uz pomoć Gita. Ako Git nije instaliran na računalu isti se može preuzeti s https://git-scm.com/downloads. Unutar novog direktorija, desnim klikom miša treba izabrati opciju „`Git Bash Here`“. Unutar Git konzole treba upisati iduću naredbu: `git clone` https://github.com/ante-solic/Release-Management-Application.git. Drugi način za preuzeti aplikaciju je preuzimanjem ZIP arhiva sa stranice https://github.com/ante-solic/Release-Management-Application.

Unutar preuzetog direktorija će se nalaziti tri direktorija. Direktorij `CarSales` sadrži testnu aplikaciju preko koje se testira biblioteka Feature flag. Direktorij `Feature-Flag-Library` sadrži navedenu biblioteku. Posljednji direktorij pod nazivom `ReleaseManagment` sadrži *frontend* i *backend* dio aplikacije.

Kako bi biblioteka Feature flag funkcionirala, potrebno ju je instalirati u lokalni maven repozitorij. To se postiže idućim koracima. Prvo je potrebno otvoriti `Feature-Flag-Library` direktorij u IntelliJu. Unutar IntelliJa se otvori maven terminal. U terminal se upisuje iduća naredba: `mvn clean install`. Ako je ovo izvršeno bez greške biblioteka će se instalirati na idućoj putanji:

`C:\Users\USERNAME\.m2\repository\com\featureflag.`

Ovo je posljednji korak koji je bio potreban kako bi se aplikacija, kao i biblioteka instalirali na računalo ili poslužitelj. Ako nije bilo grešaka *backend* aplikacija će se pokrenuti

na lokaciji `http://localhost:8080`. *Frontend* dio aplikacije se pokreće na idućoj lokaciji: `http://localhost:3000` preko naredbe `npm start` unutar Visual Studio Codea nakon što se u njemu otvori direktorij `frontend` koji se nalazi unutar direktorija `ReleaseManagement`.

7. Zaključak

U ovom završnom radu razvijena je web aplikacija za upravljanje izdanjima softvera (engl. *Release Management*) koja omogućava bolju organizaciju i praćenje procesa razvoja softvera. Aplikacija se sastoji od tri glavne komponente: pozadinskog sustava, korisničkog sučelja i biblioteke Feature Flag. Korištenjem tehnologija poput Jave, Spring Boota, Reacta, PostgreSQLa i Flywaya, uspješno je izrađen sustav koji omogućuje stabilno i efikasno upravljanje softverskim izdanjima.

Ova aplikacija može značajno unaprijediti proces razvoja softverskih rješenja, omogućujući bolju kontrolu i pravovremenu isporuku softvera. Posebno je važno istaknuti biblioteku Feature flag, koja omogućava fleksibilno upravljanje značajkama, čime olakšava proces testiranja i uvođenja novih funkcionalnosti.

Korištenjem biblioteke Feature flag, razvojni timovi mogu brzo reagirati na neočekivane probleme ili sigurnosne ranjivosti koje se mogu pojaviti nakon što je nova značajka izdana. U slučaju da neka značajka uzrokuje poteškoće u radu aplikacije, ona se može odmah deaktivirati bez potrebe za vraćanjem na stariju verziju izdanja, čime se smanjuje potencijalna šteta i osigurava kontinuitet usluge za krajnjeg korisnika.

Razvijeni sustav posebno je važan jer omogućava bolju organizaciju u razvoju softvera, što je ključno za održavanje kvalitete i zadovoljstvo krajnjih korisnika. Moguća primjena ovog sustava uključuje različite timove za razvoj softvera, gdje može poslužiti kao centralno mjesto za upravljanje i praćenje izdanja. Predložena poboljšanja uključuju dodatnu automatizaciju procesa, integraciju s drugim alatima za razvoj softvera te mogućnost prilagodbe aplikacije specifičnim potrebama korisnika.

8. Literatura

[1] „Java dokumentacija“, https://dev.java/learn/ (posjećeno 20.08.2024.)

[2] „Spring Boot dokumentacija“, https://docs.spring.io/spring-boot/index.html (posjećeno 20.08.2024.)

[3] „React dokumentacija“, https://legacy.reactjs.org/docs/getting-started.html (posjećeno 21.08.2024.)

[4] „PostgreSQL dokumentacija“, https://www.postgresql.org/docs/current/ (posjećeno 22.08.2024.)

[5] „Flyway dokumentacija“, https://documentation.red-gate.com/fd/ (posjećeno 22.08.2024.)

[6] „JWT (JSON Web Token) dokumentacija“, https://jwt.io/introduction (posjećeno 23.08.2024.)

[7] „Maven dokumentacija“, https://maven.apache.org/guides/index.html (posjećeno 24.08.2024.)