Event Driven

Studenti: Predmetni profesor:

Kristina Joksimović, 18203 Bratislav Predić

Anđelija Mijajlović, 18247 Dragan Stojanović

Mihajlo Bencun, 18081

Arhitekturni projekat

# Kontekst

Aplikacija je pogodna za upotrebu na više nivoa: od individue, radi čuvanja obaveza na jednom mestu i brzog pristupa istim, do organizacija različitih tipova i veličina (porodica – korporacija) kao pomoć pri organizaciji jednog ili više kratkoročnih i dugoročnih projekata.

# Cilj

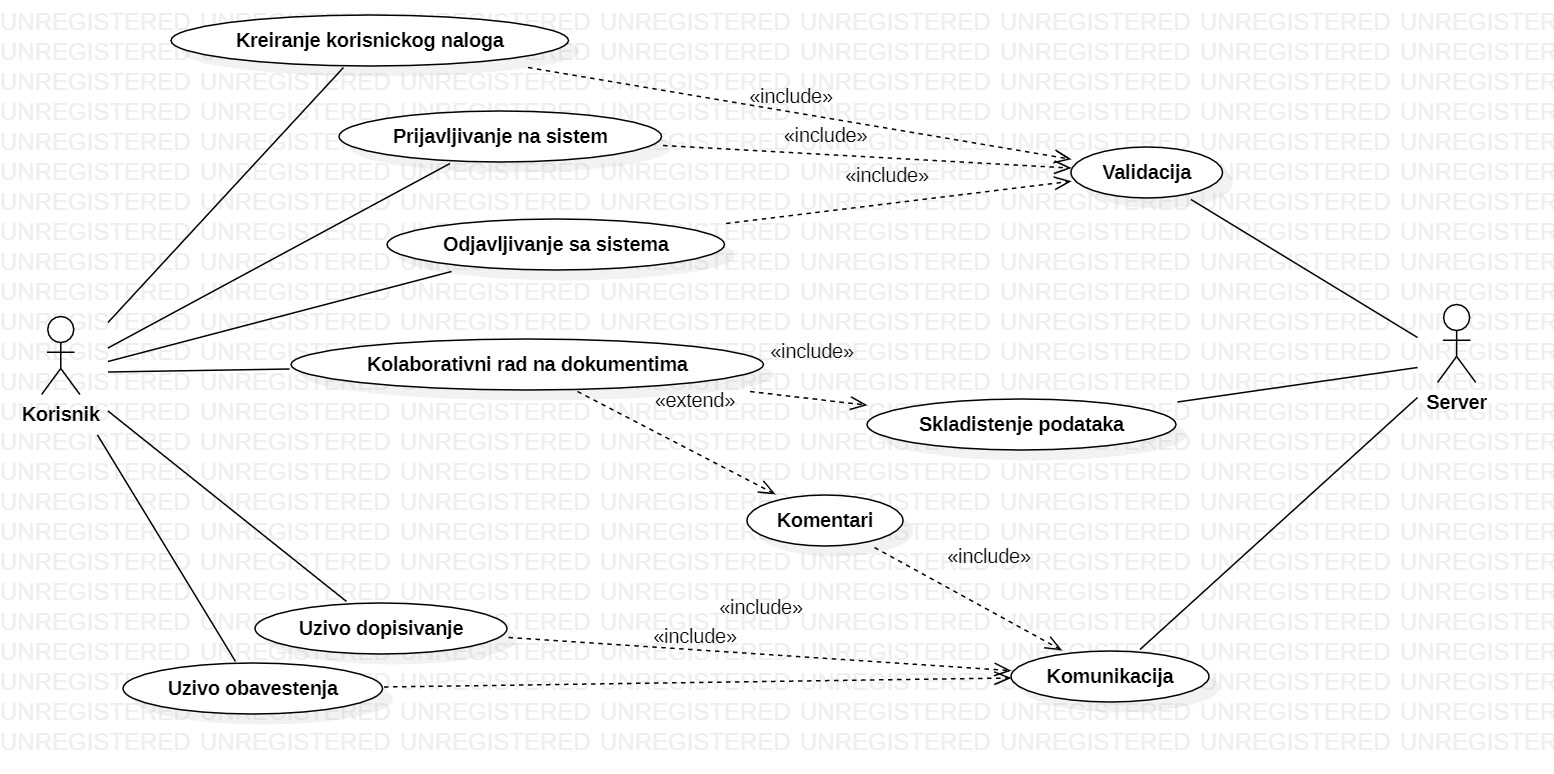
Omogućiti pojednostavljenu saradnju na projektima u okviru zajednice kroz bezbednu komunikaciju i postizanje centralizovane organizacije

# Zahtevi

## Glavni funkcionalni zahtevi

Osnovne funkcionalnosti aplikacije su izražene kroz sledeće funkcionalne zahteve:

1. Kreiranje korisničkog naloga,
2. Prijavljivanje korisničkim nalogom na sistem,
3. Pregled organizacionih elemenata u vidu kalendara na nivou zajednice,
4. Odjavljivanje sa sistema,
5. Izmene i brisanje korisničkog naloga,
6. Kreiranje i brisanje zajednice,
7. Pridruživanje i napuštanje zajednice,
8. Menadžment zajednice,
9. Uživo dopisivanje,
10. Uživo obaveštenja,
11. Podešavanja aplikacije,
12. Kreiranje i brisanje dokumenata različitih tipova,
13. Kolaborativni rad na dokumentu i
14. Komentari u toku rada na dokumentu.



## Nefunkcionalni zahtevi

Atributi kvaliteta softvera koje ova aplikacija treba da poseduje su sledeći:

1. Performanse – Brzina rada sistema, što kraće vreme odziva i što manja latencija u komunikaciji zbog rada aplikacije u realnom vremenu. React je efikasna i brza biblioteka za klijentski deo aplikacije što će obezbediti dobre performanse,
2. Pouzdanost – Otpornost sistema na padove kao i brz i efikasan oporavak od istih. Efikasan sloj perzisentcije omogućava visoku pouzdanost
3. Sigurnost – Autentifikacija, autorizacija i otpornost sistema na napad
4. Upotrebljivost – Intuitivnost korisničkog interfejsa (UI/UX)
5. Skalabilnost – Mogućnost sistema da podrži veliki broj korisnika, kao i nagle promene u opterećenju
6. Održivost – Standard kvaliteta izvornog koda koji će omogućiti da se sistem lako može ažurirati i održavati u narednom periodu

# Tehnička i poslovna ograničenja

* Aplikacija je zamišljena tako da joj se pristupa kroz Web pregledače,
* Aplikacija ima slojevitu arhitekturu tako da su serverski deo i sloj perzistencije sakriveni od krajnjih korisnika
* Aplikacija koristi i sinhronu i asinhronu komunikaciju

# Arhitekturni dizajn

## Arhitekturni obrasci

1. **Layered**

Arhitektura aplikacije se zasniva na slojevitom arhitekturnom obrascu (*Layered*). Osnovna ideja jeste da aplikacija ima prezentacioni sloj, sloj domenske logike i sloj izvora podataka (troslojna arhitektura).

1. **Domain Model**

Koristi se u sistemima gde može postojati mnogo posebnih slučajeva ponašanja i objekti na sistematski način modeliraju složene sisteme. Domenski model se sastoji od mreže međusobno povezanih objekata, odnosno entiteta domenske logike, koja se izdvaja u poseban sloj u kome objekti razmenjuju poruke. Ulančavanjem poziva u nizu poruka se obrađuje jedan slučaj upotrebe. U početku je model prosta slika relacione baze podataka, te tokom izrade divergira od strukture baze i usložnjava se

1. **Data Mapper**

Do formiranja ovog obrasca doveli su složeni domenski modeli. Objektni model korišćen u aplikaciji i relacioni model u bazi podataka rade na osnovu različitih koncepata za struktuiranje podataka. Sa porastom složenosti aplikacije se šeme ova dva modela sve više razlikuju i da bi se osigurao nezavisan razvoj i izmene šema one se potpuno razdvajaju. Maper podataka predstavlja sloj koji razdvaja klase i prebacuje podatke između objektnog i relacionog modela. U sistem se uvodi poseban maper za svaku hijerarhiju klasa, kao međurešenje po složenosti.

U slučaju našeg sistema Prisma prati ovaj obrazac.

1. **Model- View- Controller**

Prezentacioni sloj se modelira na osnovu MVC obrasca. Ovaj obrazac koristi se za organizaciju i razdvajanje komponenti u softverskoj aplikaciji radi lakšeg održavanja i razvoja.

* + Model - podaci i logika poslovanja: podaci se obrađuju i pristupa im se putem modela.
  + Pogled - sloj koji korisnik vidi i sa kojim interaguje: prikazuju se informacije iz modela na korističkom interfejsu.
  + Kontroler - posrednik između Modela i Pogleda: obrađuje korisnički unos, ažurira modele na osnovu toga i odgovara na akcije korisnika.

Ovaj obrazac omogućava odvajanje logike poslovanja od korisničkog interfejsa.

Recimo, kada bi korisnik dodao događaj u određenu zajednicu, korisnički interfejs bi omogućio unos tih informacija, nakon toga, kontroler bi primio te informacije i validirao ih, sačuvao novi događaj u modelu za događaje povezan sa odabranom zajednicom. Pogled bi se ažurirao da prikaže novi događaj na kalendaru

1. **Klijent- Server**

Klijent-server opisuje način komunikacije i organizaciju sistema. Ideja je da se podele funkcionalnosti sistema između dva tipa entiteta: klijenta, koji zahteva resurse ili usluge, i servera, koji pruža te resurse ili usluge.

Zasniva se na principu razdvajanja odgovornosti između klijenta i servera, kako bi se omogućila bolja skalabilnost, održavanje i efikasnost sistema.

Korišćenje klijent-server modela u našem slučaju, omogućilo bi centralizovano upravljanje podacima o događajima. Klijenti (korisnici) bi slali zahteve serveru kako bi pregledali, dodavali ili menjali događaje unutar zajednica kojima pripadaju. Server bi zatim vratio odgovor sa traženim informacijama ili potvrdom promena.

Ovaj model olakšava upravljanje podacima, pristupanje informacijama iz različitih klijentskih uređaja i omogućava bolju skalabilnost aplikacije. Takođe, olakšava održavanje integriteta podataka, jer se podaci o događajima čuvaju centralizovano na serveru.

1. **Publish - Subscribe**

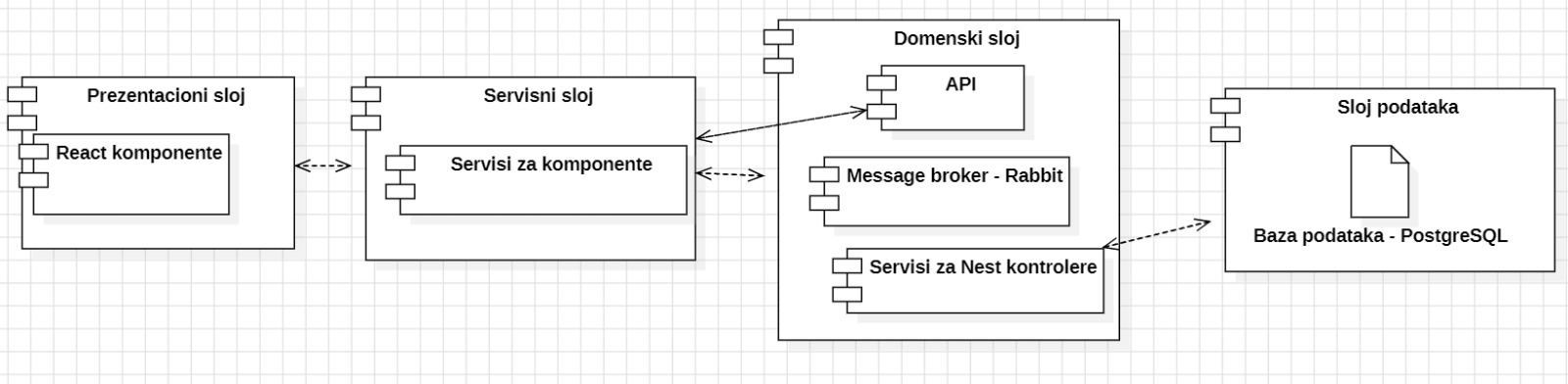
*Publish-Subscribe* ili često skraćeno pub-sub jeste obrazac u kom proizvođač šalje poruke na kanal na koji može biti pretplaćeno više potrošača umesto da šalje poruku svakom potrošaču ponaosob. Ovo pomaže kod skalabilnosti, razdvajanja komponenti i asinhronosti aplikacije. Takođe, poboljšava responzivnost proizvođača, pošto je dovoljno da on pošalje jednom poruku, a da potom nastavi sa svojim zadacima, umesto da veći deo vremena posveti slanju poruka svima koji je očekuju. Infrastruktura za slanje poruka mora biti bezbedna i robustna kako bi se obezbedilo glatko funkcionisanje sistema.

1. **Repository**

*Repository* obrazac služi da izoluje sloj izvora podataka od ostatka aplikacije kroz interfejse i metode isključivo namenjene CRUD operacijama nad određenim entitetima modela.

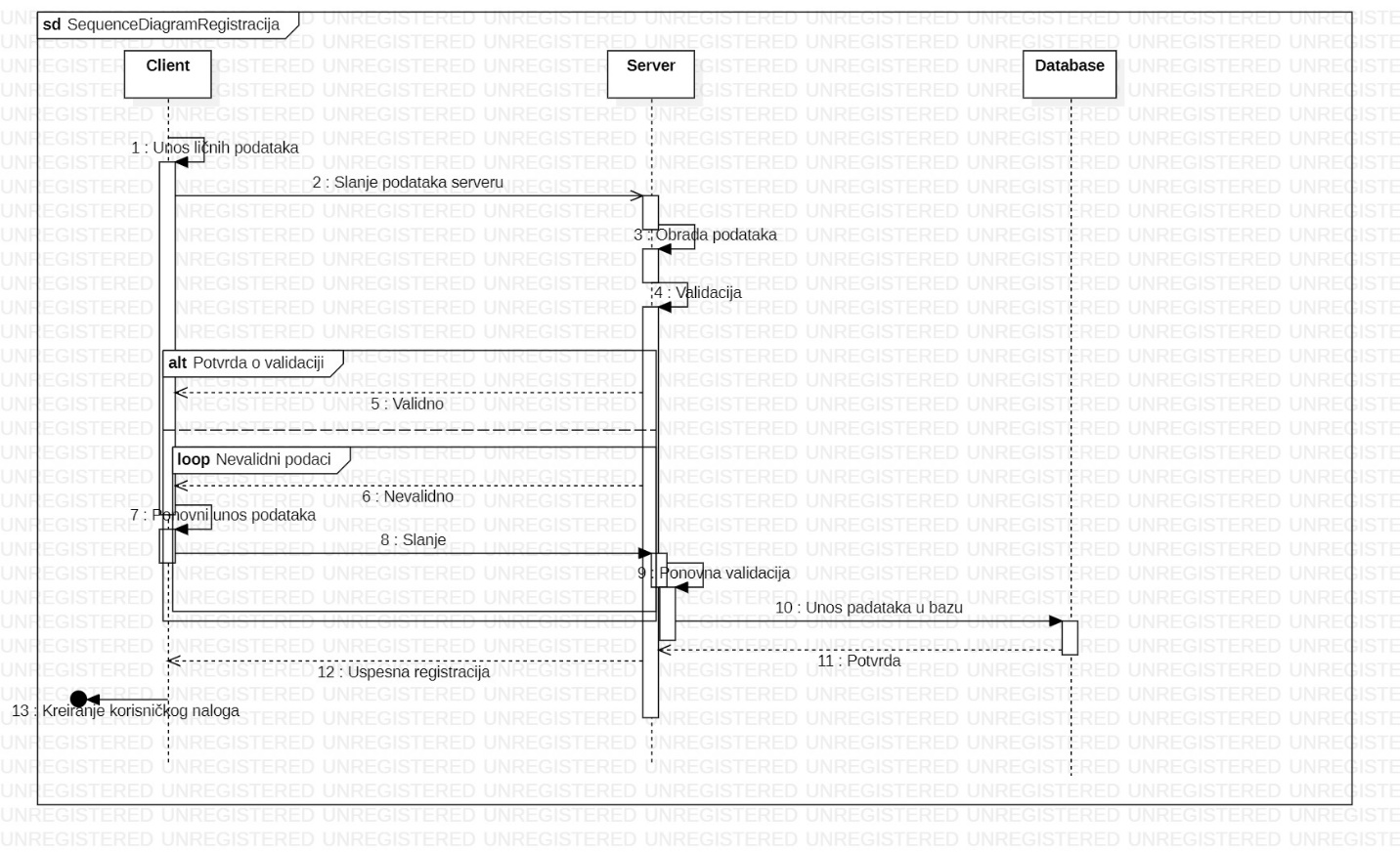
## Generalna arhitektura

## Strukturni pogled na sistem

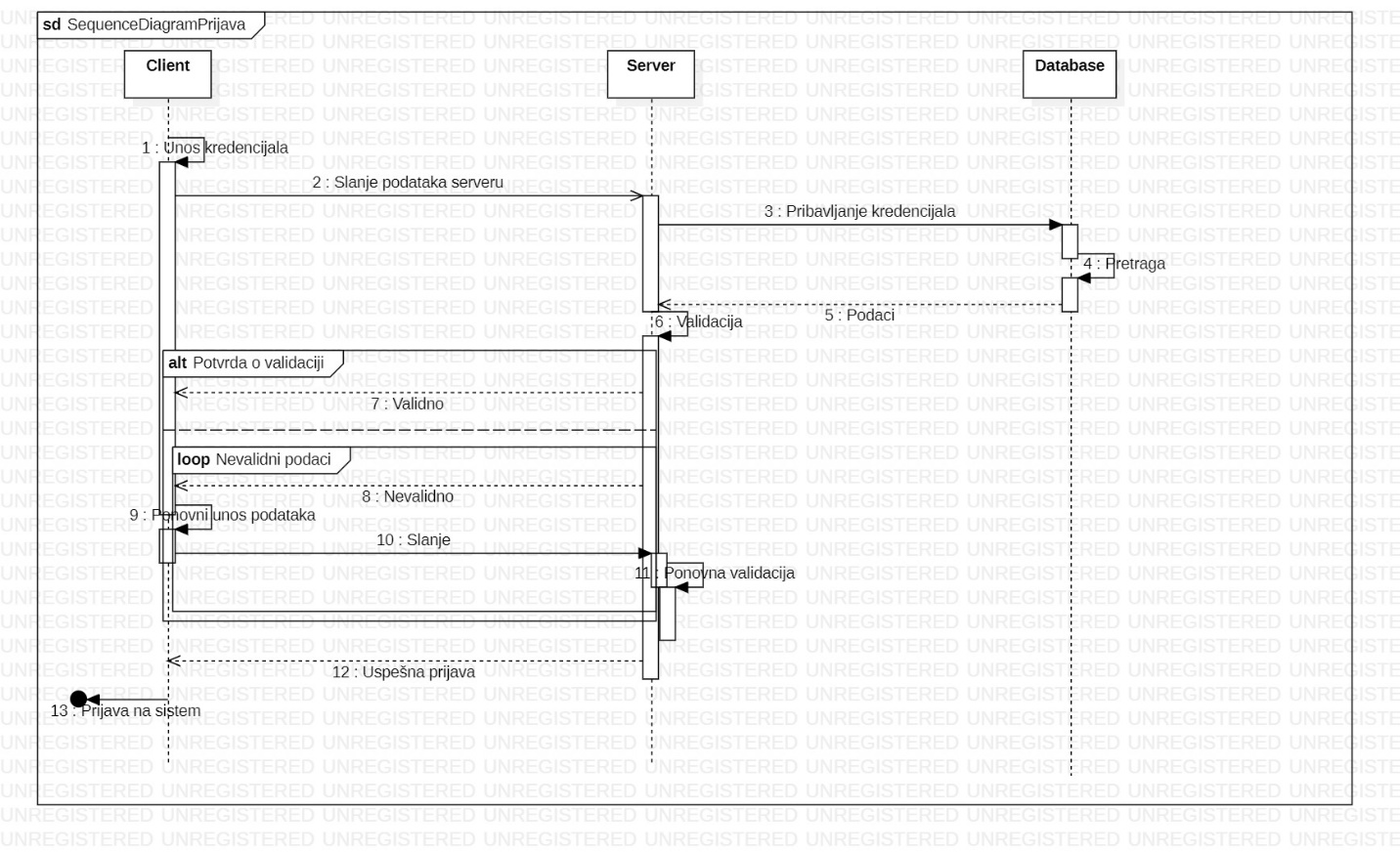


## Bihevioralni pogled na sistem

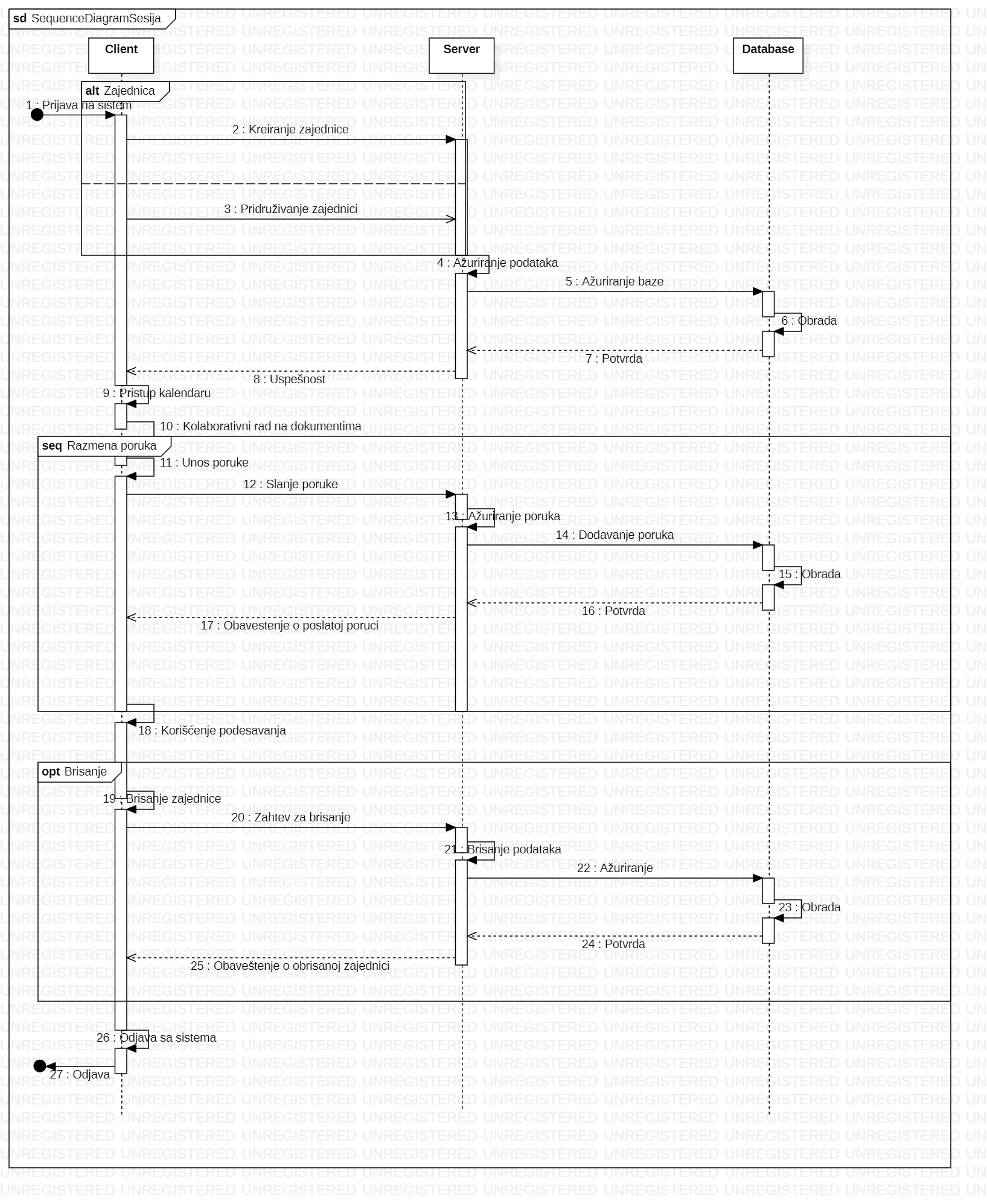
1. **Kreiranje korisničkog naloga**



1. **Prijava na sistem**



1. **Događaji u okviru sesije**



# Implementacija

Aplikacioni okviri i biblioteke koji će biti korišćeni za implementaciju klijentskih i serverskih komponenti sistema:

NestJS - server-side framework zasnovan na Node.js i Express-u.   
React - JavaScript biblioteka za izgradnju korisničkih interfejsa.  
Redux - JavaScript biblioteka za upravljanje stanjem aplikacije.  
PostgreSQL - objektno-relacioni sistem za upravljanje bazom podataka.  
Prisma ORM - Node.js i Typescript objektno-relacioni maper.  
RabbitMQ - Message broker.