Opis projektnog zadatka

Uvod

U višestambenim zgradama živi velik broj ljudi koji dijele zajedničke dijelove zgrade, poput stubišta, hodnika, dizala, parkirališta i vanjskih površina. Iako svaki stanar koristi svoj privatni prostor, zajednički prostori zahtijevaju stalno i organizirano održavanje. Financijski model za to postoji – pričuva koju plaćaju svi suvlasnici – no mnoge zgrade i dalje ostaju zapuštene, što se jasno vidjelo nakon potresa u Zagrebu.

Glavni uzrok problema nije nedostatak sredstava, nego slaba i neučinkovita komunikacija među suvlasnicima. Sastanci stanara često su slabo posjećeni, informacije se prenose sporim i nepouzdanim kanalima, a mnogi suvlasnici nemaju uvid u probleme koji se događaju izvan njihovog kata ili ulaza. Zbog toga brojni kvarovi i potrebne intervencije ostaju neprimijećeni ili neraspravljeni.

Ovaj projekt bavi se upravo tom problematikom – stvaranjem centralizirane, transparentne i suvremene platforme koja olakšava prijavu problema, komunikaciju suvlasnika i donošenje zajedničkih odluka.

Cilj projektnog zadatka

Cilj projekta je razviti web aplikaciju koja:

- služi kao digitalna oglasna ploča za sve suvlasnike,
- omogućuje pokretanje i vođenje diskusija o problemima u zgradi,
- omogućuje glasanje o prijedlozima,
- automatski generira prijedlog za sastanak kada više od 25 % suvlasnika glasa pozitivno,
- omogućuje integraciju s vanjskim sustavom StanPlan radi formalnog vođenja sastanaka,
- administraciji omogućuje upravljanje korisnicima i postavkama integracije.

Korištenjem ove aplikacije komunikacija postaje brža, jednostavnija i preglednija, što dovodi do kvalitetnijeg održavanja zgrade i učinkovitijeg donošenja odluka.

Opis problematike

Suvlasnici često nemaju uvid u stanje cijele zgrade i oslanjaju se na usmene informacije, oglasne ploče ili rad predstavnika suvlasnika. Ovakav pristup uzrokuje:

- nedovoljno prijavljivanje problema,
- nisku informiranost suvlasnika,
- spor proces donošenja odluka,
- nedostatak transparentnosti,
- lošu koordinaciju među stanarima i predstavnicima.

Dodatni problem nastaje zbog neredovitog održavanja sastanaka, niskog odaziva i nepostojanja digitalnog traga o raspravama i odlukama.

Sve navedeno rezultira lošim održavanjem zajedničkih prostora, sporim rješavanjem intervencija i stvaranjem nezadovoljstva među suvlasnicima.

Opis predloženog rješenja

Predloženo rješenje je web aplikacija koja omogućuje suvlasnicima:

Diskusije

- pokretanje rasprave o bilo kojem problemu ili prijedlogu,
- otvorene i privatne diskusije,
- određivanje ograničenja (broj poruka, zabrana sudjelovanja),
- sudjelovanje predstavnika suvlasnika u javnim diskusijama,
- vidljivost postojanja privatne diskusije svim suvlasnicima uz skrivanje sadržaja.

Glasanje

- pokretanje glasanja unutar diskusije,
- glasanje “Za” ili “Protiv”,
- automatski poziv za sastanak ako više od 25 % suvlasnika glasa pozitivno.

Sastanci

- generiranje prijedloga sastanka s naslovom, dnevnim redom i ciljem,
- povezivanje s diskusijom u kojoj je glasanje pokrenuto,
- pravno valjani zaključci sastanka kroz integraciju sa StanPlan aplikacijom,
- slanje e-mail obavijesti svim suvlasnicima.

Administracija

- izrada i upravljanje korisnicima (suvlasnici i predstavnik suvlasnika),
- izmjena korisničkih podataka,
- postavljanje StanPlan integracijskog URL-a,
- nadzor sigurnosnih aspekata aplikacije.

Autentifikacija

- prijava putem OAuth 2.0 servisa,
- sigurno rukovanje korisničkim identitetima bez lokalnog spremanja lozinki.

Potencijalna korist projekta

Implementacija aplikacije donosi višestruke koristi:

Suvlasnicima:

- veća informiranost o stanju zgrade,
- mogućnost sudjelovanja u raspravama bez odlaska na fizički sastanak,
- transparentnost u odlučivanju,
- brže rješavanje problema.

Predstavniku suvlasnika:

- manji administrativni teret,
- bolja evidencija svih prijedloga i odluka,
- jednostavnija organizacija sastanaka,
- povećano povjerenje zajednice.

Zgradi i zajednici:

- bolje održavanje zajedničkih prostora,
- racionalnije korištenje pričuve,
- brža reakcija u hitnim slučajevima,
- veći stupanj organiziranosti i suradnje.

Postojeća slična rješenja na tržištu

Trenutno postoje aplikacije koje pokrivaju dio funkcionalnosti potrebnih za upravljanje zgradama, poput:

Susjed42

- naglasak na osnovnoj međususjedskoj komunikaciji,
- ne podržava strukturirane diskusije niti automatizirano sazivanje sastanaka.

mSuvlasnik

- fokusiran na financijsko poslovanje i komunikaciju s upraviteljem,
- komunikacija među suvlasnicima nije u središtu.

E-upravitelj

- alat namijenjen profesionalnim upraviteljima,
- nudi dokumentaciju i praćenje radova, ali nema demokratičnu oglasnu ploču.

Ključna razlika našeg rješenja:
Demokratizira komunikaciju dopuštajući svakom suvlasniku pokretanje rasprava i iznošenje prijedloga, uz automatizirane mehanizme glasanja i sazivanja sastanka.

Skup korisnika

Aplikacija je namijenjena:

- suvlasnicima stanova u višestambenim zgradama,
- predstavnicima suvlasnika,
- upraviteljima zgrada,
- velikim stambenim kompleksima,
- potencijalno i poslovnim zgradama koje imaju zajedničke prostore.

Korisnici su tehnički raznoliki – od starijih osoba do digitalno aktivnih suvlasnika – pa je aplikacija dizajnirana da bude jednostavna, intuitivna i pristupačna.

Opseg projektnog zadatka

Ovaj projekt obuhvaća:

- razvoj serverske logike,
- izradu baze podataka i poslovnih pravila,
- mogućnost pokretanja diskusija i glasanja,
- automatizirano generiranje sastanaka nakon glasanja,
- integraciju s OAuth 2.0 sustavom,
- administracijske funkcije,
- integraciju sa StanPlan sustavom.

Ovaj projekt **ne** obuhvaća:

- rješavanje financijskog dijela upravljanja zgradom,
- vođenje tehničke dokumentacije zgrade,
- komercijalne funkcionalnosti upravitelja nekretninama.

Zaključno

Ovaj projekt predstavlja moderan odgovor na stvarne probleme komunikacije u višestambenim zgradama. Digitalizacijom oglasne ploče, omogućavanjem strukturiranih diskusija i uvođenjem transparentnog glasanja, aplikacija potiče angažiranost suvlasnika i ubrzava donošenje odluka. Integracija s postojećim StanPlan sustavom proširuje funkcionalnost na formalne procese donošenja odluka, čime aplikacija postaje snažan alat za suvremeno upravljanje zgradama.

Rješenje je skalabilno, prilagodljivo i lako nadogradivo te predstavlja temelj za budući razvoj naprednijih funkcionalnosti.

Analiza zahtjeva

Na ovoj se stranici nalazi detaljna analiza zahtjeva za aplikaciju za upravljanje komunikacijom i održavanjem višestambenih zgrada.
Cilj je definirati funkcionalne i nefunkcionalne zahtjeve, dionike sustava te zahtjeve vezane uz održavanje i sigurnost.
Nefunkcionalni zahtjevi i zahtjevi domene primjene dopunjuju funkcionalne zahtjeve. Oni opisuju kako se sustav treba ponašati i koja ograničenja treba poštivati (performanse, korisničko iskustvo, pouzdanost, standardi kvalitete, sigurnost itd.).

Analiza zahtjeva definira sve funkcionalne, nefunkcionalne i održavne aspekte sustava te identificira ključne dionike.
Tablični prikaz zahtjeva omogućuje transparentnost, lakšu provjeru i osigurava da svi zahtjevi projekta budu precizno dokumentirani i pratljivi tijekom razvoja.
Dodatno, opisani nefunkcionalni zahtjevi i zahtjevi domene primjene osiguravaju kvalitetu, sigurnost, pouzdanost i jednostavnu nadogradnju sustava.

2. Funkcionalni zahtjevi

ID zahtjeva	Opis	Prioritet	Kriteriji prihvaćanja
F-001	Sustav omogućuje administratoru stvaranje korisničkih računa - predstavnik suvlasnika i suvlasnici.	Visok	Administrator može stvoriti korisničko ime, lozinku i email adresu, a stvoreni korisnik može pristupiti sustavu.
F-002	Sustav omogućuje korisnicima promjenu lozinke pomoću korisničkog imena i stare lozinke.	Visok	Korisnik može promijeniti lozinku i pristupiti sustavu.
F-003	Sustav podržava autentifikaciju putem vanjskog servisa OAuth 2.0.	Srednji	Korisnik se može prijaviti u sustav pomoću servisa za autentifikaciju.
F-004	Sustav omogućuje korisnicima prijavu putem korisničkog imena i lozinke.	Visok	Korisnik se može prijaviti u sustav pomoću svoje lozinke i korisničkog imena.
F-005	Sustav omogućuje suvlasnicima pokretanje javne i privatne diskusije na oglasnoj ploči.	Visok	Korisnik može kreirati temu, postaviti naslov, opis i odrediti parametre diskusije.
F-006	Inicijator diskusije može ograničiti broj poruka po korisniku ili zabraniti sudjelovanje određenih suvlasnika.	Visok	Inicijator može postaviti parametre diskusije.
F-007	Sustav omogućuje pokretanje privatne diskusije vidljive samo odabranim sudionicima.	Visok	Samo korisnici na listi mogu čitati i komentirati privatnu diskusiju.

ID zahtjeva	Opis	Prioritet	Kriteriji prihvaćanja
F-008	Sustav automatski šalje email poruku suvlasnicima koji su dodani u privatnu diskusiju.	Srednji	Korisnik koji je dodan u diskusiju prima email obavijest.
F-009	Sustav omogućuje pokretanje glasanja unutar diskusije.	Visok	Inicijator može pokrenuti glasanje s pozitivnim i negativnim odgovorom. Glasovi se broje i prikaz se mijenja u stvarnom vremenu.
F-010	Sustav omogućuje prikaz trenutnog broja pozitivnih i negativnih glasova.	Nizak	Nakon svakog glasa, broj glasova se ažurira na sučelju.
F-011	Ako broj pozitivnih glasova premašuje 25%, može se sazvati sastanak suvlasnika.	Nizak	Sustav omogućuje kreiranje zahtjeva za sastanak ako ima više od 25% pozitivnih glasova.
F-012	Sustav koristi vanjsku aplikaciju StanPlan radi kreiranja sastanaka.	Visok	Sustav šalje podatke (naslov, termin, dnevni red, ciljeve) putem sučelja StanPlan.
F-013	Administrator unosi adresu servera StanPlan aplikacije radi integracije.	Srednji	Administrator može upisati adresu servera StanPlan.
F-014	Sustav omogućuje preuzimanje liste diskusija s pozitivnim rezultatom glasanja.	Nizak	Sustav šalje listu s naslovom, pitanjem i poveznicom na diskusiju.
F-015	Sustav omogućuje pregled svih diskusija.	Visok	Korisnik može vidjeti sve diskusije na oglasnoj ploči.
F-016	Sustav vodi evidenciju broja poruka po sudioniku u diskusiji.	Srednji	Broj poruka po sudioniku se automatski bilježi i prikazuje.
F-017	Sustav omogućuje upload datoteka uz poruke (slike, dokumenti).	Srednji	Datoteka se uspješno sprema i povezuje s odgovarajućom porukom, vidljiva sudionicima diskusije.
F-018	Administrator može dodavati i uređivati uloge korisnika.	Srednji	Administrator može mijenjati uloge i ovlasti korisnika.

3. Nefunkcionalni zahtjevi i zahtjevi domene primjene

ID zahtjeva	Opis	Prioritet	Kriteriji prihvaćanja
N-001	Sustav mora biti dostupan 24/7 uz minimalne zastoje.	Visok	Sustav je dostupan većinu vremena, downtime < 2h mjesečno.

ID zahtjeva	Opis	Prioritet	Kriteriji prihvaćanja
N-002	Aplikacija mora biti responzivna i raditi na mobilnim i desktop uređajima.	Visok	Sučelje se pravilno prikazuje na svim tipičnim rezolucijama.
N-003	Sustav mora koristiti siguran protokol (HTTPS) i enkripciju korisničkih podataka.	Visok	Komunikacija je uvijek enkriptirana, lozinke i tokeni su sigurni.
N-004	Sustav podržava više jezika korisničkog sučelja (minimalno hrvatski i engleski).	Srednji	Korisnik može odabrati jezik sučelja, tekstovi su prikazani ispravno.
N-005	Sustav mora imati vrijeme odziva do 2 sekunde za tipične operacije (pregled diskusija, glasanje, učitavanje stranice).	Visok	Sučelje se učitava i operacije izvršavaju brzo bez primjetnog kašnjenja.
N-006	Sustav mora biti kompatibilan s najčešćim web preglednicima i mobilnim platformama (Chrome, Firefox, Edge, iOS, Android).	Visok	Aplikacija se ispravno prikazuje i funkcionira na navedenim platformama.
N-007	Sustav mora imati audit log za praćenje aktivnosti korisnika.	Srednji	Sve aktivnosti korisnika su zapisane u logu za reviziju.
N-008	Sustav mora podržavati ograničen broj korisnika primjeren opsegu studentskog projekta.	Nizak	Sustav može posluživati tipičan broj korisnika u testnom okruženju (do 50-100 korisnika).
N-009	Sustav mora poštivati standarde kvalitete i sigurnosne protokole te biti skalabilan za moguće buduće nadogradnje.	Srednji	Sustav je razvijen u skladu sa sigurnosnim standardima i modularnošću.

4. Zahtjevi za održavanjem

ID zahtjeva	Opis	Prioritet	Kriteriji prihvaćanja
M-001	Sustav mora imati dokumentaciju za administratore i programere.	Visok	Dokumentacija je dostupna i ažurirana.
M-002	Sustav mora omogućiti jednostavno ažuriranje i nadogradnju baze podataka i aplikacije.	Visok	Novi update se može instalirati bez gubitka podataka ili funkcionalnosti.
M-003	Sustav mora imati procedure za backup i oporavak podataka.	Visok	Podaci se mogu obnoviti iz backupa u slučaju kvara.

ID zahtjeva	Opis	Prioritet	Kriteriji prihvaćanja
M-004	Sustav mora omogućiti praćenje grešaka i anomalija.	Srednji	Sustav generira logove i upozorenja za eventualne probleme.

5. Dionici

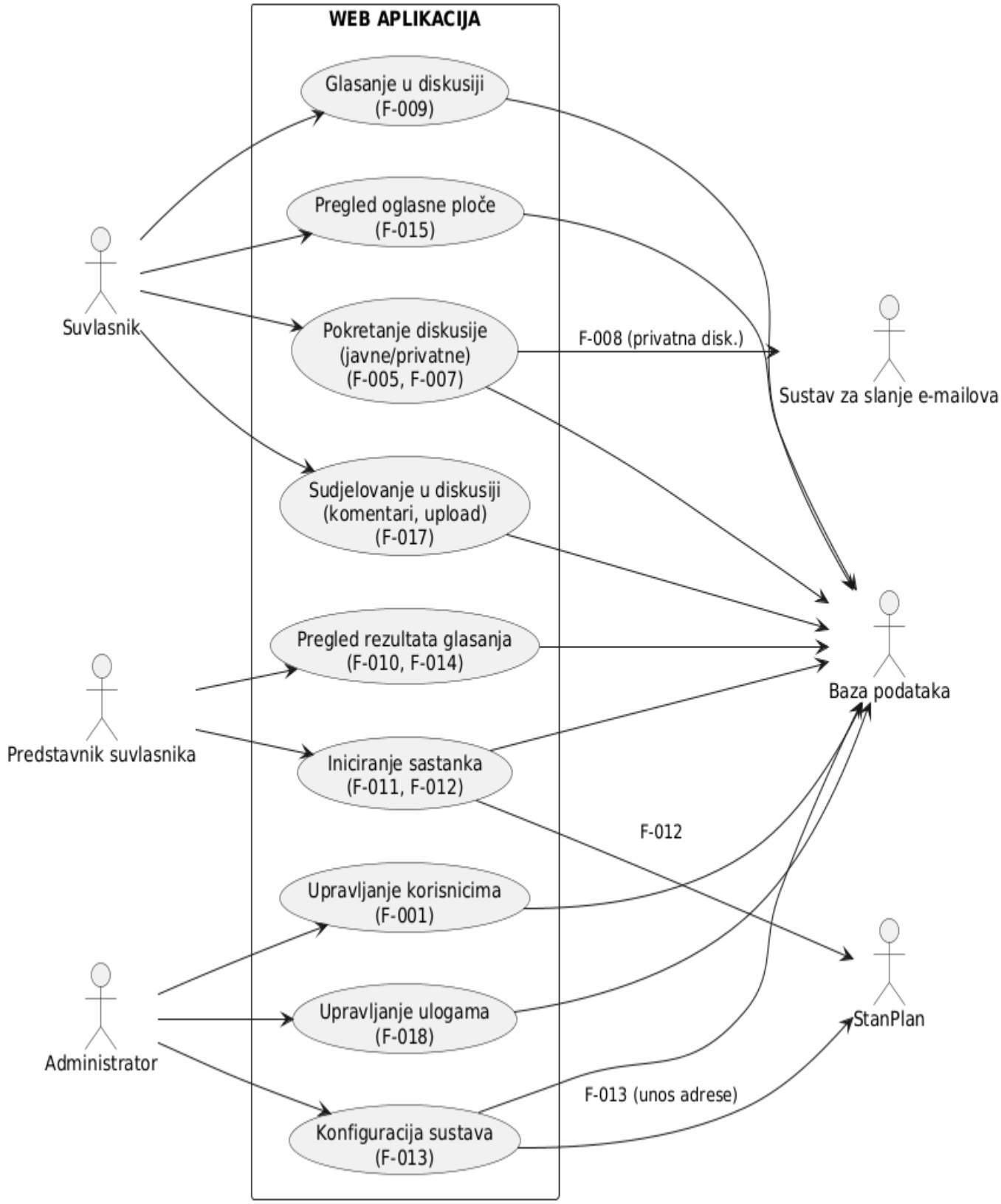
Dionik	Uloga / Interes
Suvlasnici	Glavni korisnici aplikacije; prijavljuju probleme, sudjeluju u diskusijama i glasanjima.
Predstavnici suvlasnika	Koordiniraju održavanje, pregledavaju diskusije i iniciraju sastanke.
Administrator	Upravljanje korisnicima, integracijama i održavanjem sustava.
Upravitelji zgrada	Koriste aplikaciju za pregled problema i planiranje održavanja (opcionalno).
Razvojni tim	Razvija i održava aplikaciju, implementira nove funkcionalnosti i ispravke bugova.

Obrasci uporabe

Dijagrami obrazaca uporabe

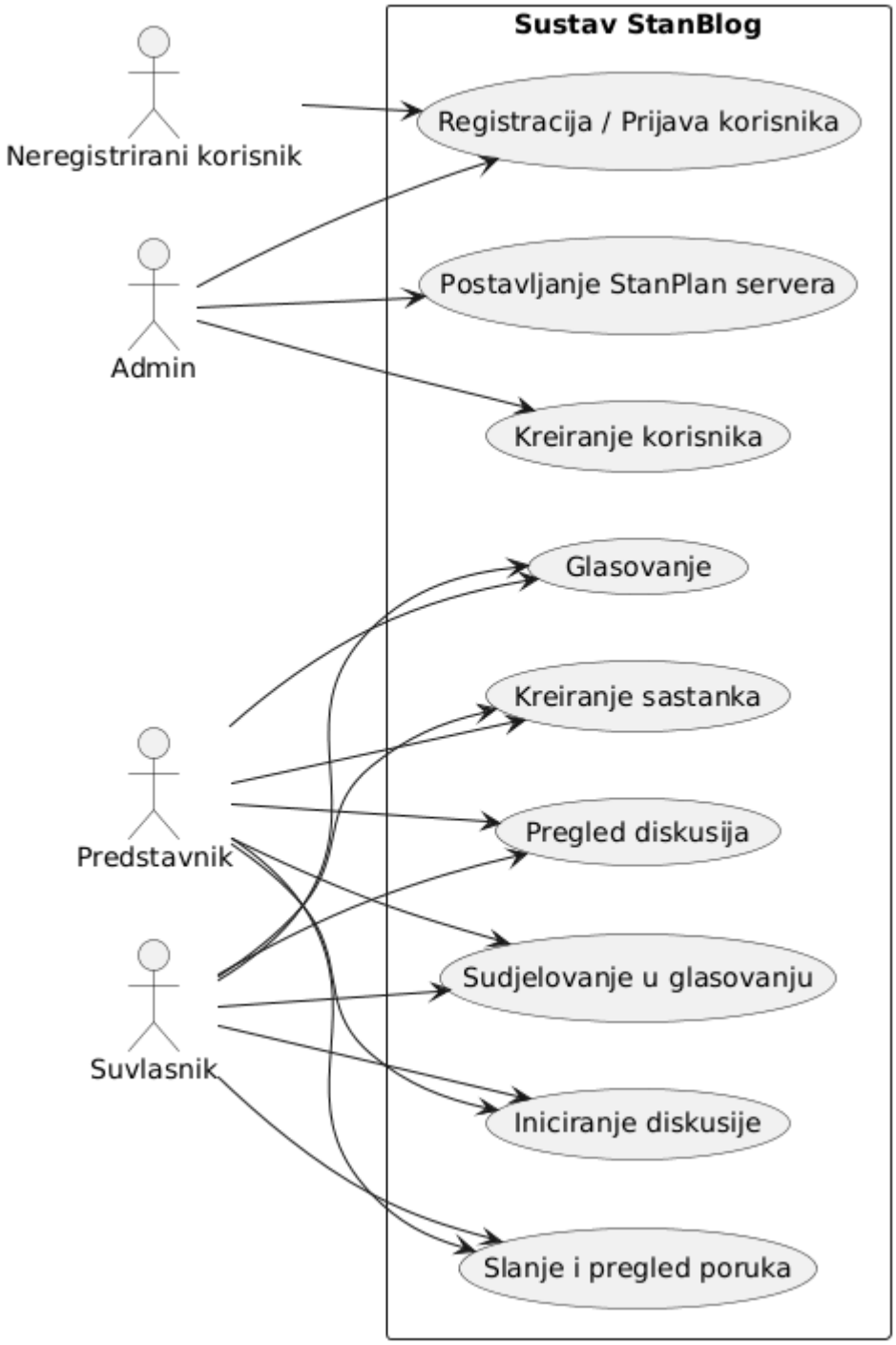
Prikazati odnos aktora i obrazaca uporabe odgovarajućim UML dijagramom. Modelirati po razinama apstrakcije i skupovima srodnih funkcionalnosti.

1. Visokorazinski dijagram obrazaca uporabe cijelog sustava



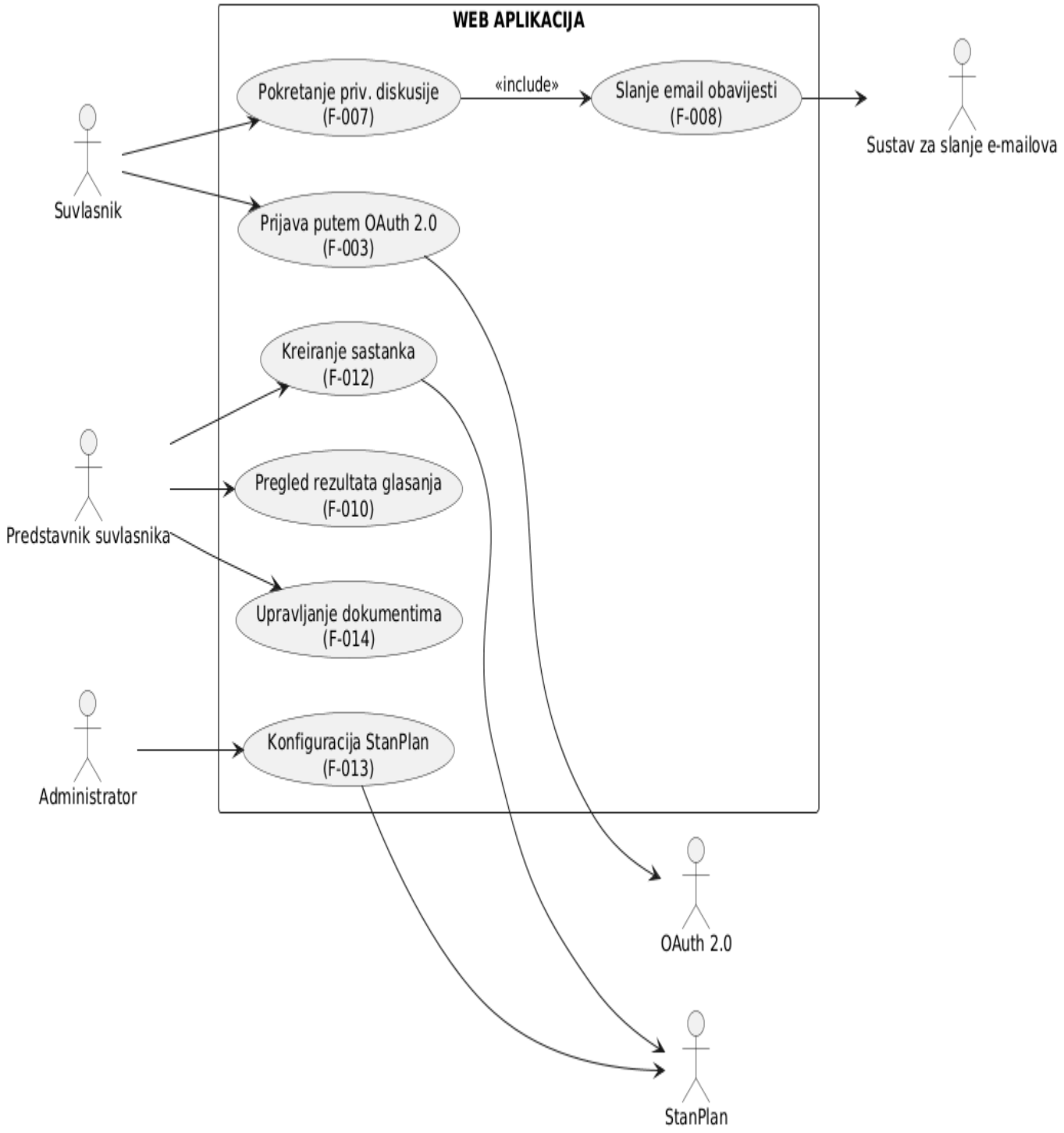
Dijagram prikazuje glavne funkcionalnosti web aplikacije i interakciju aktera s njima. Suvlasnici mogu pregledavati oglasnu ploču, pokretati i sudjelovati u diskusijama te glasati u njima. Predstavnici suvlasnika iniciraju sastanke i pregledavaju rezultate glasanja, dok administratori upravljaju korisnicima, ulogama i konfiguracijom sustava. Sustav je povezan s vanjskim servisima poput StanPlana i sustava za slanje e-mailova, a svi use case-ovi koriste bazu podataka za pohranu i dohvat informacija. Dijagram jasno prikazuje sve ključne funkcionalnosti i njihove međusobne veze.

2. dijagram obrazaca uporabe za korisničke uloge



Dijagram obrazaca uporabe za korisničke uloge – StanBlog sustav prikazuje glavne funkcionalnosti i interakciju različitih korisničkih uloga s aplikacijom. Administratori upravljaju korisnicima i konfiguracijom sustava, Predstavnici suvlasnika kreiraju sastanke i koordiniraju aktivnosti, dok Suvlasnici pregledavaju i pokreću diskusije te sudjeluju u glasovanju. Neregistrirani korisnici imaju ograničen pristup i mogu se registrirati. Dijagram jasno definira uloge, njihove funkcionalnosti i međusobne veze unutar sustava.

3. dijagram obrazaca uporabe za kritične sustave i integracije



Dijagram prikazuje glavne funkcionalnosti web aplikacije i interakciju aktera s ključnim vanjskim sustavima. Suvlasnici se prijavljuju putem OAuth 2.0 i mogu pokretati privatne diskusije, pri čemu se automatski šalju e-mail obavijesti. Predstavnici suvlasnika kreiraju sastanke, pregledavaju rezultate glasanja i upravljaju dokumentima. Administratori konfiguriraju integraciju sa StanPlan sustavom. Dijagram jasno pokazuje tko inicira akcije, kako se funkcionalnosti povezuju unutar sustava i kako aplikacija komunicira s vanjskim servisima, naglašavajući kritične integracije.

Opis obrazaca uporabe

UC01 – Registracija/Prijava

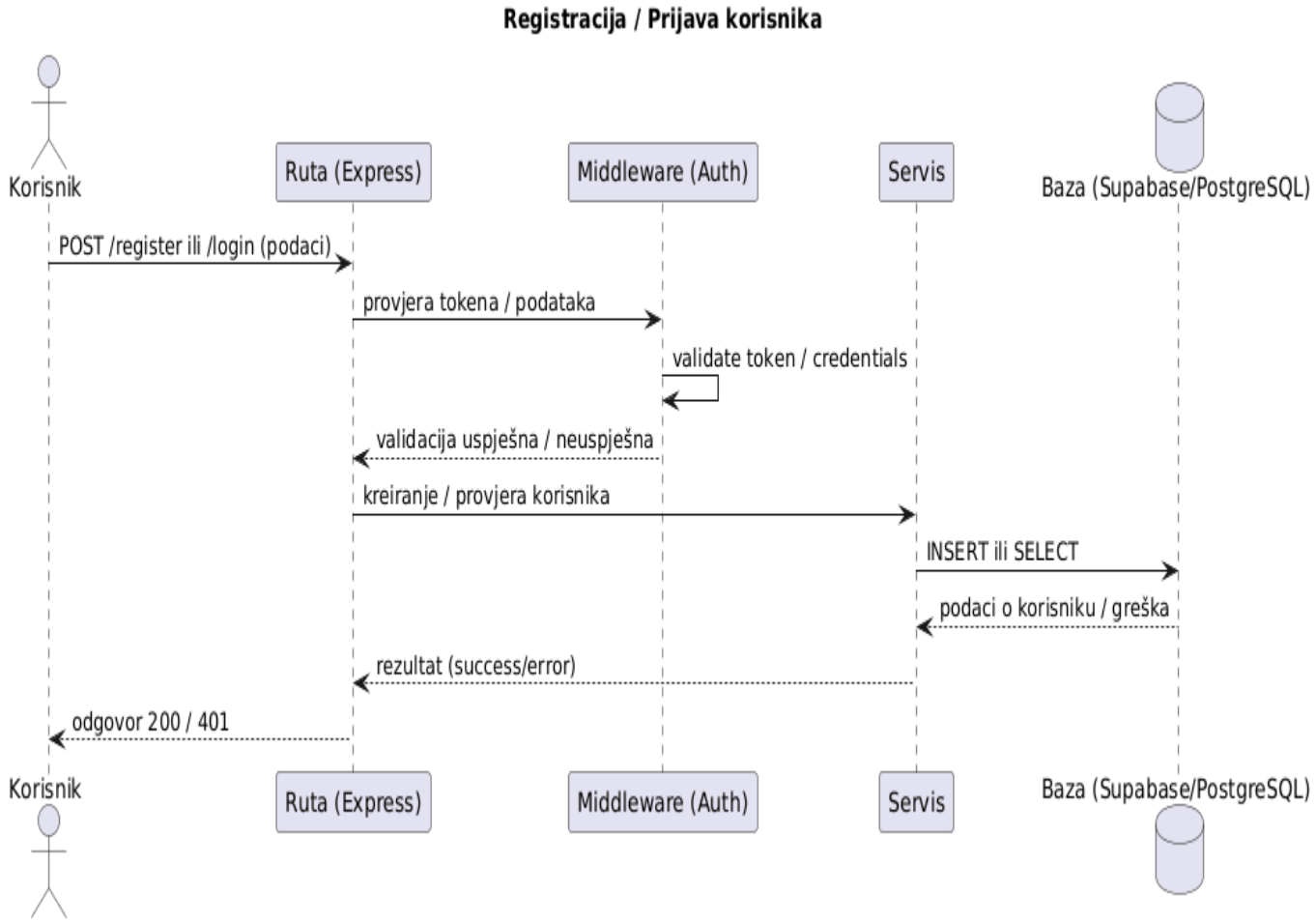
Glavni sudionik: Korisnik
Cilj: Prijaviti se u sustav
Sudionici: Korisnik, Sustav, Autentifikacijski servis
Preduvjet: Korisnik ima kreiran račun

Osnovni tijek

1. Korisnik unosi korisničko ime i lozinku.
2. Sustav provjerava vjerodajnice.
3. Sustav generira i vraća token o uspješnoj prijavi.
4. Korisnik se preusmjerava na početnu stranicu.

Odstupanja

- **1a.** Pogrešna lozinka → Poruka o grešci.
- **2a.** Korisnik ne postoji → Prikaz upozorenja.
- **3a.** OAuth servis nedostupan → Sustav nudi lokalnu prijavu.



UC02 – Kreiranje korisnika (Admin)

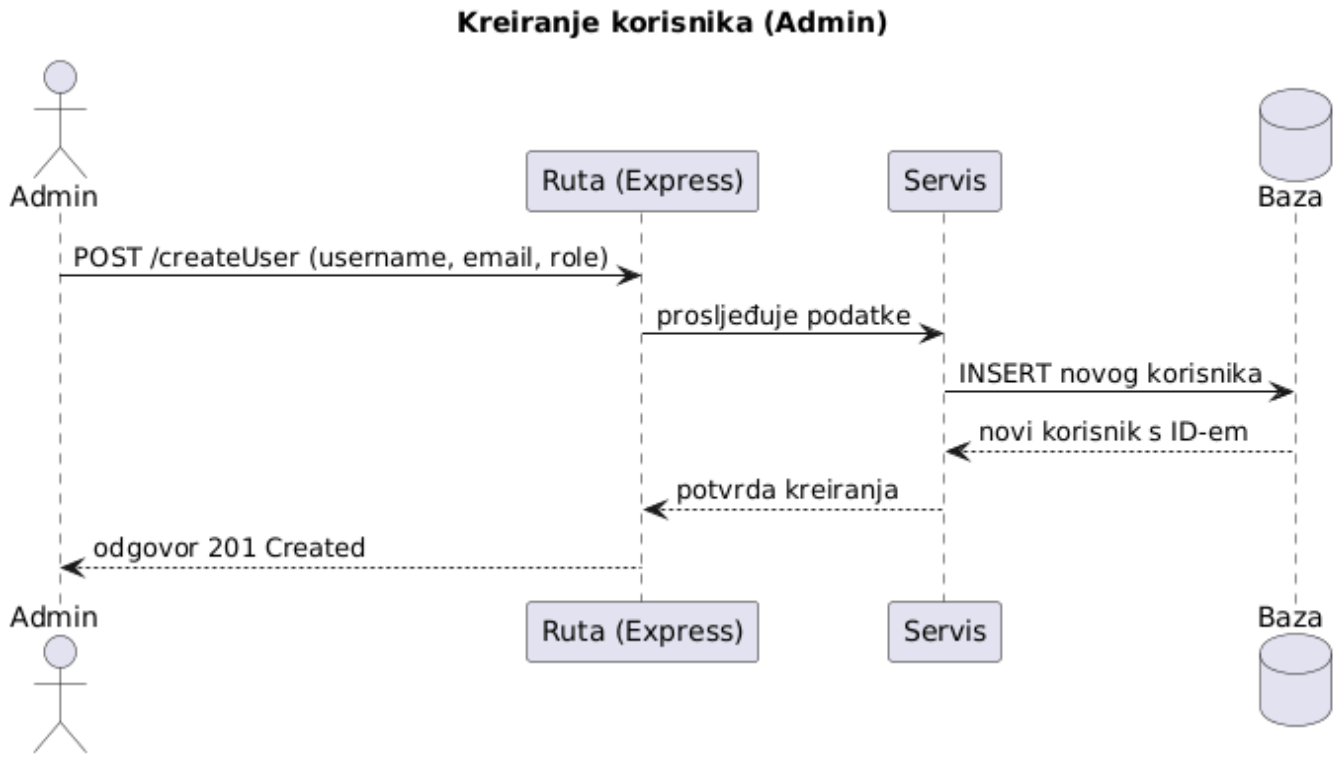
Glavni sudionik: Administrator
Cilj: Stvoriti račun suvlasnika ili predstavnika suvlasnika
Sudionici: Administrator, Sustav
Preduvjet: Administrator je prijavljen u sustav

Osnovni tijek

1. Administrator odabire opciju „Dodaj korisnika”.
2. Sustav prikazuje formu za unos podataka.
3. Administrator unosi korisničko ime, email i ulogu.
4. Sustav provjerava ispravnost podataka.
5. Sustav upisuje podatke u bazu i kreira korisnika.

Odstupanja

- **1a.** Podaci nisu ispravni → Sustav javlja grešku i vraća na unos.
- **3a.** Email već postoji → Sustav javlja poruku i traži drugi email.



UC03 – Iniciranje diskusije

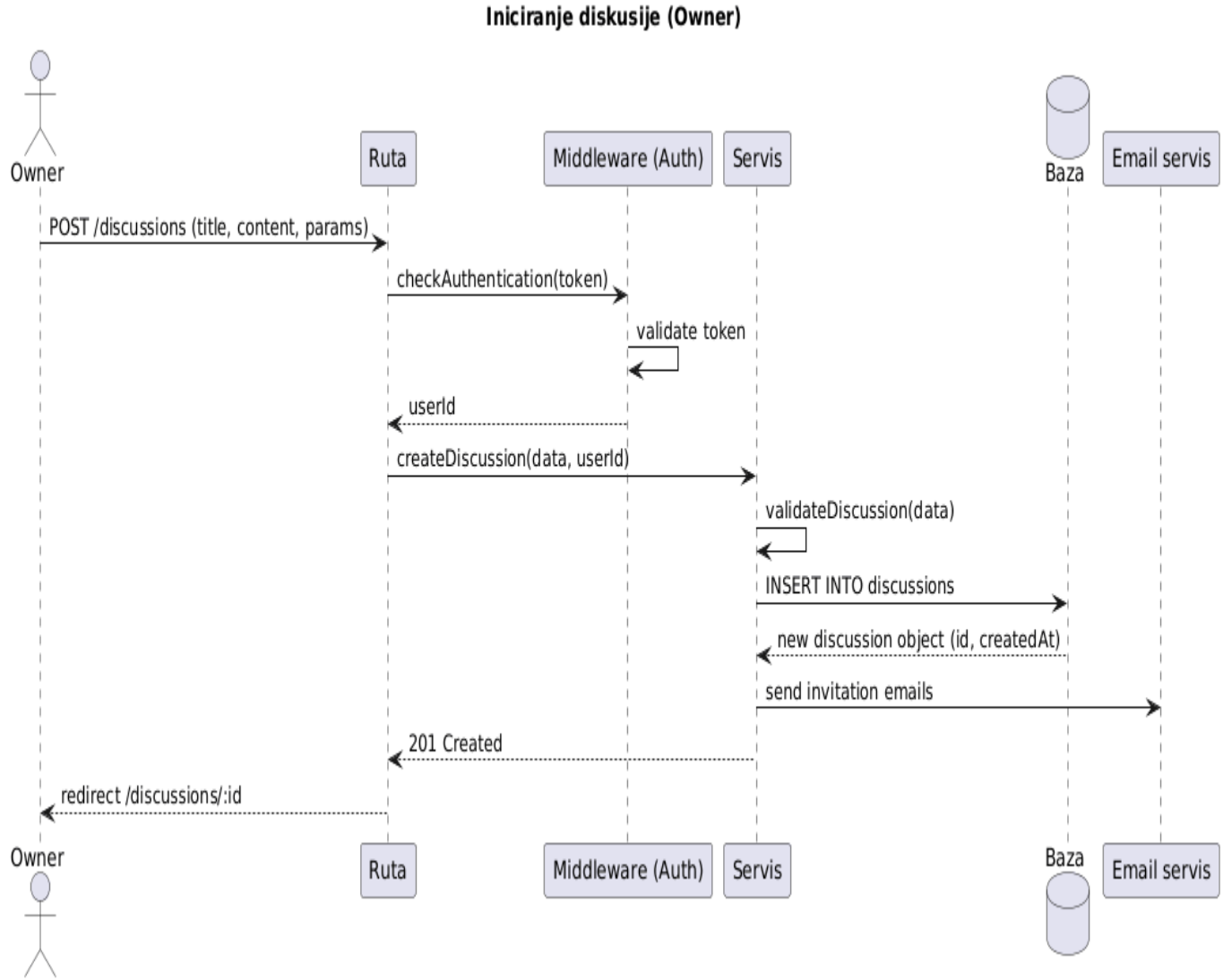
Glavni sudionik: Suvlasnik
Cilj: Kreirati javnu ili privatnu diskusiju
Sudionici: Suvlasnik, Sustav, Email servis
Preduvjet: Korisnik je prijavljen

Osnovni tijek

1. Suvlasnik odabire opciju „Nova diskusija”.
2. Suvlasnik unosi naslov, opis i parametre.
3. Sustav provjerava unose.
4. Sustav sprema diskusiju u bazu.
5. Ako je privatna — sustav šalje email pozvanim korisnicima.

Odstupanja

- **2a.** Nedostaje naslov → Sustav traži ispravak.
- **4a.** Baza nedostupna → Sustav prikazuje grešku.



UC04 – Upravljanje sudionicima

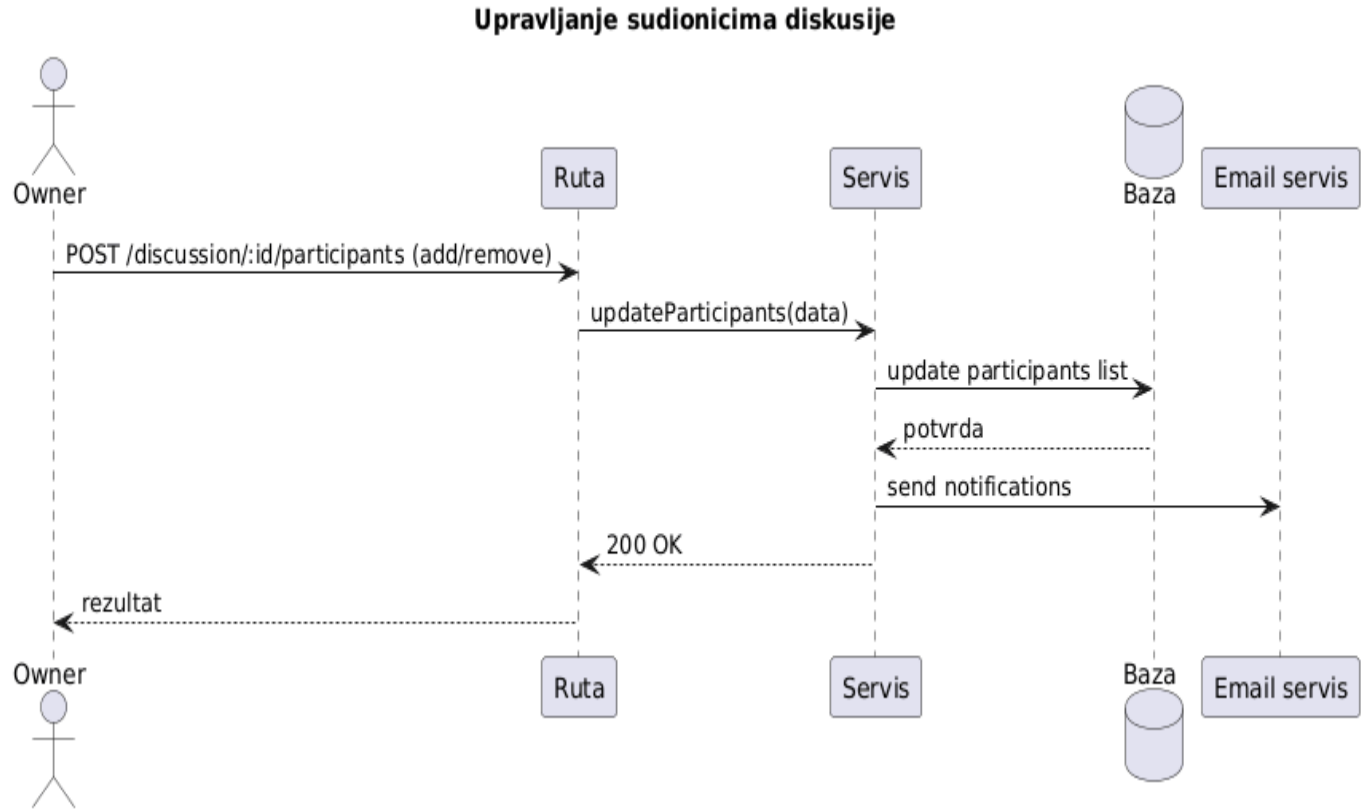
Glavni sudionik: Inicijator diskusije
Cilj: Dodati/ukloniti sudionike privatne diskusije
Sudionici: Inicijator, Sustav, Email servis
Preduvjet: Diskusija je privatna

Osnovni tijek

1. Inicijator otvara postavke diskusije.
2. Odabire dodavanje ili uklanjanje sudionika.
3. Sustav ažurira popis sudionika u bazi.
4. Sustav šalje email obavijest novim sudionicima.

Odstupanja

- **2a.** Korisnik već sudjeluje → Sustav javlja poruku.
- **3a.** Baza nedostupna → Sustav javlja problem.



UC05 – Slanje i pregled poruka

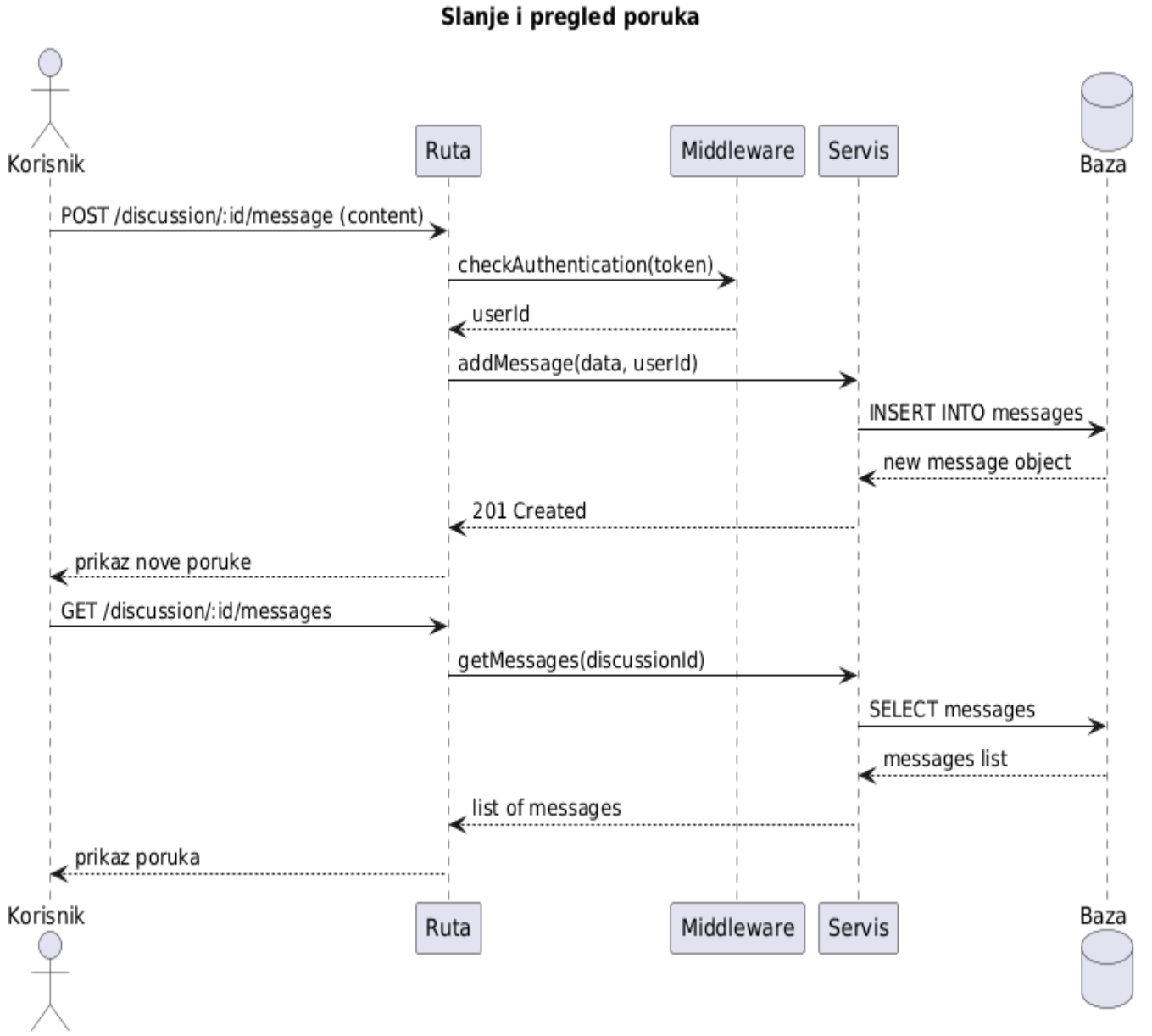
Glavni sudionik: Korisnik
Cilj: Poslati poruku u diskusiji
Sudionici: Korisnik, Sustav
Preduvjet: Korisnik ima pristup diskusiji

Osnovni tijek

1. Korisnik otvara diskusiju i unosi poruku.
2. Sustav provjerava je li korisnik ovlašten.
3. Sustav sprema poruku u bazu.
4. Sustav prikazuje novu poruku.

Odstupanja

- **2a.** Korisnik nije sudionik privatne diskusije → zabrana pristupa.
- **3a.** Neuspjeh zapisivanja → poruka o grešci.



UC06 – Glasovanje

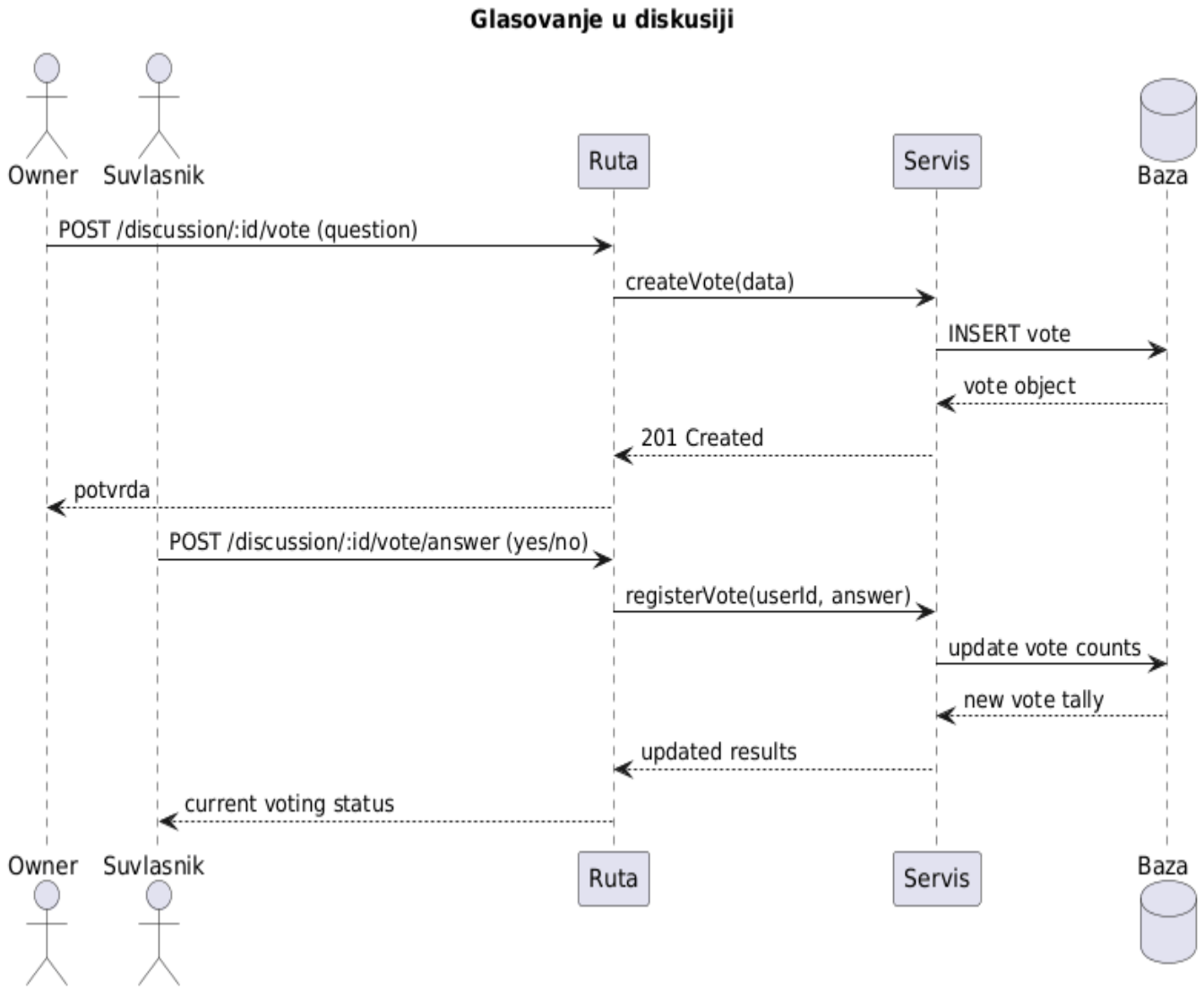
Glavni sudionik: Suvlasnik
Cilj: Glasati o prijedlogu u diskusiji
Sudionici: Suvlasnik, Sustav
Preduvjet: Glasanje je aktivno

Osnovni tijek

1. Inicijator pokreće glasanje.
2. Suvlasnik odabire svoj glas (DA/NE).
3. Sustav sprema glas.
4. Sustav ažurira broj glasova.
5. Sustav provjerava je li dosegnut prag od 25%.

Odstupanja

- **2a.** Korisnik je već glasao → sustav blokira drugi glas.
- **3a.** Neuspjeh zapisivanja → upozorenje.



UC07 – Kreiranje sastanka

Glavni sudionik: Inicijator
Cilj: Kreirati sastanak u aplikaciji StanPlan
Sudionici: Inicijator, Sustav, StanPlan API
Preduvjet: Postotak pozitivnih glasova > 25%

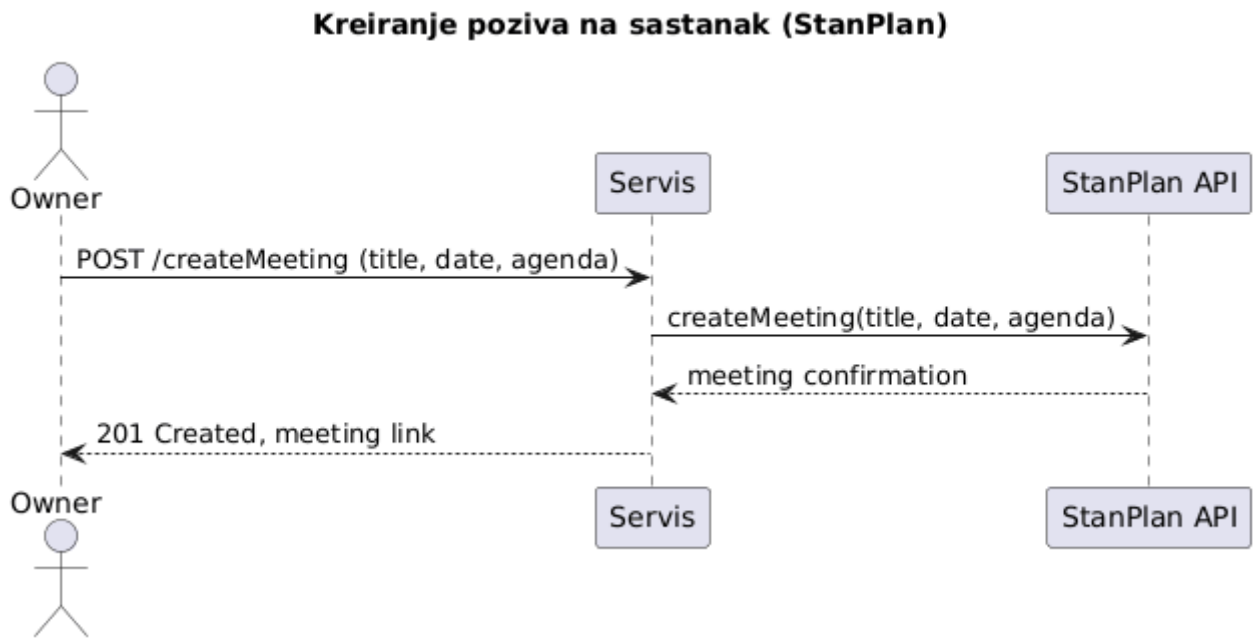
Osnovni tijek

1. Korisnik odabire „Kreiraj sastanak”.
2. Sustav priprema podatke.
3. Sustav šalje zahtjev StanPlan API-ju.
4. StanPlan API vraća link sastanka.
5. Sustav prikazuje i sprema link sastanka.

Odstupanja

- **3a.** StanPlan API nedostupan → ponovni pokušaj.

- **4a.** Povratni podaci neispravni → ručni unos.



UC08 – Pregled diskusija

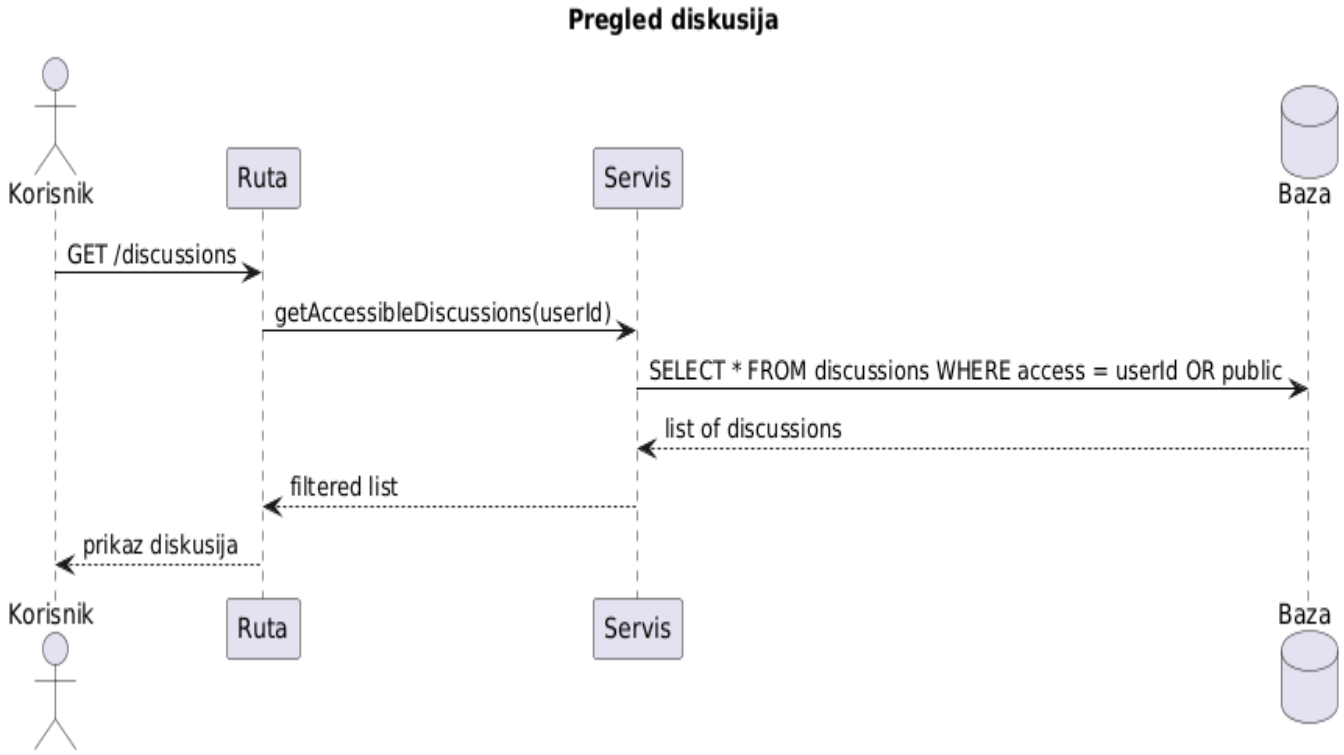
Glavni sudionik: Korisnik
Cilj: Vidjeti sve dostupne diskusije
Sudionici: Korisnik, Sustav
Preduvjet: Korisnik je prijavljen

Osnovni tijek

1. Korisnik otvara oglasnu ploču.
2. Sustav dohvaća sve diskusije.
3. Sustav filtrira privatne diskusije prema ovlaštenju.
4. Sustav prikazuje popis diskusija.

Odstupanja

- **2a.** Greška u bazi → sustav prikazuje prazan popis.



Provjera uključenosti ključnih funkcionalnosti u obrasce uporabe

UC	Funkcionalni zahtjevi
UC01 – Registracija / Prijava korisnika	F-001, F-002, F-003, F-004
UC02 – Kreiranje korisnika (Admin)	F-001, F-003
UC03 – Iniciranje diskusije	F-005, F-006, F-007, F-008
UC04 – Upravljanje sudionicima diskusije	F-006, F-007, F-008
UC05 – Slanje i pregled poruka	F-005
UC06 – Glasovanje	F-009, F-010, F-011
UC07 – Kreiranje sastanka	F-012, F-013
UC08 – Pregled diskusija	F-014, F-015

Arhitektura sustava

Opis arhitekture

•Stil arhitekture: Unutar ovog projekta korištena je klijent–poslužitelj arhitektura. Za ovaj pristup odlučili smo se zbog jednostavnosti razvoja, održivosti i potrebe za modularnošću, koja bi bila teško ostvariva u monolitskoj arhitekturi. S obzirom na relativno mali opseg i jednostavnost projekta, nije bilo potrebe za više od jednog backend servisa, pa samim time ni za primjenom mikrouslužne arhitekture.

• Podsustavi: Backend koristi Express.js, Node.js Frontend koristi EJS te Javascript. Kasnije će biti implementiran React. Za bazu podataka koristimo Supabase te PostgreSQL relacijsku bazu podataka. Aplikacija će biti pokrenuta na Azure radnoj platformi jer adekvatno zadovoljava sve potrebe i zahtjeve naše aplikacije.

• Spremišta podataka: Sustav koristi relacijsku bazu podataka, implementiranu putem Supabase platforme koja je kompatibilna s PostgreSQL-om. Podaci se pohranjuju u jasno definirane tablice kao što su korisnici, zgrade, rasprave, poruke i glasovi, pri čemu se koriste primarni i vanjski ključevi radi osiguravanja referencijalnog integriteta. Ovakav model omogućuje strogu strukturu podataka, pouzdane veze među entitetima i učinkovito izvršavanje upita. Budući da sustav zahtijeva transakcijsku konzistentnost, kontrolu pristupa i jasne odnose između entiteta, relacijska baza se pokazala najboljim izborom u odnosu na NoSQL rješenja.

• Mrežni protokoli: Za komunikaciju između klijenta i poslužitelja koristi se HTTP protokol, pri čemu se podaci razmjenjuju putem REST API ruta definiranih u Express.js aplikaciji.
• Globalni upravljački tok: Korisnik putem preglednika šalje zahtjev (npr. za raspravu), koji backend obrađuje kroz odgovarajući podsustav, dohvaća podatke iz baze, te vraća rezultat u obliku stranice.

• Sklopovskoprogramski zahtjevi:

- Programski zahtjevi:
  - Poslužiteljski dio (backend)
    - Jezik i runtime: Node.js (verzija ≥ 18.x LTS, preporučeno 20.x)
    - Framework: Express.js 5.1.0
    - Render engine: EJS 3.1.10
    - Baza podataka: PostgreSQL (putem paketa pg i postgres)
    - Autentifikacija i sigurnost: JSON Web Tokens (jsonwebtoken) te hashiranje lozinki pomoću bcryptjs
    - Cloud integracija: Supabase (@supabase/supabase-js)
    - Konfiguracija: putem .env datoteka (paket dotenv)
    - Razvojni alati: nodemon za automatsko ponovno pokretanje poslužitelja tijekom razvoja
  - Klijentski dio (frontend)
    - Trenutno koristi EJS, HTML i CSS
    - Planirana je buduća migracija na React (library)
    - Klijent komunicira s poslužiteljem putem HTTP/HTTPS protokola (REST API)
  - Operacijski sustav:
    - Kompatibilno s Windows 10+, macOS i Linux (npr. Ubuntu 20.04+)
    - Za produkcijsko okruženje preporučuje se Linux zbog stabilnosti i performansi

    - Dodatni alati:
        - Git za verzioniranje koda
        - Docker (opcionalno) za jednostavnije postavljanje produkcijskog okruženja
        - NPM (verzija ≥ 9.x) za upravljanje ovisnostima

- Sklopovski zahtjevi:

    - Za razvojno okruženje:
        - Procesor: minimalno 1 jezgra @ 2.0 GHze
        - Memorija: 1 GB RAM-a
        - Diskovni prostor: 500 MB slobodnog prostora
        - Internetska veza (za preuzimanje paketa i pristup Supabase servisu)
    - Za produkcijsko okruženje:
        - Procesor: 2+ jezgre @ 2.0 GHz
        - Memorija: 2–4 GB RAM-a (ovisno o broju istovremenih korisnika)
        - Diskovni prostor: 1 GB slobodnog prostora
        - Stalna mrežna povezanost (HTTP/HTTPS, port 3000 ili konfigurirani port)
        - Preporučeni deployment: Linux poslužitelj ili cloud platforma (npr. Vercel, Render, Supabase Hosting)

Obrazloženje odabira arhitekture

Glavni principi kojima smo se vodili prilikom odabira arhitekture bili su održljivosti i modularnost. Bilo nam je izuzetno bitno imati jasan, čitki kod tako da možemo povezati naše dijelove bez većih problema. Modularnost je bitna radi neovisnog razvijanja dijelova koje možemo spojiti na aplikaciju i nastaviti njen razvoj.
Odlučili smo se za ove arhitekture jer pružaju modularnost, jednostavnost i održljivost, a pružaju mnogo mogućnosti za moćno ostvarivanje web aplikacije.
Nismo razmatrili druge arhitekture jer nam ova arhitektura pruža sve što trebamo bez značajnih poteškoća, jednostavna je za korištenje i zadovoljava sve naše potrebe.

Organizacija sustava na visokoj razini

- Klijent-poslužitelj: Sustav je organiziran prema klijent-poslužitelj arhitekturi, gdje klijent (web preglednik) šalje zahtjeve putem HTTP protokola, a poslužitelj ih obrađuje pomoću Express.js aplikacije te vraća podatke ili renderirane stranice. Poslužitelj upravlja poslovnom logikom, autentikacijom i komunikacijom s bazom podataka.
- Baza podataka: U sustavu se koristi relacijska baza podataka PostgreSQL, koja služi za pohranu i dohvat strukturiranih podataka poput korisnika, rasprava, poruka, glasova i zgrada. Pristup bazi ostvaren je pomoću pg i postgres paketa, čime se omogućuje sigurna i učinkovita manipulacija podacima.
- Datotečni sustav: Supabase, PostgreSQL
- Grafičko sučelje: Korisničko sučelje temeljeno je na EJS i Javascript datotekama koje se renderiraju na poslužitelju i prikazuju korisniku putem preglednika. Sučelje ostvaruje korištenje funkcionalnosti sustava kao što su prijava, pregleda rasprava, glasanje i administracija putem REST API poziva.

Organizacija aplikacije

Opišite organizaciju aplikacije na složenijoj razini, uključujući strukturu slojeva i komponenata aplikacije:

• Frontend i Backend slojevi:

Backend sloj aplikacije organiziran je unutar direktorija /backend i temelji se na Node.js okruženju s frameworkom Express.js. Ovaj sloj predstavlja logičko središte sustava i zadužen je za obradu zahtjeva korisnika, komunikaciju s bazom podataka te generiranje odgovora koji se prosljeđuju frontend dijelu.

Struktura backend sloja:

- /auth – implementacija autentifikacije i autorizacije korisnika (JWT tokeni, bcrypt hashiranje lozinki)
- /data – sloj za pristup i upravljanje podacima iz baze
- /middleware – Express middleware funkcije za provjeru tokena, logiranje i obradu grešaka
- /routes – definicije REST API ruta koje povezuju zahtjeve korisnika s logikom aplikacije
- /services – pomoćne usluge i poslovna logika koje backend koristi za obradu podataka
- /utils – zajedničke pomoćne funkcije i alati
- server.js – glavni ulazni modul koji pokreće Express poslužitelj i inicijalizira povezivanje sa Supabaseom i bazom podataka

Odgovornosti backend sloja:

- Obrada HTTP zahtjeva i slanje odgovora klijentu *Upravljanje autentifikacijom i sesijama
- Sigurna pohrana i dohvat podataka iz baze
- Generiranje dinamičkih sadržaja (EJS) ili JSON odgovora
- Upravljanje konfiguracijom putem .env datoteka (pomoću paketa dotenv)
- Integracija s Supabase servisom kao cloud rješenjem za autentifikaciju i bazu

Frontend sloj nalazi se unutar direktorija /frontend i trenutno koristi EJS, HTML i CSS tehnologije za prikaz dinamičkih stranica generiranih na poslužitelju. Ovaj sloj je zadužen za korisničku interakciju te prikaz podataka koje backend vraća nakon obrade.

Struktura frontend sloja:

- /views – EJS predlošci koji definiraju izgled stranica i dinamički prikazuju podatke dobivene od backend dijela
- /public – statičke datoteke poput CSS-a, slika i JavaScript skripti

Odgovornosti frontend sloja:

- Prikaz korisničkog sučelja i rukovanje interakcijama korisnika
- Slanje zahtjeva backendu putem HTTP/HTTPS protokola (REST API)
- Prikaz rezultata koje backend vraća – u obliku renderirane HTML stranice ili JSON podataka

Planira se prepisivanje frontenda na React, čime će se dodatno odvojiti od poslužiteljske logike i omogućiti komponentni pristup razvoju korisničkog sučelja

Interakcija između slojeva:

Frontend i backend međusobno komuniciraju isključivo putem HTTP/HTTPS protokola, pri čemu backend izlaže REST API rute definirane u Express.js aplikaciji. Korisnički zahtjev (npr. prijava, dohvat podataka ili objava sadržaja) prolazi kroz backend sloj, obrađuje se u odgovarajućem servisu te se vraća kao HTML prikaz ili JSON odgovor koji frontend prikazuje korisniku.

• MVC arhitektura:

Aplikacija koristi MVC (Model-View-Controller) arhitekturu, koja omogućuje jasno odvajanje poslovne logike, korisničkog sučelja i pristupa podacima, što olakšava održavanje i proširivost sustava.

Model (M)

Model sloj implementiran je unutar direktorija /data backend dijela aplikacije.
Odgovoran je za:
• Definiranje strukture podataka (npr. korisnici, rasprave, komentari)
• Interakciju s bazom podataka (PostgreSQL putem paketa pg i postgres)
• Operacije poput dohvaćanja, spremanja, ažuriranja i brisanja podataka

Model omogućuje da se poslovna logika backend sloja odvaja od načina na koji se podaci fizički pohranjuju i dohvaćaju.

View (V)

View sloj implementiran je u direktoriju /frontend/views putem EJS predložaka, koji renderiraju HTML stranice za korisničko sučelje. Odgovornosti view sloja:

- Prikaz podataka dobivenih iz backend-a na pregledniku korisnika
- Dinamičko generiranje sadržaja pomoću EJS izraza
- Održavanje izgleda stranica i korisničkog iskustva

View sloj je potpuno odvojen od poslovne logike, što omogućuje promjenu dizajna ili frontend tehnologije (npr. prelazak na React) bez izmjene backend logike.

Controller (C)

Controller sloj organiziran je unutar direktorija /routes i djelomično kroz /services. Odgovoran je za:

- Primanje zahtjeva od frontend sloja (HTTP GET/POST/PUT/DELETE)
- Pozivanje odgovarajućih model funkcija za obradu podataka
- Pripremu podataka za view ili kao JSON odgovor klijentu
- Upravljanje poslovnom logikom i provjerom autentifikacije (pomoću middleware funkcija)

Controller povezuje frontend i backend, te osigurava da se logika prezentacije i logika podataka odvajaju od same obrade zahtjeva.

Baza podataka

Vrsta i implementacija baze

Odabrana baza podataka je **PostgreSQL**, implementirana preko **Supabase** platforme u oblaku. Za primarne ključeve koristi se tip **UUID** radi jedinstvene identifikacije unutar svih tablica. Baza je strukturirana tako da omogućuje učinkovito praćenje korisnika, zgrada, diskusija, poruka, anketa i glasova.

Tablice baze

Tablica: app_user

Atribut	Tip podatka	Opis varijable
id	UUID	Primarni ključ (PK), jedinstveni identifikator korisnika
first_name	VARCHAR	Ime korisnika
last_name	VARCHAR	Prezime korisnika
email	VARCHAR	Jedinstvena email adresa korisnika, UNIQUE
phone_number	VARCHAR	Broj telefona korisnika
role	role_t	Uloga korisnika (admin, suvlasnik, predstavnik)
created_at	TIMESTAMPTZ	Datum i vrijeme stvaranja korisničkog računa
password_hash	TEXT	Hash lozinke korisnika

Tablica: building

Atribut	Tip podatka	Opis varijable
id	UUID	Primarni ključ (PK), jedinstveni identifikator zgrade
name	VARCHAR	Naziv zgrade
address	VARCHAR	Adresa zgrade
created_at	TIMESTAMPTZ	Datum i vrijeme unosa zgrade

Tablica: building_membership

Atribut	Tip podatka	Opis varijable
building_id	UUID	Primarni i strani ključ (FK), povezuje s tablicom building
user_id	UUID	Primarni i strani ključ (FK), povezuje s tablicom app_user
user_role	role_t	Uloga korisnika unutar zgrade

Tablica: discussion

Atribut	Tip podatka	Opis varijable
id	UUID	Primarni ključ (PK), jedinstveni identifikator diskusije
title	VARCHAR	Naslov diskusije, UNIQUE u kombinaciji s building_id
poll_description	VARCHAR	Opis ankete u diskusiji
building_id	UUID	Strani ključ (FK), povezuje s tablicom building
owner_id	UUID	Strani ključ (FK), povezuje s tablicom app_user, vlasnik diskusije
visibility	visibility_t	Vidljivost diskusije (javno/privatno)
status	discussion_status_t	Status diskusije (otvoreno/zatvoreno)

Atribut	Tip podatka	Opis varijable
created_at	TIMESTAMPTZ	Datum i vrijeme kreiranja diskusije

Tablica: discussion_participant

Atribut	Tip podatka	Opis varijable
discussion_id	UUID	Primarni i strani ključ (FK), povezuje s tablicom discussion
user_id	UUID	Primarni i strani ključ (FK), povezuje s tablicom app_user
can_post	BOOLEAN	Omogućuje korisniku pisanje poruka u diskusiji
number_of_messages	INTEGER	Broj poruka koje je korisnik poslao, mora biti >= 0

Tablica: message

Atribut	Tip podatka	Opis varijable
id	UUID	Primarni ključ (PK), jedinstveni identifikator poruke
discussion_id	UUID	Strani ključ (FK), povezuje s tablicom discussion
author_id	UUID	Strani ključ (FK), povezuje s tablicom app_user, autor poruke
body	TEXT	Sadržaj poruke
created_at	TIMESTAMPTZ	Datum i vrijeme slanja poruke

Tablica: poll

Atribut	Tip podatka	Opis varijable
id	UUID	Primarni ključ (PK), jedinstveni identifikator ankete
discussion_id	UUID	Strani ključ (FK), povezuje s tablicom discussion
author_id	UUID	Strani ključ (FK), povezuje s tablicom app_user, autor ankete
question	TEXT	Pitanje ankete
created_at	TIMESTAMPTZ	Datum i vrijeme kreiranja ankete
closed	BOOLEAN	Status ankete (true ako je zatvorena)

Tablica: upload

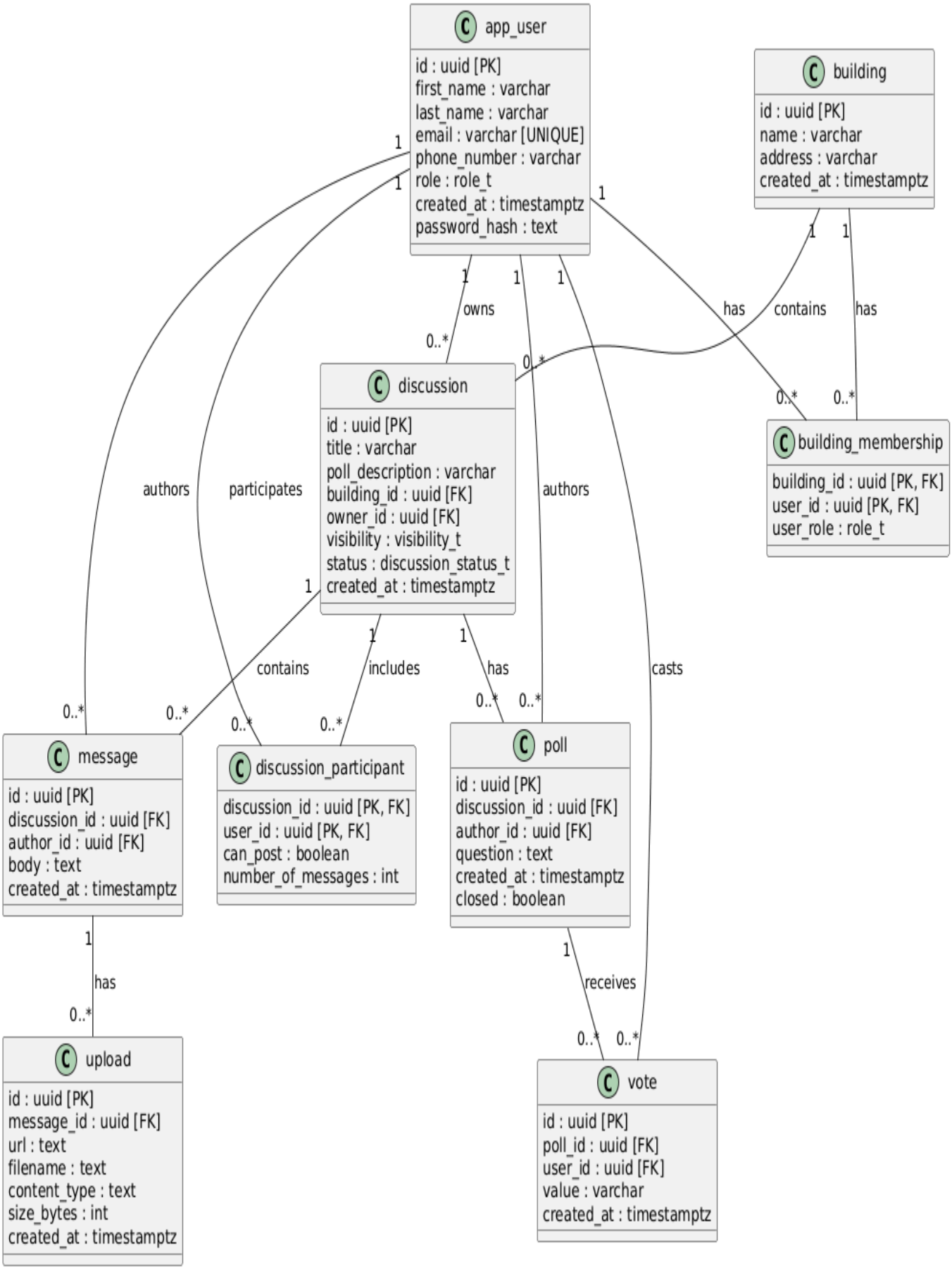
Atribut	Tip podatka	Opis varijable
id	UUID	Primarni ključ (PK), jedinstveni identifikator datoteke
message_id	UUID	Strani ključ (FK), povezuje s tablicom message
url	TEXT	URL lokacija datoteke

Atribut	Tip podatka	Opis varijable
filename	TEXT	Naziv datoteke
content_type	TEXT	MIME tip datoteke
size_bytes	INTEGER	Veličina datoteke u bajtovima
created_at	TIMESTAMPTZ	Datum i vrijeme dodavanja datoteke

Tablica: vote

Atribut	Tip podatka	Opis varijable
id	UUID	Primarni ključ (PK), jedinstveni identifikator glasa
poll_id	UUID	Strani ključ (FK), povezuje s tablicom poll
user_id	UUID	Strani ključ (FK), povezuje s tablicom app_user
value	vote_choice_t	Vrijednost glasa (da/ne)
created_at	TIMESTAMPTZ	Datum i vrijeme glasanja

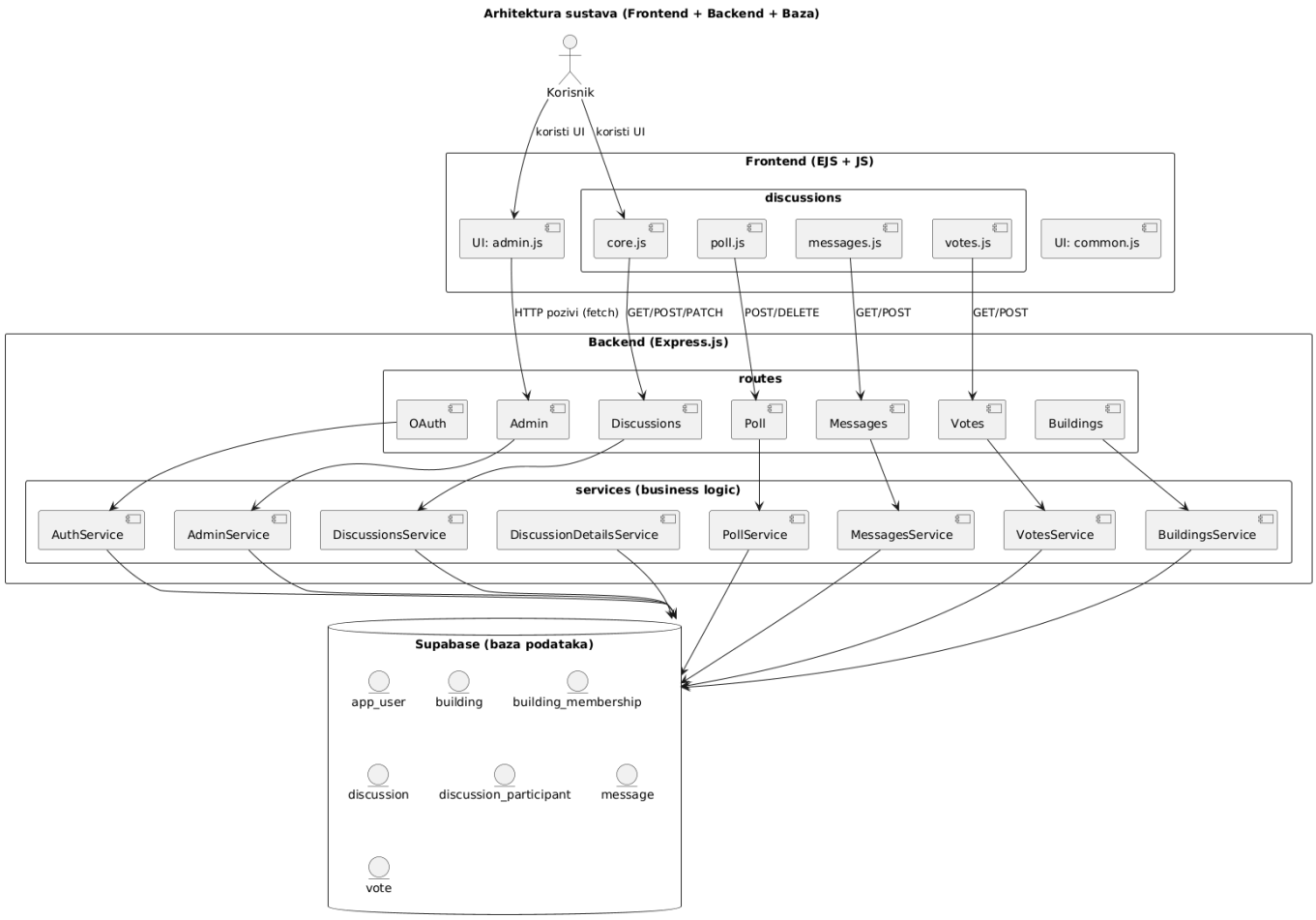
Dijagram baze podataka



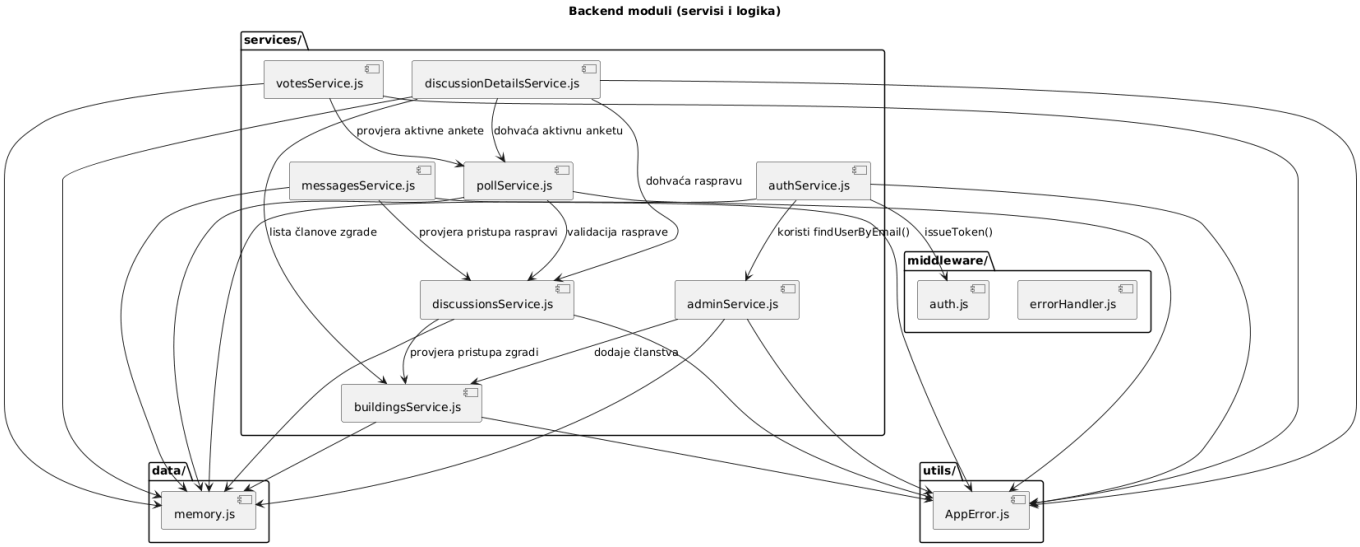
Odnosi tablica i objekata u bazi podataka.

Dijagram razreda

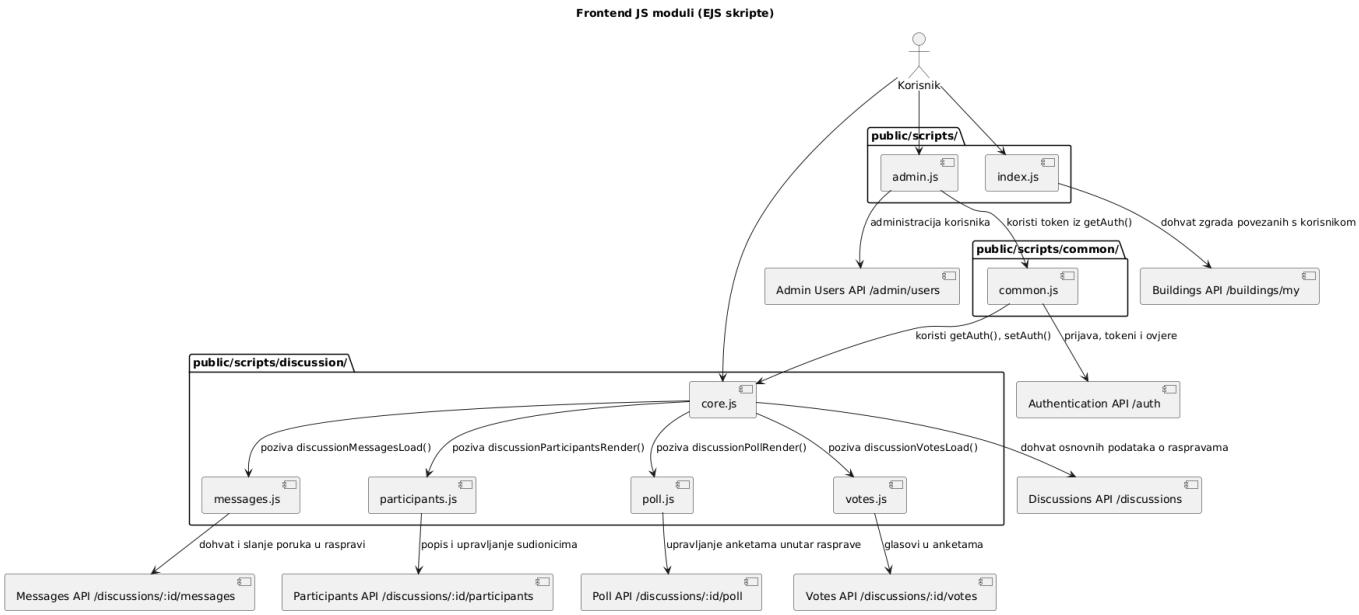
Iako aplikacija ne koristi objektno-orijentirani pristup niti definirane klase, dijagram razreda služi kao apstraktni prikaz strukture sustava. Umjesto razreda, backend i frontend dio aplikacije organizirani su kroz JavaScript module i datoteke koje grupiraju povezane funkcije i odgovornosti. Dijagram zato prikazuje pojedine .js datoteke kao logičke jedinice, zajedno s njihovim ulogama i međusobnim odnosima. Ovaj pristup omogućuje da se funkcionalna arhitektura sustava jasno prikaže, iako se implementacija temelji na modularnim fileovima, a ne na klasama u tradicionalnom smislu.



- **Dijagram arhitekture sustava**
  Ovaj dijagram prikazuje cjelokupnu arhitekturu sustava — od korisnika preko frontend sučelja do backend servisa i baze podataka. Frontend moduli (EJS/JS datoteke) komuniciraju s Express.js routerima putem HTTP zahtjeva, dok backend servisi obrađuju poslovnu logiku i koriste Supabase bazu za pohranu podataka. Dijagram jasno prikazuje slojevitu strukturu aplikacije i način na koji su sve komponente povezane u funkcionalnu cjelinu.



- **Dijagram backend modula (servisi i logika)**
  Dijagram prikazuje kako su backend moduli organizirani i međusobno povezani. Svaki servis predstavlja logičku cjelinu sustava — npr. authService.js upravlja prijavom, discussionsService.js raspravama, a buildingsService.js zgradama i članstvima. Veze između modula označavaju funkcionalnu suradnju (npr. DiscussionDetailsService koristi PollService i BuildingsService), dok svi servisi zajednički komuniciraju s bazom i dijele mehanizam za rukovanje greškama (AppError). Dijagram daje jasan pregled unutarnje strukture backend sustava bez ulaska u detalje implementacije.



- **Dijagram frontend JS modula (EJS skripte)**
  Ovaj dijagram prikazuje organizaciju frontend dijela aplikacije – niz međusobno povezanih JavaScript modula koji upravljaju prikazom, interakcijom i komunikacijom s backend API-jem. core.js djeluje kao središnji modul rasprave i povezuje ostale (poruke, ankete, sudionike, glasove), dok common.js sadrži zajedničke funkcije poput autentikacije. admin.js omogućuje administraciju korisnika i zgrada, a index.js upravlja početnim prikazom. Dijagram prikazuje organizaciju frontend modula te način na koji oni upravljaju prikazom i interakcijom korisnika.

dio 1. revizije

Prilikom prve predaje projekta, potrebno je prilo˘ziti potpuno razraden dijagram razreda vezan uz generičku funkcionalnost sustava. Ostale funkcionalnosti trebaju biti idejno razrađene u dijagramu sa sljedećim kompon entama: nazivi razreda, nazivi metoda i vrste pristupa metodama (npr. javni, zaštićeni), nazivi atributa razreda, veze i odnosi između razreda.

Napomena: Prilikom druge predaje projekta dijagram razreda i opisi moraju odgovarati stvarnom stanju implementacije

Dinamičko ponašanje aplikacije

Dinamičko ponašanje aplikacije odnosi se na način na koji objekti u sustavu evoluiraju kroz vrijeme, uključujući prijelaze između različitih stanja. To uključuje aktivnosti, događaje, odluke i interakcije unutar aplikacije. UML dijagrami stanja omogućuju vizualizaciju tih promjena i olakšavaju razumijevanje dinamike sustava.

Razumijevanje promjena stanja neophodno je za pravilno funkcioniranje aplikacije jer pruža uvid u interakcije među objektima, komponentama i korisnicima tijekom rada sustava. Korištenjem UML dijagrama stanja i aktivnosti moguće je vizualizirati prijelaze i stanja objekata, identificirati potencijalne probleme, osigurati točnu implementaciju te poboljšati komunikaciju među članovima tima.

Zadatak:

- Dokumentirajte dva dinamička dijagrama koji prikazuju značajne ili neke složene procese u aplikaciji. Registracija i prijava korisnika nisu ključni procesi u ovom kontekstu, stoga se usmjerite na specifične, značajne procese.

UML dijagrami stanja

UML dijagrami stanja nužni su za razumijevanje dinamičkog ponašanja sustava. Oni jasno prikazuju promjene stanja objekata tijekom vremena ovisno o događajima i uvjetima.

Preporuke za izradu UML dijagrama stanja:

1. Odaberite značajan objekt sustava (npr. prijava štete, obrada zahtjeva).
2. Identificirajte stanja objekta (npr. aktivan, neaktivan, obrisan).
3. Definirajte događaje i uvjete prijelaza.
4. Ako je potrebno, koristite hijerarhiju stanja za složenije probleme.
5. Dodajte bilješke za pojašnjenje važnih prijelaza i elemenata.

> Primjer: Dijagram stanja za prijavu štete može uključivati:
>
> - Stanja: *Kreirana prijava*, *Obrađena prijava*, *Zatvorena prijava*….

UML dijagrami aktivnosti

Dijagram aktivnosti prikazuje tijek izvršavanja određenog procesa. Osim za razumijevanje toka podataka unutar aplikacije, koristi se za analizu poslovnih procesa.

Zadatak:

1. Identificirajte proces koji želite modelirati (npr. obrada zahtjeva).
2. Razlomite proces na aktivnosti i povežite ih za prikaza slijeda izvršavanja.
3. Upotrijebite pseudo čvorove za jasan prikaz.

> Primjer:** Na primjer, u aplikaciji za upravljanje štetama, dijagram aktivnosti može prikazivati proces obrade zahtjeva za pomoć, što uključuje aktivnosti: *Primanje zahtjeva*, *Analiza podataka*, *Odobravanje resursa*, *Slanje pomoći*.

Arhitektura sustava predstavlja temeljni okvir za razumijevanje i implementaciju svih njegovih funkcionalnosti. U kontekstu razvojne dokumentacije aplikacija, dijagrami komponenata i razmještaja odlučujući su za prikaz povezanosti i rasporeda različitih komponenata sustava. Ovi dijagrami omogućuju sudionicima projekta razumijevanje i vizualizaciju fizičkog i logičkog dizajna sustava, uključujući interakcije između dijelova aplikacije, što je odlučujuće za efikasnu implementaciju i dugoročnu održivost sustava.

Arhitektura sustava, u kontekstu dijagrama komponenata i razmještaja, pruža uvid u strukturu i raspored ključnih dijelova aplikacije. Ovi dijagrami nisu korisni samo tijekom faza oblikovanja i implementacije, već služe i kao alati za održavanje i optimizaciju sustava u budućnosti.

Kao dio razvoja aplikacije, važno je osmisliti i dokumentirati arhitekturu sustava s naglaskom na dijagrame komponenata i razmještaja. Vaš zadatak je izraditi **dijagram komponenata** koji će jasno prikazivati ključne funkcionalne komponente aplikacije, njihovu međusobnu povezanost te sučelja za komunikaciju. Također, trebate izraditi **dijagram razmještaja** komponenata, koji treba detaljno prikazivati kako su te komponente raspoređene u infrastrukturi sustava, uključujući fizičke i virtualne resurse poput poslužitelja ili uređaja krajnjih korisnika.

Dijagram komponenata

Komponente sustava predstavljaju bitne dijelove aplikacije koji obavljaju specifične funkcije. Svaka komponenta je autonomna jedinica s vlastitim odgovornostima, ali je povezana s drugim komponentama kako bi sustav u cjelini funkcionirao. Komponente mogu biti elementi poput modula, servisa, razreda ili paketa, te komuniciraju putem jasno definiranih sučelja.

UML dijagram komponenata za vašu aplikaciju ovisi o njezinoj arhitekturi i složenosti, no općenito treba prikazivati važne funkcionalne komponente, njihovu međusobnu povezanost i sučelja za komunikaciju. Obzirom na namjenu projekta, dijagram treba biti jasan, organiziran i lako čitljiv kako bi olakšao razumijevanje strukture i suradnju unutar tima.

Dijagram razmještaja

UML dijagram razmještaja prikazuje fizičku ili virtualnu raspodjelu komponenata sustava unutar infrastrukture. Cilj je prikazati kako su komponente raspoređene (npr. na poslužiteljima, u okruženjima oblaka ili na uređajima krajnjih korisnika) te način komuni čije, API-ja ili drugih komunikacijskih protokola.

U ovoj dokumentaciji preporučuje se uključiti dijagram razmještaja instanci (engl. Instance Level Deployment Diagram) ili implementacije.

- Dijagram razmještaja instanci prikazuje način na koji su aplikacijske komponente raspoređene unutar infrastrukture, uključujući fizičke i virtualne čvorove (poslužitelje) na kojima se izvode. Ovaj dijagram detaljno opisuje kako su komponente povezane i kako međusobno komuniciraju.

> U kontekstu aplikacije, npr. ona koja koristi Docker kontejner, dijagram razmještaja instanci pokazuje kako se aplikacije raspoređuju unutar Docker kontejnera na različitim poslužiteljima ili u okruženjima oblaka.

Implementacijski oblik

- Implementacijski oblik daje detaljan prikaz rasporeda komponenata u fizičkoj ili virtualnoj infrastrukturi. Ovdje se koriste stvarni poslužitelji, mrežne veze, uređaji korisnika, aplikacijski paketi i drugi artefakti kako bi dijagram prikazao točan fizički raspored komponenata, s naglaskom na fizičko povezivanje i tehničke resurse.

> **Primjeri:** Raspored sustava unutar fizičkog podatkovnog centra, uključujući informacije o virtualnim poslužiteljima, bazama podataka, mrežnim vezama i sklopovskim resursima; prikaz rasporeda komponenata na konkretnim poslužiteljima u oblaku (npr. AWS, Google Cloud).

Ovo poglavlje treba opissti provedena ispitivanja implementiranih funkcionalnosti na razini komponenti i sustava. Fokus je na odabiru i izvedbi ispitnih slučajeva koji obuhvaćaju redovne, rubne uvjete i testiranje grešaka, kao i upotrebu odgovarajućih alata za provedbu testiranja.

Ispitivanje komponenti

Cilj ispitivanja komponenti je provjera osnovnih funkcionalnosti implementiranih u razredima sustava. Ovdje je potrebno izolirati svaku komponentu kako bi se testirala njezina ispravnost i reakcija na različite scenarije.

Zadaci:

1. Razviti minimalno **6 ispitnih slučajeva** koji obuhvaćaju:
    - **Redovne slučajeve**: testiranje uobičajenog ponašanja funkcionalnosti.
    - **Rubne uvjete**: provjera ulaznih podataka na granici valjanosti.
    - **Izazivanje pogreške (exception throwing)**: testiranje reakcije na iznimke.
    - **Nepostojeće funkcionalnosti**: provjera reakcije na poziv neimplementirane funkcionalnosti.

Struktura ispitivanja:

Za svaki ispitni slučaj potrebno je:

1. Opišite funkcionalnost koju testirate (npr. dodavanje korisnika, validacija podataka).
2. Navedite ispitni slučaj:
    - Ulazne podatke.
    - Očekivane rezultate.
    - Dobivene rezultate (prolaz/pad ispitivanja).
3. Opišite postupak prvođenja ispitivanja
4. U Gitu moraju biti dostupni izvorni kodovi ispitnih slučajeva.

**Ispitivanje sustava **

Cilj ispitivanja sustava je testiranje ponašanja cijelog sustava u uvjetima stvarnog korištenja, uz posebnu pažnju na međusobnu povezanost svih komponenti. Ispitivanje treba obuhvatiti sve aspekte sustava i njegovu interakciju s korisnicima.

Zadaci:

Razviti minimalno **4 ispitna slučaja** koji obuhvaćaju:

- **Redovne slučajeve**: očekivano ponašanje sustava.

- **Rubne uvjete**: reakcija sustava na granične ulaze.
- **Poziv nepostojećih funkcionalnosti**: testiranje kako sustav reagira na neimplementirane ili neispravne funkcije.

Struktura ispitnih slučajeva za Selenium:

1. **Ulazi**:
    - Konkretni podaci koji se unose u sustav (npr. korisničko ime i lozinka za prijavu).
    - Simulacija korisničkih akcija (klikanje, unos teksta, navigacija).
2. **Koraci ispitivanja**:
    - Npr. Detaljan opis koraka koje Selenium izvršava:
        1. Otvoriti aplikaciju u pregledniku.
        2. Unijeti podatke u formu.
        3. Kliknuti na gumb za potvrdu.
        4. Verificirati očekivane rezultate (npr. prijava uspješna).
3. **Očekivani izlaz**:
    - Očekivani rezultat ispitivanja (npr. korisnik preusmjeren na početnu stranicu nakon prijave).
4. **Dobiveni izlaz**:
    - Priložiti logove, screenshotove ili izvještaje s generiranim rezultatima (npr. iz JUnit-a ili Selenija).

Alati za ispitivanje sustava:

- **Selenium IDE** ili prikladan obzirom na vaše razvojno okruženje:
    - Jednostavan alat za snimanje korisničkih akcija u pregledniku i automatsko ponavljanje testova.
    - Preporučuje se za osnovne ispitne slučajeve.
- **Selenium WebDriver**:
    - Omogućuje pisanje naprednih testova u različitim programskim jezicima (Java, Python, C#).
    - Preporučuje se za složenije testove, koji zahtijevaju detaljno prilagodbu i automatizaciju.

Primjeri ispitivanjasa Seleniumom:

1. **Formular za prijavu**:

> u dokumentaciji obavezna su specifičnija ispitivanja vaše aplikacije! - **Ulaz**: Korisničko ime = "user@fer.ugnz.hr", Lozinka = "password123".

```
- **Koraci**:
  1. Otvoriti aplikaciju.
  2. Unijeti korisničko ime i lozinku.
  3. Kliknuti na "Prijava".
  4. Provjeriti je li korisnik preusmjeren na početnu stranicu.
- **Očekivani izlaz**: "Prijava uspješna."
```

2. **Rubni uvjet – nevažeća lozinka**:
    - **Ulaz**: Korisničko ime = "user@example.com", Lozinka = "malimedo".
    - **Koraci**:
        1. Unijeti podatke u formu za prijavu.
        2. Kliknuti na "Prijava".
        3. Provjeriti je li prikazana poruka o grešci.
    - **Očekivani izlaz**: "Pogrešno korisničko ime ili lozinka."

Prezentacija rezultata

Za oba tipa ispitivanja (komponenti i sustava) potrebno je:

- Jasno dokumentirati ulaze, korake, očekivane i dobivene rezultate.
- Priložiti slike ekrana, logove ili izvješća generirana alatima (npr. JUnit ili Selenium).
- Detaljno opisati ponašanje sustava, posebno u rubnim uvjetima.
- Navesti broj otkrivenih grešaka!

Korištene tehnologije i alati

Cilj: Jasno i precizno opisati tehnologije korištene u projektu kako bi se olakšalo održavanje, proširenje i suradnja u timu. Uključite informacije:

1. **Programski jezici**: Navesti korištene jezike i njihove verzije (npr. JavaScript 16.13).
2. **Radni okviri i biblioteke**: Detaljno opisati alate za frontend i backend (npr. React 18, Node.js 16).
3. **Baza podataka**: Navesti vrstu baze (npr. PostgreSQL 13).
4. **Razvojni alati**: Popis korištenih IDE-ova, alata za verzioniranje (npr. VS Code, Git 2.34).
5. **Alati za ispitivanje**: Jedinični, integracijski ili UI ispitni sljučajevi (npr. Jest 27, Selenium 4.0).
6. **Alati za razmještaj**: Korišteni alati za implementaciju (npr. Docker 20.10).
7. **Cloud platforma**: Ako je aplikacija hostana, navesti platformu (npr. AWS).

Preporuke za opis:

- **Jasno i precizno**: Izbjegavati tehnički žargon i navesti točne verzije.
- **Obrazloženje izbora**: Objasniti zašto su odabrane određene tehnologije.
- **Opis konfiguracije**: Istaknuti specifične postavke alata i baza.

Primjer:
Za razvoj klijentskog dijela aplikacije korišten je **React** (verzija 18) [ref.], popularna JavaScript biblioteka za izgradnju interaktivnih korisničkih sučelja. React omogućuje stvaranje samostalnih komponenti koje se mogu ponovno koristiti i lako ažurirati, čime se poboljšava učinkovitost razvoja. Za stiliziranje su upotrijebljene **styled-components** (verzija 5.3), što omogućava integraciju stilova izravno unutar Ručat komponenti koristeći **CSS-in-JS** pristup.

Ovaj odjeljak dokumentacije treba dati detaljne smjernice za instalaciju, konfiguraciju, pokretanje i administraciju aplikacije. Cilj je olakšati postavljanje aplikacije na razvojnom, ispitnom i produkcijskom okruženju.

1. Instalacija

Ovdje treba navesti korake potrebne za instalaciju svih potrebnih komponenti:

- **Preduvjeti**: Popis potrebnog softvera i njihovih verzija (npr. Node.js 16, Docker 20.10).
- **Preuzimanje**: Upute za preuzimanje izvornog koda (npr. kloniranje Git repozitorija).

Primjer:

```
git clone https://github.com/Projekt/primjer.git

cd repo
```

Instalacija ovisnosti: Upute za instaliranje ovisnosti.

```
npm install
```

2. Postavke

Detaljne upute za konfiguraciju aplikacije:

- **Konfiguracijske datoteke**: Gdje se nalaze (npr. config.json, .env) i što treba prilagoditi.
    - Primjer .env datoteke:

```
DATABASE_URL=postgres://user:password@localhost:5432/dbname

API_KEY=your_api_key
```

- **Postavke baze podataka**: Upute za inicijalizaciju baze podataka, uključujući migracije i postavljanje inicijalnih podataka.

```
npm run db:migrate

npm run db:seed
```

3. Pokretanje aplikacije

Upute za pokretanje aplikacije u različitim okruženjima:

- **Razvojno okruženje**:

```
npm run dev
```

- **Produkcijsko okruženje**:
    - Prevođenje aplikacije:

```
npm run build
```

- Pokretanje poslužitelja:

```
npm start
```

- **Provjera rada**: Navesti npr. URL (npr. http://localhost:3000) .

4. Upute za administratore

Smjernice za administratore aplikacije nakon puštanja u pogon:

- **Pristup administratorskom sučelju**:
  - URL za admin panel (npr. /admin).
  - Početni podaci za prijavu (ako postoje).
- **Redovito održavanje**:
  - Arhiviranje baze podataka.
  - Pregled logova.
  - Ažuriranje aplikacije (primjer: povlačenje novih verzija iz Git repozitorija i ponovno pokretanje aplikacije).

```
git pull origin main

npm install

npm run build

npm start
```

- **Rješavanje problema**: Kako pristupiti logovima i dijagnosticirati greške (npr. logs/error.log ili docker logs).

---

5. Primjer za Render platformu (Cloud Deploy)

Render je popularna cloud platforma za jednostavno smješanje aplikacija.

- **Priprema repozitorija**:
  - Osigurajte da vaš projekt ima datoteku render.yaml ili Dockerfile za konfiguraciju deploya.
  - Primjer render.yaml:

```
services:

- type: web

name: my-web-app

env: node

buildCommand: npm install && npm run build
```

> startCommand: npm start
>
> plan: free

- **Postavljanje na Render**:
  - Prijavite se na Render.
  - Kreirajte novi **Web Service** i povežite ga s vašim GitHub repozitorijem.
  - Konfigurirajte postavke (npr. build i start komande).
  - Dodajte environment varijable (npr. DATABASE_URL, API_KEY).
- **Pokretanje aplikacije**:
  Render će automatski preuzeti repozitorij, instalirati ovisnosti i pokrenuti aplikaciju. Nakon deploya, aplikaciji možete pristupiti putem generiranog URL-a (npr. https://my-web-app.onrender.com).

Opis prisutpa aplikaciji na javnom poslužitelju

Pristup aplikaciji Dokumentirajte postupak i pružite jasne smjernice za korištenje aplikacije na javnom poslužitelju.

- Navedite ograničenja!
- U uputama obuhvatite kako korisnici mogu pristupiti aplikaciji putem internetskog preglednika.
- Priložite korake za pristup administratorskom sučelju ako je primjenjivo.

U ovom poglavlju potrebno je napisati osvrt na vrijeme izrade projektnog zadatka, koji su tehnički izazovi prepoznati, jesu li riješeni ili kako bi mogli biti riješeni, koja su znanja stečena pri izradi projekta, koja bi znanja bila posebno potrebna za brže i kvalitetnije ostvarenje projekta i koje bi bile perspektive za nastavak rada u projektnoj grupi. Potrebno je točno popisati funkcionalnosti koje nisu implementirane u ostvarenoj aplikaciji.

Rev.	Opis promjene/dodatka	Autori	Datum
0.1	Otvorena organizacija i stvoren Wiki	Jakov Svalina	14.10.2025.
0.2	Uređen README.md	Jakov Svalina	18.10.2025.
0.3	Dodan opis projektnog zadatka	Jakov Svalina	21.10.2025.
0.4	Odrađena analiza zahtjeva	Jakov Svalina	21.10.2025.
0.5	Nadopunjen dnevnik sastanaka	Jakov Svalina	14.11.2025.
0.6	Objašnjena arhitektura i dizajn	Luka Zorić	14.11.2025.
0.7	Dodani dijagrami razreda,mvc arhitektura, sklopoprogramski zahtjevi	Marko Vrkić	14.11.2025.
0.8	Napravljena analiza zahtjeva i detaljan opis projektnog zadatka	Vinko Šapina	14.11.2025.
0.8	Napravljena Specifikacija zahtjeva sustava i dopunjena Arhitektura i dizajn sustava	Oleksandr Malik	14.11.2025.

Kontinuirano osvježavanje Popisati sve reference i literaturu koja je pomogla pri ostvarivanju projekta.

1. Programsko inženjerstvo, FER ZEMRIS, http://www.fer.hr/predmet/proinz
2. I. Sommerville, "Software engineering", 8th ed, Addison Wesley, 2007.
3. T.C.Lethbridge, R.Langaniere, "Object-Oriented Software Engineering", 2nd ed. McGraw-Hill, 2005.
4. I. Marsic, Software engineering book", Department of Electrical and Computer Engineering, Rutgers University, http://www.ece.rutgers.edu/˜marsic/ books/SE
5. The Unified Modeling Language, https://www.uml-diagrams.org/
6. Astah Community, http://astah.net/editions/uml-new

Dnevnik sastajanja

1. sastanak
    - datum: 18. listopada 2025.
    - prisustvovali: M.Vrkić, L.Zorić, O.Malik, V.Šapina, I.Trobradović, J.Svalina
    - teme sastanka:
        - odabir projektnog zadatka - prihvaćanje StanBlog zadatka
        - uspostavljanje WhatsApp grupe svih članova tima
        - dogovaranje uloga u razvoju projekta
2. sastanak
    - datum: 22. listopada 2025.
    - prisustvovali: M.Vrkić, L.Zorić, O.Malik, V.Šapina, I.Trobradović, J.Svalina
    - teme sastanka:
        - odabir konkretnih tehnologija koje će se koristiti za razvoj projekta - React, NodeJS, Supabase, API-jevi
        - osmišljavanje prve inačice dizajna web-stranice
        - određivanje prioritetnih funkcionalnih zahtjeva za razvoj do prve predaje projekta
3. sastanak
    - datum: 4. studenog 2025.
    - prisustvovali: M.Vrkić, L.Zorić, O.Malik, V.Šapina, I.Trobradović, J.Svalina
    - teme sastanka:
        - uspostavljanje rokova svakom članu za isporuku svoga dijela zadatka
        - modifikacija baze podataka u svrhu lakšeg backend razvoja
    - poveznica na issue:
        - #1
        - #5

Plan rada

Tjedan	Ključne aktivnosti	Angažirani članovi tima
3.	raspodjela dužnosti, odabir projektnog zadatka	MV, LZ, OM, VŠ, IT, JS
4.	razrada plana razvoja web-aplikacije, dogovor oko implementiranja željenih usluga i dodatnih funkcionalnosti te odabir razvojnih tehnologija	MV, LZ, OM, VŠ, IT, JS
5.	razvoj baze podataka, razvoj prototipa projekta koristeći privremenu lokalnu bazu	VŠ, IT
6.	povezivanje aplikacije sa Supabase bazom i refactoring kako bi aplikacija radila s njom	JS, IT
7.	deployment	JS

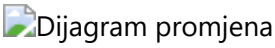
Tablica aktivnosti

Kontinuirano osvježavanje

Zadatak	Luka Zorić	Marko Vrkić	Jakov Svalina	Isa Trobradović	Vinko Šapina	Oleksandr Malik
Upravljanje projektom					6	
Opis projektnog zadatka					3	
Funkcionalni zahtjevi					3	
Opis pojedinih obrazaca						5
Dijagram obrazaca						4
Sekvencijski dijagrami						6
Opis ostalih zahtjeva						2
Arhitektura i dizajn sustava	7	7				
Baza podataka					6	3
Dijagram razreda	5	4				
Dijagram stanja	3	3				
Dijagram komponenti						4
Korištene tehnologije i alati			4			
Ispitivanje programskog rješenja				10		
Dijagram razmještaja	3	3				
Upute za puštanje u pogon			5			
Dnevnik sastajanja			3			
Izrada aplikacije	6	6	8	12		
Izrada početne stranice	5	4				
Izrada baze podataka					7	3
Spajanje s bazom podataka			6			

Zadatak	Luka Zorić	Marko Vrkić	Jakov Svalina	Isa Trobradović	Vinko Šapina	Oleksandr Malik
Izrada programske potpore			7	18		
Ukupno	29	27	33	40	25	26

Dijagram pregleda promjena


Dijagram promjena

Kjučni izazovi i rješenja

- izazovi projekta:
  - loša komunikacija i posljedični manjak koordinacije
    - rješenje: sastanci uživo i jasno podijeljeni zadatci i dobro definirani vremenski rokovi
  - korištenje nepoznatih tehnologija
    - rješenje: čitanje dokumentacije i istraživanje o uporabi vanjskih servisa poput Supabase i NodeJS
  - nedovoljno dobro definirani zahtjevi od backend tima prema timu za bazi, stoga je baza bila problematična za koristiti
    - rješenje: jasno definiranje svih atributa i parametara koje backend tim treba kako bi se mogao ostvariti prijelaz na vanjsku bazu Supabase
  - kašnjenje u razvoju
    - rješenje: sastanci na kojima se jasno definira opseg projekta koji će se implementirati u zadanom vremenskom intervalu

Programsko inženjerstvo ak.god 25/26

Sveučilište u Zagrebu

Fakultet elektrotehnike i računarstva

Naziv projekta: StanBlog

Tim: <TG10.2>

Ime tima: stanari123

Nastavnik: Vlado Sruk