

Faza 1 – Arhitektura softverskog sistema

1. Kontekst i cilj softverskog projekta

1.1 Kontekst projekta

Predmet ovog projekta je razvoj **web softverskog sistema za online igranje šaha**, funkcionalno sličnog platformama kao što je *chess.com*. Sistem je namenjen velikom broju korisnika koji putem internet pregledača mogu igrati šah u realnom vremenu, pratiti svoj napredak i komunicirati sa drugim igračima.

Aplikacija je zasnovana na **klijent–serverskoj arhitekturi**, gde korisnici pristupaju sistemu preko web klijenta, dok se kompletan poslovni logika, obrada zahteva i upravljanje podacima nalaze na serverskoj strani. Sistem mora obezbediti pouzdanu real-time komunikaciju između korisnika, kao i dosledno i bezbedno upravljanje šahovskim partijama.

1.2 Cilj projekta

Cilj projekta je projektovanje i specifikacija **skalabilnog, pouzdanog i bezbednog softverskog sistema** koji omogućava korisnicima da:

- igraju šah protiv drugih korisnika u realnom vremenu,
 - upravljaju svojim korisničkim nalozima,
 - prate statistiku i istoriju odigranih partija.
-

2. Arhitektурно specifični zahtevi

2.1 Glavni funkcionalni zahtevi

Sistem treba da podrži sledeće osnovne funkcionalne zahteve:

1. Registracija korisnika

Sistem mora omogućiti kreiranje novog korisničkog naloga unosom osnovnih podataka.

2. Prijava korisnika (autentifikacija)

Registrovani korisnici moraju imati mogućnost bezbedne prijave na sistem.

3. Upravljanje korisničkim profilom

Korisnik može pregledati i izmeniti osnovne podatke svog profila.

4. Kreiranje i pridruživanje šahovskoj partiji

Korisnici mogu započeti novu partiju ili se pridružiti postojećoj.

5. Igranje šaha u realnom vremenu

Sistem mora omogućiti razmenu poteza između igrača u realnom vremenu.

6. **Validacija poteza**
Server mora proveravati ispravnost svakog poteza u skladu sa pravilima šaha.
 7. **Čuvanje stanja i istorije partije**
Sistem mora čuvati sve odigrane poteze i konačne rezultate partija.
 8. **Pregled statistike i rangiranja**
Korisnik može pregledati statistiku svojih partija i rang u odnosu na druge korisnike.
-

2.2 Atributi kvaliteta (ne-funkcionalni zahtevi)

Sistem mora zadovoljiti sledeće ne-funkcionalne zahteve:

- **Performanse**
Sistem mora omogućiti brzo procesiranje poteza i minimalno kašnjenje u realnom vremenu.
 - **Skalabilnost**
Arhitektura mora omogućiti podršku za veliki broj istovremenih korisnika i partija.
 - **Pouzdanost**
Sistem mora obezbediti tačnost stanja partije i sprečiti gubitak podataka.
 - **Bezbednost**
Podaci o korisnicima moraju biti zaštićeni mehanizmima autentifikacije i autorizacije.
 - **Dostupnost**
Sistem treba da bude dostupan korisnicima 24/7, uz minimalne prekide u radu.
 - **Upotrebljivost**
Korisnički interfejs mora biti intuitivan i prilagođen različitim veličinama ekrana.
 - **Održivost i proširivost**
Sistem mora biti lako održiv i omogućiti dodavanje novih funkcionalnosti bez značajnih izmena postojeće arhitekture.
-

2.3 Tehnička i poslovna ograničenja

Tehnička ograničenja

- Sistem mora biti realizovan kao web aplikacija.
- Komunikacija između klijenta i servera vrši se putem HTTP i WebSocket protokola.
- Podaci se čuvaju u centralizovanoj bazi podataka.
- Sistem mora biti nezavisан od platforme i dostupan kroz standardne web pregledače.

Poslovna ograničenja

- Projekat se realizuje u okviru akademskog kursa.
- Vreme i obim razvoja su ograničeni nastavnim planom i rokovima.
- Fokus projekta je na arhitekturi sistema, a ne na potpunoj produkcionoj implementaciji.

3. Projektovanje arhitekturnog dizajna softverskog sistema

Sistem je projektovan primenom kombinacije više arhitekturnih obrazaca, kako bi se obezbedila jasna struktura, održivost i skalabilnost softverskog rešenja. Svaki obrazac je primenjen na odgovarajućem nivou sistema radi rešavanja specifičnih arhitekturnih problema.

3.1 Arhitekturni obrasci i njihova primena

1. Client–Server (Klijent–Server)

- Omogućava razdvajanje korisničkog interfejsa i centralnog serverskog dela sistema.
- Frontend (React aplikacija) komunicira sa backend-om (.NET API) preko HTTP i WebSocket protokola, omogućavajući više korisnika da istovremeno koriste sistem.

2. Layered (Slojevita arhitektura)

- Primenjena unutar backend-a radi organizacije poslovne logike i pristupa podacima.
- Slojevi uključuju:
 - **Controller layer** – prima zahteve od klijentskog dela sistema
 - **Service layer** – implementira poslovna pravila i logiku igre
 - **Domain layer** – definiše modele sistema (npr. Game, Player, Move)
 - **Repository layer** – apstrakcija pristupa bazi podataka
 - **Infrastructure** – integracija sa spoljnim servisima, kao što je Apache Kafka

3. MVC (Model–View–Controller)

- Primena u prezentacionom sloju (React frontend).
- Omogućava razdvajanje stanja aplikacije (Model), prikaza podataka (View) i korisničkih akcija (Controller).
- Na ovaj način se postiže lakše održavanje i event-driven interakcija sa korisnikom.

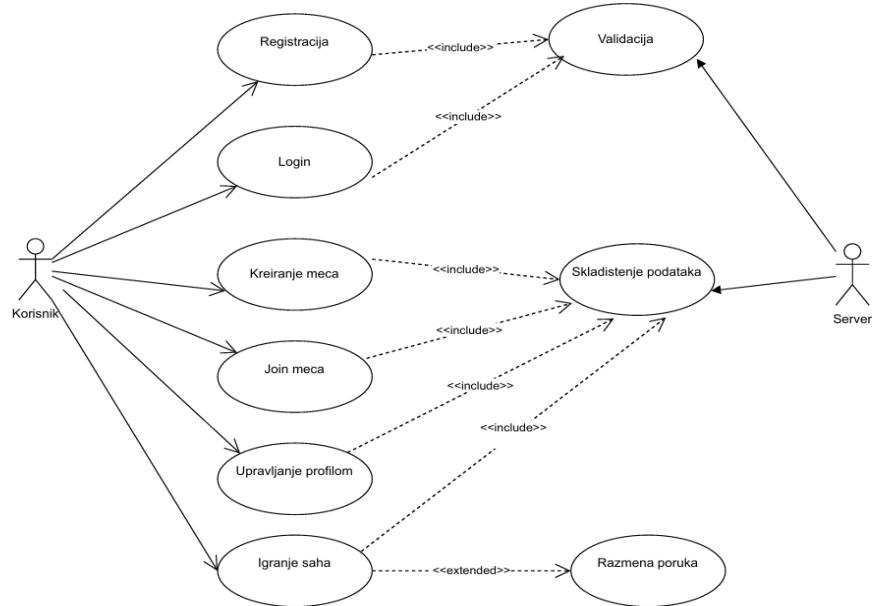
4. Repository

- Koristi se za izolaciju pristupa PostgreSQL bazi podataka od poslovne logike.
- Backend servisi pristupaju bazi preko Repository sloja (npr. GameRepository, UserRepository), što omogućava enkapsulaciju SQL operacija i jednostavnije testiranje poslovne logike.

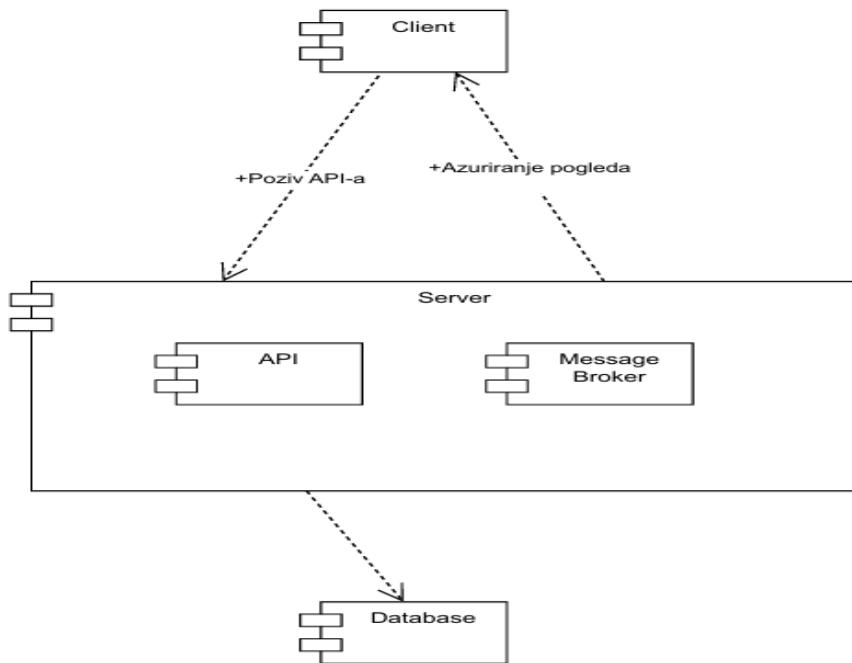
5. Publish–Subscribe / Event-based

- Realizovano korišćenjem Apache Kafka sistema za asinhronu razmenu događaja.
 - Omogućava komponentama sistema da reaguju na događaje, kao što su odigrani potezi, poruke u chatu ili završene partije, bez direktnog pozivanja metoda.
-

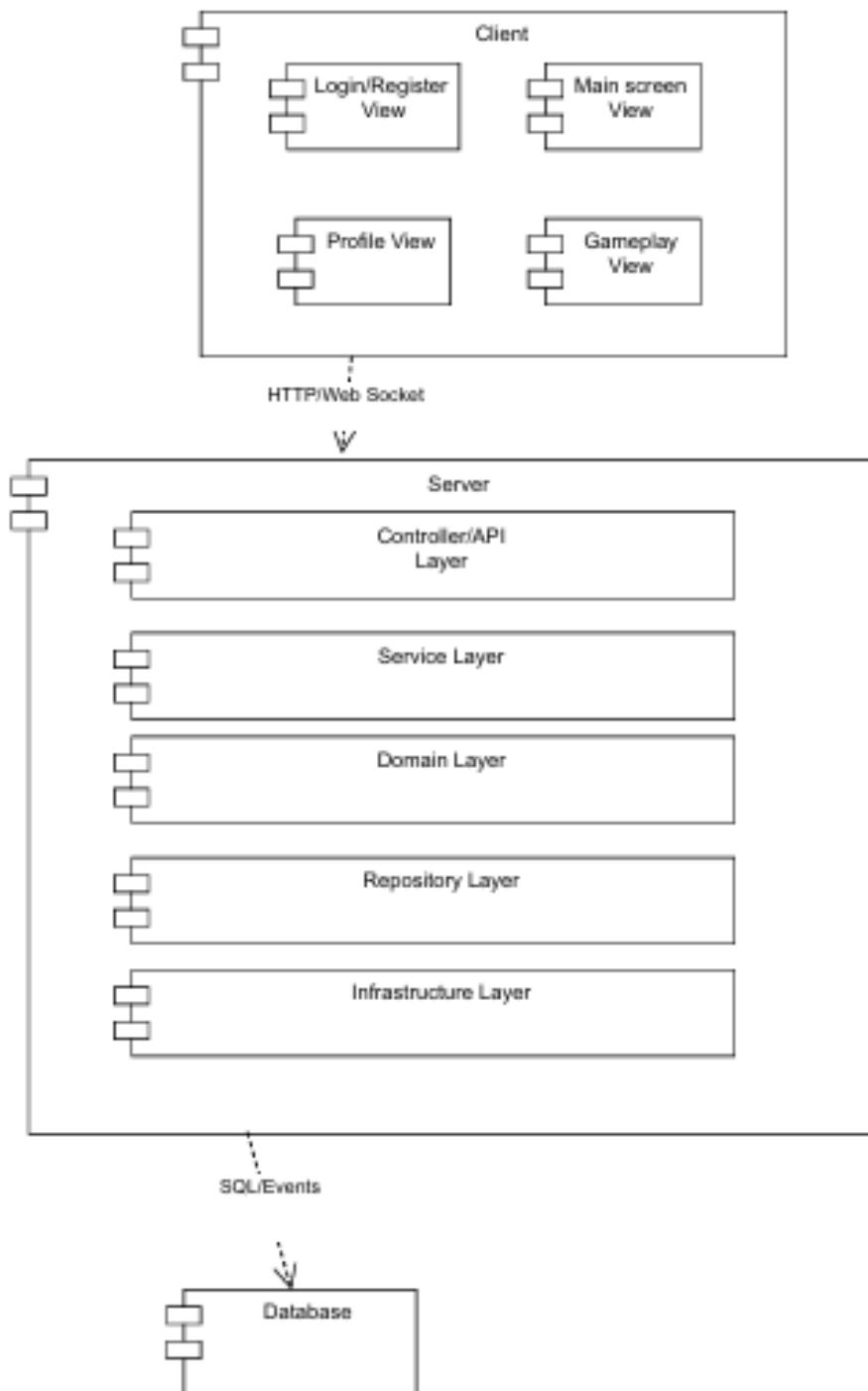
3.2 Use Case dijagram



3.3 Generalna arhitektura (box-line dijagram)

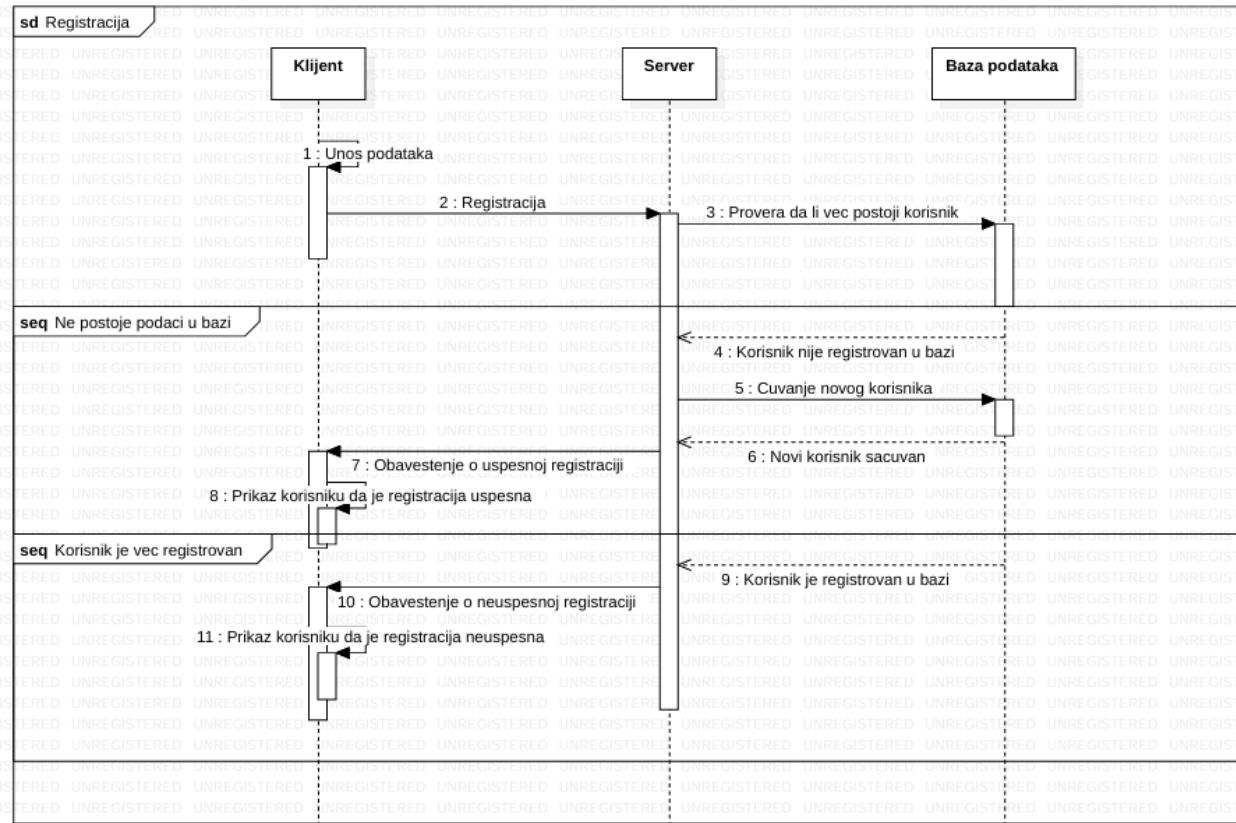


3.4 Strukturni dijagram

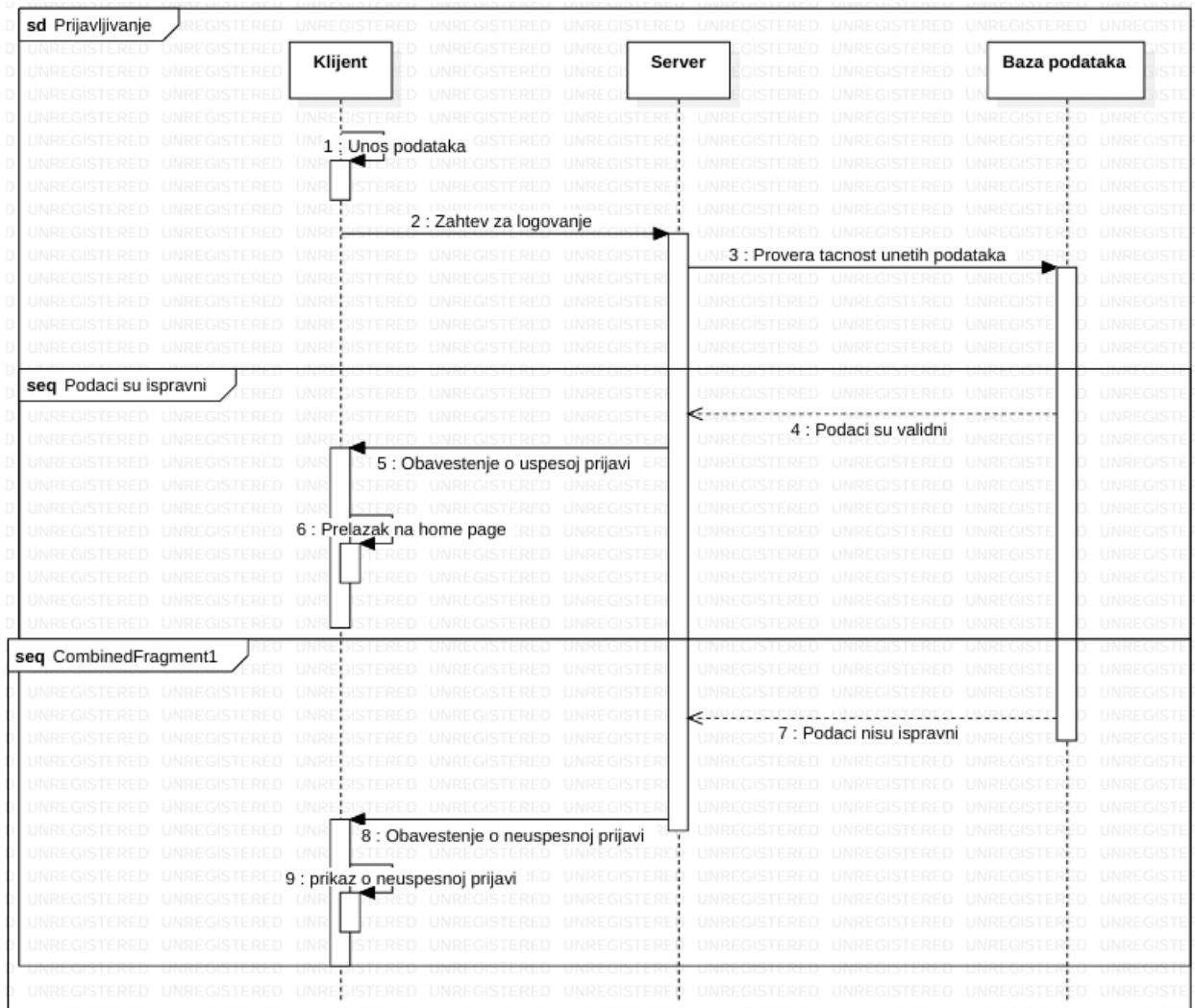


3.5 Bihevioralni dijagram

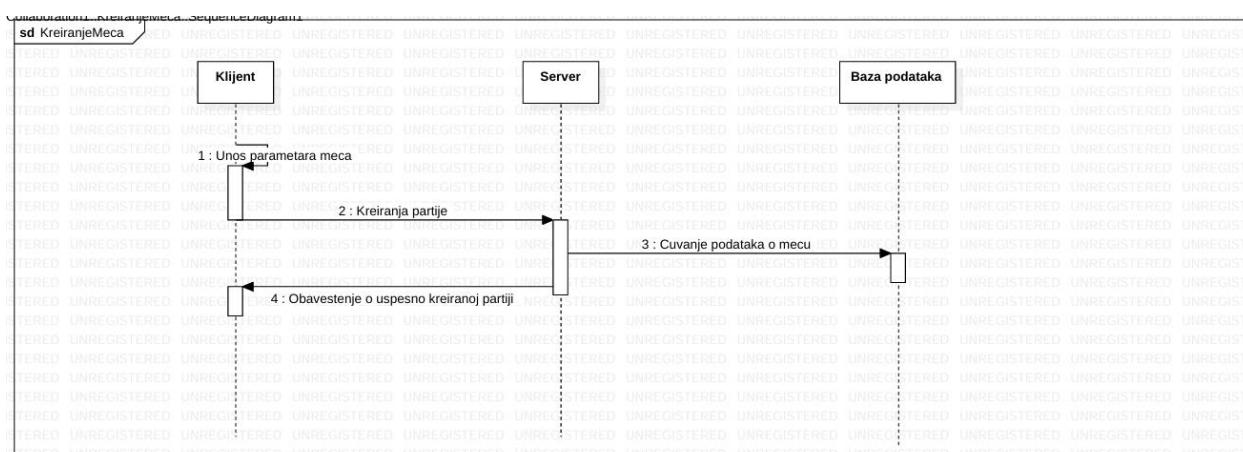
3.5.1 Registracija



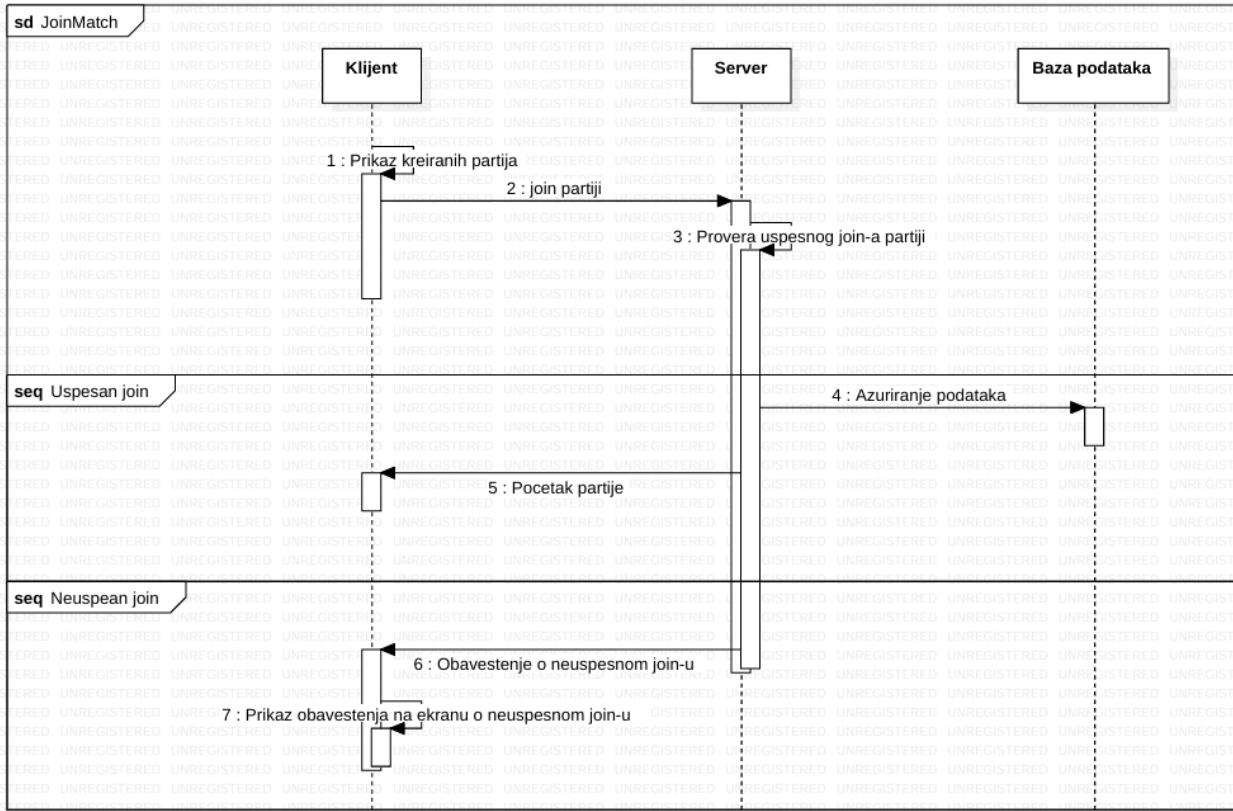
3.5.2 Prijavljivanje



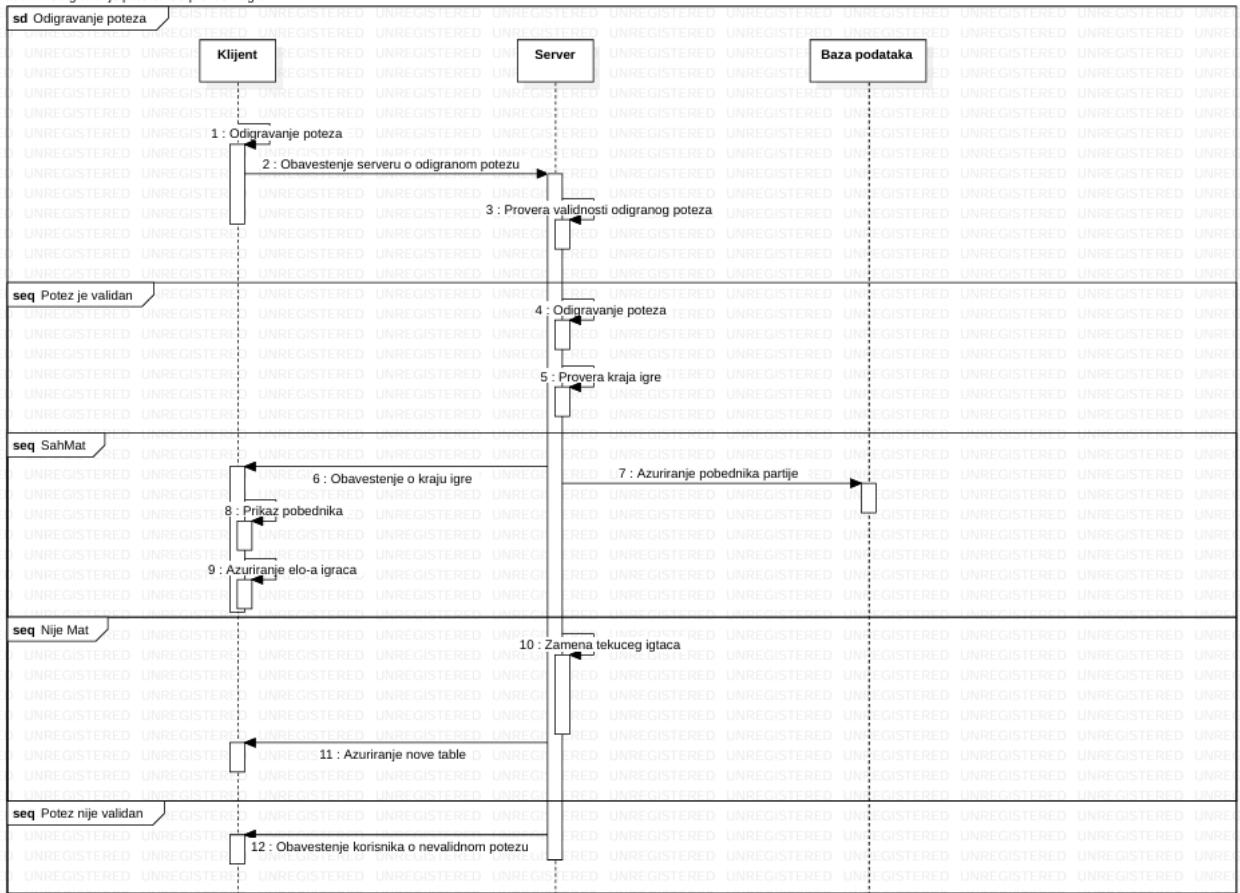
3.5.3 Kreiranje partije



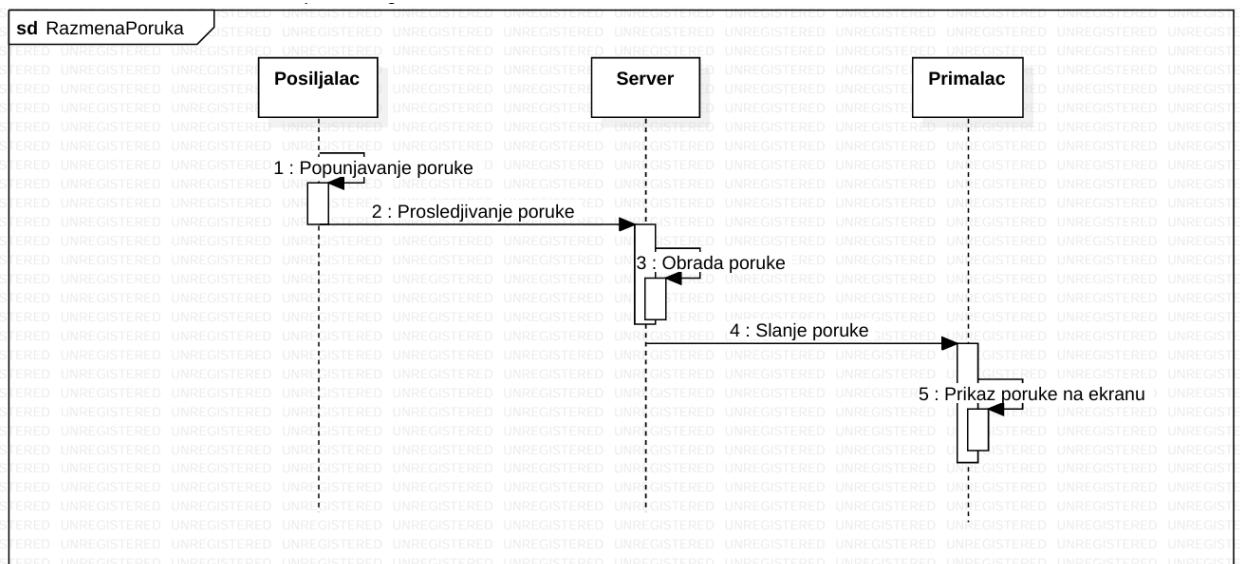
3.5.4 Pridruzivanje partiji



3.5.5 Odigravanje poteza



3.5.6 Razmena poruka



4. Framework i biblioteke

Backend – .NET (ASP.NET Core)

Serverski deo sistema implementiran je korišćenjem **ASP.NET Core (.NET)** framework-a. .NET omogućava izradu performantnog i skalabilnog backend-a sa jasnom podrškom za slojevitu arhitekturu, REST API-je i real-time komunikaciju.

Frontend – React

Za klijentski deo sistema koristi se **React** JavaScript biblioteka.

Frontend komunicira sa backend-om putem HTTP protokola (REST) i WebSocket veze za real-time funkcionalnosti.

Real-time komunikacija – WebSocket (SignalR)

Za real-time komunikaciju koristi se **WebSocket protokol**, implementiran kroz **SignalR** biblioteku u .NET okruženju.

SignalR se koristi za razmenu poteza, sinhronizaciju stanja partije i chat između igrača.

Asinhroni događaji – Apache Kafka

Apache Kafka se koristi za asinhronu obradu događaja i implementaciju event-driven arhitekture.

Baza podataka – PostgreSQL

Za skladištenje podataka koristi se **PostgreSQL** relaciona baza podataka.

U bazi se čuvaju korisnički podaci, partije, potezi i statistika. Pristup bazi je realizovan preko Repository sloja.