Duel Nexus

Arhitekturni projekat

Lazar Mančić 19206
Momčilo Marjanović 19206

Datum: 30.12.2025.

Sadržaj

1 KONTEKST I CILJ SOFTVERSKOG PROJEKTA

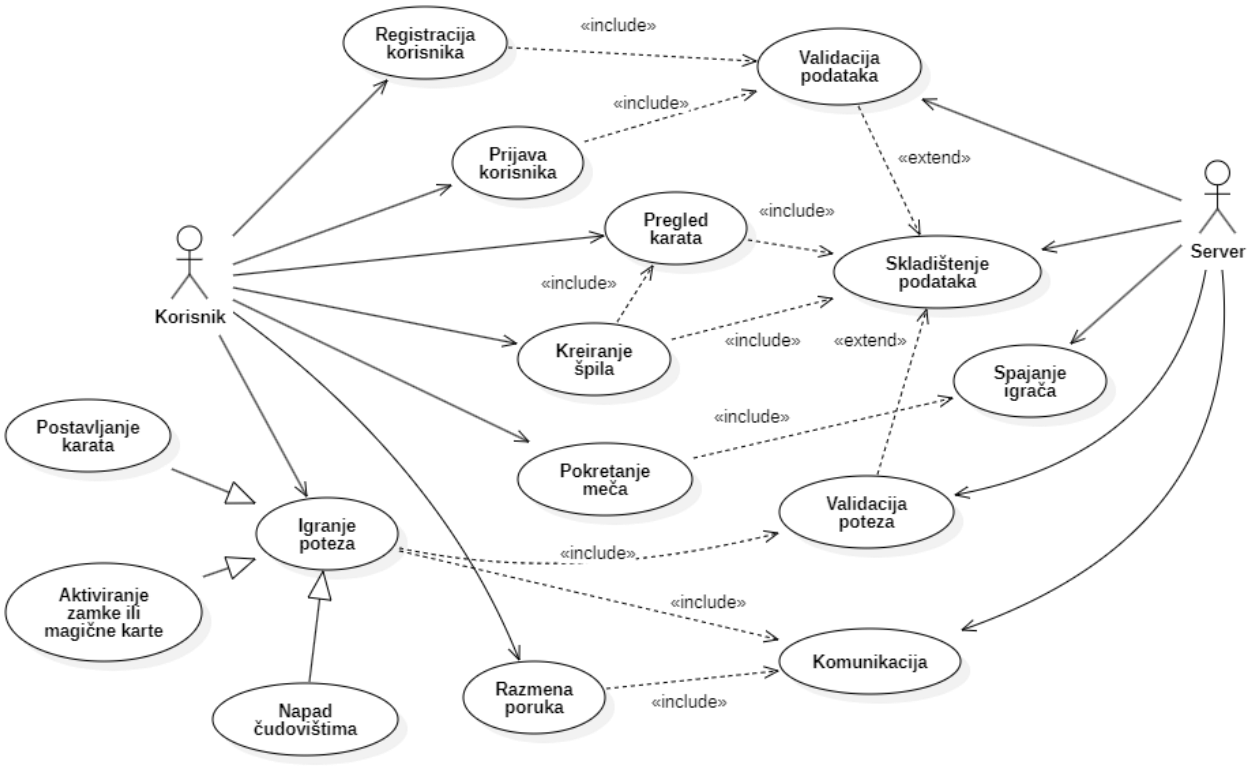
Softverski projekat predstavlja projektovanje i implementaciju *online* kartaške igre za više igrača baziranu na potezima (*turn-based*) u vidu veb aplikacije pod nazivom *Duel Nexus*. Na početku, svaki od igrača sastavlja špil karata koji će koristiti za bitku sa drugim igračima. U špilu se mogu naći više tipova karata: normalna čudovišta, čudovišta sa efektima, magičnih karata (*spell card*) i karte zamki (*trap card*). Bitka se može započeti pridruživanjem nasumičnom meču ili kreiranjem meča sa prijateljima.

Cilj igre je pobediti u bici strateškim odabirom karata koje se izvlače iz špila za vreme partije. Na početku partije svi igrači dobijaju životne poene i izvlače 5 karata iz špila, da bi pobedili oni moraju napasti protivnikove karte ili direktno napasti njega kako bi smanjili životne poene. Onog trenutka kada ostane jedan igrač sa životnim poenima većim od nule. Bitka se odvija u ciklusima, svaki ciklus se sastoji od izvlačenja i postavljanja karata na tabli, bitke i aktiviranja magičnih i karata zamki.

2 ARHITEKTURNI ZAHTEVI

2.1. Glavni funkcionalni zahtevi

- **Registracija korisnika** – Potrebno je omogućiti novim korisnicima kreiranje naloga kako bi mogli da igraju igru.
- **Prijava korisnika** – Potrebno je omogućiti korisnicima pristup njihovom nalogu.
- **Validacija podataka** – Sistem treba da omogući proveru podataka korisnika koji pokušavaju da se prijave ili registruju.
- **Pregled karata** – Potrebno je omogućiti korisniku da pregleda svoj špil karata.
- **Kreiranje špila** – Potrebno je omogućiti korisniku kreiranje i promenu svog špila od karata koje su mu dostupne, odnosno koje su u njegovom posedu.
- **Pokretanje meča** – Potrebno je omogućiti korisniku da pokrene meč sa drugim korisnicima.
- **Spajanje korisnika** – Sistem treba da omogući spajanje korisnike koji si pristupili pokreranju meča.
- **Igranje poteza** – Potrebno je omogućiti korisniku da odigra potez kada je u toku meča i kada je njegov red.
- **Validacija poteza** – Sistem treba da omogući proveru i izvršavanje poteza koji je igrač koji je na redu odigrao.
- **Završetak meča** – Sistem treba da omogući proveru da li je meč završen, odnosno da li je neki od igrača pobedio.
- **Razmena poruka** – Potrebno je omogućiti korisnicima da razmenjuju poruke.
- **Komunikacija** – Sistem treba da omogući komunikaciju između korisnika.



Slika 1 – Dijagram slučajeva korišćenja (Use-Case Diagram).

2.2. Ne-fukncionalni zahtevi (atributi kvaliteta)

- **Pouzdanost** – Sistem treba da omogući perzistenciju podataka poteza igrača. U slučaju gubitka konekcije između klijenta i servera, pri ponovom povezivanju korisnik može da nastavi tamo gde je stao, ukoliko se u međuvremenu meč nije završio.
- **Performanse** – Aplikacija treba da korisnicima obezbedi što manje vreme odziva, kako se radi o igri u kojoj se potezi odigravaju u realnom vremenu, i vreme akcije je bitno za sami tok mečeva.
- **Dostupnost** – Aplikacija treba da bude dostupna u svakom trenutku i da obezbedi kratko vreme oporavka u slučaju privremene nedostupnosti.
- **Modifikabilnost** – Sistem treba da bude razvijen tako da podržava lako dodavanje novih funkcionalnosti i laku izmenu postojećih bez većih uticaja na sistem.
- **Skalabilnost** – Sistem treba da bude u mogućnosti da podrži rast broja korisnika u budućnosti, sa što manjim uticajem na ostale atribute kvaliteta.
- **Upotrebljivost** – Aplikacija treba da bude intuitivna i jednostavna za korišćenje.
- **Sigurnost** – Sistem treba da pruža korisniku sigurnost zbog toga što svaki korisnik ima svoj nalog. Iz tog razloga su potrebne autentifikacija i autorizacija korisnika.
- **Bezbednost** – Sistem treba da poseduje zaštitu od namernih, zlonamernih akcija, kako ne bi došlo do kompromitovanja servera.

2.3. Tehnička i poslovna ograničenja

Tehnička ograničenja su sledeća:

- **Pristup preko veb pregledača** – Potrebno je omogućiti korisnicima pristup aplikaciji preko većine stabilnih veb pregledača, shodno tome potrebno je odabrati tehnologije za izradu veb aplikacije koje podržavaju razne vrste veb pregledača i omogućavaju što bolju optimizaciju same aplikacije.
- **Komunikacija** – Sistem treba da podrži dva tipa komunikacije, sinhronu (između klijenata i servera) koja se odnosi na samu prirodu veb aplikacija, i asinhronu (prosleđivanje poruke/akcije jednog klijenta ostalima, odnosno razmena poruka među klijentima) koja se odnosi na interakciju unutar same igre.
- **Apstrakcija podataka** – Korisnicima su dostupni samo podaci koji su predviđeni za prikaz, dok se od njih krije način reprezentacije podataka u bazi, odnosno sama šema baze podataka.

Poslovna ograničenja sistema se baziraju na samim pravilima igre i od njih zavise akcije koje će korisnik moći da obavi u određenom trenutku.

3 ARHITEKTURNI DIZAJN

3.1. Arhitekturni obrasci

U nastavku su navedeni arhitekturni obrasci koji će biti korišćeni za realizaciju sistema. Na odabir ovih obrazaca je uticala sama priroda sistema i aplikacije (*web*), ali i sam kontekst projekta, odnosno sama igra.

3.1.1. *Layered* obrazac

Sistem će se sastojati od tri sloja:

- **Prezentacioni sloj** – Sloj koji će se izvršavati na klijentskoj strani i ima ulogu posrednika između aplikacije i klijenata u vidu interakcije sa sistemom. Prezentacioni sloj komunicira sa slojem ispod sebe, odnosno sa serverskim slojem.
- **Serverski sloj** – Sloj koji će se izvršava na serverskoj strani i implementirati sve bitne funkcionalnosti koje aplikacija treba da pruži. U okviru ovog sloja, implementiraće se poslovna logika sistema, odnosno funkcije za igranje same igre, ali i funkcije za čuvanje (perzistenciju) podataka koje generiše aplikacija na osnovu interakcije sa klijentima. Kako bi došlo do interakcije ovaj sloj komunicira sa prezentacionim slojem, a da bi došlo do čuvanja podataka komunicira sa slojem ispod sebe, slojem perzistencije.
- **Sloj perzistencije** – Sloj koji će predstavljati baza podataka, koja će skladištiti podatke koje je potrebno sačuvati i ponovno koristiti. Može se nalaziti na istom serveru kao i serverski sloj, ali je svakako logički odvojen, i ne mora se nalaziti tamo.

3.1.2. *Model-View-Controller (MVC)* obrazac

Sistem će biti organizovan po *MVC* obrascu. Modeli (*Models*) u sistemu su domenske klase orijentisane prema samoj aplikaciji, odnosno igri, a podaci unutar baze se odgovarajući mapiraju. Pogled (*View*) jeste ono što klijent vidi, odnosno deo aplikacije koji se izvršava na klijentu. Preko pogleda se prikazuju samo relevatni podaci za samog klijenta, ostali se od njega sakrivaju. Delovanje *View*-a na modele se odvija preko kontrolera (*Controller*) koji se izvršavaju na serveru. Kontroleri takođe preuzimaju i ulogu izmene modela na osnovu korisničkih instrukcija, a sa druge strane ažuriraju pogled na osnovu promenjenih modela, pa tako predstavljaju vrstu posrednika između korisnika i podataka i poslovne logike.

3.1.3. *Publish-Subscriber* obrazac

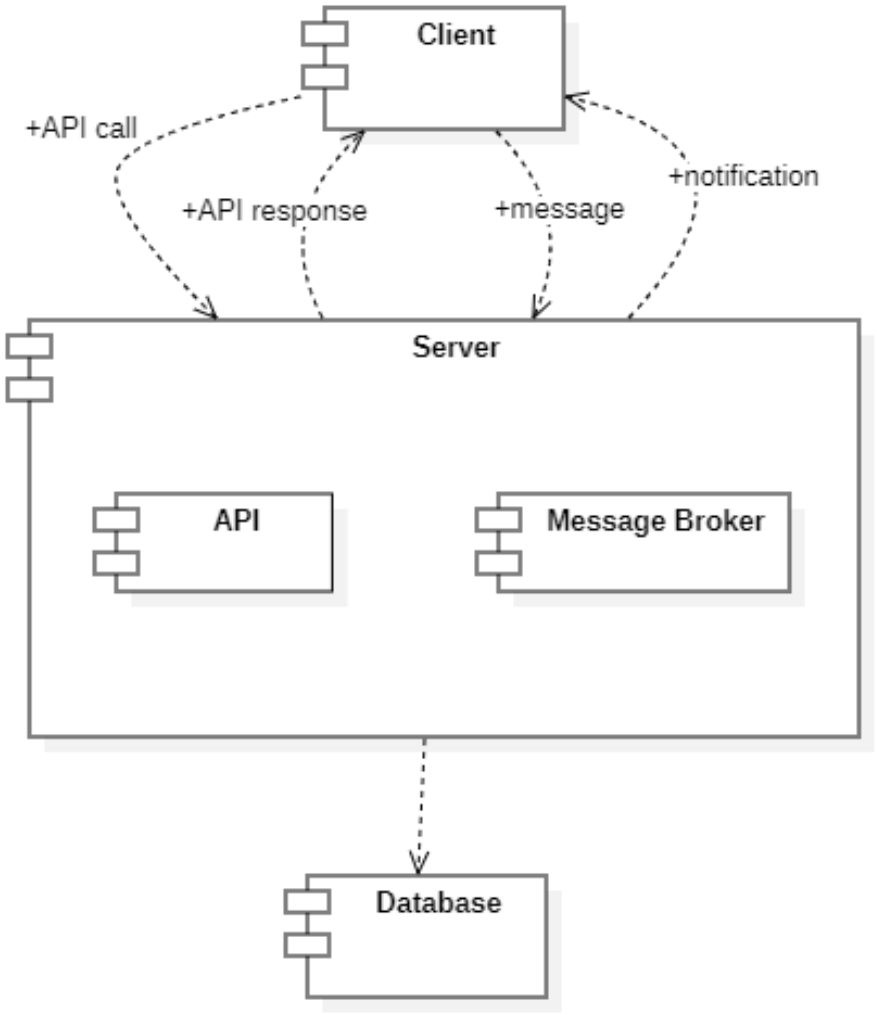
Igranje igre, odnosno sam rad aplikacije, zahteva komunikaciju između korisnika u realnom vremenu. Obrazac kojim će se ovo implementirati je *Publish-Subscriber* obrazac, koji će koristiti *message-broker* komponentu za ostvarivanje pravovremenog ažuriranja na klijentima. Klijenti se pretplaćuju (*subscribe*) za prijem specifičnih poruka ili specifičnog sadržaja, kako bi dobijali ažuriranja kada je to potrebno, tako da oni predstavljaju pretplatnike. Publikatori, koji se izvršavaju na serverskoj strani, šalje poruke/sadržaj po *broadcast* principu pretplatnicima na asinhron način.

3.1.4. *Repository* obrazac

Implementacija perzistencije će biti organizovana kroz centralizovano skladište pasivnog tipa. Komponente severskog sloja komuniciraju sa skladištem pozivom odgovarajućih metoda interfejsa između aplikacije i skladišta. Postojanje više istovremenih partija igara, samim tim i stanja partija, u kojima se oglega stanje sistema, potrebno je obezbediti konkurentan pristup skladištu podataka. Obezbeđuje se perzistencija domenskih modela poslovne logike mapiranjem na modele podataka u sloju perzistencije, odnosno u bazi podataka. Implementacija je obezbeđena kroz *Dotnet Entity Framework*, koji je iskorišćen za realizaciju baze podataka po *code-first* principu i pristupu, odnosno komunikaciju, sa samom bazom podataka.

3.2. Generalna arhitektura sistema

Arhitektura sistema podrazumeva postojanje klijenata, servera i baze podataka u kojoj će se čuvati informacije o korisnicima i njihovim igrama, kartama i druge relevatne informacije za sam sistem i korisnike. Primenom obrazaca je obezbeđeno da klijent ne komunicira direktno sa bazom podataka, samim tim i nema pristup svim podacima. Klijent komunicira sa serverom i to sinhrono preko *API*-a ili asinhrono razmenom poruka putem *message-broker* komponente, preko čega on dobija podatke koji su potrebni. Server na osnovu zahteva koje prima preko *API*-a, ili poruka koje je dobio *message-passing* komponentom, izvršava poslovnu logiku, i ukoliko je potrebno čita ili ažurira podatke u bazi podataka.

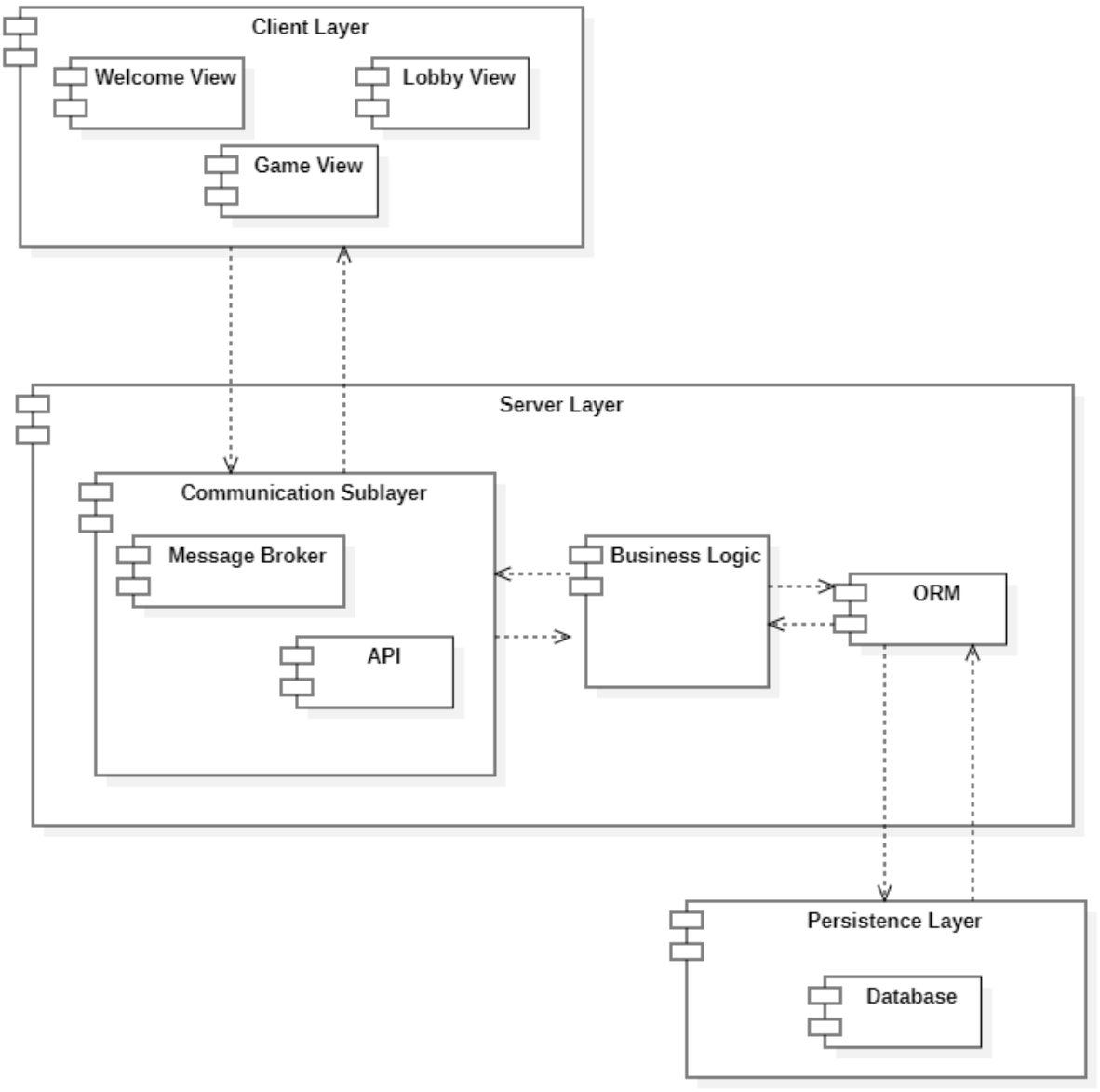


Slika 2 – Dijagram generalne arhitekture sistema.

3.3. Strukturni pogledi

Na sledećem dijagramu može se videti ilustrovana struktura sistema sa navedenim komponentama sistema kao i njihove međusoble relacije, odnosno povezanosti. Komponente najvišeg nivoa predstavljaju ustvari slojepe po *Layered* arhitekturnom obrascu. Klijentski sloj predstavlja prezentacioni sloj. Klijentski i serverski sloj komuniciraju sinhrono putem *RESTful API* servisa, dok je asinhrona komunikacija ostvarna pomoću *Publish-Subscriber* arhitenturnog obrasca.

Serverski sloj sadrži podslojeve i komponente za komunikaciju, poslovnu logiku i za interakciju sa bazom podataka. Komunikacioni podsloj sadrži komponente za sinhronu (*API*) i asinhronu (*Message-Broker*) komunikaciju. Komponenta za interakciju sa bazom podataka predstavlja *ORM (Object-Relational Mapping)* koji se koristi za mapiranje modela pri implementaciji *Repository* arhitekturnog obrasca.

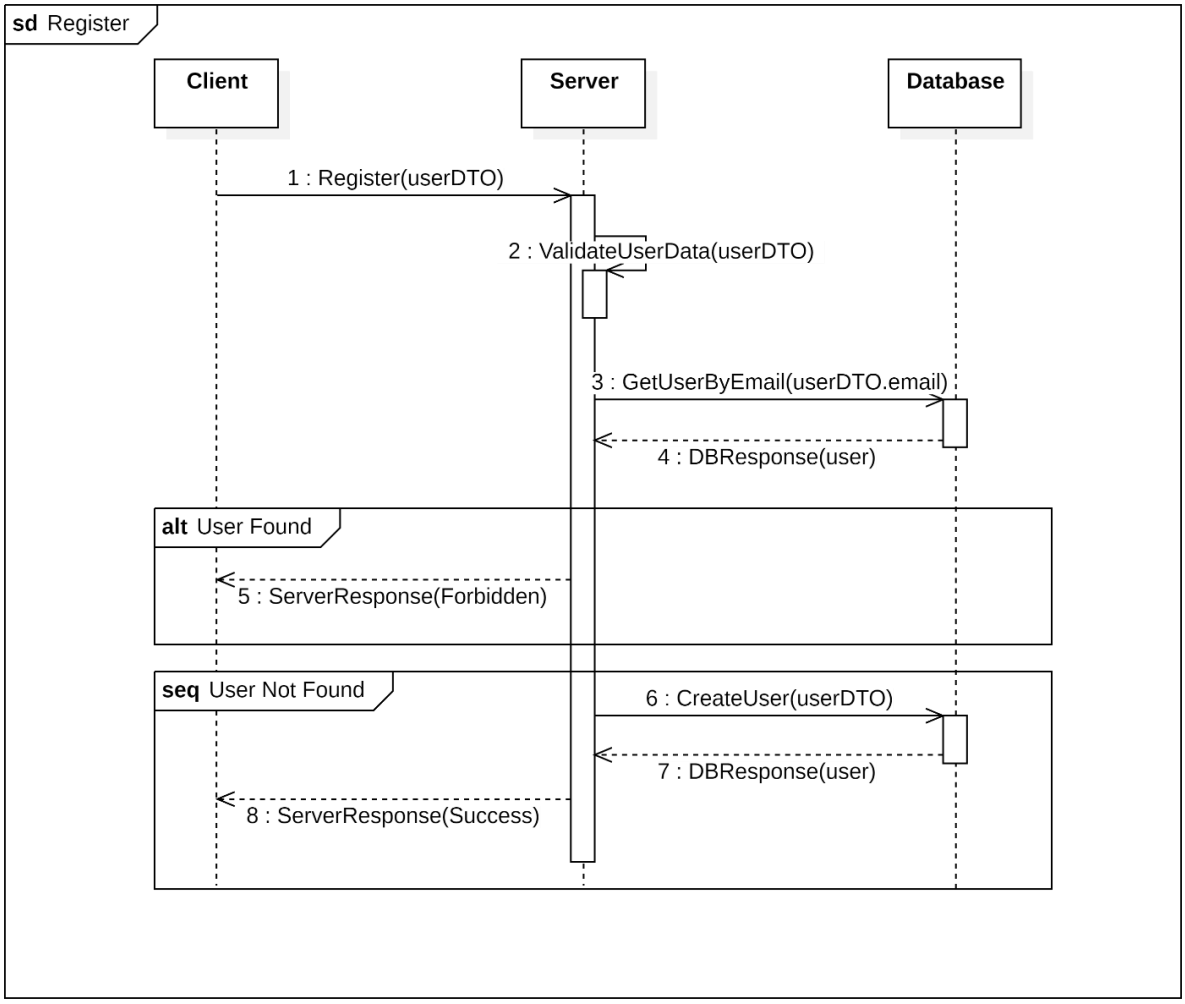


Slika 3 – Dijagram strukturnog pogleda sistema.

3.4. Bihejvioralni pogledi

3.4.1. Registracija korisnika

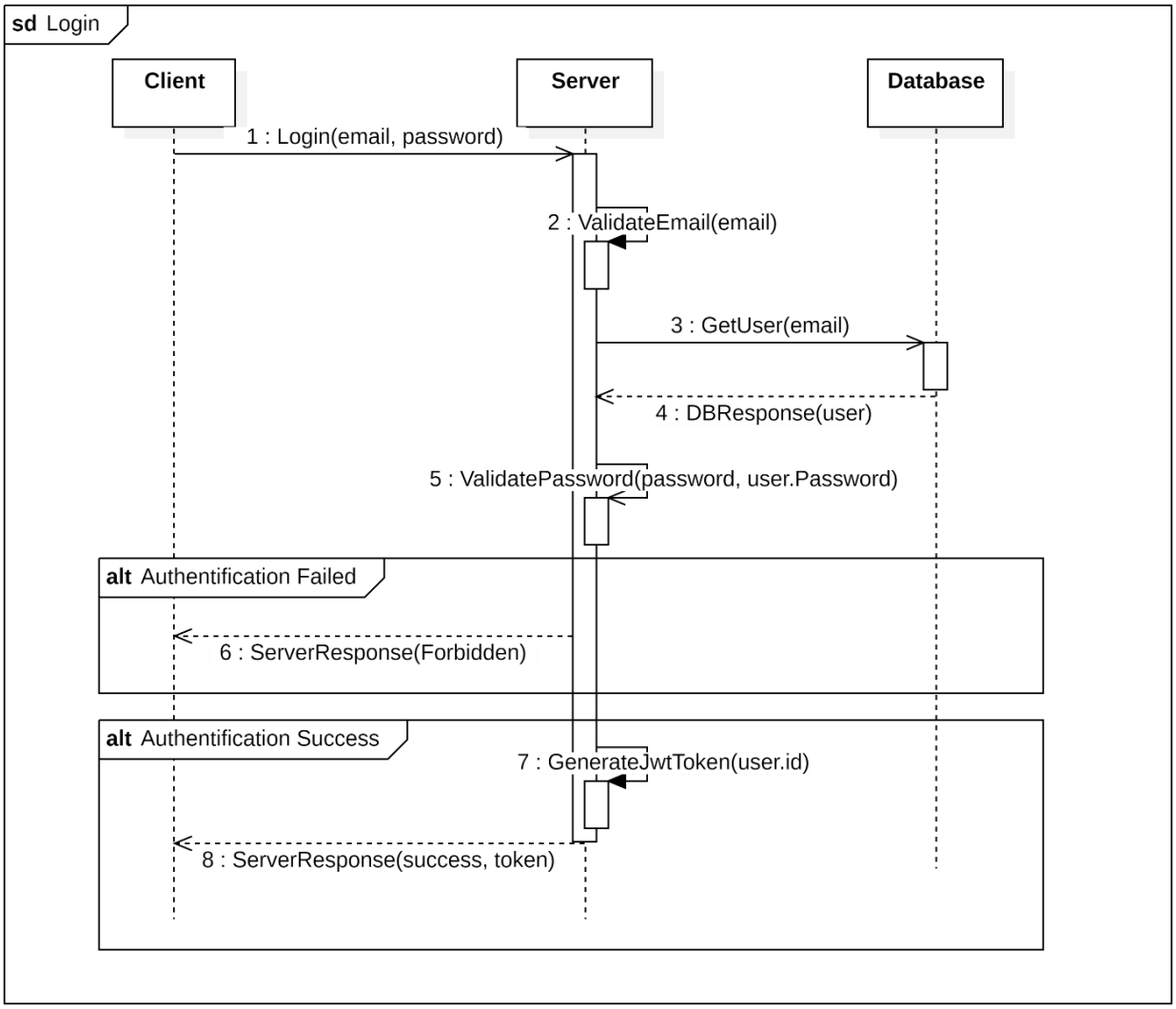
Klijent unosi podatke i šalje ih serveru. Server vrši validaciju podataka, uključujući proveru broja godina, validnosti e-mail adrese i da li je korisnik već registrovan. Ukoliko dođe do greške, server obaveštava klijenta. U suprotnom, novi korisnik se upisuje u bazu podataka i klijent dobija obaveštenje da je nalog uspešno kreiran.



Slika 4 – Dijagram sekvence registracije korisnika.

3.4.2. Prijavljivanje korisnika

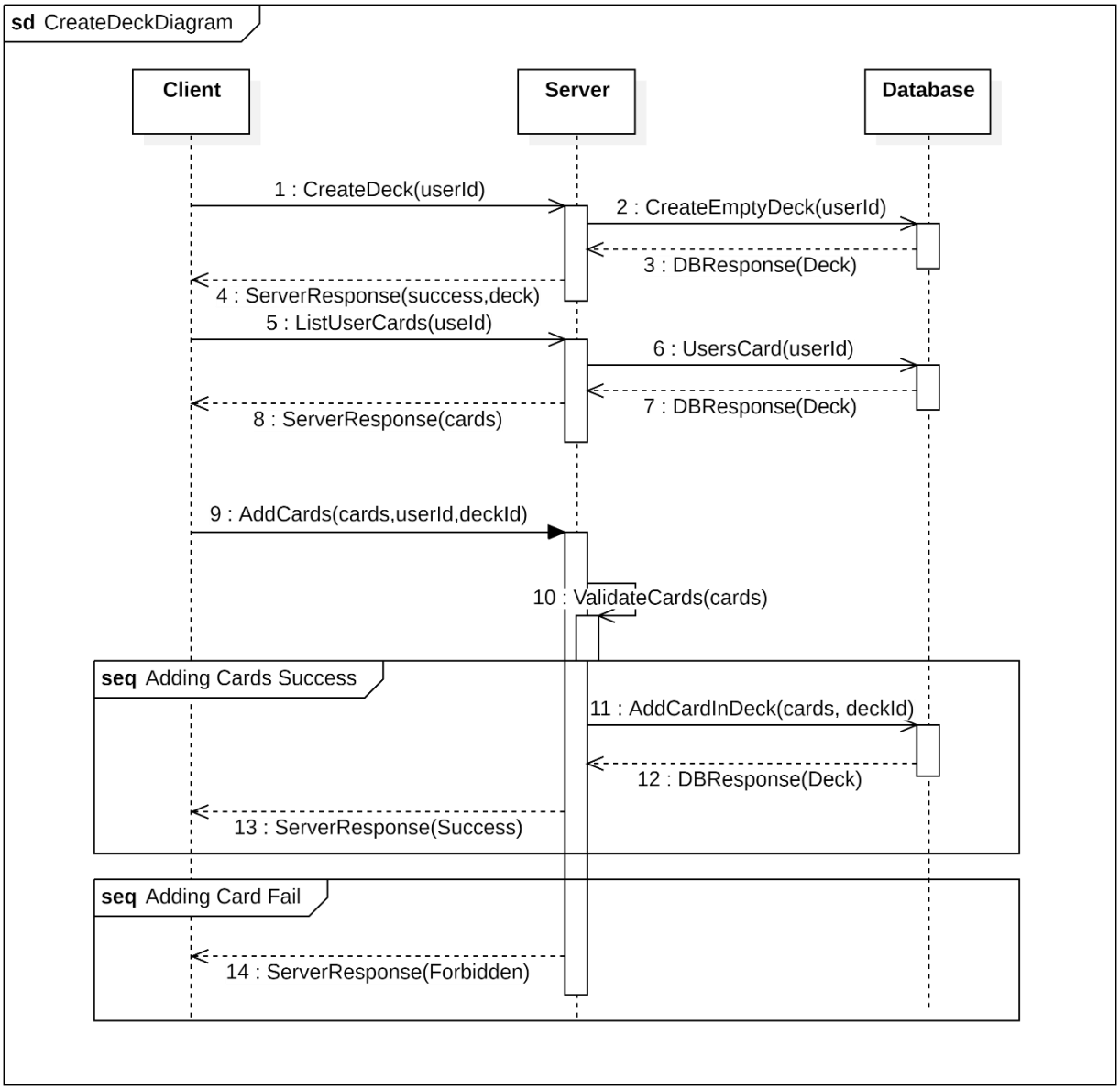
Klijent unosi podatke i šalje zahtev za prijavljivanje serveru. Server proverava da li su kredencijali validni i da li je korisnik već prijavljen. Ukoliko je korisnik već ulogovan, vrši se redirekcija na početnu (home) stranicu. Ako korisnik nije prijavljen, a kredencijali su ispravni, server kreira JWT token i šalje ga klijentu. U slučaju neispravnih podataka, klijent se obaveštava da prijava nije uspešna.



Slika 5 – Dijagram sekvence prijavljivanja korisnika.

3.4.3. Kreiranje špila

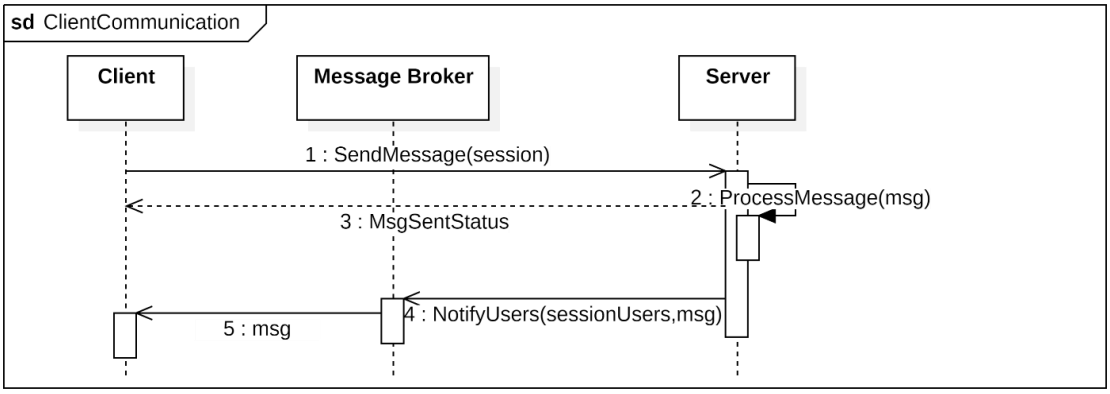
Klijent šalje zahtev serveru za kreiranje špila (deck-a). Server kreira špil i obaveštava klijenta da je uspešno kreiran. Nakon toga, klijent šalje zahtev serveru za listu svih svojih karata. Ukoliko klijent uspešno dobije listu, šalje novi zahtev sa kartama koje je odabrao. Ako uslovi nisu ispunjeni, server odbija zahtev i obaveštava klijenta o grešci.



Slika 6 – Dijagram sekvence kreiranja špila.

3.4.4. Komunikacija klijenata

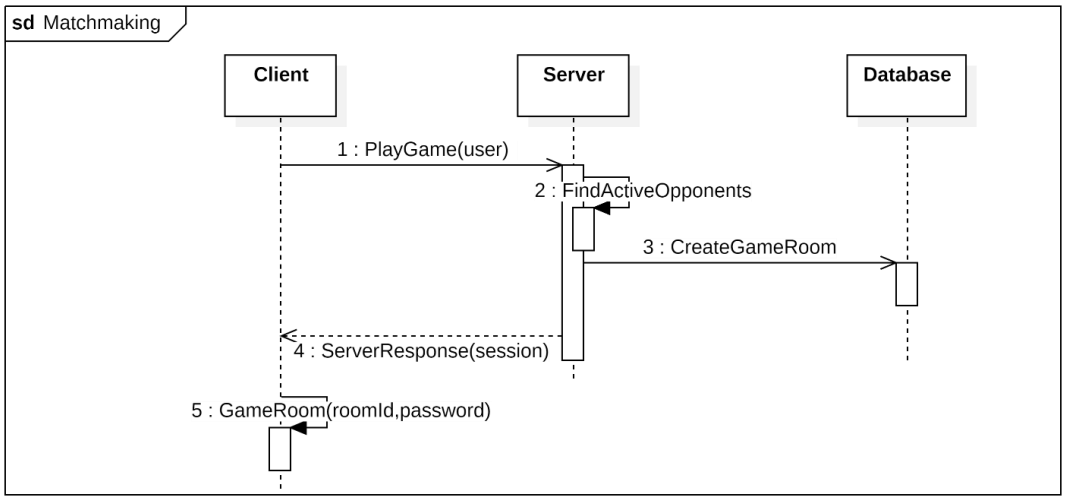
Komunikacija između korisnika odvija se grupno unutar jednog gameRoom-a. Klijent šalje poruku serveru, server je obrađuje i upisuje u bazu podataka, a zatim poziva MessageBroker koji ima zadatak da obavesti sve igrače u toj sobi.



Slika 7 – Dijagram sekvence komunikacije klijenata.

3.4.5. *Matchmaking*

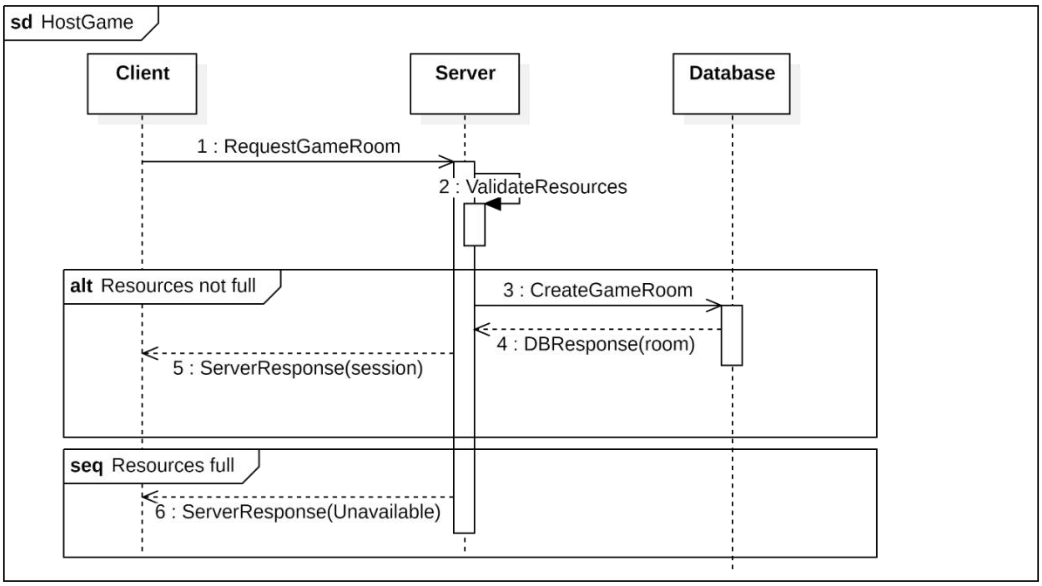
Kada klijent želi da započne igru, šalje zahtev serveru. Server prikuplja sve zahteve za igru i uparuje igrače na osnovu njihovog ELO rejtinga. Nakon toga se kreira gameRoom, koji se upisuje u bazu podataka. Klijentu se šalju ID sobe i lozinka, pri čemu je lozinka u ovom slučaju prazno polje.



Slika 8 – Dijagram sekvence matchmaking-a igrača.

3.4.6. Kreiranje nove partije

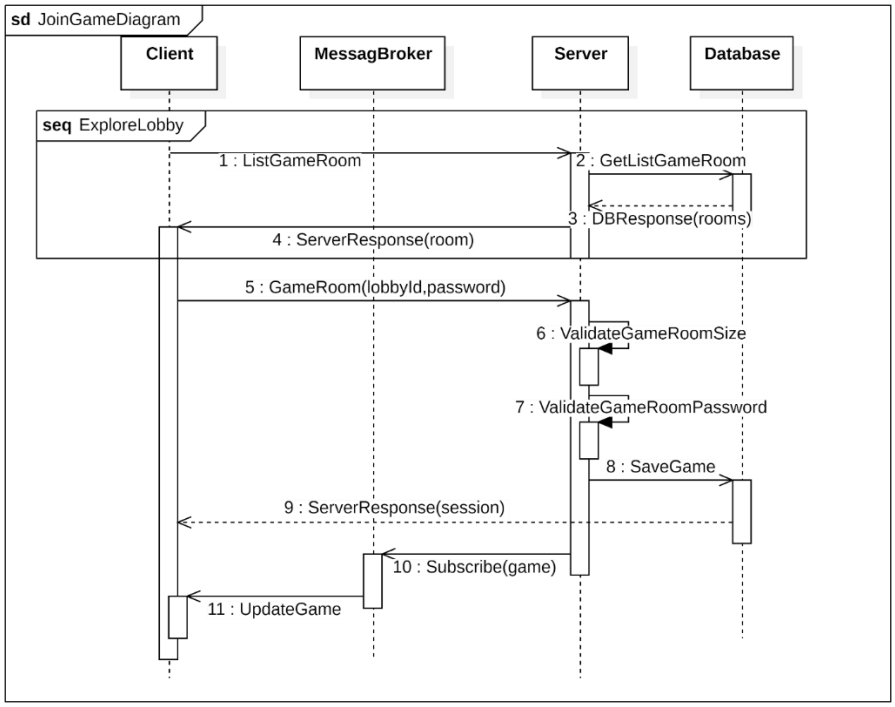
Klijent šalje zahtev serveru za kreiranje novog gameRoom-a. Server proverava da li postoje dovoljni resursi za podršku novoj sobi. Kao administrator sobe postavlja se korisnik koji je poslao zahtev. Ukoliko postoje dovoljni resursi, server kreira sobu, upisuje je u bazu podataka i obaveštava klijenta.



Slika 9 – Dijagram sekvence kreiranja nove partije.

3.4.7. Pridruživanje sobi

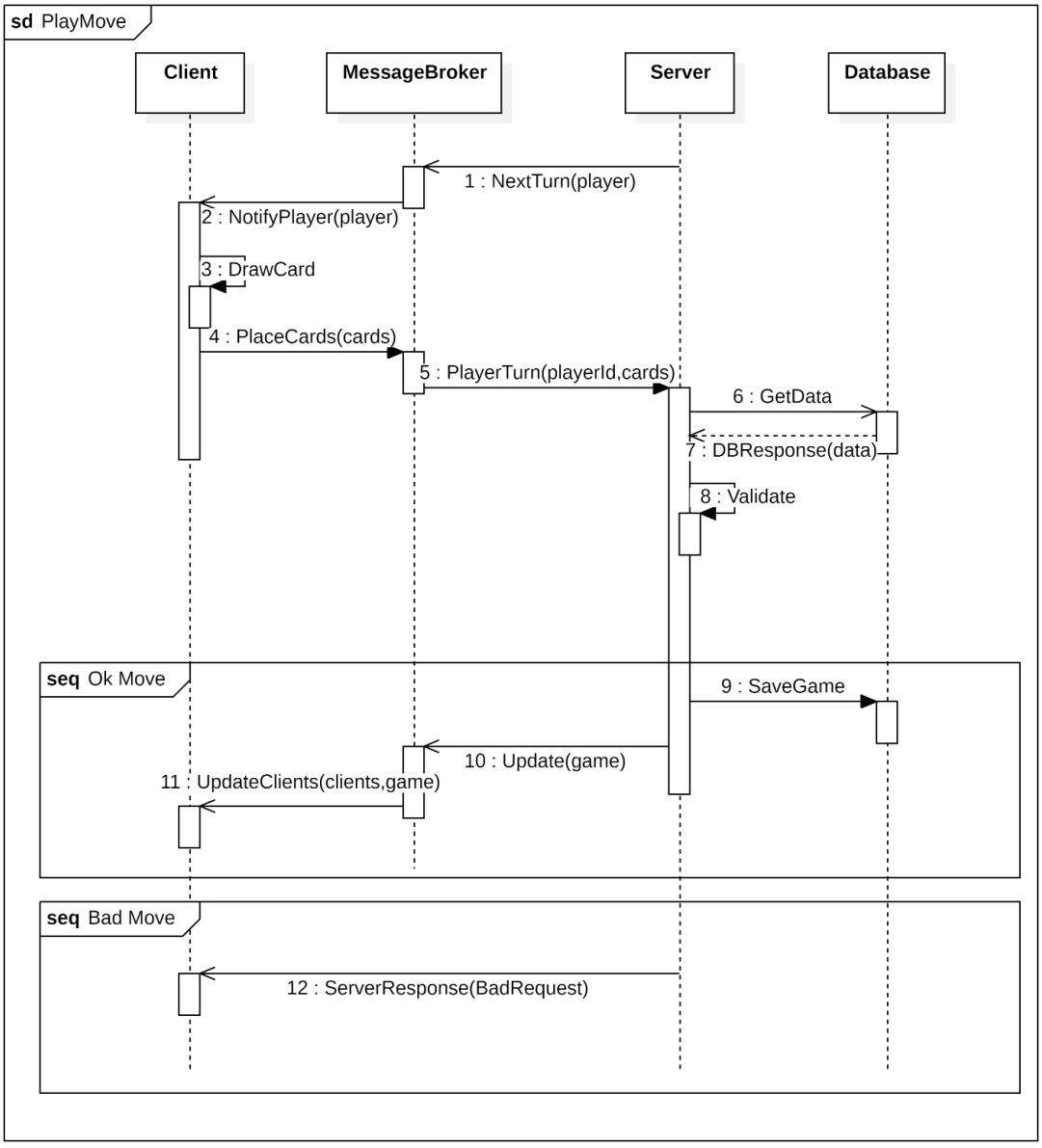
Klijent može da izabere sobu u kojoj želi da igra ili da od servera zatraži listu dostupnih soba. Ako klijent zna ID sobe i lozinku, može se direktno priključiti. U suprotnom, može da dobije listu soba koje odgovaraju njegovim kriterijumima.



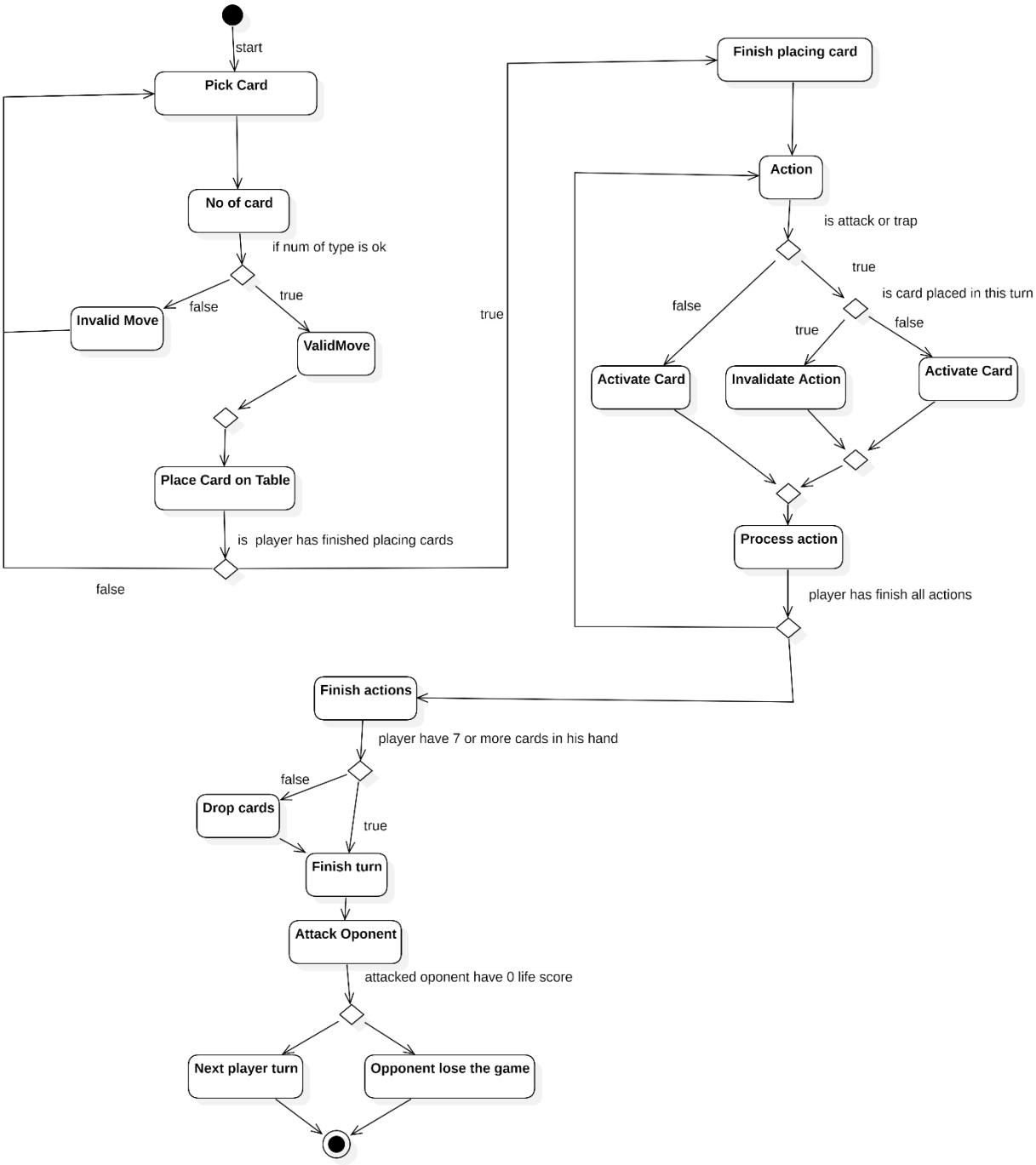
Slika 10 – Dijagram sekvence pridruživanja sobi partije.

3.4.8. Odigravalje poteza

Igrač dobija notifikaciju od MessageBroker-a da je njegov potez. Na početku poteza igrač vuče kartu, a zatim bira karte koje želi da odigra. Dozvoljeno je odigrati maksimalno jednu monster kartu, jednu trap kartu i više spell karata. Na tabli može biti najviše 5 monster karata i 5 spell/trap karata. Igrač ne može odigrati kartu koja je odbačena u istom potezu. Ukoliko igrač na kraju poteza ima 7 ili više karata u ruci, mora odbaciti višak, jer je maksimalno dozvoljeno 6 karata.



Slika 11 – Dijagram sekvence odigravanja poteza.



Slika 12 – Dijagram aktivnosti odigravanja poteza.

3.5. Implementaciona pitanja

Specifikacija biblioteka i programskih okvira koji će biti korišćeni prilikom implementacije sistema:

- **React** sa programskim jezikom **TypeScript** – *JavaScript* programski okvir za pisanje klijentskog dela *Web* aplikacija korišćenjem koncepata deklarativnog programiranja.
- **ASP.NET** sa programskim jezikom **C#** - Programski okvir .NET jezika za pisanje serverskog dela aplikacije, odnosno implementaciju poslovne logike i komunikacije sa bazom.
- **Entity Framework** sa programskim jezikom **C#** - *ORM* okvir koji se koristi zajedno sa .NET okvirom i omogućava rad sa bazom podataka iz aplikacija pisanih .NET okvirom.
- **SignalR** sa programskim jezikom **C#** i **Typescript** – Uloga *message-broker*-a, biblioteka za komunikaciju u realnom vremenu.
- **PostgreSQL** – *SQL* baza podataka, korišćena za skladištenje podataka.

4 ANALIZA ARHITEKTURE

4.1. Potencijalni rizici u implementaciji i strategije prevazilaženja

Jedan od najznačajnih potencijalnih rizika u implementaciji jeste problem kapaciteta servera i optimizacije aplikacije za veliki broj korisnika. Rizik koji je direktno povezan sa ovim je činjenica da se koristi centralizovano skladište podataka, i ukoliko se broj korisnika ekstremno poveća, skladište neće biti u stanju da odgovori na sve zahteve u potrebnom vremenu. Kako se radi o *online* igri, vreme odziva je ključno, pa ovaj rizik može postati fatalan. Strategija za prevazilaženje ovog rizika može biti testiranje performansi i opterećenja servera, uz distribuiranje skladišta i potencijalno samog serverskog izvršavanja aplikacije.