Arhitekturni projekat

Higher-Lower

Članovi tima:

Filip Dojčinović, 18135
Jovan Dojčinović, 18134

Naziv tima:

GangOfD

Datum: 30.11.2023..

SADRŽAJ

1 KONTEKST I CILJ PROJEKTA

Higher-Lower predstavlja online multiplayer igru za minimalno 2 igrača koji imaju mogućnost takmičenja među sobom tako što će odgovarati na različita pitanja. Pitanja će obuhvatati različite teme i događaje. Ekran se deli na dva dela gde se sa jedne strane nalazi pogrešan odgovor a sa druge ispravan. Korisnik bira tačan odgovor (levu ili desnu stranu ekrana) i u zavisnosti od pogotka dobija poene. Korisnik može kreirati ili se pridružiti postojećoj igri ukoliko nije zauzeta, odnosno popunjena.

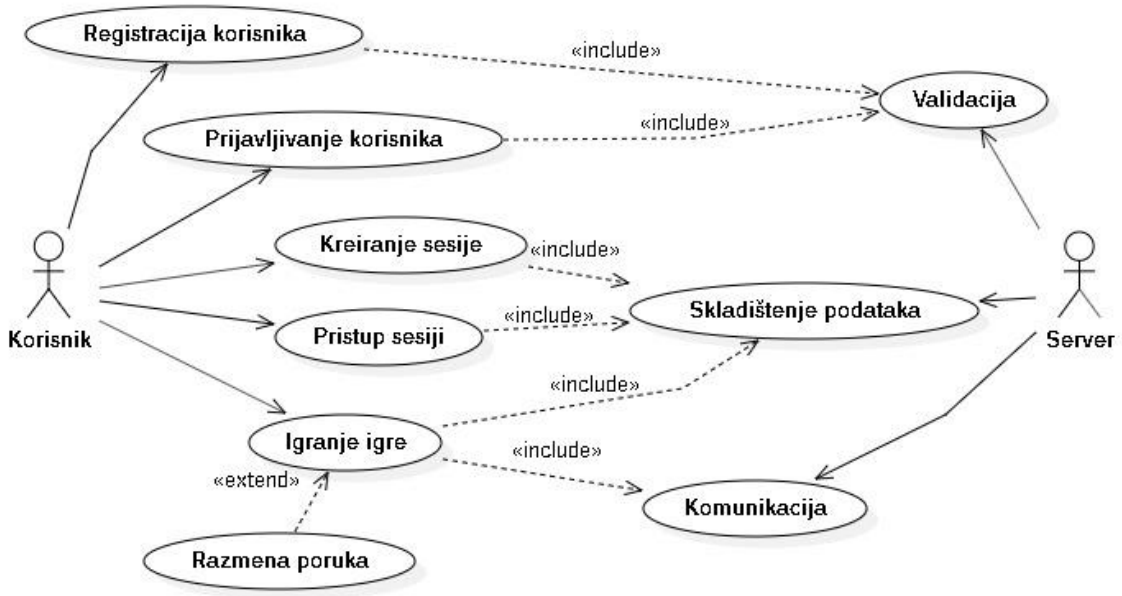
Cilj igre je da korisnik za određeno vreme koje traje igra skupi što više poena. Korisnik koji je skupio najviše poena, odnosno ima najviše tačnih odgovora pobeđuje u igri i dobija određenu nagradu.

2 ARHITEKTURNI ZAHTEVI

2.1 ARHITEKTURNO ZNAČAJNI SLUČAJEVI KORIŠĆENJA

- Registracija korisnika
- Prijava korisnika
- Kreiranje sesije
- Pristup sesiji
- Igranje igre
- Razmena poruka

2.1.1 Use Case dijagram sistema



2.2 NEFUNKCIONALNI ZAHTEVI

- **Pouzdanost** – sistem treba da omogući perzistenciju podataka na početku/kraju poteza igrača. U slučaju prekida konekcije između servera i klijenta, pri ponovnom povezivanju klijent može da nastavi tamo gde je stao.
- **Performanse** – aplikacija treba da obezbedi što manje vreme odziva budući da je reč o sistemu koji radi u realnom vremenu.
- **Dostupnost** – potrebno je da aplikacija bude dostupna u svakom trenutku.
- **Modifikabilnost** – budući da nije moguće predvideti dalji život sistema, potrebno je realizovati sistem tako da podržava lako dodavanje novih funkcionalnosti i laku izmenu već postojećih bez većih uticaja na sam sistem.
- **Skalabilnost** – potrebno je podržati rast broja korisnika u budućnosti.
- **Upotrebljivost** – potrebno je da aplikacija bude intuitivna i jednostavna za korišćenje.
- **Sigurnost** – bitno je da aplikacija pruža korisniku sigurnost zbog toga što svaki korisnik ima lični profil. Iz tog razloga potrebno je implementirati sistem autentifikacije korisnika.

2.3 TEHNIČKA I POSLOVNA OGRANIČENJA

- **Pristup preko web-a** – potrebno je obezbediti korisnicima da pristupe sistemu putem weba, te shodno tome koristiti adekvatne web tehnologije koje će omogućiti interakciju i komunikaciju između korisnika i sistema.
- **Komunikacija** – sistem treba da podrži dva različita tipa komunikacije, sinhronu (između klijenta i servera) i asinhronu (propagacija izmena od jednog klijenta ka ostalima, kao i razmena poruka među klijentima).
- **Skrivenost šeme baze podataka** – korisnicima su dostupni samo podaci predviđeni za prikaz, dok se od njih krije način reprezentacije podataka u bazi.

Poslovna ograničenja sistema baziraju se na pravilima same igre i od njih zavisi koje će akcije korisnik moći da obavi u datom trenutku.

3 ARHITEKTURNI DIZAJN

3.1 ARHITEKTURNI OBRASCI

U nastavku su dati arhitekturni obrasci koji će biti iskorišćeni za realizaciju sistema.

3.1.1 *Layered* obrazac

Aplikacija će imati tri sloja – prezentacioni sloj klijenta, serverski sloj, kao i sloj perzistencije, koji obuhvata skladište podataka. Prezentacioni sloj se izvršava na klijentu i ima ulogu posrednika između aplikacije i klijenta u vidu interakcije sa sistemom. Prezentacioni sloj će komunicirati sa slojem ispod sebe, a to je serverski sloj.

Serverski sloj će se izvršavati na serveru i implementirati sve bitne funkcionalnosti koje aplikacija treba da pruži. U okviru serverskog sloja, implementiraće se poslovna logika sistema, odnosno metode potrebne za odigravanje igre, ali i metode potrebne za perzistenciju podataka koje generiše aplikacija. Serverski sloj komunicira sa prezentacionim slojem, a komunikacija sa klijentima se odigrava direktno preko implementiranog RESTful API-ja, ili posredno, korišćenjem *Message broker-a (Socket.IO)*.

Sloj perzistencije predstavlja samu bazu podataka, koja će skladištiti podatke relevantne za aplikaciju.

3.1.2 *Model-View-Controller* obrazac

MVC (Model-View-Controller) arhitektura predstavlja organizacioni obrazac u softverskom dizajnu koji razdvaja komponente sistema radi lakšeg održavanja i skaliranja. Model predstavlja domenske klase koje su odgovorne za predstavljanje podataka i logike poslovanja aplikacije. Komunicira sa bazom podataka i održava integritet podataka. Model ne zna ništa o prikazu ili načinu na koji se podaci prikazuju korisnicima. View ne sadrži poslovnu logiku, već samo prikazuje podatke koji su joj dostavljeni od strane modela, reaguje na promene u modelu i ažurira se kako bi odražavala trenutno stanje podataka. Kontroleri su odgovorni za obradu korisničkih zahteva i interakciju sa modelom. Oni prihvataju ulazne komande od korisnika (npr. klik na dugme) i prosleđuju ih modelu ili view-u, ažuriraju model na osnovu korisničkih akcija i ažuriraju view prema promenama u modelu.

3.1.3 *Publish-Subscribe* obrazac

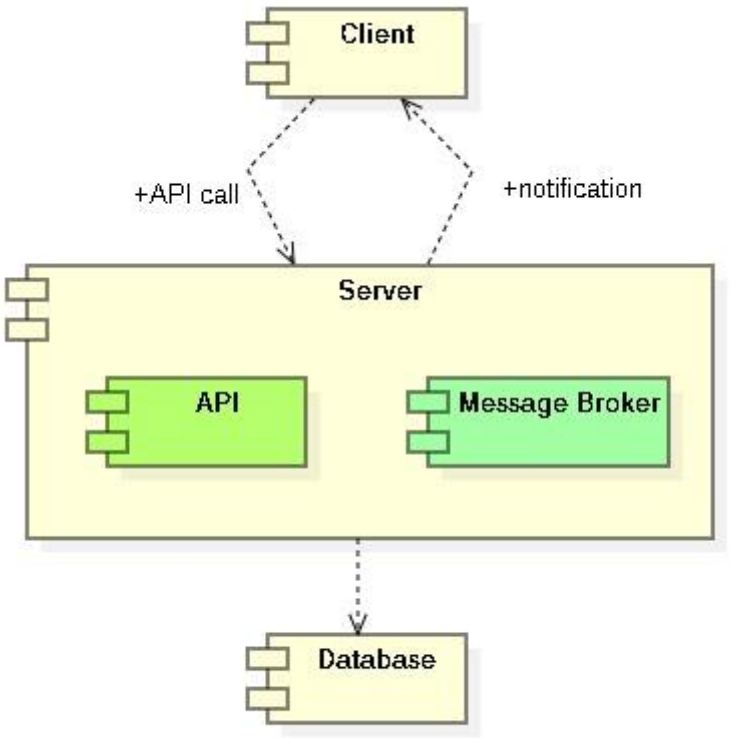
Aplikacija zahteva rad u realnom vremenu i komunikaciju između korisnika. Ovo će se postići implementacijom *Publish-Subscribe* obrasca koji će koristiti *Socket.IO* biblioteku za ostvarivanje pravovremenog ažuriranja na klijentima. Klijenti će se pretplatiti na temu koja je za njih relevantna i dobijati ažuriranja onda kada je to potrebno. U slučaju kada je potrebno izvršiti akciju nad nekom temom, klijent će slati poziv API-ju koji će aktivirati *publish* sistem, koji zatim obaveštava ostale klijente pretplaćene na pomenutu temu.

3.1.4 *Repository* obrazac

Predstavlja centralnu tačku pristupa podacima u aplikaciji. Odgovoran je za komunikacju sa bazom podataka. Ono što je prednost ovog obrasca je da omogućava da ostatak aplikacije ne mora da zna kako se podaci čuvaju i dobijaju iz baze. Takođe i promene u logici pristupa podacima ne utiču na druge delove aplikacije. Komponente sloja servera komuniciraju sa skladištem pozivom odgovarajućih metoda. Stanje sistema ogleda se u stanju partija igara koje su pokrenute u datom trenutku, i pošto ih ima više, potrebno je obezbediti konkurentan pristup skladištu.

3.2 GENERALNA ARHITEKTURA

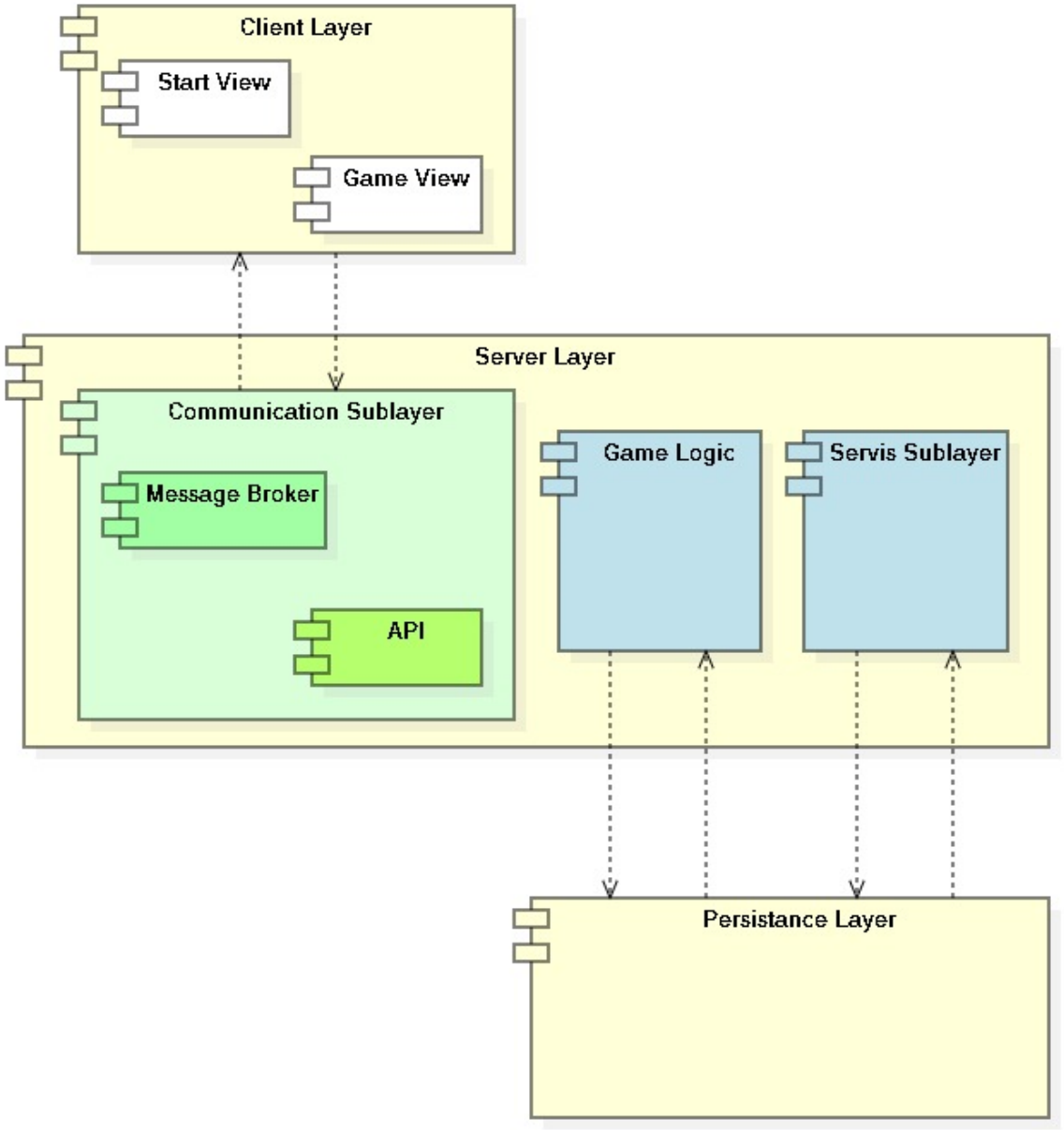
Arhitektura sistema podrazumeva postojanje klijenta, servera i baze podataka u kojoj će se čuvati informacije o korisnicima i njihovim igrama.



3.3 STRUKTURNI POGLEDI

Sledeći dijagram ilustruje strukturu sistema navodeći komponente sistema kao i njihovu međusobnu povezanost. Struktura klijentske aplikacije zasnovana je na *Model-View-Controller* arhitekturnom obrascu. Klijentska i serverska aplikacija ostvaruju sinhronu komunikaciju korišćenjem RESTful API servisa. Asinhrona komunikacija između klijentske i serverske aplikacije ostvarena je pomoću *Publish-Subscribe* arhitekturnog obrasca. Serverska aplikacija je zadužena za komunikaciju sa bazom podataka.

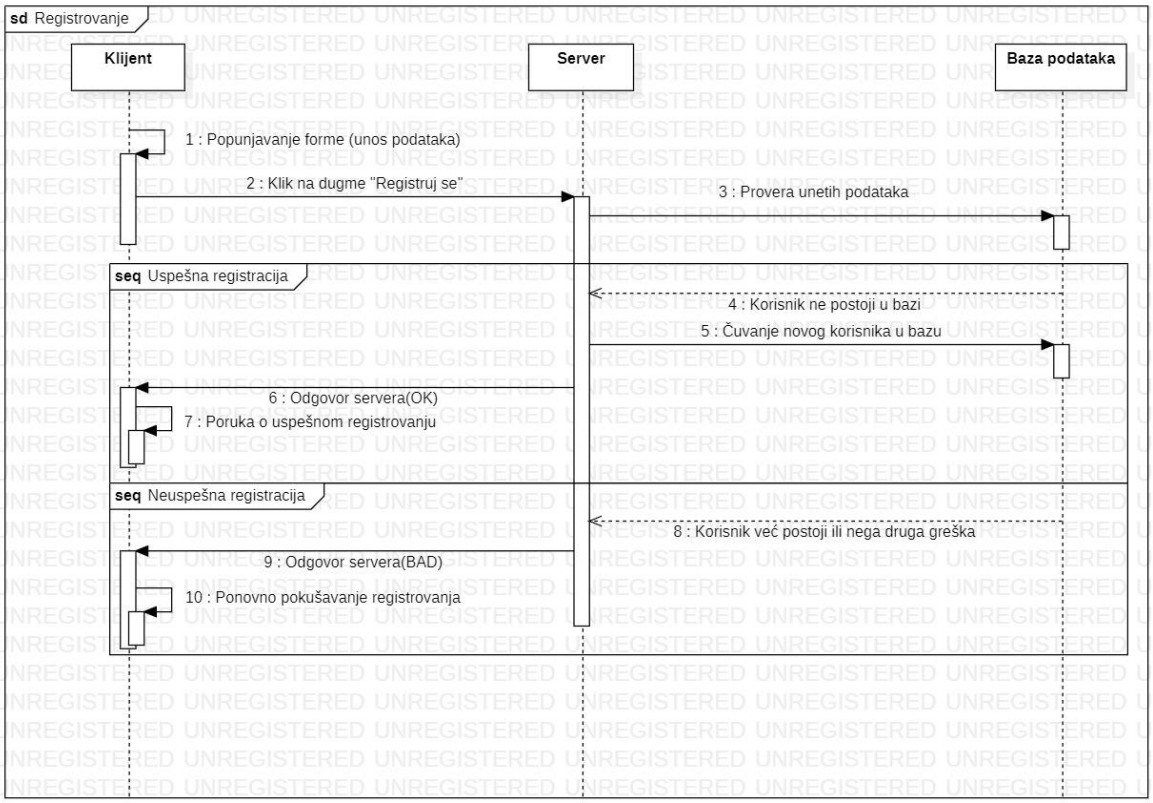
Klijentska aplikacija je implementirana kao Web aplikacija i predviđena je za rad na svim računarima sa pristupom Internetu. Serverska aplikacija se realizuje pomoću Express JS-a, dok se kao sistem za upravljanje bazom podataka koristi MongoDB. Za komunikaciju između klijenata se koristi biblioteka *Socket.IO* koja igra ulogu brokera poruka (*message broker*).



3.4 BIHEJVIORALNI POGLEDI

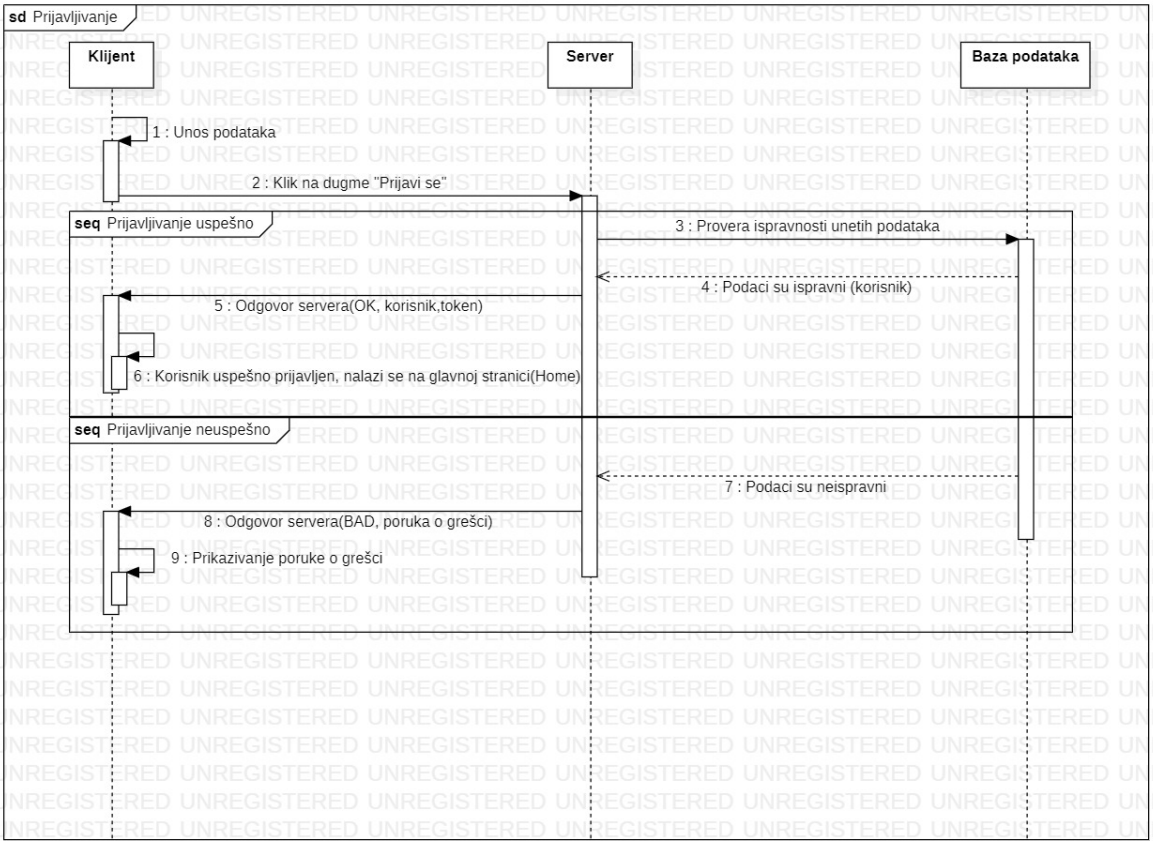
3.4.1 Registracija korisnika

Korisnik unosi odgovarajuće podatke popunjavanjem forma, klijent šalje zahtev za registraciju serveru sa unetim podacima. Server proverava da li se u bazi nalazi korisnik sa već izabranim korisničkim imenom. Ukoliko da, šalje se obaveštenje klijentu da je korisničko ime već zauzeto. U suprotnom, novi korisnik se pamti u bazi, i klijent se obaveštava o uspešnoj registraciji.



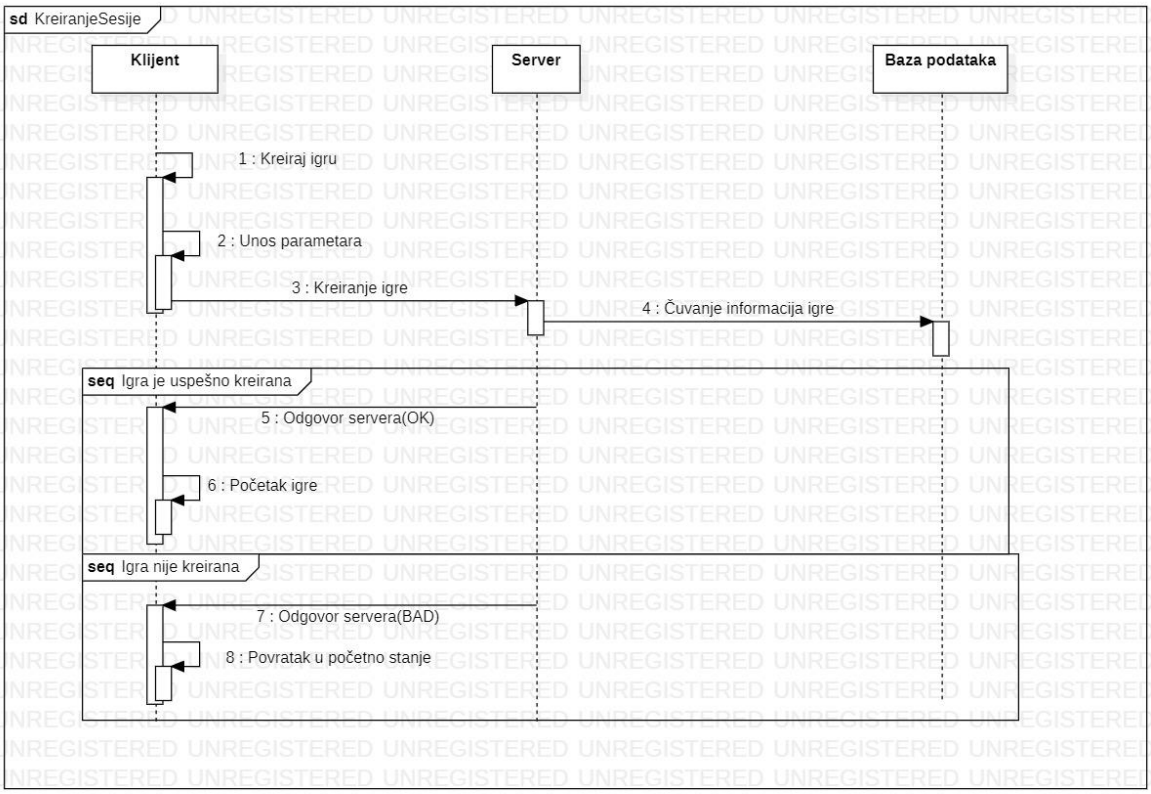
3.4.2 Prijava korisnika

Klijent unosi neophodne podatke za prijavljivanje i šalje zahtev serveru. Server proverava validnost podataka, i ako nisu validni klijent se obaveštava o tome. U suprotnom, nakon obaveštenja o uspešnosti prijave, klijent pristupa početnoj strani.



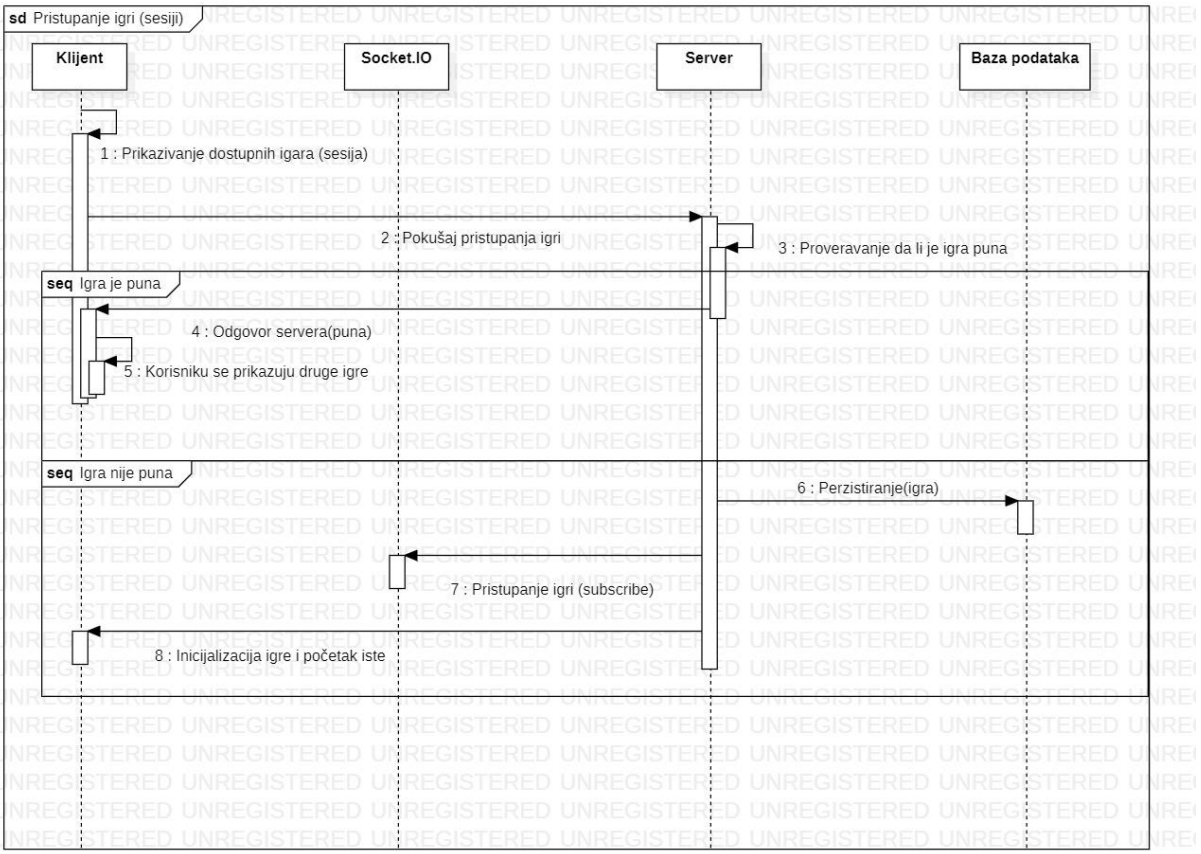
3.4.3 Kreiranje sesije

Klikom na dugme „*Host Game*“ (Kreiraj igru) klijent se opredeljuje za kreiranje nove sesije i prikazuje se forma za kreiranje igre. Nakon unosa određenih parametara, klikom na dugme „*Create Game*“ serveru se, sa unetim parametrima, šalje zahtev za kreiranje nove igre. Parametri se zatim čuvaju u bazi, a klijentu se šalje povratna informacija o njegovom zahtevu kako u slučaju uspešnog tako i u slučaju nesupešog kreiranja igre. Klijent zatim pristupa *game* stranici gde čeka da se ostali igrači priključe.



3.4.4 Pristup sesiji

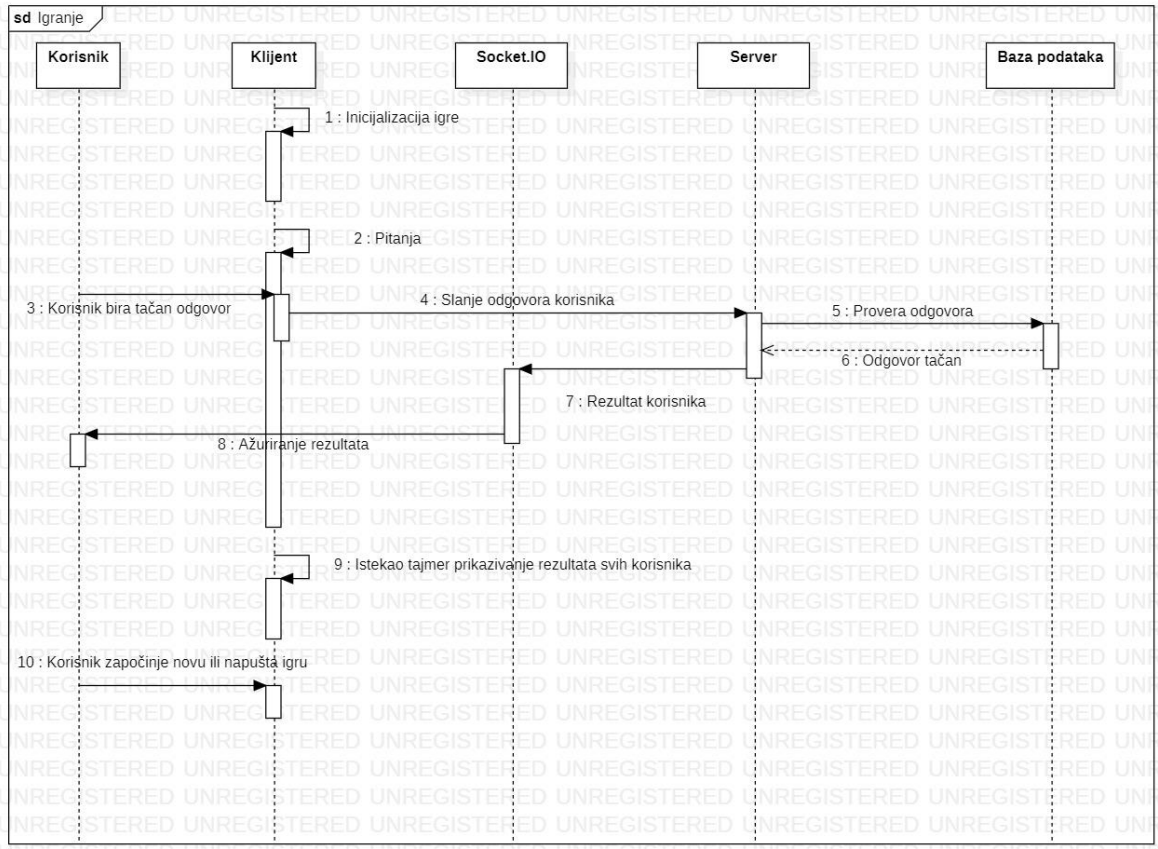
Klikom na dugme „*Join game*“ klijent šalje zahtev serveru za pristup igri , server proverava da li je pun, i ako jeste, klijent se obaveštava o tome i vraća se nazad na osvežen izbor. U slučaju da *game* nije pun, baza se osvežava, klijent se obaveštava o uspehu i pristupa izabranom *game-u*, a server šalje informaciju soketu koji obaveštava sve klijente iz *game-*a o novom igraču, i svim igračima se osvežava strana.



3.4.5 Odigravanje poteza

Pre nego što *game* počne postavlja se štoperica kojom se određuje trajanje partije, kao i postavljanje broja poena svih igrača na nulu. Igra počinje i ekran se deli u dva dela, na sredini ekrana se nalazi pitanje. Na jednoj strani se nalazi jedan odgvor koji je prezentovan određenom ilustracijom tako da asocira odnosno da je povezana sa odgovorom koji može biti pogrešan ili ispravan. Klikom na levi ili desni deo ekrana korisnik se opredeljuje za svoj odgovor i dobija poen ukoliko je tačno odgovorio na pitanje. Pitanja obuhvataju širko spektar znanja opšte informisanosti tako da zabava neće biti problem. Nakon isteka vremena određuje se pobednik. Korisnici imaju mogućnost napuštanja trenutnog *game-a* ili ponovno učestvovanje u istom.

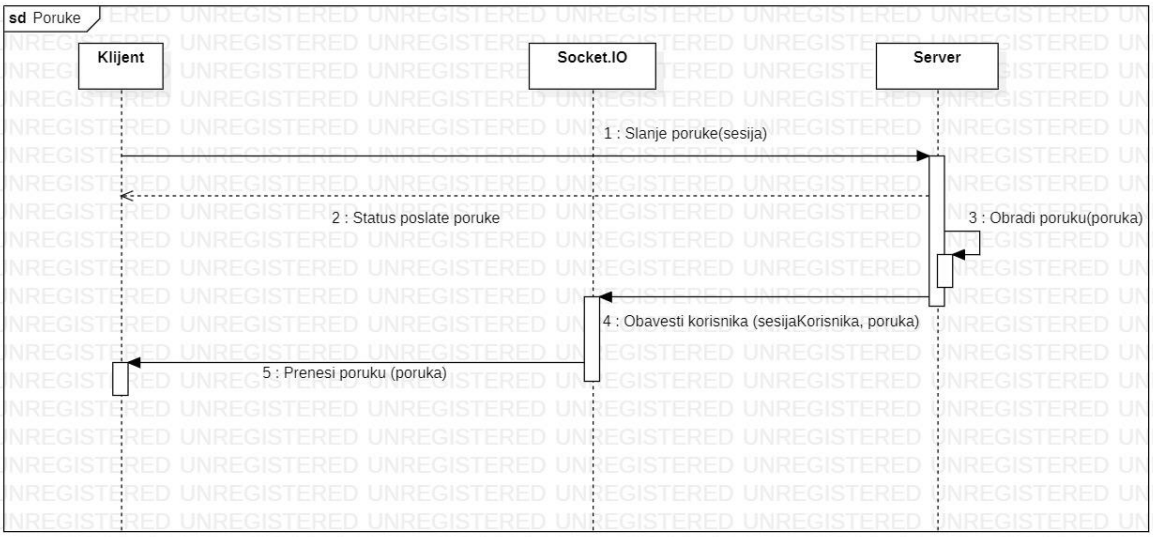
Dijagram interakcije koji opisuje odigravanje poteza dat je na sledećoj slici:



3.4.6 Razmena poruka

Razmena poruka se odvija grupno, u okviru jedne partije. Klijent šalje poruku koju prihvata server, obrađuje odakle je stigla i kojoj partiji pripada, i zatim poziva soket koji ima zadatak da poruku prosledi svim zainteresovanim stranama, odnosno, igračima u toj partiji.

Sledeći dijagram interakcije predstavlja pojednostavljenu sliku o razmeni poruka kroz sistem:



3.5 IMPLEMENTACIONA PITANJA

Specifikacija biblioteka i programskih okvira:

- **React** – JavaScript biblioteka za pisanje Web korisničkih interfejsa,
- **SocketIO** – Biblioteka koja koristi Internet soket (Websocket koji igra ulogu brokera poruka) , omogućava API za real-time klijent-server komunikaciju,
- **Expres JS** – Serverska aplikacija,
- **MongoDB** – Baza podataka orijentisana dokumentima
- **Mongoose-** MongoDB ODM

4 ANALIZA ARHITEKTURE

4.1 POTENCIJALNI RIZICI U IMPLEMENTACIJI I STRATEGIJE PREVAZILAŽENJA

Jedan od rizika koji postoje jeste činjenica da se koristi centralizovano skladište podataka. Ovaj rizik može prerasti u problem ukoliko se broj korisnika ekstremno poveća do tačke kada centralizovano skladište nije u stanju da odgovori na sve zahteve pravovremeno. Budući da se radi o online igrici, vreme odziva je ključno, te sistem treba projektovati tako da je razdvajanje skladišta podataka na više distribuiranih skladišta lako izvodljivo. Ka tome će se težiti postizanjem slabe sprege između sloja perzistencije i viših slojeva u okviru sistema.