UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Natalija Asanović

PROJEKTOVANJE I IMPLEMENTACIJA VEB APLIKACIJE ZA CENTRALIZOVANO UPRAVLJANJE ORGANIZACIJOM SVEČANIH DOGAĐAJA

master rad

Beograd, 2026.

Mentor:

dr Mirjana Maljković Ružičić, docent
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Nina Radojičić Matić, docent
Univerzitet u Beogradu, Matematički fakultet

dr Staša Vujičić Stanković, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Naslov master rada: Projektovanje i implementacija veb aplikacije za centralizovano upravljanje organizacijom svečanih događaja

Rezime: Master rad bavi se analizom, projektovanjem i implementacijom veb aplikacije **PlanIT**, namenjene organizaciji i koordinaciji svečanih događaja. Cilj rada je razvoj sistema koji omogućava pouzdanu razmenu informacija između koordinatora, klijenata i spoljnih saradnika, uz jasno definisanu kontrolu pristupa podacima i evidentiranje aktivnosti korisnika. Posebna pažnja posvećena je modelovanju podataka i primeni kombinacije relacione i nerelacione baze podataka u okviru jedinstvenog sistema. Za skladištenje podataka primenjen je pristup polyglot persistence, kojim se kombinuju relacioni i dokumentno orijentisani modeli. Rad obuhvata prikaz arhitekture, implementaciju najvažnijih funkcionalnosti i testiranje razvijenog rešenja. Implementirana aplikacija omogućava centralizovano upravljanje događajima, komunikacijom i korisničkim aktivnostima, kao i dalje proširenje sistema u skladu sa potrebama različitih organizacionih procesa.

Ključne reči: veb aplikacija, kontrola pristupa, Python, Django, React, SQLite, MongoDB, organizacija svečanih događaja

Sadržaj

Glava 1

Uvod

Organizacija svečanih događaja predstavlja složen proces koji podrazumeva koordinaciju većeg broja učesnika, blagovremenu razmenu informacija i praćenje različitih aktivnosti tokom pripreme i realizacije događaja. U ovom procesu učestvuju koordinatori, klijenti i spoljni saradnici, čije aktivnosti moraju biti međusobno usklađene kako bi događaj bio realizovan u skladu sa definisanim zahtevima i rokovima. Savremeni način poslovanja zato nameće potrebu za digitalizacijom organizacionih procesa, sa ciljem unapređenja komunikacije, centralizovanog upravljanja podacima i pouzdanijeg praćenja aktivnosti svih učesnika.

U okviru ovog rada razvijena je veb aplikacija PlanIT, namenjena organizaciji i koordinaciji svečanih događaja. Cilj rada je analiza, projektovanje i implementacija sistema koji omogućava razmenu informacija između koordinatora, klijenata i spoljnih saradnika, uz jasno definisane mehanizme kontrole pristupa podacima i evidentiranja aktivnosti korisnika. Istraživačko pitanje rada odnosi se na modelovanje kontrole pristupa i pouzdane evidencije aktivnosti u višekorisničkom sistemu u kojem prava korisnika zavise ne samo od njihove uloge, već i od odnosa prema konkretnom događaju.

Aplikacija omogućava upravljanje događajima, korisnicima i njihovim ulogama, internu komunikaciju učesnika, komunikaciju sa spoljnim saradnicima, vođenje privatnih beleški i praćenje značajnih promena u sistemu putem revizijskog dnevnika (engl. *audit log*). Na taj način formirano je centralizovano okruženje u kojem su informacije o događajima, učesnicima i aktivnostima dostupne u skladu sa pravima pristupa svakog korisnika.

Aplikacija je realizovana kao veb sistem zasnovan na klijent-server arhitekturi. Serverska strana razvijena je u programskom jeziku Python korišćenjem razvojnog

okvira Django i biblioteke Django REST Framework, koji omogućavaju implementaciju poslovne logike i izlaganje funkcionalnosti sistema putem REST programskog interfejsa. Klijentska strana realizovana je korišćenjem biblioteke React i programskog jezika TypeScript, u obliku jednostranične veb aplikacije (engl. *Single Page Application*). Komunikacija između klijentske i serverske strane ostvarena je putem HTTP zahteva prema REST API-ju. Autentifikacija korisnika zasnovana je na JWT tokenima, dok je autorizacija realizovana kombinovanjem kontrole zasnovane na korisničkim ulogama i provera odnosa korisnika prema konkretnom resursu.

Za skladištenje podataka primenjen je pristup *polyglot persistence*, odnosno kombinovanje relacione i nerelacione baze podataka u okviru istog sistema. Relacioni sistem za upravljanje bazama podataka SQLite koristi se za čuvanje strukturisanih podataka sa jasno definisanim vezama, kao što su podaci o korisnicima, događajima i spoljnim saradnicima. Nerelacioni dokumentno orijentisani sistem za upravljanje bazama podataka MongoDB se koristi za skladištenje poruka, privatnih beleški i revizijskih zapisa, odnosno podataka čija je struktura fleksibilnija. Ovakva podela omogućava da se različite kategorije podataka modeluju i čuvaju u skladu sa njihovom prirodom i načinom korišćenja.

Ostatak rada organizovan je na sledeći način. U drugoj glavi prikazane su tehnologije i alati korišćeni prilikom razvoja aplikacije, sa posebnim osvrtom na serverske i klijentske veb tehnologije, autentifikaciju i modele skladištenja podataka. U trećoj glavi opisana je aplikacija PlanIT, njena arhitektura, model podataka, organizacija klijentske i serverske strane, kao i implementacija najznačajnijih funkcionalnosti. U okviru iste glave prikazana je i strategija testiranja, uz razmatranje postojećih ograničenja sistema. U četvrtoj glavi dat je zaključak, u kojem su sumirani ostvareni rezultati i navedene mogućnosti daljeg razvoja aplikacije.

Glava 2

Tehnologije

2.1 Programski jezik Python

Python [35] je programski jezik opšte namene koji se odlikuje jednostavnom sintaksom, visokim nivoom apstrakcije i naglašenom čitljivošću izvornog koda. Zahvaljujući ovim osobinama, pogodan je za razvoj različitih vrsta softvera, uključujući veb aplikacije, automatizaciju procesa i obradu podataka [19].

Programski jezik Python podržava više programskih paradigmi, među kojima su proceduralno, objektno orijentisano i funkcionalno programiranje, što omogućava da se način organizovanja programa prilagodi prirodi problema koji se rešava [19]. Funkcije omogućavaju izdvajanje ponovljivih postupaka u zasebne celine, dok klase i objekti pružaju osnovu za modelovanje složenijih pojmova iz domena aplikacije. Jezik je dinamički tipiziran, što doprinosi brzini razvoja, ali zahteva pažljivo testiranje i disciplinovanu organizaciju programa – u većim projektima se zbog toga često koriste anotacije tipova, automatski testovi i alati za statičku analizu.

Pored osnovnih jezičkih mogućnosti, programski jezik Python poseduje razvijen ekosistem biblioteka i razvojnih okvira, koji omogućava implementaciju rada sa bazama podataka, mrežnom komunikacijom i izgradnjom veb aplikacija bez potrebe da se svaka infrastrukturna funkcionalnost razvija od početka [19]. Ovakav ekosistem, uz dostupnost Django okvira i biblioteke Django REST Framework, predstavlja osnovni razlog za izbor programskog jezika Python za serversku stranu aplikacije PlanIT.

2.2 Okvir Django i biblioteka Django REST Framework

Okvir Django

Django [6] je razvojni okvir otvorenog koda namenjen izradi veb aplikacija u programskom jeziku Python. Njegova osnovna namena jeste da obezbedi skup međusobno povezanih komponenti za rešavanje uobičajenih problema u veb razvoju, kao što su obrada HTTP zahteva, rutiranje, rad sa bazama podataka, autentifikacija korisnika, validacija podataka i administracija sistema [41]. Zahvaljujući tome, veći deo pažnje se može posvetiti modelovanju poslovnog domena i implementaciji specifičnih funkcionalnosti aplikacije, umesto ponovnoj izradi osnovnih infrastrukturnih mehanizama.

Django projekat se organizuje kao celina koja sadrži opštu konfiguraciju i jednu ili više aplikacija. Svaka aplikacija predstavlja funkcionalno zaokružen deo sistema i može obuhvatati sopstvene modele, poglede (engl. *views*), URL obrasce i pomoćne komponente [41]. Ovakva podela podstiče modularnost i olakšava razvoj, testiranje i održavanje većih softverskih sistema. Zvanična dokumentacija okvira Django takođe razlikuje projekat kao skup konfiguracije i aplikacija od aplikacije kao komponente koja realizuje određenu funkcionalnost [6].

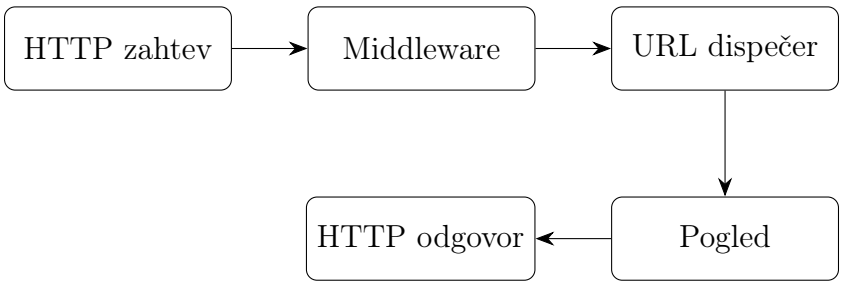
Arhitektura Django aplikacije najčešće se opisuje pomoću obrasca *Model–Template–View*: *model* predstavlja strukturu i pravila podataka, *template* sloj je zadužen za prikaz sadržaja, dok *view* obrađuje zahtev i priprema odgovor [41]. U aplikacijama u kojima se Django koristi prvenstveno kao serverski deo REST sistema, *template* sloj može imati manju ulogu, jer se podaci prosleđuju klijentskoj aplikaciji u strukturisanom formatu, najčešće kao JSON. Ipak, osnovno razdvajanje odgovornosti ostaje prisutno: modeli opisuju podatke, pogledi obrađuju zahteve, a rutiranje povezuje URL obrasce sa odgovarajućom logikom [6].

Jedna od centralnih komponenti Django okvira jeste objektno-relaciono mapiranje, odnosno ORM, koje omogućava da se tabele relacione baze podataka predstave pomoću Python klasa, dok se redovi u tabelama predstavljaju njihovim instancama [41]. Polja modela odgovaraju kolonama baze, a veze između modela mogu se izraziti korišćenjem relacija *jedan-prema-jedan*, *jedan-prema-više* i *više-prema-više*. Na taj način veliki broj operacija nad bazom podataka može se realizovati bez neposrednog pisanja SQL naredbi [6].

Django ORM podržava pravljenje, čitanje, izmenu i brisanje podataka pomoću Python izraza. Upiti se formiraju korišćenjem objekata tipa `QuerySet`, koji podržavaju filtriranje, sortiranje, agregaciju i povezivanje podataka iz više tabela. Važna osobina `QuerySet` objekata jeste odloženo izvršavanje (engl. *lazy loading*): upit se ne šalje sistemu za upravljanje bazom podataka u trenutku njegovog definisanja, već tek kada je rezultat zaista potreban [6].

Osim upita nad postojećim podacima, ORM upravlja i promenama same strukture baze – putem sistema migracija. Migracije predstavljaju verzionisane opise promena šeme, kao što su pravljenje tabele, dodavanje polja, izmena ograničenja ili uklanjanje relacije. Komanda `makemigrations` generiše migracione datoteke na osnovu promena u modelima, dok komanda `migrate` primenjuje te promene na konkretnu bazu [41]. Time se struktura baze usklađuje sa izvornim kodom i omogućava kontrolisano menjanje šeme tokom razvoja sistema. Zvanična dokumentacija detaljno opisuje tok migracija, njihove zavisnosti, transakcije, istorijske modele i migracije podataka [6].

Kada je struktura podataka definisana, sledeći korak je obrada zahteva koji nad njom operišu. Dolazni HTTP zahtev najpre prolazi kroz posredničke slojeve (engl. *middleware*), nakon čega URL dispečer određuje pogled koji treba da ga obradi; pogled zatim prima objekat zahteva, izvršava odgovarajuću poslovnu logiku i vraća HTTP odgovor [6]. Tok obrade zahteva prikazan je na slici 2.1.



Slika 2.1: Tok obrade HTTP zahteva u Django okviru

Django podržava i funkcijske i klasne poglede, pri čemu klasni pogledi omogućavaju ponovno korišćenje zajedničkog ponašanja i lakšu organizaciju složenijih funkcionalnosti [6]. Posrednički slojevi, pomenuti u prethodnom koraku, izvršavaju se pre ili posle pogleda i obrađuju zahteve i odgovore na nivou čitave aplikacije – kroz njih se najčešće realizuju upravljanje sesijama, povezivanje autentifikovanog korisnika sa zahtevom, bezbednosne provere i evidentiranje aktivnosti. Njihov redosled

je značajan, jer svaki od njih prima rezultat prethodnog i prosleđuje obradu sledećem [6], pa njihova konfiguracija mora biti usklađena sa međusobnim zavisnostima u toku obrade zahteva.

Django dolazi i sa ugrađenim sistemom za autentifikaciju i autorizaciju korisnika, koji obuhvata korisničke naloge, grupe, dozvole, upravljanje lozinkama i povezivanje autentifikovanog korisnika sa zahtevom [41]. Podrazumevani sistem može se proširiti dodatnim poljima, prilagođenim korisničkim modelom i sopstvenim pravilima pristupa [6]. Prilagođeni korisnički model se najčešće definiše na početku projekta, pre primene prvih migracija, jer naknadna zamena podrazumevanog modela može zahtevati složene izmene šeme i zavisnosti.

Django sadrži i ugrađene mehanizme zaštite od čestih bezbednosnih problema, uključujući falsifikovanje zahteva između sajtova (engl. *Cross-Site Request Forgery*), SQL umetanje i napade podmetanjem korisničkog interfejsa (engl. *clickjacking* napade) [5].

Biblioteka Django REST Framework

Django REST Framework, u nastavku DRF, predstavlja skup komponenti namenjenih razvoju veb programskih interfejsa nad Django okvirom. DRF uvodi apstrakcije prilagođene REST servisima, među kojima su serijalizatori, API pogledi, generički pogledi, `ViewSet` klase, ruteri, autentifikacione politike i klase dozvola [8]. Na taj način funkcionalnosti Django aplikacije mogu se izložiti klijentskim sistemima kroz standardizovane HTTP metode i strukturisane reprezentacije podataka.

Serijalizatori [10] imaju dvostruku ulogu. Oni pretvaraju složene Python i Django objekte u strukture pogodne za formate kao što je JSON, ali istovremeno obavljaju validaciju ulaznih podataka i njihovo pretvaranje u interne objekte. Serijalizator određuje koja polja se izlažu klijentu, koja se mogu menjati, koja su samo za čitanje i koja pravila validacije moraju biti zadovoljena.

Povezivanje serijalizatora sa Django modelima omogućava automatsko formiranje polja na osnovu strukture modela, ali javni API ne bi trebalo poistovetiti sa unutrašnjim modelom baze [10]. Neka polja mogu sadržati poverljive ili tehničke podatke koje nije primereno izložiti klijentskoj strani. Zbog toga serijalizator ima i bezbednosnu ulogu, jer kontroliše sadržaj koji se prihvata i vraća kroz API.

Ova bezbednosna uloga serijalizatora prirodno se nadovezuje na način na koji se pogledi organizuju – DRF podržava više nivoa apstrakcije pri njihovom pisanju. Osnovna klasa `APIView` [11] pruža neposrednu kontrolu nad HTTP metodama, dok

generički pogledi i mixin klase nude unapred definisano ponašanje za uobičajene operacije nad resursima. `ViewSet` [12] klase grupišu logiku povezanih akcija, kao što su listanje, prikaz pojedinačnog objekta, pravljenje i izmena, a ruteri zatim mogu automatski da formiraju odgovarajuće URL obrasce.

Autentifikacija i autorizacija u DRF-u predstavljaju odvojene mehanizme [7, 9]. Autentifikaciona klasa pokušava da odredi identitet korisnika i autentifikacione podatke povezane sa zahtevom, dok klase dozvola odlučuju da li je konkretna operacija dozvoljena. Provera dozvola vrši se na početku obrade pogleda, pre izvršavanja njegove poslovne logike, a odluka se može zasnivati na korisniku, njegovoj ulozi, vrsti zahteva i objektu kome pristupa.

Za složenije sisteme mogu se definisati i prilagođene klase dozvola, koje omogućavaju proveru korisničke uloge, vlasništva nad resursom, pripadnosti određenom događaju ili drugih pravila poslovnog domena [9]. DRF podržava i objektne dozvole, koje se primenjuju na konkretnu instancu resursa nakon njenog pronalaženja.

U okviru aplikacije PlanIT okvir Django je korišćen kao osnova serverske strane sistema. Django modeli predstavljaju strukturisane podatke o korisnicima, događajima i spoljnim saradnicima, ORM obezbeđuje pristup relacionoj bazi, dok migracije omogućavaju kontrolisano menjanje šeme. Prilagođeni korisnički model koristi se za razlikovanje administratora, koordinatora i klijenata.

Django REST Framework korišćen je za izlaganje funkcionalnosti sistema kroz REST API. Serijalizatori obavljaju validaciju ulaznih podataka i formiranje odgovora, dok pogledi i servisni sloj obrađuju zahteve koji se odnose na događaje, korisnike, poruke, beleške, saradnike i revizijske zapise. Prilagođene klase dozvola obezbeđuju kontrolu pristupa zasnovanu na korisničkoj ulozi i povezanosti korisnika sa konkretnim događajem.

2.3 Biblioteka React i programski jezik TypeScript

Biblioteka React

React je biblioteka otvorenog koda namenjena razvoju korisničkih interfejsa u veb aplikacijama [22]. Njegov razvojni model zasniva se na komponentama, deklarativnom opisivanju prikaza i jednosmernom protoku podataka. Umesto neposrednog menjanja elemenata interfejsa kao odgovora na svaku korisničku akciju, opisuje se izgled interfejsa za određeno stanje aplikacije, dok React usklađuje prikaz sa pro-

menama podataka [3].

Korisnički interfejs organizuje se kao skup komponenti koje predstavljaju funkcionalno zaokružene celine i mogu se kombinovati radi izgradnje složenijih prikaza [28]. Komponente se najčešće definišu kao funkcije, dok se za opisivanje strukture prikaza koristi JSX, sintaksno proširenje koje omogućava zapis sličan HTML-u unutar JavaScript ili TypeScript koda [27]. JSX se tokom izgradnje aplikacije transformiše u odgovarajuće JavaScript izraze, pri čemu se u prikaz mogu uključiti promenljive, pozivi funkcija i drugi izrazi [3].

Podaci se komponentama mogu prosleđivati pomoću svojstava, odnosno *props*, dok se podaci koji se menjaju tokom rada komponente čuvaju u njenom stanju [23, 25]. Promena stanja dovodi do ponovnog formiranja odgovarajućeg prikaza, što omogućava da korisnički interfejs reaguje na korisničke akcije i promene podataka. U funkcijskim komponentama stanje se najčešće definiše pomoću Hook funkcije `useState` [21].

React komponente mogu reagovati i na događaje koje pokreće korisnik, kao što su klik, unos teksta ili slanje obrasca, pri čemu odgovarajuće funkcije mogu menjati stanje komponente ili pokrenuti druge operacije [24]. Za operacije povezane sa sistemima izvan same komponente, kao što su slanje mrežnih zahteva, povezivanje sa spoljnim sistemom ili registrovanje pretplate, može se koristiti Hook funkcija `useEffect` [20].

Deklarativni i komponentni pristup olakšavaju razdvajanje prikaza od komunikacione i poslovne logike. Ipak, kvalitet React aplikacije zavisi od načina na koji su određene granice komponenti i raspoređeno stanje. Prevelike komponente mogu obuhvatiti više nepovezanih odgovornosti, dok preterano usitnjavanje može nepotrebno povećati složenost. Zbog toga je komponente korisno formirati prema jasnim funkcionalnim celinama.

Programski jezik TypeScript u React aplikacijama

Programski jezik TypeScript predstavlja nadskup programskog jezika JavaScript koji uvodi statički sistem tipova. Kod napisan u programskom jeziku TypeScript prevodi se u JavaScript, koji se zatim izvršava u pregledaču. Uvođenjem tipova moguće je eksplicitno opisati oblik podataka, ulaze i rezultate funkcija, strukturu objekata i ugovore između komponenti – odnosno jasno definisan oblik podataka koji jedna komponenta očekuje od druge.

U React aplikacijama TypeScript se koristi za opisivanje svojstava komponenti, stanja i događaja. Tipovi i interfejsi omogućavaju definisanje podataka koje komponenta očekuje, dok se datoteke sa ekstenzijom `.tsx` koriste kada se JSX kombinuje sa TypeScript kodom [26].

Poseban značaj statičkog tipiziranja ogleda se u komunikaciji sa REST API-jem. Odgovori serverske strane mogu se predstaviti TypeScript interfejsima koji opisuju korisnike, događaje, poruke i druge domenske objekte, čime se preciznije definiše struktura podataka koju koriste komponente klijentske aplikacije. Ipak, TypeScript proverava tipove tokom razvoja i ne garantuje da podaci pristigli preko mreže zaista imaju očekivanu strukturu, pa je u kritičnim slučajevima potrebna i provera tokom izvršavanja.

U aplikaciji PlanIT React i TypeScript korišćeni su za razvoj klijentske strane, pri čemu su TypeScript tipovi korišćeni za opisivanje korisničkih podataka, događaja, poruka i drugih struktura koje se razmenjuju sa serverskom stranom. Kombinacija biblioteke React i programskog jezika TypeScript omogućila je modularnu organizaciju klijentske strane, jasnije ugovore između komponenti i pouzdanije izmene tokom razvoja.

2.4 JWT autentifikacija

Autentifikacija predstavlja postupak utvrđivanja identiteta korisnika, dok autorizacija određuje kojim resursima i operacijama autentifikovani korisnik može da pristupi. U savremenim klijent-server aplikacijama jedan od često korišćenih pristupa za rešavanje ovog prvog koraka zasniva se na JSON Web Token (JWT) standardu.

JWT predstavlja kompaktan format za prenos skupa tvrdnji (engl. *claims*) između dve strane [17]. Token se sastoji od zaglavlja (engl. *header*), korisničkog sadržaja (engl. *payload*) i potpisa (engl. *signature*). Zaglavlje sadrži podatke o tipu tokena i algoritmu potpisa, korisnički sadržaj obuhvata skup tvrdnji, dok potpis omogućava proveru integriteta tokena i potvrđuje da njegov sadržaj nije izmenjen nakon izdavanja.

Standardno JWT kodiranje ne predstavlja šifrovanje. Podaci iz zaglavlja i korisničkog sadržaja mogu se dekodirati bez posedovanja tajnog ključa. Zbog toga token ne bi trebalo da sadrži lozinke, poverljive lične podatke ili druge informacije čije bi otkrivanje moglo da ugrozi korisnika ili sistem. Potpis štiti integritet tokena, ali ne obezbeđuje poverljivost njegovog sadržaja [17].

Ova razlika između zaštite integriteta i poverljivosti neposredno se odražava na sam tok autentifikacije. Korisnik najpre prosleđuje svoje kredencijale serverskoj strani, a nakon njihove provere server generiše token i vraća ga klijentu. Klijent zatim token prosleđuje pri svakom narednom zahtevu prema zaštićenom resursu, najčešće u HTTP zaglavlju `Authorization`, uz oznaku `Bearer`. Serverska strana proverava potpis, vreme važenja i druge relevantne tvrdnje, nakon čega određuje identitet korisnika i nastavlja obradu zahteva.

JWT autentifikacija razlikuje se od klasičnog pristupa zasnovanog na serverskim sesijama. Kod sesijskog modela server čuva stanje o prijavljenom korisniku, dok se kod JWT pristupa većina podataka potrebnih za proveru identiteta može nalaziti u samom tokenu. Time se olakšava rad distribuiranih sistema, jer svaka serverska instanca koja poseduje odgovarajući ključ može proveriti token. Ipak, u praksi se često uvode mehanizmi za opoziv tokena i osvežavanje tokena, pa ovakav sistem nije nužno potpuno bez čuvanja stanja (engl. *stateless*) [39].

Uobičajeno je korišćenje dve vrste tokena: pristupnog tokena (engl. *access token*) i osvežavajućeg tokena (engl. *refresh token*). Pristupni token ima kraći period važenja i koristi se za pristup zaštićenim API krajnjim tačkama. Osvežavajući token traje duže i služi za dobijanje novog pristupnog tokena bez ponovnog unošenja korisničkog imena i lozinke. Ovakva podela smanjuje posledice kompromitovanja pristupnog tokena, ali istovremeno zahteva strožu zaštitu osvežavajućeg tokena [39].

Bezbednost JWT autentifikacije zavisi od pravilnog izbora algoritma, upravljanja ključevima i provere standardnih tvrdnji: tvrdnja `exp` (engl. *expiration time*) određuje trenutak isteka tokena, `iat` (engl. *issued at*) vreme njegovog izdavanja, dok se `iss` (engl. *issuer*) i `aud` (engl. *audience*) mogu koristiti za proveru izdavaoca i ciljne publike tokena [17]. Zanemarivanje ovih provera može dovesti do prihvatanja tokena koji je istekao ili nije namenjen konkretnom sistemu.

Autentifikaciju ne treba izjednačavati sa autorizacijom. JWT može sadržati identifikator korisnika ili njegovu ulogu, ali serverska aplikacija i dalje mora da proveri da li korisnik ima pravo pristupa konkretnom resursu. Pravila pristupa zato se sprovode na nivou krajnjih tačaka i pojedinačnih objekata, u skladu sa korisničkom ulogom i odnosom korisnika prema traženom resursu.

U praksi se ovakva podela odgovornosti sprovodi kroz konkretne biblioteke. U okviru biblioteke Django REST Framework JWT autentifikacija može se realizovati pomoću biblioteke `djangorestframework-simplejwt`, koja podržava izdavanje pristupnih i osvežavajućih tokena, njihovu proveru, obnavljanje pristupnog tokena,

rotaciju (engl. *rotation*) osvežavajućih tokena i njihovo stavljanje na crnu listu (engl. *blacklist*) [16]. Rotacija podrazumeva izdavanje novog osvežavajućeg tokena pri svakoj uspešnoj obnovi, dok se prethodni token može opozvati kako bi se smanjila mogućnost njegove ponovne zloupotrebe.

U aplikaciji PlanIT JWT autentifikacija koristi se za zaštitu komunikacije između klijentske i serverske strane.

2.5 Modeli kontrole pristupa

Kontrola pristupa predstavlja mehanizam kojim se određuje kojim resursima korisnik može da pristupi i koje operacije nad njima može da izvrši. U višekorisničkim sistemima pravila pristupa mogu se modelovati na različite načine, u zavisnosti od toga na kojim svojstvima se zasniva odluka o dozvoli pristupa.

Kontrola pristupa zasnovana na ulogama (engl. *Role-Based Access Control*, RBAC) povezuje dozvole sa unapred definisanim korisničkim ulogama [37]. Korisniku se dodeljuje odgovarajuća uloga, a njegove mogućnosti u sistemu proizlaze iz skupa dozvola povezanih sa tom ulogom. Ovakav pristup je pogodan za sisteme u kojima postoje jasno razgraničene kategorije korisnika i njihove odgovornosti.

Kontrola pristupa zasnovana na atributima (engl. *Attribute-Based Access Control*, ABAC) odluku o pristupu donosi na osnovu atributa korisnika, resursa, zahtevane operacije i, po potrebi, konteksta u kojem se zahtev izvršava [15]. U odnosu na RBAC, ovaj pristup omogućava izražavanje detaljnijih pravila, ali istovremeno povećava složenost njihovog definisanja i održavanja.

Kontrola pristupa zasnovana na odnosima (engl. *Relationship-Based Access Control*, ReBAC) polazi od odnosa između korisnika i resursa, odnosno između različitih entiteta sistema [13]. Pravo pristupa tako može zavisiti od toga da li je korisnik vlasnik određenog resursa, da li je sa njim neposredno povezan ili se nalazi u drugom odgovarajućem odnosu.

Navedeni modeli ne moraju biti međusobno isključivi. U sistemima u kojima opšte mogućnosti korisnika zavise od njegove uloge, ali pristup pojedinačnim resursima zavisi i od odnosa prema njima, moguće je kombinovati kontrolu zasnovanu na ulogama sa dodatnim proverama na nivou konkretnog objekta. Izbor odgovarajućeg pristupa zato zavisi od strukture korisničkih uloga, odnosa između učesnika i resursa, kao i potrebnog nivoa granularnosti pravila pristupa

2.6 Revizijski dnevnik i evidencija aktivnosti

U višekorisničkim informacionim sistemima nije dovoljno odrediti samo koje operacije je određenom korisniku dozvoljeno da izvrši. Za pouzdano praćenje rada sistema važno je evidentirati i operacije koje su zaista izvršene, vreme njihovog nastanka i resurse nad kojima su sprovedene. Ovakva evidencija najčešće se organizuje u obliku revizijskog dnevnika (engl. *audit log*), koji predstavlja hronološki skup revizijskih zapisa o značajnim aktivnostima i promenama u sistemu. Pojedinačni revizijski zapis predstavlja podatke o konkretnoj evidentiranoj aktivnosti, dok niz takvih zapisa omogućava formiranje revizijskog traga (engl. *audit trail*), odnosno praćenje istorije aktivnosti i promena tokom vremena.

Revizijski zapis treba da sadrži dovoljno podataka da se naknadno može utvrditi kontekst evidentirane aktivnosti. U zavisnosti od prirode sistema, među tim podacima mogu se nalaziti identitet izvršioca, vrsta operacije, objekat nad kojim je operacija izvršena, vreme njenog nastanka i dodatni podaci koji opisuju konkretnu promenu [32].

Posebno značajna svojstva revizijskog dnevnika jesu integritet i nepromenljivost zapisa. Da bi revizijski trag predstavljao pouzdanu evidenciju aktivnosti, potrebno je sprečiti neovlašćeno menjanje ili uklanjanje prethodno evidentiranih podataka. Jedan od pristupa jeste organizovanje dnevnika po principu dodavanja novih zapisa bez izmene prethodnih (engl. *append-only*). Na taj način čuva se hronološki sled aktivnosti i smanjuje mogućnost narušavanja integriteta prethodno evidentiranih podataka [18].

Zaštita revizijskog dnevnika podrazumeva i odgovarajuću kontrolu pristupa. Budući da revizijski zapisi mogu sadržati podatke o korisnicima, izvršenim operacijama i promenama nad resursima sistema, pristup dnevniku treba ograničiti na ovlašćene korisnike. Posebno je važno kontrolisati mogućnost izmene i brisanja zapisa, jer bi neovlašćena promena njihovog sadržaja mogla da naruši pouzdanost celokupnog revizijskog traga [18].

Revizijski dnevnik i kontrola pristupa imaju različite, ali međusobno povezane uloge. Kontrola pristupa određuje da li korisnik ima pravo da izvrši određenu operaciju, dok revizijski dnevnik omogućava naknadno praćenje operacija koje su izvršene. Ovakav pristup naročito je značajan u sistemima u kojima više nezavisnih učesnika izvršava operacije nad zajedničkim resursima.

Modelovanje mehanizama kontrole pristupa i evidencije aktivnosti predstavlja

jedno od ključnih istraživačkih pitanja ovog rada. U kontekstu sistema za organizaciju događaja, potrebno je omogućiti ne samo ograničavanje pristupa podacima već i pouzdano povezivanje značajnih aktivnosti sa njihovim izvršiocem, vremenom nastanka i odgovarajućim događajem. Zbog toga revizijski dnevnik predstavlja važan deo arhitektonskog rešenja, dok je način njegove konkretne realizacije u aplikaciji PlanIT predstavljen u poglavlju 3.

2.7 Baze podataka

Baze podataka predstavljaju jednu od osnovnih komponenti savremenih informacionih sistema, jer omogućavaju organizovano, trajno i pouzdano čuvanje podataka. Izbor odgovarajućeg modela skladištenja zavisi od strukture podataka, načina njihovog korišćenja, zahteva u pogledu konzistentnosti i očekivanog opterećenja sistema. U praksi se najčešće koriste relacione baze podataka, zasnovane na tabelama i unapred definisanim relacijama, i nerelacione baze (engl. *NoSQL*), koje koriste fleksibilnije modele kao što su dokumenti, parovi ključ–vrednost (engl. *key-value*) ili grafovi.

U okviru aplikacije PlanIT korišćena su dva različita sistema za upravljanje bazama podataka. SQLite je primenjen za strukturisane podatke sa jasno definisanim međusobnim vezama, dok je MongoDB korišćen za podatke čija je struktura promenljivija i koji se tokom rada sistema pretežno dodaju, kao što su poruke, privatne beleške i revizijski zapisi. Ovakva podela omogućava da se svaka grupa podataka čuva u modelu koji bolje odgovara njenim osobinama i načinu korišćenja.

SQLite sistem za upravljanje bazama podataka

SQLite je relacioni sistem za upravljanje bazama podataka (engl. *database management system*) koji implementira veliki deo SQL standarda i podržava transakcionu obradu podataka [38]. Za razliku od klasičnih relacionih sistema koji zahtevaju pokretanje posebnog serverskog procesa, SQLite funkcioniše kao biblioteka ugrađena neposredno u aplikaciju (engl. *embedded database*). Celokupna baza podataka najčešće se čuva u jednoj datoteci, čime se pojednostavljuju instalacija, konfiguracija i prenos između različitih okruženja.

Relacioni model podrazumeva predstavljanje podataka pomoću tabela, pri čemu svaka tabela sadrži redove i kolone. Red predstavlja jednu instancu posmatranog

entiteta, dok kolone opisuju njegova svojstva. Veze između različitih entiteta uspostavljaju se primarnim (engl. *primary key*) i stranim ključevima (engl. *foreign key*). Ovakav pristup omogućava precizno definisanje strukture podataka, ograničenja integriteta i međusobnih odnosa između različitih tabela [38].

Jedna od važnih osobina relacionih baza jeste mogućnost definisanja ograničenja. Ograničenja jedinstvenosti (engl. *unique constraints*) sprečavaju postojanje dupliranih vrednosti u kolonama za koje je to nedozvoljeno, dok strani ključevi obezbeđuju da se povezani zapisi odnose na postojeće entitete. Na taj način deo poslovnih pravila može biti sproveden neposredno na nivou baze podataka, čime se smanjuje mogućnost nastanka nekonzistentnih zapisa.

SQLite podržava ACID svojstva transakcija (engl. *atomicity, consistency, isolation, durability*), odnosno atomičnost, konzistentnost, izolovanost i trajnost [38]. Atomičnost podrazumeva da se skup povezanih operacija izvršava kao jedinstvena celina, pa se u slučaju greške sve promene moraju poništiti. Konzistentnost zahteva da transakcija prevede bazu iz jednog ispravnog stanja u drugo, dok izolovanost određuje način na koji istovremene operacije međusobno utiču jedna na drugu. Trajnost obezbeđuje da uspešno potvrđene promene ostanu sačuvane i nakon prekida rada sistema.

Zahvaljujući ovim osobinama, SQLite je posebno pogodan za razvojna i testna okruženja i lokalne aplikacije, dok jednostavan format zasnovan na jednoj datoteci dodatno olakšava pravljenje rezervnih kopija (engl. *backup*).

Odsustvo posebnog serverskog procesa predstavlja i ograničenje u sistemima sa velikim brojem istovremenih operacija upisa. SQLite dozvoljava više paralelnih operacija čitanja, ali je konkurentni upis ograničen načinom zaključavanja datoteke (engl. *file locking*). Zbog toga nije optimalan izbor za sisteme sa velikim brojem istovremenih korisnika i intenzivnim promenama podataka. U takvim okruženjima pogodniji su serverski relacioni sistemi, kao što su PostgreSQL ili MySQL.

U aplikaciji PlanIT SQLite se koristi za skladištenje strukturisanih podataka o korisnicima, događajima, spoljnim saradnicima i njihovim međusobnim vezama. Ovi podaci imaju stabilnu strukturu i zahtevaju jasno definisane relacije.

Rad sa SQLite bazom realizovan je preko Django ORM sistema. Modeli definisani u serverskom delu aplikacije preslikavaju se u relacione tabele, dok se strani ključevi i ograničenja jedinstvenosti koriste za održavanje integriteta podataka. Ovakav pristup omogućava da se većina operacija nad bazom podataka izrazi pomoću Python koda, dok ORM generiše odgovarajuće SQL upite.

Izbor sistema za upravljanje bazom podataka SQLite odgovara razvojnom karakteru aplikacije i očekivanom obimu korišćenja. Njegova jednostavna konfiguracija omogućava lako pokretanje projekta bez dodatne infrastrukture. Istovremeno, pošto Django ORM odvaja poslovni model od konkretnog sistema za upravljanje bazom, moguća je kasnija zamena sistema SQLite drugim relacionim sistemom uz ograničene izmene konfiguracije i proveru eventualnih razlika u podržanim funkcionalnostima.

MongoDB sistem za upravljanje bazama podataka

MongoDB je dokumentno orijentisan sistem za upravljanje bazama podataka (engl. *document-oriented database*) koji podatke organizuje u dokumente (engl. *documents*) i kolekcije (engl. *collections*) [4, 29]. Dokument predstavlja osnovnu jedinicu skladištenja i sastoji se od parova polje–vrednost. Njegova struktura slična je JSON objektu, dok se podaci interno čuvaju u binarnom BSON formatu. Kolekcija predstavlja skup dokumenata i ima ulogu približno sličnu tabeli u relacionoj bazi.

Za razliku od relacionog modela, dokumentni model ne zahteva da svi zapisi unutar kolekcije imaju potpuno isti skup polja – pristup poznat i kao fleksibilna šema (engl. *schema-less* ili *flexible schema*) [4, 29]. Dokumenti mogu posedovati različitu strukturu, a pojedina polja mogu sadržati ugnježdene objekte i nizove. Ovakva fleksibilnost dozvoljava da se podaci čuvaju u obliku koji je bliži strukturi objekata u aplikaciji i da se model postepeno prilagođava promenama poslovnih zahteva.

Fleksibilna šema ne znači da struktura podataka treba da bude proizvoljna. Dokumenti unutar iste kolekcije u praksi najčešće dele zajednički skup osnovnih polja, dok se razlike javljaju u opcionim ili dodatnim podacima. MongoDB dozvoljava definisanje pravila validacije šeme (engl. *schema validation*) kada je potrebno ograničiti dozvoljene tipove i strukturu dokumenata [29]. Time se može uspostaviti ravnoteža između fleksibilnosti dokumentnog modela i potrebe za konzistentnošću podataka.

Modelovanje odnosa u MongoDB-u može se realizovati ugnježđivanjem (engl. *embedding*) povezanih podataka u isti dokument ili čuvanjem referenci (engl. *referencing*) ka drugim dokumentima [4, 29]. Ugnježđivanje je pogodno kada se povezani podaci najčešće čitaju zajedno i pripadaju istoj logičkoj celini. Reference su primerenije kada se podaci nezavisno menjaju, koriste na više mesta ili predstavljaju složene relacije. Izbor između ova dva pristupa zavisi od očekivanih obrazaca pristupa podacima, učestalosti izmena i veličine dokumenata.

MongoDB podržava različite vrste upita (engl. *queries*) nad dokumentima, uključujući filtriranje po poljima, rad sa ugnježdenim strukturama, sortiranje i agregaciju podataka [29]. Agregacioni okvir (engl. *aggregation framework*) omogućava formiranje složenijih tokova obrade u kojima se dokumenti filtriraju, grupišu, transformišu i sortiraju kroz niz uzastopnih faza.

Za efikasno izvršavanje upita koriste se indeksi (engl. *indexes*) [4, 29]. Indeks se može formirati nad jednim ili više polja, uključujući polja ugnježdenih dokumenata i nizova. Pravilno odabrani indeksi smanjuju potrebu za pregledom svih dokumenata u kolekciji i ubrzavaju često korišćene upite.

Ova fleksibilnost dokumentnog modela posebno dolazi do izražaja kod podataka kod kojih se pojedinačni zapisi mogu razlikovati po dodatnim atributima. Primer u aplikaciji predstavljaju revizijski zapisi, kod kojih različite vrste aktivnosti zahtevaju različite kontekstualne podatke. Promena statusa događaja može sadržati prethodni i novi status, dok zapis o slanju poruke može sadržati kanal komunikacije i identifikator primaoca. U relacionom modelu ovakve razlike mogu zahtevati veliki broj opcionih kolona ili dodatnih tabela, dok se u dokumentnom modelu mogu predstaviti fleksibilnim objektom sa metapodacima.

U aplikaciji PlanIT MongoDB se koristi za čuvanje poruka i privatnih beleški pored revizijskih zapisa, čija struktura i pravila korišćenja detaljnije su opisani u poglavlju posvećenom aplikaciji.

Komunikacija sa sistemom za upravljanje bazom podataka MongoDB realizovana je pomoću biblioteke `pymongo` [30], bez dodatnog ORM sloja. Na ovaj način serverski deo aplikacije neposredno formira dokumente, izvršava upite i upravlja indeksima.

Primena sistema za upravljanje bazama podataka SQLite i MongoDB u istom sistemu omogućava da se strukturisani podaci sa strogim relacijama odvoje od komunikacije i evidencionih podataka sa promenljivijom strukturom. SQLite obezbeđuje jednostavan relacioni model i integritet osnovnih poslovnih entiteta, dok MongoDB dozvoljava fleksibilno predstavljanje poruka, beleški i revizijskih zapisa. Razlozi za kombinovanje ova dva modela detaljnije su predstavljeni u narednom odeljku posvećenom pristupu *polyglot persistence*.

2.8 Pristup polyglot persistence

Polyglot persistence predstavlja arhitektonski pristup u kojem se u okviru jednog informacionog sistema koristi više različitih tehnologija za skladištenje podataka

[14]. Osnovna ideja ovog pristupa jeste da različite kategorije podataka ne moraju imati iste strukturne osobine, obrasce pristupa ni zahteve u pogledu konzistentnosti. Umesto korišćenja jednog univerzalnog sistema za upravljanje bazom podataka, bira se ona tehnologija koja najbolje odgovara konkretnom tipu podataka i načinu njegove upotrebe.

Ovakav pristup postao je značajan sa razvojem složenih veb i distribuiranih sistema, u kojima se istovremeno obrađuju strogo strukturisani poslovni podaci, zapisi sa promenljivom šemom, poruke, događaji, revizijski tragovi i druge vrste podataka. Relacione baze pogodne su za podatke sa jasno definisanim vezama, ograničenjima i potrebom za transakcionom obradom, dok dokumentne baze omogućavaju fleksibilnije predstavljanje zapisa i jednostavnije prilagođavanje promenama njihove strukture [14, 36].

Jedna od osnovnih prednosti ovog pristupa jeste mogućnost boljeg usklađivanja modela podataka sa zahtevima konkretnog domena, umesto prilagođavanja svih podataka jednom modelu koji možda nije podjednako pogodan za sve slučajeve [36]. Odgovarajući izbor skladišta može pojednostaviti upite, smanjiti potrebu za složenim transformacijama i doprineti efikasnijem izvršavanju operacija.

Ovakva podela može doprineti i jasnijem razdvajanju odgovornosti unutar sistema. Svako skladište dobija precizno određenu ulogu, pri čemu jedan sistem može biti zadužen za osnovne poslovne entitete, a drugi za komunikacione, evidencione ili analitičke podatke. Time se omogućava da se pravila modelovanja i optimizacije primenjuju nezavisno, u skladu sa osobinama konkretne kategorije podataka.

Međutim, korišćenje više baza uvodi dodatnu arhitektonsku i operativnu složenost [36]. Sistem mora upravljati različitim drajverima, načinima povezivanja, strategijama indeksiranja, pravilima validacije i postupcima pravljenja rezervnih kopija.

Poseban izazov predstavlja konzistentnost podataka između različitih baza [36]. Spoljni ključevi, ograničenja i lokalne transakcije mogu se primenjivati unutar jedne baze, ali najčešće ne postoje neposredno između dva različita sistema za skladištenje. Zbog toga se međubazne veze obično predstavljaju logičkim identifikatorima, dok aplikacioni ili servisni sloj proverava postojanje povezanih entiteta i sprovodi poslovna pravila.

Sličan problem javlja se kod operacija koje zahtevaju istovremenu izmenu podataka u više skladišta [36]. Transakcija nad jednom bazom ne garantuje automatski uspeh operacije nad drugom, pa sistem u slučaju delimičnog neuspeha može ostati u nekonzistentnom stanju. Ovaj rizik se ublažava pažljivim projektovanjem servi-

snih operacija i evidentiranjem promena, a u zavisnosti od zahteva sistema može se prihvatiti i eventualna konzistentnost za podatke kod kojih trenutno usklađivanje nije neophodno.

Iz ovih rizika proizilazi i potreba da se, pri projektovanju sistema sa više skladišta, jasno odredi koje skladište predstavlja primarni izvor istine (engl. *source of truth*) za svaki tip podatka – odnosno koja baza sadrži merodavnu, izvornu vrednost na koju se sistem oslanja u slučaju neslaganja. Pojedine vrednosti mogu se denormalizovati i čuvati u drugoj bazi radi efikasnijeg čitanja, ali tada mora biti poznato da je reč o izvedenoj kopiji. Denormalizacija može smanjiti broj dodatnih upita, ali istovremeno povećava rizik da se kopija i izvorni podatak razlikuju nakon naknadnih izmena [36].

U sistemu PlanIT ovaj pristup je primenjen kombinovanjem SQLite i MongoDB sistema za upravljanje bazom podataka, čija je detaljna realizacija predstavljena u poglavlju posvećenom arhitekturi i implementaciji aplikacije.

Pristup polyglot persistence je opravdan kada različite kategorije podataka imaju dovoljno različite zahteve da prednosti specijalizovanih modela nadmašuju dodatnu složenost. Njegova primena nije cilj sama po sebi, već sredstvo za jasnije modelovanje podataka, bolje razdvajanje odgovornosti i efikasnije korišćenje pojedinačnih tehnologija. Uspešna primena zato zahteva pažljivo definisane granice između skladišta, jasan servisni sloj i dosledno dokumentovana pravila konzistentnosti.

Glava 3

Aplikacija PlanIT

Pri projektovanju sistema PlanIT posebna pažnja posvećena je dvema međusobno povezanim oblastima: kontroli pristupa u višekorisničkom okruženju sa više nezavisnih učesnika i evidenciji aktivnosti kao mehanizmu za praćenje izvršenih operacija i promena u sistemu. Pristup podacima i funkcionalnostima određuje se na osnovu korisničke uloge i odnosa korisnika prema konkretnom događaju, dok se značajne aktivnosti beleže u revizijskom dnevniku zajedno sa podacima o izvršiocu, vremenu nastanka i kontekstu u kojem je promena izvršena. Poseban izazov predstavlja praćenje aktivnosti koje nisu neposredna posledica akcije korisnika unutar sistema, već nastaju kao rezultat spoljašnjih događaja, kao što je prijem elektronske pošte od spoljnog saradnika. U takvim slučajevima potrebno je pouzdano povezati primljenu komunikaciju sa odgovarajućim događajem i učesnicima pre njenog evidentiranja u sistemu. Pažnja je posvećena i modelovanju podataka, odnosno načinu na koji se u okviru jedinstvenog sistema organizuju strukturisani podaci, komunikacija i istorija aktivnosti. Arhitektonsko rešenje na taj način objedinjuje koordinaciju učesnika, razmenu informacija, kontrolu pristupa i praćenje aktivnosti, što predstavlja jedan od centralnih aspekata ovog rada.

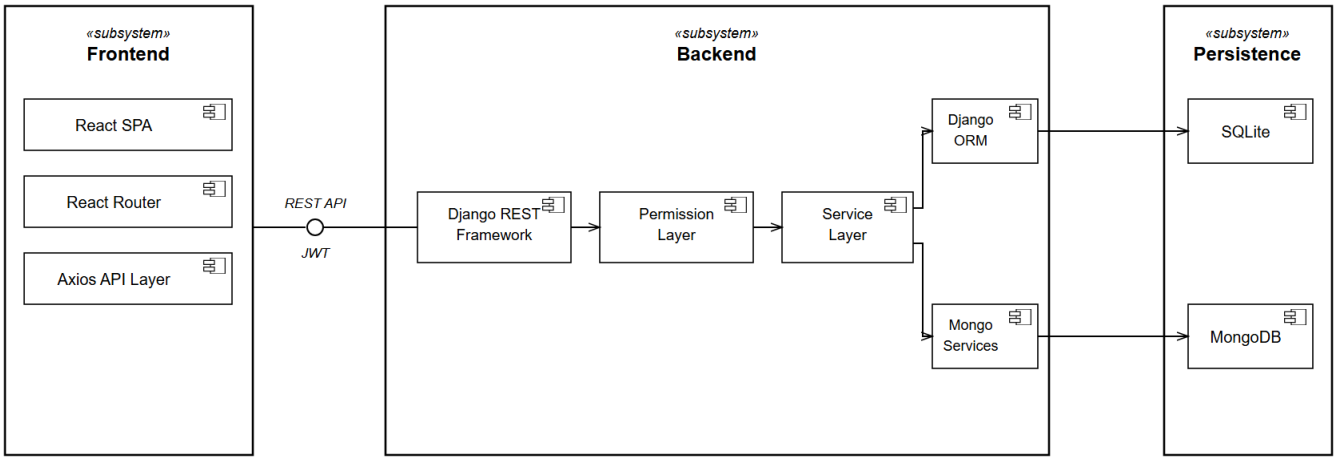
U nastavku glave predstavljeni su arhitektura sistema i model podataka, korisničke uloge i pravila pristupa, ključne poslovne funkcionalnosti, kao i implementacija klijentske i serverske strane aplikacije. Nakon toga opisani su REST API i pristup testiranju, a na kraju su razmotrena ograničenja postojećeg rešenja i mogućnosti njegovog daljeg razvoja.

Izvorni kod serverske i klijentske strane aplikacije, zajedno sa uputstvom za pokretanje, dostupan je u GitHub repozitorijumu [1].

3.1 Arhitektura sistema

Aplikacija PlanIT realizovana je kao veb sistem zasnovan na klijent-server arhitekturi. Klijentska i serverska strana predstavljaju jasno razdvojene celine koje međusobno komuniciraju putem REST API-ja. Ovakva organizacija omogućava nezavisan razvoj korisničkog interfejsa i serverske poslovne logike, kao i jednostavnije održavanje i proširenje pojedinačnih delova sistema.

Na slici 3.1 prikazana je arhitektura sistema i odnosi između njegovih glavnih komponenti. Klijentsku stranu čini React aplikacija napisana u programskom jeziku TypeScript i izgrađena pomoću alata Vite [42]. Ona komunicira sa serverskom stranom preko REST API-ja, pri čemu se za pristup zaštićenim resursima koristi JWT autentifikacija. Serverska strana realizovana je pomoću okvira Django i biblioteke Django REST Framework, dok se podaci čuvaju u relacionoj bazi kojom upravlja sistem SQLite i dokumentnoj bazi kojom upravlja sistem MongoDB.



Slika 3.1: Arhitektura sistema PlanIT

Klijentska strana razvijena je kao jednostranična veb aplikacija (*Single Page Application*), pri čemu se navigacija između prikaza odvija na klijentskoj strani bez ponovnog učitavanja cele stranice. Ovakva organizacija omogućava da korisnik prelazi između različitih funkcionalnih celina aplikacije uz očuvanje zajedničkog konteksta.

React komponente zadužene su za prikaz korisničkog interfejsa i obradu korisničkih akcija, pri čemu su funkcionalnosti organizovane kroz celine kao što su navigacija, prikaz događaja, komunikacija, beleške, kalendar i revizijski dnevnik. Komunikacija sa serverskom stranom izdvojena je u poseban API sloj koji za slanje HTTP zahteva

koristi biblioteku Axios [2], tako da komponente ne moraju neposredno da poznaju detalje slanja zahteva, već koriste unapred definisane funkcije za pristup resursima sistema. Zahtevi i odgovori razmenjuju se u JSON formatu prema krajnjim tačkama sa prefiksom `/api/`, dok serverska strana obrađuje svaki zahtev i vraća odgovarajući HTTP status i sadržaj odgovora.

Za pristup zaštićenim resursima koristi se JWT autentifikacija. Pristupni i osvežavajući token čuvaju se na klijentskoj strani u skladištu `localStorage`. Axios presretač zahteva (engl. *interceptor*) automatski preuzima pristupni token i dodaje ga u `Authorization` zaglavlje zahteva. Ukoliko server vrati odgovor sa statusom 401, klijentska aplikacija pokušava da osveži pristupni token pomoću osvežavajućeg tokena. Nakon uspešnog osvežavanja novi pristupni i osvežavajući token čuvaju se u `localStorage`, a originalni zahtev se ponavlja. U slučaju neuspešnog osvežavanja tokeni se uklanjaju, a korisnik se preusmerava na stranicu za prijavljivanje.

Čuvanje tokena u skladištu `localStorage` pojednostavljuje njihovu upotrebu i omogućava njihovo očuvanje nakon osvežavanja stranice, ali podrazumeva da su tokeni dostupni JavaScript kodu koji se izvršava u okviru aplikacije. Zbog toga eventualno izvršavanje neovlašćenog koda na klijentskoj strani može ugroziti sačuvane tokene. Bezbednija alternativa za čuvanje osvežavajućeg tokena bila bi upotreba `HttpOnly` kolačića, koji nije dostupan JavaScript kodu. Ovaj izbor predstavlja kompromis između jednostavnosti implementacije i nivoa zaštite tokena.

Serverska strana sistema organizovana je prema slojevitoj arhitekturi. Prezentacioni sloj čine API pogledi i serijalizatori. Pogledi primaju HTTP zahteve i prosleđuju obradu odgovarajućim komponentama, dok serijalizatori proveravaju strukturu i ispravnost ulaznih podataka i formiraju podatke koji se vraćaju klijentskoj strani.

Kontrola pristupa realizovana je kroz poseban sloj dozvola. Kod pristupa događajima administrator može da vidi sve događaje, koordinator one koje koordinira, a klijent događaje čiji je vlasnik. Za druge resurse primenjuju se dodatna, uža pravila pristupa, detaljnije opisana u odeljku o autentifikaciji i autorizaciji. Ovakva kontrola pristupa sprovodi se na serverskoj strani, nezavisno od ograničenja postavljenih u korisničkom interfejsu.

Poslovna logika sistema izdvojena je u servisni sloj. API pogledi ne pristupaju bazama neposredno, već delegiraju operacije odgovarajućim servisima. Servisi proveravaju poslovna pravila, izvršavaju operacije nad podacima i, kada je potrebno, kreiraju revizijske zapise. Na primer, servis zadužen za događaje proverava dozvoljene promene statusa, dok servis za komunikaciju proverava pravila vezana za slanje

poruka. Ovakvo izdvajanje omogućava bolje razdvajanje odgovornosti: API sloj ostaje usmeren na prijem i formiranje zahteva, dok pravila domena ostaju na jednom mestu, čime se smanjuje dupliranje koda, olakšava testiranje i omogućava ponovno korišćenje istih operacija u različitim delovima sistema.

Poseban deo serverske arhitekture predstavlja posrednički sloj (engl. *middleware*). U sistemu PlanIT definisane su tri prilagođene komponente ovog sloja. Klasa `SimpleCorsMiddleware` zadužena je za primenu CORS (engl. *Cross-Origin Resource Sharing*) pravila. Ona proverava poreklo zahteva, za dozvoljene izvore dodaje odgovarajuća CORS zaglavlja i obrađuje prethodne provere zahteva (engl. *preflight check*) koji koriste HTTP metod `OPTIONS`. Klasa `ForcePasswordChangeMiddleware` ograničava pristup funkcionalnostima korisnicima koji još nisu promenili jednokratnu lozinku. Evidentiranje osnovnih podataka o HTTP zahtevima i odgovorima obavlja klasa `RequestLoggingMiddleware`. Na ovaj način pravila koja se primenjuju na veći broj krajnjih tačaka izdvojena su iz pojedinačnih pogleda i primenjuju se centralizovano.

Servisni sloj je ujedno odgovoran i za evidentiranje značajnih promena u revizijskom dnevniku. Većina operacija izmene podataka, kao što su pravljenje korisnika i događaja, promena statusa, upravljanje saradnicima, porukama i beleškama, povezana je sa odgovarajućim revizijskim zapisom. Poseban slučaj predstavlja sistemska poruka koja se automatski kreira nakon promene statusa događaja: sama promena statusa se evidentira, dok generisanje sistemske poruke nema zaseban revizijski zapis.

Za čuvanje podataka koriste se relacioni sistem za upravljanje bazama podataka SQLite i dokumentni sistem za upravljanje bazama podataka MongoDB, u skladu sa pristupom *polyglot persistence* predstavljenim u prethodnom poglavlju. Za pristup relacionoj bazi podataka koristi se Django ORM i u njoj se čuvaju podaci o korisnicima, događajima, spoljnim saradnicima i vezama između događaja i saradnika, dok se u dokumentnoj bazi podataka čuvaju poruke, privatne beleške i revizijski zapisi.

Veze između ova dva skladišta ostvaruju se logičkim identifikatorima, a njihovu ispravnost proverava servisni sloj. Pojedina polja u MongoDB dokumentima čuvaju se denormalizovano kako bi se smanjila potreba za dodatnim pristupom relacionoj bazi. Razlozi za ovakvu podelu podataka detaljnije su razmotreni u odeljku 2.8 o *polyglot persistence* pristupu.

Ovakva organizacija uspostavlja jasne granice između korisničkog interfejsa, kontrole pristupa, poslovne logike i skladištenja podataka. Naredni odeljci detaljnije

prikazuju pojedine delove ove arhitekture i njihovu ulogu u sistemu.

Konzistentnost između skladišta podataka

Korišćenje relacione i dokumentne baze podataka u okviru istog sistema zahteva razmatranje konzistentnosti podataka između dva nezavisna skladišta. MongoDB dokumenti čuvaju identifikatore pojedinih entiteta iz relacione baze, ali između njih ne postoje referencijalna ograničenja kakva postoje između relacionih tabela. Zbog toga brisanje entiteta iz relacione baze ne dovodi automatski do brisanja ili izmene povezanih MongoDB dokumenata. Takvi dokumenti ostaju sa sačuvanim identifikatorima, a denormalizovani podaci, poput korisničkog imena, omogućavaju da deo istorijskog konteksta ostane dostupan i nakon brisanja povezanog entiteta.

Dodatni izazov predstavljaju operacije koje zahtevaju upis u oba skladišta. Transakcije relacione baze ne obuhvataju MongoDB operacije, pa sistem ne obezbeđuje atomsku transakciju nad oba skladišta. Redosled izvršavanja operacija i transakcije relacionog dela omogućavaju poništavanje relacionih izmena u pojedinim slučajevima neuspeha, ali jednom uspešno izvršen MongoDB upis ne može biti automatski poništen naknadnim vraćanjem (engl. *rollback*) relacione transakcije. Zbog toga je pri delimičnom neuspehu moguće privremeno ili trajno odstupanje između stanja u dva skladišta. Stroža garancija konzistentnosti zahtevala bi dodatni mehanizam koordinacije.

3.2 Korisničke uloge i model pristupa

Sistem PlanIT zasniva kontrolu pristupa na tri osnovne korisničke uloge: administratoru, koordinatoru i klijentu. Svaka uloga obuhvata skup dozvoljenih aktivnosti i ograničenja koja proizlaze iz odgovornosti korisnika u procesu organizacije događaja. Spoljni saradnik predstavlja zaseban domenski entitet, ali nije korisnik sistema, ne poseduje nalog i ne pristupa aplikaciji neposredno. Komunikacija sa spoljnim saradnicima ostvaruje se posredstvom koordinatora.

Administrator je zadužen za administrativne funkcionalnosti i upravljanje osnovnom strukturom sistema. Njegove aktivnosti obuhvataju pravljenje korisničkih naloga sa jednokratnom lozinkom, kao i pravljenje događaja i njihovo povezivanje sa odgovarajućim klijentom i koordinatorom. Administrator ima pristup svim doga-

đajima, administrativnom kalendaru i globalnom revizijskom dnevniku, zbog čega njegova uloga ima administrativni i nadzorni karakter.

Koordinator je odgovoran za neposredno vođenje događaja koji su mu dodeljeni. Može da pregleda podatke o tim događajima, menja njihov status u skladu sa dozvoljenim tranzicijama, komunicira sa klijentima i upravlja spoljnim saradnicima. Upravljanje saradnicima obuhvata pravljenje njihovih zapisa, povezivanje sa događajem i uklanjanje postojeće veze. Koordinator može da šalje interne poruke klijentu i eksterne poruke saradnicima koji su prethodno povezani sa odgovarajućim događajem.

Klijent predstavlja vlasnika događaja i pristupa isključivo događajima povezanim sa njegovim korisničkim nalogom. Može da pregleda osnovne podatke o događaju, razmenjuje interne poruke sa koordinatorom i vodi privatne beleške. Pristup beleškama dodatno je ograničen autorstvom, pa korisnik može da vidi, menja i briše samo sopstvene beleške.

U sistemu PlanIT kontrola pristupa zasniva se prvenstveno na RBAC modelu, dopunjenom elementima ReBAC pristupa predstavljenim u delu 2.5. Osnovne dozvole određene su ulogama administratora, koordinatora i klijenta, dok se za pojedine operacije dodatno proverava odnos korisnika prema konkretnom resursu. Na taj način korisnička uloga određuje opšti skup dostupnih funkcionalnosti, a pristup pojedinačnim objektima ograničava se u skladu sa odnosima definisanim u domenskom modelu.

Primena čistog RBAC modela ne bi bila dovoljna za izražavanje svih pravila pristupa u sistemu PlanIT, jer dva korisnika iste uloge mogu imati različita prava nad konkretnim resursom. ABAC model omogućio bi definisanje pravila na osnovu većeg broja atributa korisnika, resursa i konteksta, ali takav nivo opštosti nije bio potreban za postojeći skup zahteva. Potpuno oslanjanje na ReBAC pristup takođe ne bi odgovaralo strukturi sistema, pošto značajan deo dozvola neposredno proizlazi iz jasno definisanih korisničkih uloga. Zbog toga je primenjen kombinovani pristup u kojem RBAC određuje osnovne dozvole, dok objektne provere ograničavaju skup resursa nad kojima se te dozvole mogu primeniti.

Postojeći model pristupa odgovara trenutnom skupu od tri korisničke uloge. Uvođenje dodatne uloge zahtevalo bi proširenje modela korisnika i pregled postojećih pravila na mestima na kojima se uloga eksplicitno proverava. Bilo bi potrebno odrediti dostupne funkcionalnosti nove uloge, način njenog pristupa pojedinačnim događajima i drugim resursima, kao i eventualne nove odnose sa postojećim entitetima.

U zavisnosti od njenih odgovornosti, izmene bi obuhvatile klase dozvola, filtriranje podataka i odgovarajuće prikaze na klijentskoj strani. Sa povećanjem broja uloga i odnosa raste i broj pravila koja je potrebno međusobno uskladiti, zbog čega bi kod složenijeg modela pristupa bilo opravdano razmotriti veću centralizaciju autorizacione logike.

Pravila pristupa sprovode se na serverskoj strani. Ograničenja u korisničkom interfejsu, kao što su skrivanje određenih stranica ili dugmadi, unapređuju korisničko iskustvo, ali ne predstavljaju osnovni bezbednosni mehanizam. Svaki zahtev prema zaštićenoj API krajnjoj tački prolazi kroz proveru autentifikacije, korisničke uloge i, kada je to potrebno, prava pristupa konkretnom događaju ili podatku. Time se pravila pristupa primenjuju nezavisno od ograničenja prikazanih u korisničkom interfejsu.

Za realizaciju pravila pristupa na serverskoj strani definisano je sedam prilagođenih klasa dozvola. Četiri klase odluku donose prvenstveno na osnovu korisničke uloge: `IsAdmin`, `IsCoordinator`, `IsCoordinatorOrAdmin` i `IsCoordinatorOrClient`. Njima se ograničava pristup funkcionalnostima koje su namenjene određenim kategorijama korisnika, kao što su administratorske operacije, rad sa spoljnim saradnicima, pregled korisnika i pristup notifikacijama.

Preostale tri klase uključuju i proveru odnosa korisnika prema konkretnom resursu. Klasa `CanAccessEvent` određuje pristup događajima i povezanim porukama u zavisnosti od toga da li je korisnik administrator, koordinator dodeljen događaju ili klijent povezan sa tim događajem. Klasa `IsEventParticipant` ograničava pristup funkcionalnostima namenjenim učesnicima konkretnog događaja, kao što su privatne beleške i označavanje notifikacija kao pročitanih, dok `IsEventCoordinator` dozvoljava određene operacije, poput promene statusa događaja, samo koordinatoru kome je taj događaj dodeljen. Na taj način deo pravila pristupa može da se izrazi samo ulogom korisnika, dok je za operacije nad konkretnim objektima neophodna i provera odnosa korisnika prema tom objektu.

U tabeli 3.1 prikazan je sažet pregled ključnih dozvola prema korisničkim ulogama. Izrazi *„samo dodeljeni“*, *„samo sopstveni“* i *„samo sopstvena“* označavaju dodatno ograničenje pristupa na nivou konkretnog događaja ili zapisa.

Tabela 3.1 pokazuje da korisničke uloge u sistemu PlanIT nisu organizovane hijerarhijski, odnosno da uloga sa širim administrativnim ovlašćenjima ne obuhvata automatski dozvole ostalih uloga. Razlog za takvu organizaciju jeste razdvajanje administrativnih i operativnih odgovornosti. Administrator upravlja sistemom i ima

Tabela 3.1: Pregled dozvola prema korisničkim ulogama

Akcija	**Administrator**	**Koordinator**	**Klijent**
Kreiranje korisnika	da	ne	ne
Kreiranje događaja	da	ne	ne
Pregled događaja	svi događaji	samo dodeljeni	samo sopstveni
Promena statusa događaja	ne	samo dodeljeni	ne
Kreiranje spoljnog saradnika	ne	da	ne
Povezivanje ili uklanjanje saradnika sa događajem	ne	samo dodeljeni	ne
Slanje internih poruka	ne	samo dodeljeni	samo sopstveni
Slanje eksternih poruka	ne	samo dodeljeni	ne
Kreiranje privatne beleške	ne	samo dodeljeni	samo sopstveni
Izmena ili brisanje beleške	ne	samo sopstvena	samo sopstvena
Primanje notifikacija	ne	da	da
Pristup revizijskom dnevniku	globalni pristup	ne	ne
Pregled administrativnog kalendara	da	ne	ne

nadzornu ulogu, ali ne učestvuje neposredno u vođenju pojedinačnih događaja.

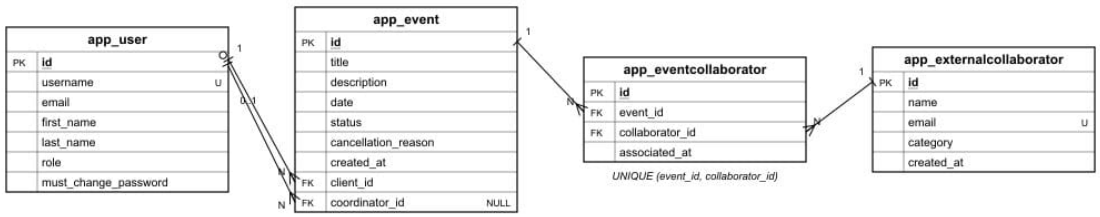
Operativne aktivnosti povezane sa konkretnim događajem poverene su koordinatoru koji je tom događaju dodeljen. Zbog toga administrator, iako ima uvid u sve događaje, ne menja njihov status, ne upravlja saradnicima povezanim sa događajem i ne učestvuje u komunikaciji sa klijentima i spoljnim saradnicima. Ovakvom podelom administrativne i nadzorne odgovornosti ostaju odvojene od aktivnosti koje pripadaju neposrednoj koordinaciji događaja.

3.3 Model podataka

Model podataka aplikacije PlanIT zasniva se na relacionom delu sistema koji obuhvata korisnike, događaje, spoljne saradnike i eksplicitnu vezu između događaja i saradnika. Ovi entiteti predstavljaju jezgro domenskog modela, jer opisuju osnovne aktere sistema, podatke o događajima i njihove međusobne odnose. Relacioni pristup izabran je zbog potrebe za jasno definisanim vezama, ograničenjima jedinstvenosti i očuvanjem referencijalnog integriteta.

Na slici 3.2 prikazan je dijagram tabela i njihovih veza u relacionom delu sistema. Prikazane tabele formirane su na osnovu Django modela `User`, `Event`, `ExternalCollaborator`

i `EventCollaborator`, a dijagram prikazuje i kardinalnosti veza između njih.



Slika 3.2: Dijagram tabela i njihovih veza relacionog dela sistema PlanIT

Osnovna namena modela i njihova najvažnija polja sažeti su u tabeli 3.2, dok su ključna ograničenja i ponašanje veza pri brisanju prikazani u tabeli 3.3.

Tabela 3.2: Osnovne osobine Django modela domenskog sloja

Model	**Namena**	**Najvažnija polja**
`User`	Autentifikovani korisnik sistema	`username`, `email`, `role`, `must_change_password`
`Event`	Događaj kojim se upravlja u sistemu	`title`, `description`, `date`, `status`, `cancellation_reason`, `created_at`
`ExternalCollaborator`	Spoljni saradnik bez korisničkog naloga	`name`, `email`, `category`, `created_at`
`EventCollaborator`	Veza između događaja i spoljnog saradnika	`event`, `collaborator`, `associated_at`

Tabela 3.3: Ključna ograničenja i ponašanje veza Django modela

Polje ili veza	**Ograničenje / ponašanje**
`User.username`	Korisničko ime je jedinstveno.
`ExternalCollaborator.email`	Adresa elektronske pošte je jedinstvena.
`Event.client`	Brisanjem klijenta brišu se i njegovi događaji (`CASCADE`).
`Event.coordinator`	Brisanjem koordinatora događaj se zadržava, a vrednost polja koordinatora postavlja se na `NULL` (`SET_NULL`).
`EventCollaborator.event`	Brisanjem događaja brišu se pripadajuće veze sa saradnicima (`CASCADE`).
`EventCollaborator.collaborator`	Brisanjem saradnika brišu se njegove veze sa događajima (`CASCADE`).
`EventCollaborator`	Par `(event, collaborator)` mora biti jedinstven.

Korisnici

Korisnici sistema predstavljeni su modelom `User`, koji proširuje Django klasu `AbstractUser`. Time su zadržana standardna autentifikaciona polja i ponašanje, dok su dodatno uvedena polja `role` i `must_change_password`. Polje `role` razlikuje administratore, koordinatore i klijente, a `must_change_password` označava obavezu promene jednokratne lozinke pri prvom prijavljivanju.

Jedan prošireni korisnički model koristi se za sve tri uloge zato što one dele isti skup autentifikacionih podataka i isti tok prijavljivanja. Njihova osnovna razlika ne nalazi se u strukturi podataka, već u skupu dozvoljenih operacija. Ovakvo rešenje izbegava dupliranje istih podataka u više modela i omogućava da se kontrola pristupa zasniva na vrednosti polja `role`. Pomoćna svojstva modela, kao što su `is_coordinator`, `is_client` i `is_admin_role`, pojednostavljuju proveru uloge u servisnom sloju i klasama dozvola.

Model `User` povezan je sa modelom `Event` kroz uloge klijenta i koordinatora. Polje `client` određuje vlasnika događaja, dok polje `coordinator` označava korisnika zaduženog za njegovo vođenje. Ponašanje ovih veza pri brisanju prikazano je u tabeli 3.3.

Događaji

Model `Event` predstavlja centralni entitet domenskog modela. Sadrži osnovne podatke o događaju, uključujući naziv, opis, datum održavanja, status, razlog otkazivanja i vreme kreiranja. Njegova osnovna polja prikazana su u tabeli 3.2. Svaki događaj povezan je sa klijentom i koordinatorom, pri čemu obe veze vode ka modelu `User`.

Klijent predstavlja vlasnika događaja i njegova veza sa događajem je obavezna. U slučaju brisanja klijenta primenjuje se pravilo `CASCADE`, zbog čega se zajedno sa korisničkim nalogom uklanja i povezani događaj. Ovakvo ponašanje odgovara modelu pristupa u kojem klijent pristupa događajima upravo na osnovu veze definisane poljem `client`.

Veza sa koordinatorom koristi pravilo `SET_NULL`. Kada se korisnički nalog koordinatora obriše, događaj se zadržava u sistemu, dok se vrednost polja `coordinator` postavlja na `NULL`. Događaj tada ostaje bez dodeljenog koordinatora. Postojeća implementacija ne sadrži funkcionalnost za naknadnu dodelu novog koordinatora postojećem događaju.

Status događaja predstavljen je enumerisanim poljem sa vrednostima `PLANNED`, `CONFIRMED`, `DONE` i `CANCELLED`. Dozvoljene promene između ovih stanja nisu proizvoljne, već su definisane poslovnim pravilima sistema. U ovoj sekciji status se posmatra samo kao deo strukture modela, dok će dijagram stanja i pravila dozvoljenih prelaza biti prikazani u odeljku 3.5, posvećenom poslovnoj logici sistema.

Spoljni saradnici

Model `ExternalCollaborator` predstavlja spoljne saradnike sistema. Sadrži naziv saradnika, jedinstvenu adresu elektronske pošte, kategoriju i vreme kreiranja zapisa. Kategorija je enumerisano polje sa vrednostima `FLORIST`, `PHOTOGRAPHER`, `BAND`, `DJ`, `CATERING`, `DECORATION`, `BALLOONS`, `VENUE`, `CAKE` i `OTHER`. Ove vrednosti omogućavaju klasifikaciju saradnika prema vrsti usluge koju pružaju.

Spoljni saradnik nije korisnik sistema i ne poseduje nalog za prijavljivanje. Model zato ne nasleđuje `AbstractUser` i ne sadrži autentifikaciona polja karakteristična za korisnike sistema. Odvajanje ovog entiteta od korisničkog modela jasno razdvaja autentifikovane aktere sistema od spoljnih poslovnih partnera koji nemaju neposredan pristup aplikaciji.

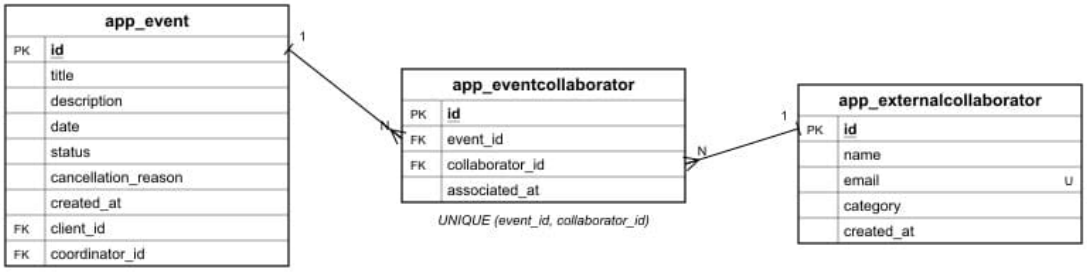
Adresa elektronske pošte je jedinstvena, čime se sprečava dodavanje više zapisa

sa istom adresom elektronske pošte. Osnovne osobine ovog modela navedene su u tabeli 3.2, dok su njegova ograničenja i veze prikazani u tabeli 3.3.

Veza događaj–saradnik

Odnos između događaja i spoljnih saradnika ima kardinalnost više-prema-više. Jedan događaj može uključivati više saradnika, dok isti saradnik može biti angažovan na više različitih događaja. Umesto implicitno generisane veze, ovaj odnos realizovan je eksplicitnim veznim (*through*) modelom `EventCollaborator`.

Na slici 3.3 izdvojen je deo dijagrama tabela i njihovih veza koji prikazuje odnos između događaja, saradnika i veznog entiteta.



Slika 3.3: Veza između događaja i spoljnih saradnika

Model `EventCollaborator` sadrži spoljne ključeve `event` i `collaborator`, kao i polje `associated_at`, koje beleži trenutak povezivanja saradnika sa događajem. Eksplicitni vezni model omogućava da se, pored uspostavljanja veze između odgovarajućih entiteta, čuvaju i dodatni podaci koji tu vezu opisuju.

Eksplicitni model omogućava i neposredno adresiranje veze u servisnom sloju. Operacije povezivanja, provere postojećeg odnosa i uklanjanja saradnika izvršavaju se nad zapisima ovog modela, dok se odgovarajuće aktivnosti mogu evidentirati u revizijskom dnevniku. Na taj način veza događaj–saradnik predstavlja eksplicitno modelovan odnos sa sopstvenim atributima i pravilima.

Nad parom `event` i `collaborator` definisano je ograničenje jedinstvenosti, čime se obezbeđuje da isti saradnik ne može biti povezan sa istim događajem više puta. Pri brisanju događaja ili saradnika primenjuje se pravilo `CASCADE` nad veznim zapisima, pa se automatski uklanjaju veze sa entitetom koji više ne postoji, dok se ostali događaji i saradnici zadržavaju.

3.4 Dokumentni model

Dokumentni deo modela podataka u sistemu PlanIT zasniva se na bazi podataka kojom upravlja sistem MongoDB i namenjen je čuvanju poruka, revizijskih zapisa i privatnih beleški, čija struktura i način korišćenja razlikuju se od osnovnih relacionih entiteta. Dokumenti iz ovih kolekcija povezuju se sa entitetima iz relacione baze putem logičkih identifikatora, dok se provera njihove ispravnosti sprovodi u servisnom sloju.

Poruke

Kolekcija `messages` čuva poruke povezane sa konkretnim događajem. Svaki dokument sadrži identifikator događaja, kanal komunikacije, sadržaj poruke, vreme slanja i status pročitanosti, kao i smer poruke (engl. `direction`) koji određuje da li je poruku uputio korisnik sistema (`OUTBOUND`) ili je reč o poruci primljenoj od spoljnog saradnika (`INBOUND`). Za poruke koje potiču od korisnika sistema čuva se identifikator pošiljaoca, kao i njegovo korisničko ime u denormalizovanom obliku, čime se izbegava dodatni upit prema relacionoj bazi prilikom prikaza poruke; kod primljenih poruka ovo polje ostaje prazno, dok se pošiljalac umesto toga identifikuje preko polja `sender_external_id`, koje upućuje na odgovarajućeg spoljnog saradnika. Dokument sadrži i polje `metadata`, namenjeno dodatnim podacima čija struktura zavisi od vrste poruke. Na primer, kod eksterne komunikacije ono može sadržati predmet poruke, dok kod interne komunikacije takvi podaci nisu potrebni.

Sistem razlikuje tri kanala komunikacije. Kanal `INTERNAL` koristi se za razmenu poruka između koordinatora i klijenta u okviru događaja. Kanal `EXTERNAL` koristi se za komunikaciju koordinatora sa spoljnim saradnicima povezanim sa događajem. Pored poruka koje koordinator upućuje saradniku, sistem prima i beleži i odgovore koje saradnik pošalje elektronskom poštom. Kanal `SYSTEM` namenjen je automatski generisanim sistemskim porukama, kao što su obaveštenja koja nastaju nakon promene statusa događaja. Status pročitanosti čuva se u polju `is_read`, a nepročitane interne i sistemske poruke predstavljaju osnovu za prikaz notifikacija korisniku.

Revizijski dnevnik

Evidencija aktivnosti predstavlja jedan od važnih aspekata projektovanja sistema PlanIT i neposredno je povezana sa istraživačkim pitanjem ovog rada, koje obuhvata

modelovanje mehanizama kontrole pristupa i evidencije aktivnosti u višekorisničkom sistemu sa više nezavisnih učesnika. U tu svrhu koristi se revizijski dnevnik, čiji se zapisi čuvaju u MongoDB kolekciji `audit_logs`. Njegova uloga je da omogući praćenje značajnih operacija i promena nastalih tokom korišćenja sistema.

Svaki revizijski zapis sadrži podatke o korisniku koji je izvršio operaciju, tipu akcije, ciljanom objektu, povezanom događaju, vremenu izvršenja i dodatnom kontekstu u okviru polja `metadata`. Na primer, prilikom promene statusa događaja formira se zapis koji identifikuje korisnika koji je izvršio promenu i događaj na koji se ona odnosi, beleži vreme izvršenja i sadrži podatke o prethodnom i novom statusu. Na osnovu takvog zapisa moguće je naknadno utvrditi ko je izvršio promenu, nad kojim događajem, kada je ona nastala i šta je promenjeno.

Korisničko ime izvršioca operacije čuva se denormalizovano u samom revizijskom zapisu, čime se izbegavaju dodatni upiti prema relacionoj bazi prilikom prikaza istorije aktivnosti. Polje `metadata` omogućava čuvanje dodatnih podataka čija struktura zavisi od vrste evidentirane aktivnosti. Tako, na primer, zapis o promeni statusa može da sadrži prethodni i novi status, dok zapis o slanju elektronske pošte spoljnom saradniku može da sadrži podatke o primaocu i predmetu poruke.

Revizijski dnevnik obuhvata aktivnosti povezane sa upravljanjem korisnicima, događajima, spoljnim saradnicima, komunikacijom i privatnim beleškama. Evidentiraju se pravljenje korisničkih naloga, promena lozinke, prijavljivanje i odjavljivanje korisnika, pravljenje događaja i promena njegovog statusa, pravljenje spoljnih saradnika i njihovo povezivanje sa događajem ili uklanjanje postojeće veze, kao i pravljenje, izmena i brisanje privatnih beleški. U oblasti komunikacije evidentiraju se slanje internih poruka, slanjeelektronske pošte spoljnim saradnicima, prijem njihovih odgovora putem elektronske pošte, kao i aktivnosti povezane sa otvaranjem poruka i notifikacija. Na taj način revizijski dnevnik objedinjuje podatke o značajnim aktivnostima koje nastaju u različitim funkcionalnim delovima sistema.

Revizijski zapisi imaju *append-only* karakter. Servisni sloj nove aktivnosti evidentira dodavanjem novih dokumenata i ne sadrži operacije za izmenu ili brisanje postojećih revizijskih zapisa. Ovo svojstvo nije garantovano mehanizmom samog skladišta podataka, već predstavlja pravilo primenjeno na nivou aplikacije. Ovakav pristup omogućava očuvanje hronološkog traga evidentiranih aktivnosti i naknadni pregled promena koje su nastale tokom korišćenja sistema.

Beleške

Dokument u kolekciji `notes` predstavlja privatnu belešku koju korisnik vezuje za konkretan događaj. Svaki dokument sadrži identifikator događaja, identifikator i korisničko ime autora u denormalizovanom obliku, sadržaj beleške, kao i vreme kreiranja i poslednje izmene. Za razliku od poruka i revizijskih zapisa, beleške mogu biti naknadno izmenjene ili obrisane, ali samo u skladu sa pravilima pristupa sistema.

Privatnost beleški zasniva se na autorstvu. Korisnik može da pristupi samo beleškama čiji se `author_id` poklapa sa njegovim identifikatorom, a ista provera primenjuje se i pri izmeni i brisanju. Beleške zato predstavljaju lični sadržaj korisnika u okviru konkretnog događaja, a ne sadržaj zajednički svim učesnicima. Dodavanje, izmena i brisanje beleški nisu dozvoljeni nakon što događaj dostigne terminalno stanje `DONE` ili `CANCELLED`.

Dokumentni model tako dopunjuje relacioni deo sistema podacima sa drugačijim načinom korišćenja: poruke podržavaju različite komunikacione kanale, revizijski zapisi čuvaju istoriju aktivnosti, dok beleške predstavljaju privatni sadržaj korisnika vezan za događaj.

3.5 Poslovna logika sistema

Poslovna logika sistema PlanIT određuje pravila prema kojima se izvršavaju ključne operacije nad događajima, komunikacijom, saradnicima i privatnim beleškama. Ova pravila definišu koje su akcije dozvoljene u određenom trenutku i pod kojim uslovima mogu biti izvršene, čime se održava konzistentnost podataka i sprečavaju operacije koje nisu u skladu sa životnim ciklusom događaja.

Kreiranje događaja

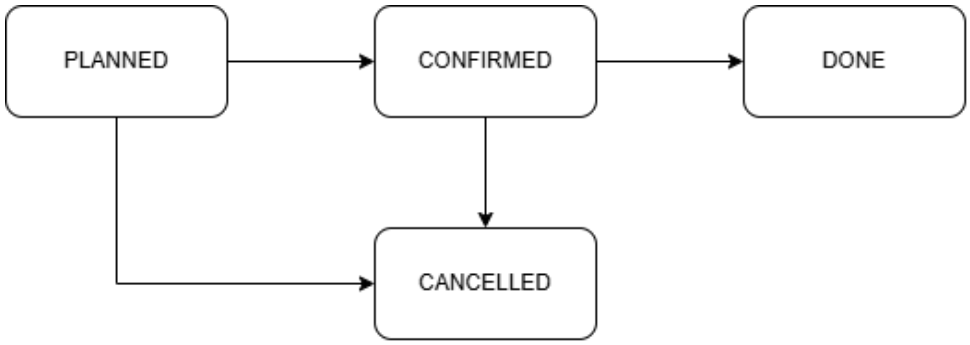
Novi događaj može da kreira administrator. Prilikom kreiranja neophodno je definisati osnovne podatke o događaju i povezati ga sa odgovarajućim klijentom i koordinatorom. Klijent predstavlja vlasnika događaja, dok je koordinator odgovoran za njegovo vođenje.

Sistem dodatno proverava da li korisnici koji se dodeljuju događaju imaju odgovarajuće uloge, čime se sprečava povezivanje proizvoljnog korisnika u svojstvu klijenta ili koordinatora. Novi događaj dobija početni status `PLANNED`, iz kojeg se njegov životni ciklus dalje razvija prema definisanim pravilima tranzicije.

Promena statusa događaja

Status događaja predstavlja važno poslovno svojstvo sistema, jer određuje trenutnu fazu njegovog životnog ciklusa i utiče na dostupnost drugih funkcionalnosti. Sistem razlikuje četiri stanja: `PLANNED`, `CONFIRMED`, `DONE` i `CANCELLED`.

Dozvoljene tranzicije prikazane su na slici 3.4.



Slika 3.4: Dijagram stanja događaja

Svaki novokreirani događaj nalazi se u stanju `PLANNED`. Iz ovog stanja događaj može biti potvrđen, čime prelazi u `CONFIRMED`, ili otkazan i preći u `CANCELLED`. Iz stanja `CONFIRMED` moguć je prelaz u `DONE`, kada se događaj smatra završenim, ili u `CANCELLED`, ukoliko dođe do njegovog otkazivanja. Stanja `DONE` i `CANCELLED` predstavljaju terminalna stanja. Nakon što događaj dostigne jedno od njih, dalje promene statusa nisu dozvoljene. Time se sprečava ponovno aktiviranje već završenog ili otkazanog događaja i očuvava konzistentan tok njegovog životnog ciklusa.

Prelazak u stanje `CANCELLED` zahteva navođenje razloga otkazivanja, čime se uz promenu statusa čuva i razlog zbog kojeg je događaj otkazan. Promena statusa evidentira se u revizijskom dnevniku zajedno sa prethodnim i novim statusom.

Promena statusa utiče i na druge funkcionalnosti sistema – dostizanjem terminalnog stanja ograničava se dalja komunikacija i rad sa privatnim beleškama, kao što je detaljnije opisano u narednim odeljcima. Status događaja zato ne predstavlja samo informativno polje, već neposredno utiče na ponašanje sistema.

Komunikacija

Komunikacija u sistemu odvija se kroz tri različita kanala: interni, eksterni i sistemski. Pravila za svaki od njih razlikuju se prema vrsti učesnika i načinu na koji poruka nastaje.

Interne poruke koriste se za komunikaciju između klijenta i koordinatora u okviru konkretnog događaja. Poruku nije moguće poslati proizvoljnom korisniku sistema, već primalac mora biti klijent ili koordinator tog događaja. Time se komunikacija zadržava u okviru jasno definisanog konteksta događaja i ograničava na korisnike koji su sa njim neposredno povezani.

Eksterne poruke namenjene su komunikaciji koordinatora sa spoljnim saradnicima. Poruka može biti poslata samo saradniku koji je prethodno povezan sa konkretnim događajem. Eksterna komunikacija je dvosmerna: saradnik odgovara elektronskom poštom, a sistem primljeni odgovor povezuje sa odgovarajućim događajem i saradnikom na osnovu adrese pošiljaoca i oznake događaja sadržane u prepisci, nakon čega se poruka evidentira na isti način kao i poruke koje šalje koordinator.

Sistemske poruke ne nastaju neposrednom akcijom korisnika, već ih automatski generiše sistem kao posledicu promene statusa događaja. Na ovaj način klijent dobija obaveštenje o promeni statusa, uključujući i prelazak događaja u završno ili otkazano stanje.

Administrator ne može da šalje interne ni eksterne poruke, ali mu je omogućen pristup postojećim porukama isključivo u režimu čitanja.

Mogućnost slanja novih internih i eksternih poruka zavisi i od statusa događaja. Prilikom prelaska događaja u terminalno stanje `DONE` ili `CANCELLED`, sistem generiše odgovarajuću sistemsku poruku kojom se klijent obaveštava o promeni statusa. Nakon što ova tranzicija bude završena, dalje slanje internih i eksternih poruka nije dozvoljeno, niti nastaju nove sistemske poruke kroz postojeće funkcionalnosti sistema.

Upravljanje spoljnim saradnicima

Spoljni saradnici mogu biti angažovani na jednom ili više događaja. Koordinator može da evidentira nove spoljne saradnike nezavisno od konkretnog događaja; povezivanje i uklanjanje veze sa događajem ograničeno je na koordinatora kome je taj događaj dodeljen. Administrator nema ovlašćenje za upravljanje saradnicima.

Jedan saradnik može učestvovati na više različitih događaja, ali isti saradnik ne može biti više puta povezan sa istim događajem. Time se sprečavaju duplirani zapisi i obezbeđuje jednoznačna povezanost.

Postojeća veza može biti uklonjena kada saradnik više nije angažovan na određenom događaju. Uklanjanjem veze ne briše se sam saradnik iz sistema, već on ostaje evidentiran i može biti povezan sa drugim događajima.

Povezanost saradnika sa događajem predstavlja i preduslov za eksternu komunikaciju, jer koordinator može poslati poruku samo saradniku koji je prethodno povezan sa tim događajem.

Privatne beleške

Privatne beleške omogućavaju korisnicima da vode sopstvene zapise povezane sa konkretnim događajem. Za razliku od poruka, beleške nisu namenjene komunikaciji između učesnika, već predstavljaju lični sadržaj autora.

Beleška ne poseduje poseban naslov, već se u celosti sastoji od slobodnog tekstualnog sadržaja koji unosi autor. Za razliku od poruka i revizijskih zapisa, beleške mogu biti naknadno izmenjene ili obrisane.

Osnovno pravilo pristupa zasniva se na autorstvu. Korisnik može da vidi, menja i briše samo beleške koje je sam kreirao. Čak i kada više korisnika ima pristup istom događaju, njihove privatne beleške ostaju međusobno odvojene.

Mogućnost rada sa beleškama dodatno zavisi od stanja događaja. Dok je događaj aktivan, korisnik može da kreira nove beleške i menja ili briše postojeće. Nakon prelaska u stanje `DONE` ili `CANCELLED`, dodavanje, izmena i brisanje beleški više nisu dozvoljeni. Time se sprečavaju naknadne promene sadržaja vezanog za završen ili otkazan događaj.

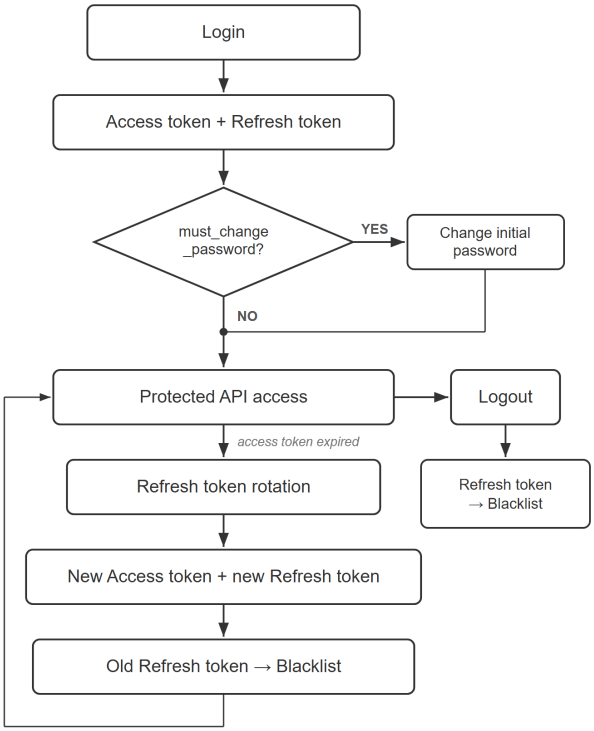
Opisana pravila povezuju životni ciklus događaja sa komunikacijom, saradnicima i privatnim beleškama, čime se obezbeđuje dosledno ponašanje sistema u različitim fazama organizacije događaja.

3.6 Autentifikacija i autorizacija

Autentifikacija u sistemu PlanIT realizovana je primenom JWT mehanizma, uz dodatnu podršku za naloge sa obaveznom promenom početne lozinke. Nakon uspešne prijave, serverska strana vraća pristupni i osvežavajući token zajedno sa osnovnim podacima o korisniku. Pristupni token koristi se za pristup zaštićenim API krajnjim tačkama, dok se osvežavajući token koristi za izdavanje novog pristupnog tokena nakon njegovog isteka.

Tok autentifikacije u aplikaciji prikazan je na slici 3.5.

Prijavljivanje korisnika izvršava se slanjem korisničkog imena i lozinke autentifikacionoj krajnjoj tački. Poseban slučaj predstavljaju korisnici čiji je nalog kreiran sa



Slika 3.5: Tok autentifikacije u sistemu PlanIT

jednokratnom lozinkom. Kod njih je postavljena zastavica `must_change_password`, pa im je nakon prvog prijavljivanja pristup ostalim funkcionalnostima ograničen sve dok ne postave novu lozinku. Na taj način početna lozinka služi samo za prvi pristup sistemu.

Pristupni token u sistemu PlanIT ima ograničeno trajanje i prosleđuje se pri zahtevima prema zaštićenim resursima. Kada pristupni token istekne, klijentska aplikacija koristi osvežavajući token za dobijanje novog para tokena. Sistem koristi rotaciju osvežavajućih tokena, tako da se nakon uspešnog osvežavanja izdaje novi osvežavajući token, dok se prethodni automatski stavlja na crnu listu i više ne može da se koristi. Time se sprečava njegova ponovna upotreba nakon uspešne rotacije.

Prilikom odjavljivanja poništava se osvežavajući token koji je prosleđen zahtevu za odjavu. Taj token se stavlja na crnu listu, nakon čega više ne može biti iskorišćen za izdavanje novih pristupnih tokena. Osvežavajući tokeni koji su prethodno zamenjeni tokom rotacije već su poništeni mehanizmom rotacije, dok sam postupak odjavljivanja ne vrši dodatno poništavanje drugih tokena korisnika. Za evidenciju izdatih i poništenih tokena koriste se odgovarajuće tabele biblioteke

`djangorestframework-simplejwt`.

Autorizacija je zasnovana na korisničkoj ulozi i odnosu korisnika prema konkretnom događaju. Na serverskoj strani definisan je skup klasa dozvola koje kontrolišu pristup različitim funkcionalnostima sistema, uključujući administraciju korisnika, događaje, komunikaciju, beleške, notifikacije, spoljne saradnike i revizijski dnevnik.

Klasa `IsAdmin` ograničava pristup na korisnike sa administratorskom ulogom, odnosno *superuser* naloge. Koristi se za operacije namenjene administraciji sistema, kao što su kreiranje korisničkih naloga, kreiranje događaja i pristup revizijskom dnevniku.

Klasa `IsCoordinatorOrAdmin` koristi se pri listanju korisnika i omogućava pristup administratoru i koordinatoru. Na taj način obe uloge mogu da pristupe odgovarajućim listama korisnika, dok su operacije kreiranja korisnika ograničene na administratora.

Za pristup događajima i čitanje pripadajućih poruka koristi se objektna dozvola `CanAccessEvent`. Administrator može da pristupi svim događajima, koordinator događajima koje koordinira, dok klijent može da pristupi samo događajima čiji je vlasnik. Ista pravila pristupa primenjuju se i pri čitanju poruka povezanih sa događajem. Administrator može da pregleda postojeće poruke, ali ne može da šalje nove.

Pojedine operacije zahtevaju stroža ograničenja od opšte objektne dozvole. Promenu statusa događaja može da izvrši samo koordinator kome je događaj dodeljen, što proverava klasa `IsEventCoordinator`. Rad sa privatnim beleškama i označavanje notifikacija kao pročitanih zasnivaju se na klasi `IsEventParticipant`, kojom se pristup ograničava na klijenta i koordinatora konkretnog događaja.

Klasa `IsCoordinatorOrClient` omogućava koordinatoru i klijentu pregled liste notifikacija. Kreiranje zapisa o spoljnom saradniku dostupno je svakom koordinatoru, dok se za povezivanje i uklanjanje veze sa konkretnim događajem koristi klasa `IsCoordinator`, uz dodatnu proveru da korisnik koordinira taj događaj.

Pravila autorizacije proveravaju se na serverskoj strani pri svakom zaštićenom zahtevu. Korisnički interfejs može da sakrije opcije koje određenom korisniku nisu dostupne, ali takvo ograničenje ima prvenstveno prezentacionu ulogu. Konačnu odluku o tome da li je operacija dozvoljena donosi serverska strana na osnovu autentifikovanog korisnika, njegove uloge i odnosa prema traženom resursu.

Na ovaj način PlanIT kombinuje JWT autentifikaciju sa kontrolom pristupa zasnovanom na korisničkim ulogama i odnosu prema konkretnim resursima. Svi

korisnici koriste isti autentifikacioni mehanizam, dok se njihove dozvole razlikuju u skladu sa ulogom i kontekstom operacije.

3.7 Implementacija klijentske aplikacije

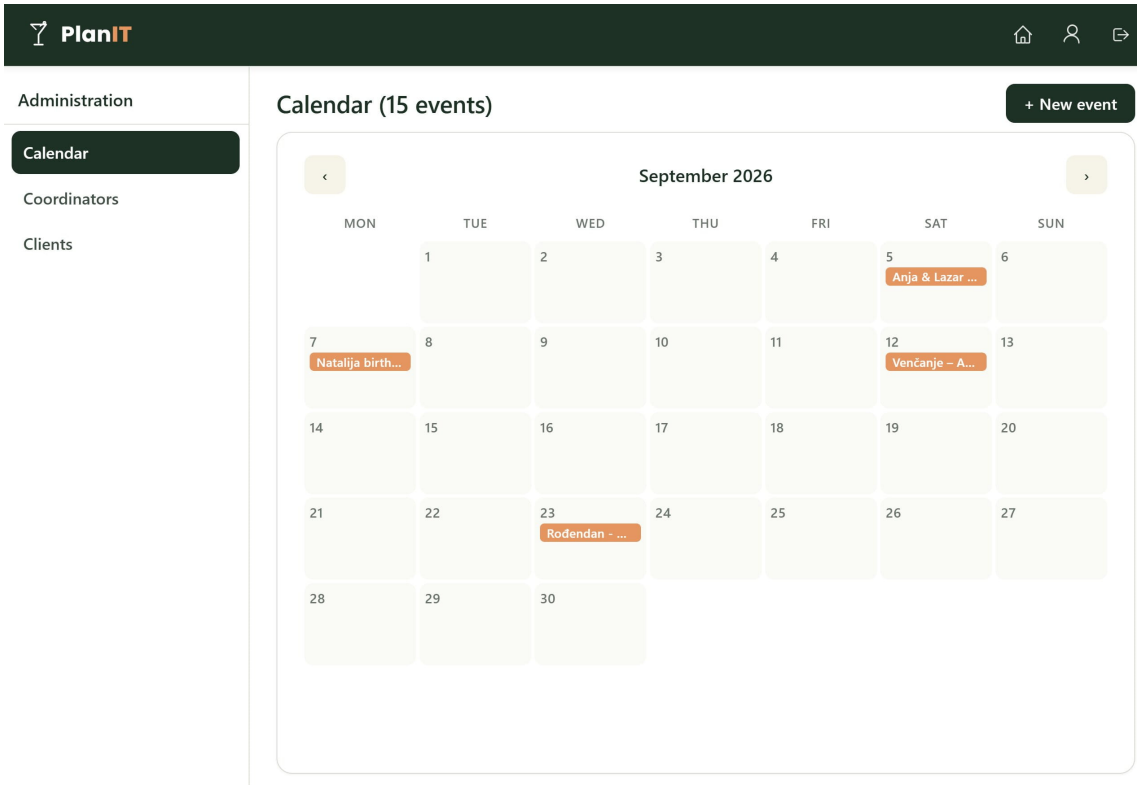
Klijentska aplikacija PlanIT organizovana je kroz prikaze prilagođene korisničkoj ulozi. Administrator koristi poseban administrativni deo aplikacije, dok koordinator i klijent nakon prijavljivanja dobijaju pregled događaja koji su im dodeljeni. Izborom konkretnog događaja prelaze na prikaz namenjen radu u njegovom kontekstu, u okviru kojeg su dostupne funkcionalnosti odgovarajuće korisničkoj ulozi. Pored toga, svim korisnicima je dostupna zajednička globalna navigacija za pristup početnoj stranici, korisničkom profilu i odjavljivanje, dok klijenti i koordinatori imaju pristup i notifikacijama. U nastavku su predstavljeni ključni delovi implementacije klijentske aplikacije, uključujući kalendarski i početni prikaz, organizaciju radnog prostora i navigacije, kao i korisničke interfejse za komunikaciju, privatne beleške i revizijski dnevnik.

Kalendar događaja

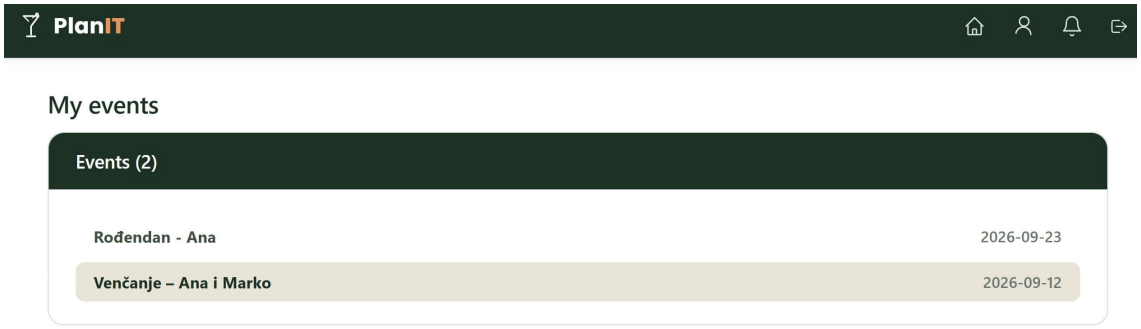
Kalendarski prikaz namenjen je administratoru i pruža mesečni pregled događaja prema datumima održavanja. Kalendar predstavlja alternativu listnom prikazu i omogućava pregled rasporeda događaja u okviru administratorskog dela aplikacije. Izgled kalendarskog prikaza dat je na slici 3.6.

Početni pregled

Nakon prijavljivanja korisniku se prikazuje početni pregled (engl. *dashboard*) prilagođen njegovoj ulozi. Administratoru se podrazumevano prikazuje kalendarski pregled događaja, dok koordinator i klijent dobijaju pregled događaja sa kojima su povezani. Iz ovog pregleda koordinator i klijent mogu da izaberu odgovarajući događaj i nastave rad u njegovom kontekstu. Primer početnog pregleda namenjenog koordinatoru i klijentu prikazan je na slici 3.7.



Slika 3.6: Administratorski kalendarski prikaz događaja



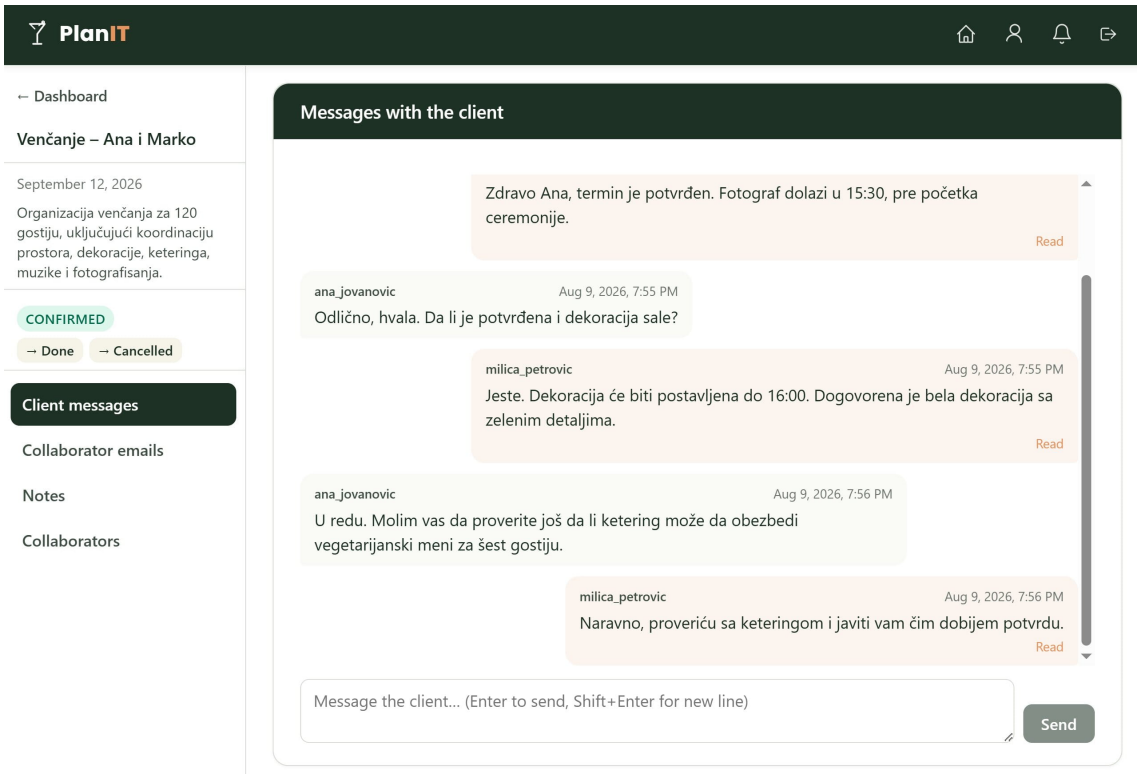
Slika 3.7: Početni pregled aplikacije

Radni prostor i navigacija

Navigacija u aplikaciji organizovana je različito za administratorske i događajne funkcionalnosti. Administrator koristi poseban raspored, u okviru kojeg bočna navigacija omogućava pristup kalendaru, koordinatorima i klijentima.

Koordinatoru i klijentu se nakon prijavljivanja najpre prikazuje lista događaja sa kojima su povezani. Izborom jednog od njih otvara se prikaz konkretnog događaja. Njegov podrazumevani sadržaj predstavlja interna komunikacija između klijenta i koordinatora, dok se osnovni podaci o događaju, uključujući datum, opis i trenutni status, prikazuju u bočnom delu interfejsa.

Bočna navigacija istovremeno omogućava pristup ostalim funkcionalnostima događaja i prilagođava se ulozi prijavljenog korisnika. Koordinator može da pristupi internoj komunikaciji sa klijentom, komunikaciji sa spoljnim saradnicima, privatnim beleškama i upravljanju saradnicima, dok su klijentu dostupne interna komunikacija sa koordinatorom i privatne beleške. Primer prikaza događaja iz perspektive koordinatora dat je na slici 3.8.

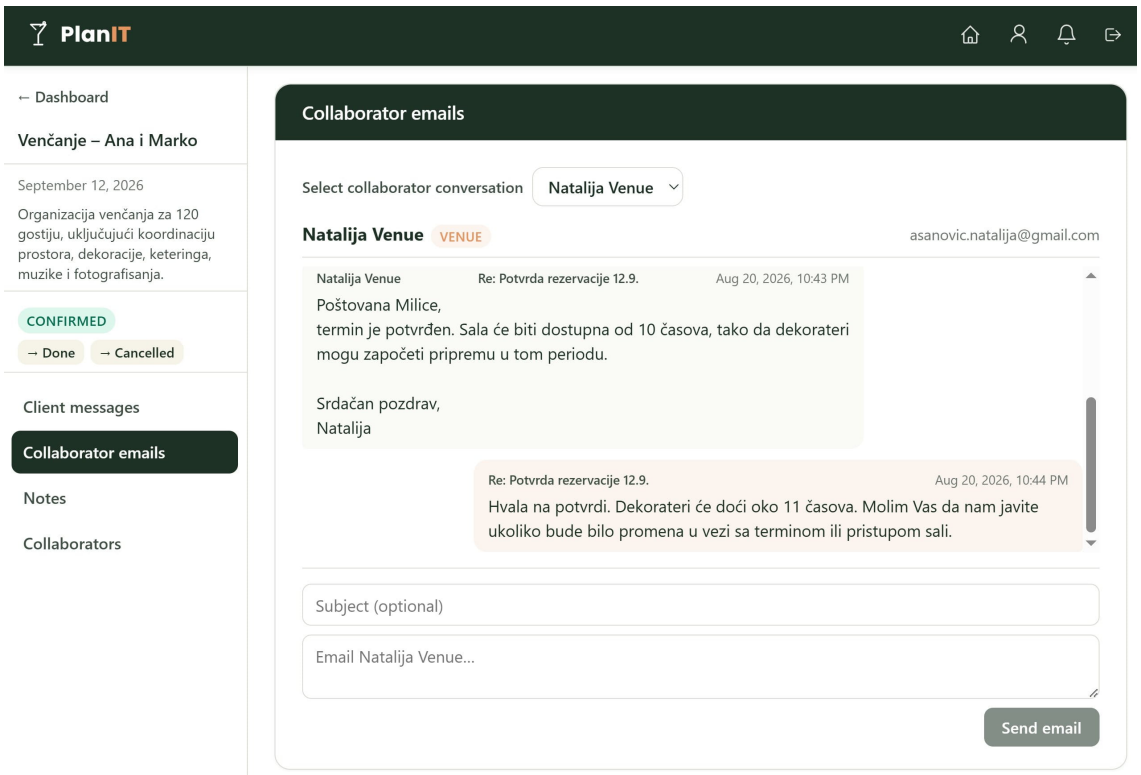


Slika 3.8: Prikaz izabranog događaja iz perspektive koordinatora

Komunikacija

Komunikacioni deo aplikacije organizovan je u okviru izabranog događaja. Interna komunikacija prikazuje se kao nit poruka između koordinatora i klijenta, dok je koordinatoru dostupan i poseban prikaz komunikacije sa spoljnim saradnicima povezanim sa događajem.

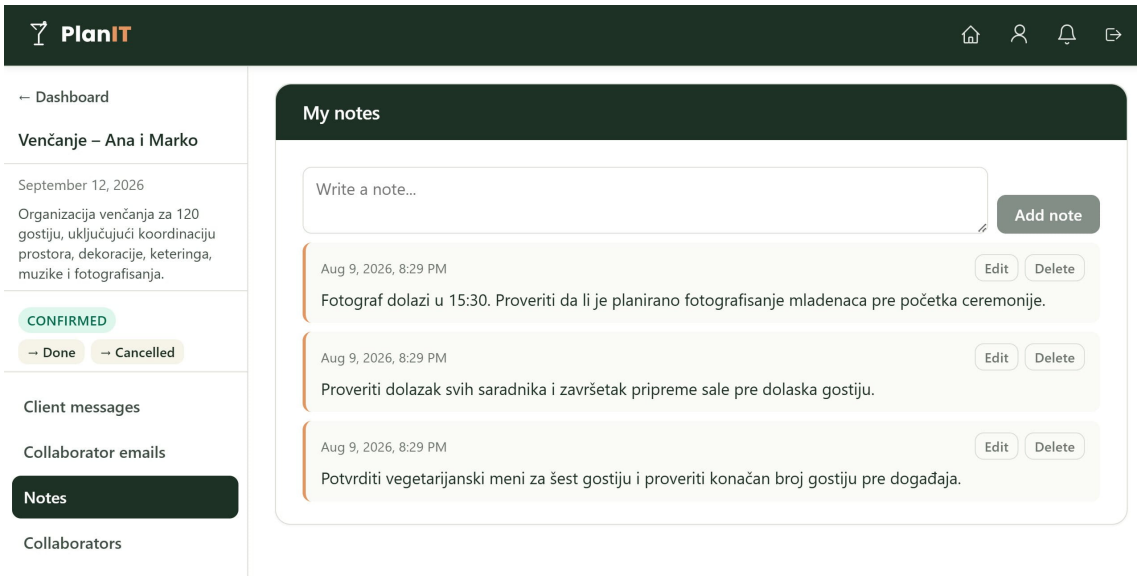
Eksterna komunikacija organizovana je prema saradnicima, tako da se za svakog povezanog saradnika prikazuje odgovarajuća istorija poruka i forma za slanje nove poruke. Poruke koje je poslao koordinator i poruke primljene od saradnika vizuelno su razdvojene u prikazu niti, u skladu sa smerom poruke. Prikaz se periodično osvežava, tako da se odgovori saradnika pristigli putem elektronske pošte pojavljuju bez potrebe za ručnim osvežavanjem stranice, dok se saradnik sa najnovijom porukom ističe na vrhu liste saradnika. Sistemska obaveštenja prikazuju se kroz notifikacioni mehanizam aplikacije. Na slici 3.9 prikazan je interfejs za eksternu komunikaciju sa saradnikom povezanim sa događajem.



Slika 3.9: Komunikacija u okviru događaja

Privatne beleške

Beleške u okviru događaja predstavljaju privatni sadržaj korisnika. Interfejs omogućava njihov pregled, kreiranje, izmenu i brisanje u skladu sa poslovnim pravilima sistema, pri čemu dostupnost pojedinih operacija zavisi od autorstva i trenutnog statusa događaja. Prikaz interfejsa za rad sa beleškama dat je na slici 3.10.

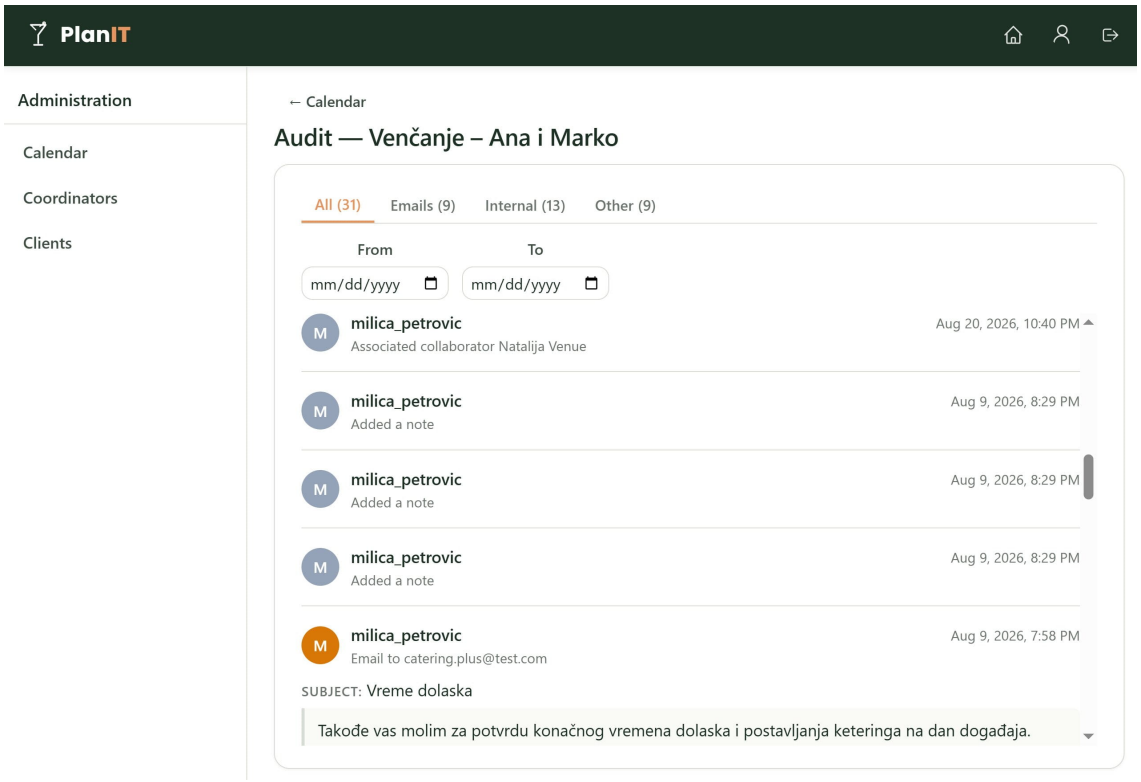


Slika 3.10: Prikaz privatnih beleški

Revizijski dnevnik

Revizijski dnevnik omogućava administratoru pregled značajnih aktivnosti evidentiranih tokom korišćenja sistema. Zapisi su prikazani u tabelarnom obliku i omogućavaju pregled vrste izvršene akcije, njenog izvršioca i vremena nastanka. Pristup ovom prikazu ograničen je na administratora, u skladu sa pravilima autorizacije opisanim u prethodnim odeljcima. Primer prikaza revizijskog dnevnika dat je na slici 3.11.

Klijentska aplikacija na ovaj način razdvaja administratorske funkcionalnosti od rada nad konkretnim događajem, dok se dostupna navigacija i funkcionalnosti prilagođavaju ulozi prijavljenog korisnika.



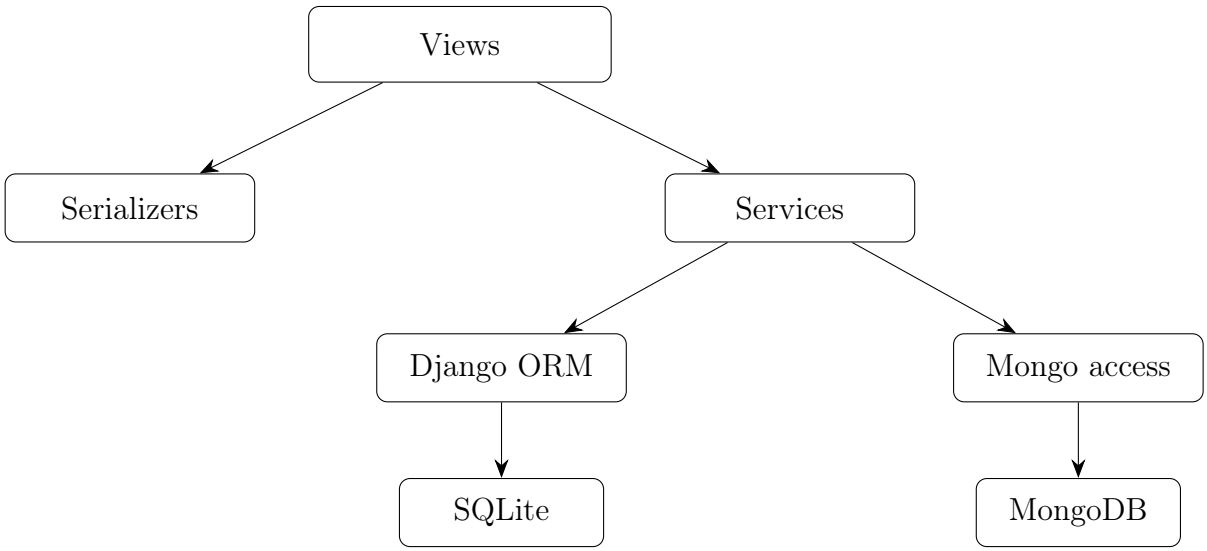
Slika 3.11: Prikaz revizijskog dnevnika

3.8 Implementacija serverske strane

Serverska strana aplikacije PlanIT organizovana je prema slojevitoj arhitekturi. Odgovornosti su raspoređene između pogleda, serijalizatora, servisnog sloja i modela, dok se pristup podacima ostvaruje preko Django ORM-a i funkcija za rad sa MongoDB kolekcijama. Ovakva organizacija razdvaja HTTP obradu, validaciju podataka, poslovnu logiku i skladištenje podataka.

Međusobni odnosi glavnih delova serverske strane prikazani su na slici 3.12. Pogledi predstavljaju centralnu tačku obrade HTTP zahteva i, u zavisnosti od vrste operacije, koriste serijalizatore za validaciju ili formiranje podataka i servisni sloj za izvršavanje poslovne logike.

Pogledi predstavljaju ulaznu tačku serverske aplikacije za HTTP zahteve. U zavisnosti od resursa i zahtevane operacije, oni primenjuju odgovarajuće klase dozvola, obrađuju podatke iz zahteva i delegiraju dalje aktivnosti serijalizatorima ili servisima. Poslovna pravila pri tome ostaju izdvojena iz sloja pogleda i sprovode se u



Slika 3.12: Organizacija serverske strane i pristup skladištima podataka

servisnom sloju.

Serijalizatori imaju različite uloge u zavisnosti od vrste operacije. Kod operacija koje primaju strukturisane podatke iz tela zahteva koriste se za proveru njihove ispravnosti pre prosleđivanja servisnom sloju. Kod pojedinih operacija čitanja koriste se za formiranje izlazne reprezentacije podataka. Zbog toga serijalizatori i servisi ne čine strogo sekvencijalan lanac, već predstavljaju odvojene komponente koje pogled koristi prema potrebi konkretne operacije.

Centralno mesto za poslovna pravila zauzima servisni sloj. U njemu se izvršavaju operacije nad događajima, korisnicima, saradnicima, porukama, beleškama i revizijskim zapisima. Izdvajanje ove logike iz pogleda smanjuje dupliranje poslovnih pravila, pojednostavljuje njihovo održavanje i omogućava nezavisnije testiranje servisnih funkcija.

Primer ovakve organizacije predstavlja promena statusa događaja. Pogled proverava odgovarajuću dozvolu i prosleđuje zahtev servisnom sloju, u kojem se primenjuju pravila vezana za tranziciju statusa i izvršavaju povezane operacije. Detaljna pravila promene statusa predstavljena su u odeljku 3.5, pa ovaj primer ovde služi samo za prikaz razdvajanja odgovornosti između HTTP i poslovnog sloja.

Servisni sloj pristupa relacionim entitetima preko Django ORM-a, dok se dokumentnim podacima pristupa pomoću funkcija za rad sa MongoDB kolekcijama. Na ovaj način servisni sloj koordinira operacije koje koriste relaciono i dokumentno

skladište u skladu sa podelom podataka definisanom u arhitekturi sistema.

Pored slojevite organizacije, serverska strana podeljena je i na funkcionalne module. Posebni moduli postoje za korisnike, događaje, komunikaciju, saradnike, beleške, revizijski dnevnik i autentifikaciju. Većina ovih oblasti ima odgovarajuće module u slojevima pogleda, serijalizatora i servisa, dok pojedine funkcionalnosti dele određene komponente sa povezanim modulima. Ovakva organizacija olakšava razdvajanje odgovornosti između različitih domenskih oblasti sistema.

Servisni sloj obuhvata i pozadinski proces koji periodično proverava dolaznu elektronsku poštu i povezuje primljene odgovore sa odgovarajućim događajem i saradnikom. Ovaj proces izvršava se nezavisno od obrade korisničkih zahteva i uključuje mehanizam za sprečavanje ponovne obrade već preuzete poruke.

Ovakva struktura razdvaja HTTP obradu, validaciju i predstavljanje podataka, poslovnu logiku i pristup skladištima, čime se serverska aplikacija organizuje u jasno definisane funkcionalne i arhitektonske celine.

3.9 REST API

REST API predstavlja komunikacioni sloj između klijentske i serverske strane aplikacije PlanIT. Krajnje tačke su organizovane prema funkcionalnim celinama sistema, tako da svaka grupa obuhvata operacije nad povezanim resursima. Umesto pojedinačnog navođenja svih dostupnih ruta, u ovom odeljku dat je sažet pregled njihove osnovne namene.

Sve API krajnje tačke imaju prefiks `/api/`. Pristup zaštićenim resursima zahteva autentifikovanog korisnika, dok se konkretne operacije dodatno ograničavaju odgovarajućim klasama dozvola i poslovnim pravilima opisanim u prethodnim odeljcima.

Autentifikacioni deo API-ja obuhvata prijavljivanje, osvežavanje tokena, odjavljivanje, pristup podacima o trenutno prijavljenom korisniku i promenu lozinke. Zaseban skup krajnjih tačaka namenjen je upravljanju korisnicima, pri čemu administrator može da kreira korisničke naloge, uključujući naloge sa jednokratnom lozinkom za prvi pristup sistemu.

Događaji predstavljaju centralni API resurs. Sistem podržava pregled liste i detalja događaja, njihovo kreiranje i promenu statusa u skladu sa pravilima životnog ciklusa. Prilikom čitanja podataka primenjuje se filtriranje prema ulozi korisnika: administrator ima pristup svim događajima, koordinator događajima koje koordinira, a klijent događajima čiji je vlasnik.

Komunikacioni deo API-ja organizovan je u kontekstu događaja. Omogućeni su pregled i slanje poruka, kao i označavanje poruka kao pročitanih, dok se za rad sa notifikacijama koriste odgovarajuće krajnje tačke. API takođe omogućava pregled i kreiranje zapisa o spoljnim saradnicima, kao i njihovo povezivanje sa konkretnim događajem i uklanjanje postojeće veze.

Privatne beleške dostupne su kroz operacije pregleda, kreiranja, izmene i brisanja u okviru konkretnog događaja, uz proveru autorstva i ostalih poslovnih ograničenja. Revizijski dnevnik dostupan je administratoru preko krajnje tačke `/api/audit-logs/`, a zapisi se u aplikaciji prikazuju filtrirani prema izabranom događaju.

Sažet pregled najvažnijih API celina prikazan je u tabeli 3.4.

Tabela 3.4: Pregled glavnih REST API modula

Modul	**Glavne operacije**
Autentifikacija	Prijavljivanje, osvežavanje tokena, odjavljivanje, podaci o trenutnom korisniku i promena lozinke
Korisnici	Pregled korisnika i kreiranje korisničkih naloga, uključujući naloge sa jednokratnom lozinkom
Događaji	Pregled liste i detalja događaja, kreiranje događaja i promena statusa
Poruke i notifikacije	Pregled i slanje poruka, označavanje poruka kao pročitanih, pregled i otvaranje notifikacija
Spoljni saradnici	Pregled i kreiranje saradnika, povezivanje sa događajem i uklanjanje postojeće veze
Beleške	Pregled, kreiranje, izmena i brisanje privatnih beleški
Revizijski dnevnik	Globalni pregled revizijskih zapisa i filtriranje prema događaju

Ovakva organizacija omogućava klijentskoj aplikaciji pristup resursima kroz jasno razdvojene API celine, dok poslovna logika, kontrola pristupa i način skladištenja podataka ostaju odgovornost serverske strane.

3.10 Testiranje

Testiranje aplikacije PlanIT obuhvata serversku i klijentsku stranu sistema, sa ciljem provere ispravnosti poslovne logike, pravila pristupa i očekivanog ponašanja korisničkog interfejsa. Testovi pokrivaju različite nivoe aplikacije, od pojedinačnih modela i klasa dozvola do servisnog sloja i REST API-ja, dok se na klijentskoj strani proveravaju pomoćne funkcije, komponente i pojedini tokovi korisničke interakcije.

Testiranje serverske strane

Serverska strana testira se korišćenjem alata `pytest` [33] i dodatka `pytest-django` [34]. Jedinični testovi obuhvataju ponašanje modela i ograničenja definisana na nivou relacionog sloja, kao i klase dozvola koje određuju prava pristupa različitih korisničkih uloga. Ovim testovima proverava se da osnovna pravila vezana za modele i dozvole ostaju ispravna i nakon izmena implementacije.

Zaseban nivo čine testovi servisnog sloja, čiji je cilj da potvrde ispravnost poslovne logike nezavisno od HTTP interfejsa. Proveravaju se operacije nad događajima, korisnicima, saradnicima, porukama, notifikacijama i revizijskim zapisima. Time se utvrđuje da servisni sloj dosledno primenjuje pravila domena i da se povezane operacije, poput evidentiranja aktivnosti u revizijskom dnevniku, izvršavaju u očekivanom trenutku.

Pored toga, postoje API testovi integracionog karaktera koji obuhvataju HTTP sloj aplikacije. Njihov cilj je da provere saradnju ruta, serijalizatora, dozvola i servisnog sloja u različitim scenarijima korišćenja. Ovim testovima obuhvaćeni su autentifikacija i odjava, kontrola pristupa prema korisničkim ulogama, pristup i pravljenje događaja, komunikacija, upravljanje saradnicima, beleške, revizijski dnevnik i notifikacije. Posebno su značajni negativni scenariji, kojima se proverava da sistem odbija zahteve korisnika koji nemaju potrebne dozvole ili pokušavaju da pristupe resursima koji im nisu dostupni. Posebna grupa testova pokriva funkcionalnost prijema elektronske pošte, uključujući ispravnost povezivanja primljene poruke sa događajem i saradnikom, sprečavanje ponovne obrade iste poruke i ponašanje sistema kada ova funkcionalnost nije konfigurisana.

Za funkcionalnosti koje koriste MongoDB, u testovima se primenjuje biblioteka `mongomock` [31]. Umesto povezivanja sa stvarnim MongoDB serverom, pymongo klijent zamenjuje se memorijskom implementacijom koja oponaša njegov interfejs. Na taj način mogu se izolovano proveravati operacije nad kolekcijama poruka, beleški i revizijskih zapisa, kao i rad sa indeksima, bez zavisnosti od spoljnog MongoDB servisa.

Testiranje klijentske strane

Klijentska strana testira se korišćenjem alata `Vitest` [43] i biblioteke `React Testing Library` [40], u okruženju zasnovanom na `jsdom`-u. Testovi imaju jedinični, komponentni i integracioni karakter. Obuhvaćene su pomoćne funkcije povezane

sa autentifikacijom i čuvanjem tokena, pojedine API funkcije, komponente korisničkog interfejsa i određeni tokovi korisničke interakcije, uključujući i tok prijavljivanja i rutiranje ka odgovarajućem prikazu.

API pozivi se u testovima zamenjuju mock implementacijama, čime se ponašanje klijentske aplikacije proverava bez oslanjanja na stvarni serverski sloj. Na ovaj način utvrđuje se ispravnost pojedinih tokova prijavljivanja, prikaza određenih komponenti, obrade korisničkih akcija i poziva prema API sloju.

Kombinovanjem testova serverske i klijentske strane proveravaju se različiti nivoi aplikacije. Testovi serverske strane prvenstveno potvrđuju ispravnost modela, poslovnih pravila, dozvola i API ponašanja, dok testovi klijentske strane proveravaju pomoćnu logiku, prikaz pojedinih komponenti i osnovne korisničke interakcije. Cilj ovakvog pristupa jeste rano otkrivanje grešaka i smanjenje rizika da naknadne izmene naruše postojeće funkcionalnosti.

Raspodela testova po nivoima testiranja prikazana je u tabeli 3.5.

Tabela 3.5: Broj testova po nivoima testiranja

Nivo	**Broj testova**
Jedinični (modeli)	12
Jedinični (dozvole)	19
Servisni	55
Integracioni (API)	106
Specijalizovani (prijem e-pošte)	19
Serverska strana, ukupno	**211**
Jedinični (autentifikacija)	7
Jedinični (API funkcije)	7
Komponentni	6
Integracioni	2
Klijentska strana, ukupno	**22**
Ukupno u projektu	**233**

Ukupno su implementirana 233 testa, od kojih 211 obuhvata serversku, a 22 klijentsku stranu aplikacije. Svi navedeni testovi uspešno prolaze. Rezultati testiranja potvrđuju očekivano ponašanje obuhvaćenih funkcionalnosti i pravila pristupa u testiranim scenarijima, uključujući i negativne slučajeve u kojima korisnik pokušava da izvrši operaciju za koju nema odgovarajuću dozvolu.

3.11 Ograničenja sistema

Fokus ovog rada bio je na istraživačkom pitanju koje se odnosi na kontrolu pristupa i pouzdanu evidenciju aktivnosti u višekorisničkom sistemu sa više nezavisnih učesnika. Ostale funkcionalnosti realizovane su u obimu potrebnom za celovito funkcionisanje aplikacije, bez zahteva da svaki njen deo bude prilagođen produkcionom okruženju. U skladu sa tim, trenutna implementacija ima određena ograničenja koja se prvenstveno odnose na relaciono skladištenje podataka, način osvežavanja podataka na klijentskoj strani, komunikaciju sa spoljnim saradnicima putem elektronske pošte i upravljanje koordinatorom događaja nakon brisanja njegovog korisničkog naloga.

Prvo ograničenje odnosi se na relacioni deo sistema, za koji se koristi SQLite sistem za upravljanje bazom podataka. SQLite pojednostavljuje razvojno okruženje, ali ima ograničenja u scenarijima sa većim brojem istovremenih operacija upisa i intenzivnijim produkcionim opterećenjem. U slučaju proširenja sistema na veći broj korisnika i izraženiji paralelni pristup podacima, bilo bi opravdano razmotriti prelazak na serverski relacioni sistem za upravljanje bazom podataka.

Drugo ograničenje odnosi se na osvežavanje podataka koji se menjaju tokom rada aplikacije. Podaci o događaju, interne poruke i notifikacije preuzimaju se sa serverske strane u intervalima od pet sekundi. Na taj način primenjen je pristup periodičnog ispitivanja servera (engl. *polling*), pri kojem klijentska strana u pravilnim vremenskim intervalima proverava dostupnost novih podataka. Promene se zato ne prosleđuju klijentu neposredno nakon njihovog nastanka, već postaju vidljive prilikom narednog zahteva. Pored toga, zahtevi se šalju i u slučajevima kada nije došlo do promene podataka, čime se stvara dodatna komunikacija sa serverskom stranom. Za sisteme sa izraženijim zahtevima za razmenom podataka u realnom vremenu bilo bi opravdano razmotriti drugačiji komunikacioni mehanizam.

Treće ograničenje odnosi se na komunikaciju sa spoljnim saradnicima putem elektronske pošte. Sistem podržava slanje i prijem poruka korišćenjem SMTP i IMAP protokola, pri čemu se primljeni odgovori povezuju sa odgovarajućim događajem i saradnikom. Ukoliko pristupni podaci za servis elektronske pošte nisu podešeni, slanje poruka se simulira, dok je njihov prijem onemogućen. Pouzdanost ovog dela sistema zato zavisi i od ispravne konfiguracije i dostupnosti spoljnog servisa za elektronsku poštu. Napredniji mehanizmi, kao što su detaljnije praćenje neuspelih isporuka, ponovni pokušaji slanja i složenija obrada grešaka, nisu obuhvaćeni tre-

nutnom implementacijom.

Četvrto ograničenje odnosi se na upravljanje vezom između događaja i koordinatora. U trenutnoj implementaciji brisanje korisničkog naloga koordinatora nije omogućeno kroz korisnički interfejs, ali ukoliko do brisanja naloga dođe na nivou sistema, primenom pravila `SET_NULL` uklanja se veza između koordinatora i događaja, dok sam događaj ostaje sačuvan. Pošto promenu statusa može da izvrši isključivo koordinator kome je događaj dodeljen, takav događaj ostaje bez korisnika koji može da izvrši narednu promenu statusa. Istovremeno, postojeća implementacija ne omogućava administratoru da postojećem događaju naknadno dodeli drugog koordinatora, pa događaj kroz raspoloživi API ostaje trajno vezan za tekući status. Ovo ograničenje moglo bi se otkloniti uvođenjem administratorske operacije za naknadnu promenu koordinatora postojećeg događaja.

Navedena ograničenja određuju granice trenutne implementacije i ukazuju na funkcionalnosti koje bi mogle biti unapređene u daljem razvoju sistema. Ona se pre svega odnose na infrastrukturne i pomoćne aspekte aplikacije, dok su mehanizmi kontrole pristupa i evidencije aktivnosti, koji predstavljaju centralni fokus rada, realizovani u okviru definisanog modela sistema.

3.12 Mogućnosti daljeg razvoja

Dalji razvoj sistema PlanIT može biti usmeren na unapređenje postojećih tehničkih rešenja, uklanjanje ograničenja trenutne implementacije i proširenje funkcionalnosti sistema. Mogući pravci razvoja odnose se na infrastrukturu i performanse, pouzdanost komunikacionih kanala, razmenu podataka u realnom vremenu, integraciju sa spoljnim servisima i nove načine pristupa aplikaciji.

Jedan od mogućih pravaca razvoja jeste prelazak sa SQLite sistema za upravljanje bazom podataka na serverski relacioni sistem, kao što je PostgreSQL, naročito u slučaju povećanja broja korisnika i intenziteta istovremenog pristupa podacima. U istom kontekstu moglo bi se razmotriti i uvođenje sistema Redis kao pomoćnog sloja za keširanje podataka koji se često čitaju, čime bi se smanjio broj ponovljenih upita prema primarnim skladištima.

Značajan prostor za unapređenje postoji u oblasti pouzdanosti komunikacije sa spoljnim saradnicima. Postojeća integracija sa servisom za elektronsku poštu mogla bi se dodatno ojačati mehanizmima za praćenje neuspešnih pokušaja slanja, ponovno slanje poruka i obaveštavanje koordinatora o problemima u isporuci. Pored toga,

sistem bi mogao da podrži i SMS komunikaciju kao dodatni kanal za vremenski osetljiva obaveštenja, kao što su značajne promene statusa događaja.

Moguće je unaprediti i način na koji klijentska aplikacija dobija promene sa serverske strane. PlanIT trenutno koristi *polling* za periodično preuzimanje podataka o događajima, porukama i notifikacijama. Uvođenjem WebSocket komunikacije bilo bi moguće prosleđivati promene klijentskoj aplikaciji neposredno nakon njihovog nastanka, čime bi se smanjila potreba za ponavljanim zahtevima i omogućilo ažurnije prikazivanje podataka.

Dalji razvoj mogao bi obuhvatiti i integraciju sa spoljnim servisima, kao što je sinhronizacija događaja sa servisom Google Calendar, kao i izradu posebne mobilne aplikacije koja bi, oslanjajući se na postojeći REST API, korisnicima omogućila pristup događajima, komunikaciji i beleškama van veb pregledača. Razvoj mobilnog klijenta dodatno bi otvorio mogućnost uvođenja push notifikacija, kojima bi korisnik bio obavešten o značajnim promenama neposredno nakon njihovog nastanka, umesto oslanjanja na periodičnu proveru novih podataka.

Navedene mogućnosti ne predstavljaju deo postojeće implementacije, već potencijalne pravce njenog daljeg razvoja. Njihova opravdanost zavisila bi od budućih zahteva za skalabilnošću, komunikacijom, integracijama sa spoljnim servisima i načinom pristupa aplikaciji.

Glava 4

Zaključak

U ovom master radu analizirana je, projektovana i implementirana veb aplikacija PlanIT, namenjena organizaciji i koordinaciji svečanih događaja, sa ciljem razvoja sistema koji podržava razmenu informacija između učesnika, uz jasno definisana pravila pristupa podacima i evidenciju aktivnosti u sistemu.

Glavni doprinos rada ogleda se u razvoju arhitektonskog rešenja koje objedinjuje koordinaciju učesnika, razmenu informacija, kontrolu pristupa i praćenje aktivnosti u okviru jedinstvene veb aplikacije. Sistem je realizovan prema klijent-server arhitekturi. Klijentska strana razvijena je korišćenjem biblioteke React i jezika TypeScript, a serverska u programskom jeziku Python uz okvir Django i Django REST Framework, sa autentifikacijom zasnovanom na JWT mehanizmu i poslovnim pravilima izdvojenim u servisni sloj. Pristup podacima i izvršavanje operacija određuju se kombinovanjem korisničke uloge i odnosa korisnika prema konkretnom događaju, čime funkcionalnosti kao što su pregled događaja, promena statusa, komunikacija, privatne beleške i upravljanje saradnicima ostaju usklađene sa odgovornostima pojedinih korisnika.

Podaci u sistemu modelovani su primenom pristupa *polyglot persistence*: strukturisani podaci o korisnicima, događajima i spoljnim saradnicima čuvaju se u relacionom sistemu za upravljanje bazama podataka SQLite, dok se poruke, privatne beleške i revizijski zapisi čuvaju u dokumentnoj MongoDB bazi, uz povezivanje putem logičkih identifikatora. Interna komunikacija omogućava razmenu poruka između klijenta i koordinatora, eksterna komunikacija povezuje koordinatora sa spoljnim saradnicima, dok revizijski dnevnik evidentira značajne aktivnosti i omogućava praćenje korisničkih akcija nad relevantnim resursima.

Životni ciklus događaja modelovan je kroz definisana stanja i dozvoljene tranzici-

je, pri čemu status događaja neposredno utiče na dostupnost komunikacije, beležaka i drugih povezanih funkcionalnosti, a poslovna pravila se sprovode na serverskoj strani nezavisno od korisničkog interfejsa. Na ovaj način PlanIT ne posmatra događaj samo kao izolovani zapis, već kao centralni kontekst u okviru kojeg se povezuju korisnici, saradnici, komunikacija i evidencija aktivnosti, uz zadržane jasno definisane granice pristupa između učesnika.

Ispravnost implementiranog rešenja proveravana je testovima serverske i klijentske strane.

Razvijeno rešenje ima i određena ograničenja koja proizlaze iz obima trenutne implementacije. Korišćeni relacioni sistem za upravljanje bazom podataka SQLite nije prilagođen većem produkcionom opterećenju, postojeća integracija sa servisom za elektronsku poštu ne uključuje mehanizme za praćenje neuspešnih pokušaja slanja i ponovno slanje poruka, osvežavanje podataka o događajima, porukama i notifikacijama zasniva se na periodičnom *polling* pristupu, a sistem nema mehanizam za naknadnu dodelu koordinatora događaju čiji je prethodni koordinator obrisan iz sistema. Ova ograničenja ne sprečavaju ostvarivanje osnovne namene sistema, ali ukazuju na pravce njegovog daljeg razvoja.

Moguća unapređenja obuhvataju prelazak na drugi, serverski sistem za upravljanje bazom podataka, uvođenje Redis sistema za keširanje, WebSocket komunikaciju za razmenu podataka u realnom vremenu i jačanje pouzdanosti komunikacije sa spoljnim saradnicima. Dalji razvoj mogao bi da uključi i integraciju sa spoljnim servisima kao što je Google Calendar, dodatne kanale obaveštavanja putem SMS i push notifikacija, kao i razvoj mobilne aplikacije zasnovane na postojećem REST API-ju.

Sprovedeni rad pokazuje da je moguće projektovati i implementirati jedinstven sistem za koordinaciju svečanih događaja koji povezuje više učesnika, različite komunikacione tokove i više tipova skladišta podataka, uz jasno definisana pravila pristupa i evidenciju aktivnosti, čime se kombinovanjem kontrole pristupa, servisnog sloja i pristupa *polyglot persistence* podržava koordinacija složenih organizacionih procesa u kojima učestvuje više nezavisnih aktera.

Upotreba alata zasnovanih na veštačkoj inteligenciji

Tokom izrade ovog master rada korišćeni su ChatGPT, Claude i Claude Code kao pomoćni alati zasnovani na veštačkoj inteligenciji. ChatGPT je korišćen prvenstveno za rad sa tekstom, uključujući proveru akademskog stila pisanja, uočavanje ponavljanja, preformulisanje pojedinih delova teksta i poboljšanje njegove jasnoće i povezanosti.

Claude i Claude Code korišćeni su kao podrška tokom razvoja aplikacije, prvenstveno radi ubrzavanja implementacije, testiranja i analize programskog koda. Claude je korišćen i kao pomoć pri generisanju dijagrama kojima su predstavljeni pojedini procesi i delovi arhitekture sistema.

Predlozi i sadržaji dobijeni korišćenjem navedenih alata nisu preuzimani bez dodatne provere. Ispravnost implementacionih rešenja proveravana je pregledom programskog koda i testiranjem funkcionalnosti, dok je tekst rada dodatno pregledan, korigovan i prilagođen sadržaju i akademskom stilu master rada.

Bibliografija

[1] Natalija Asanović. PlanIT – izvorni kod aplikacije. `https://github.com/Natalija9/event_planning/tree/main`, 2026. Pristupljeno: avgust 2026.

[2] Axios. Axios documentation. `https://axios.rest/pages/getting-started/first-steps`. Pristupljeno: avgust 2026.

[3] Alex Banks and Eve Porcello. *Learning React: Modern Patterns for Developing React Apps*. O'Reilly Media, Sebastopol, California, 2 edition, 2020.

[4] Kristina Chodorow. *MongoDB: The Definitive Guide*. O'Reilly Media, Sebastopol, California, 2 edition, 2013.

[5] Django Software Foundation. Deployment checklist. `https://docs.djangoproject.com/en/5.2/howto/deployment/checklist/`. Pristupljeno: avgust 2026.

[6] Django Software Foundation. Django documentation. `https://docs.djangoproject.com/en/5.2/`. Pristupljeno: avgust 2026.

[7] Encode OSS Ltd. Django rest framework: Authentication. `https://www.django-rest-framework.org/api-guide/authentication/`. Pristupljeno: avgust 2026.

[8] Encode OSS Ltd. Django rest framework documentation. `https://www.django-rest-framework.org/`. Pristupljeno: avgust 2026.

[9] Encode OSS Ltd. Django rest framework: Permissions. `https://www.django-rest-framework.org/api-guide/permissions/`. Pristupljeno: avgust 2026.

[10] Encode OSS Ltd. Django rest framework: Serializers. `https://www.django-rest-framework.org/api-guide/serializers/`. Pristupljeno: avgust 2026.

[11] Encode OSS Ltd. Django rest framework: Views. `https://www.django-rest-framework.org/api-guide/views/`. Pristupljeno: avgust 2026.

[12] Encode OSS Ltd. Django rest framework: Viewsets. `https://www.django-rest-framework.org/api-guide/viewsets/`. Pristupljeno: avgust 2026.

[13] Philip W. L. Fong. Relationship-based access control: Protection model and policy language. In *Proceedings of the First ACM Conference on Data and Application Security and Privacy*, 2011.

[14] Martin Fowler. Polyglot persistence. `https://martinfowler.com/bliki/PolyglotPersistence.html`, 2011. Pristupljeno: avgust 2026.

[15] Vincent C. Hu, David Ferraiolo, Richard Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. Guide to attribute based access control (ABAC) definition and considerations. NIST Special Publication 800-162, National Institute of Standards and Technology, 2014.

[16] Jazzband. Simple jwt documentation. `https://django-rest-framework-simplejwt.readthedocs.io/`. Pristupljeno: avgust 2026.

[17] Michael B. Jones, John Bradley, and Nat Sakimura. JSON web token (JWT). RFC 7519, Internet Engineering Task Force, 2015.

[18] Karen Kent and Murugiah Souppaya. Guide to computer security log management. `https://doi.org/10.6028/NIST.SP.800-92`, 2006. Pristupljeno: avgust 2026.

[19] Mark Lutz. *Learning Python*. O'Reilly Media, Sebastopol, California, 5 edition, 2013.

[20] Meta Platforms, Inc. React api reference: useeffect. `https://react.dev/reference/react/useEffect`. Pristupljeno: avgust 2026.

[21] Meta Platforms, Inc. React api reference: usestate. `https://react.dev/reference/react/useState`. Pristupljeno: avgust 2026.

[22] Meta Platforms, Inc. React documentation. `https://react.dev/`. Pristupljeno: avgust 2026.

[23] Meta Platforms, Inc. React documentation: Passing props to a component. `https://react.dev/learn/passing-props-to-a-component`. Pristupljeno: avgust 2026.

[24] Meta Platforms, Inc. React documentation: Responding to events. `https://react.dev/learn/responding-to-events`. Pristupljeno: avgust 2026.

[25] Meta Platforms, Inc. React documentation: State – a component's memory. `https://react.dev/learn/state-a-components-memory`. Pristupljeno: avgust 2026.

[26] Meta Platforms, Inc. React documentation: Using typescript. `https://react.dev/learn/typescript`. Pristupljeno: avgust 2026.

[27] Meta Platforms, Inc. React documentation: Writing markup with jsx. `https://react.dev/learn/writing-markup-with-jsx`. Pristupljeno: avgust 2026.

[28] Meta Platforms, Inc. React documentation: Your first component. `https://react.dev/learn/your-first-component`. Pristupljeno: avgust 2026.

[29] MongoDB, Inc. Mongodb manual. `https://www.mongodb.com/docs/manual/`. Pristupljeno: avgust 2026.

[30] MongoDB, Inc. Pymongo documentation. `https://pymongo.readthedocs.io/`. Pristupljeno: avgust 2026.

[31] mongomock. mongomock. `https://github.com/mongomock/mongomock`. Pristupljeno: avgust 2026.

[32] OWASP Foundation. Logging cheat sheet. `https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html`. Pristupljeno: avgust 2026.

[33] pytest. pytest documentation. `https://docs.pytest.org/en/stable/`. Pristupljeno: avgust 2026.

[34] pytest-django. pytest-django documentation. `https://pytest-django.readthedocs.io/en/stable/`. Pristupljeno: avgust 2026.

[35] Python Software Foundation. Python programming language. `https://www.python.org/`. Pristupljeno: avgust 2026.

[36] Pramod J. Sadalage and Martin Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley, 2012.

[37] Ravi Sandhu, David Ferraiolo, and Richard Kuhn. The NIST model for role-based access control: Towards a unified standard. In *Proceedings of the Fifth ACM Workshop on Role-Based Access Control*, 2000.

[38] SQLite Development Team. Sqlite documentation. `https://www.sqlite.org/docs.html`. Pristupljeno: avgust 2026.

[39] Dafydd Stuttard and Marcus Pinto. *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*. Wiley, 2 edition, 2011.

[40] Testing Library. React testing library documentation. `https://testing-library.com/docs/react-testing-library/intro/`. Pristupljeno: avgust 2026.

[41] William S. Vincent. *Django for Professionals: Production Websites with Python and Django*. WelcomeToCode, 2020.

[42] Vite. Vite – getting started. `https://vite.dev/guide/`. Pristupljeno: avgust 2026.

[43] Vitest. Vitest documentation. `https://vitest.dev/`. Pristupljeno: avgust 2026.

Biografija autora

Natalija Asanović rođena je 7. septembra 1999. godine u Beogradu. Završila je Devetu beogradsku gimnaziju. Nakon toga je upisala studijski program Informatika na Matematičkom fakultetu Univerziteta u Beogradu, gde je završila osnovne akademske studije 2023. godine. Trenutno je student master akademskih studija na istom fakultetu.