Arhitekturni projekat

Planet Wars

Članovi tima:

Đorđe Jovanović, 17150
Nevena Tufegdžić, 17506

Naziv tima:

Negatim

Datum: 24.12.2021.

SADRŽAJ

1 KONTEKST I CILJ PROJEKTA

Planet Wars predstavlja online multiplayer igru zamišljenu po principima Rizika i popularne igre *Civilization*. U pitanju je kompetitivna igra koja se može igrati sa minimalno dva igrača, gde je cilj pokoriti svaku planetu. Na početku, jedan igrač kreira partiju kojoj se priključuju ostali igrači. Svako dobije svoju startnu planetu i početan broj armija. Broj planeta i igrača podešava igrač koji kreira partiju.

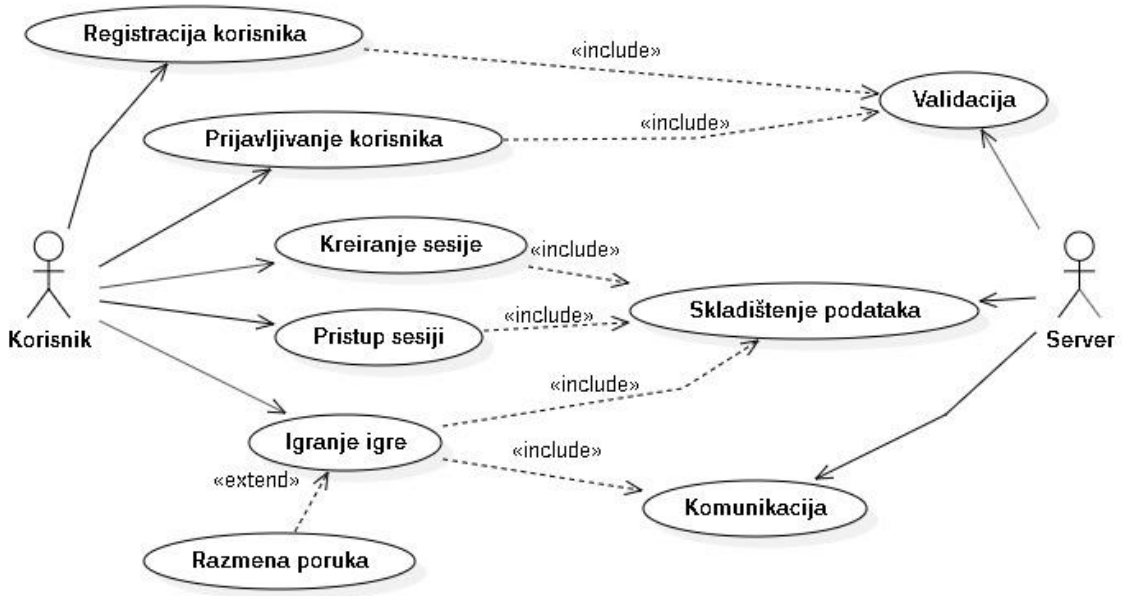
Cilj igre jeste da pametnim rasporedom armija i zauzimanjem planeta sa resursima korisnik nadvlada ostale igrače i pokori planete. U tu svrhu treba implementirati sistem resursa na planetama, koji podržava 3 tipa resursa: za napad, za odbranu i za pokretljivost armija. Resursi sa jedne planete deluju samo na armije raspoređene na toj planeti.

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, ali i slojem perzistencije, a komunikacija sa klijentima se odigrava direktno preko implementiranog RESTful API-ja, ili posredno, korišćenjem *third-party message broker*-a.

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

3.1.2 *Model-View-Controller* obrazac

Kroz sistem se provlači MVC obrazac. Modeli su domenske klase relevantne za aplikaciju, a podaci unutar baze odgovaraju definisanim modelima. *View* jeste ono što klijent vidi – deo aplikacije koji se izvršava na klijentu, u obliku *fat client*-a. Klijent preko *controller*-a, koji se izvršavaju na serveru, deluje na modele. *Controller*-i imaju ulogu izmene modela u skladu sa korisničkim komandama, a sa druge strane treba da na osnovu modela ažuriraju *view* na klijentu.

3.1.3 *Publish-Subscribe* obrazac

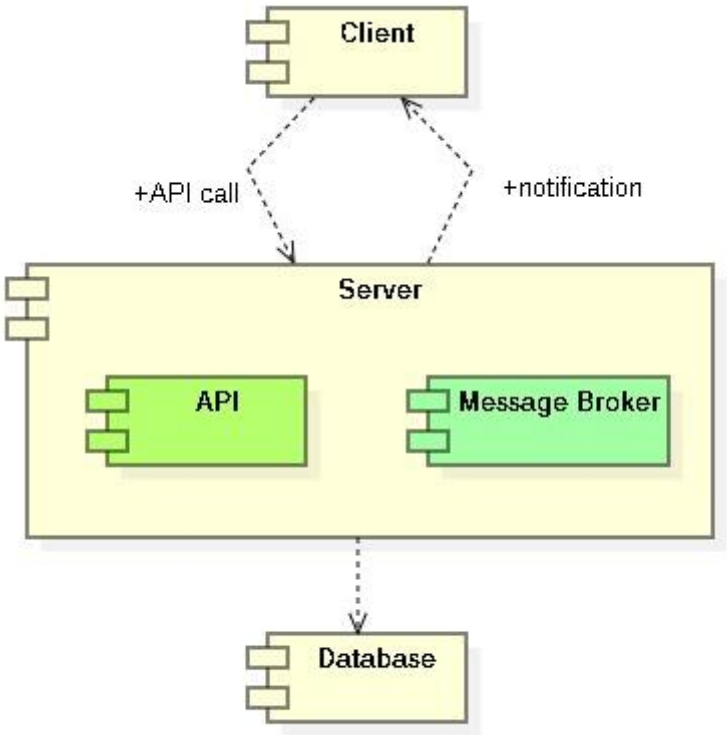
Aplikacija zahteva rad u realnom vremenu i komunikaciju između korisnika. Ovo će se postići implementacijom *Publish-Subscribe* obrasca koji će koristiti *message broker* komponentu za ostvarivanje pravovremenog ažuriranja na klijentima. Klijenti će se pretplatiti na temu koja je za njih relevantna i 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

Za implementaciju perzistencije koristiće se centralizovano skladište pasivnog tipa. 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. Svi preduslovi za implementaciju ovog obrasca obezbeđeni su kroz *Dotnet Entity Framework*, koji će biti iskorišćen za realizaciju baze podataka i pristupanju samoj bazi.

3.2 GENERALNA ARHITEKTURA

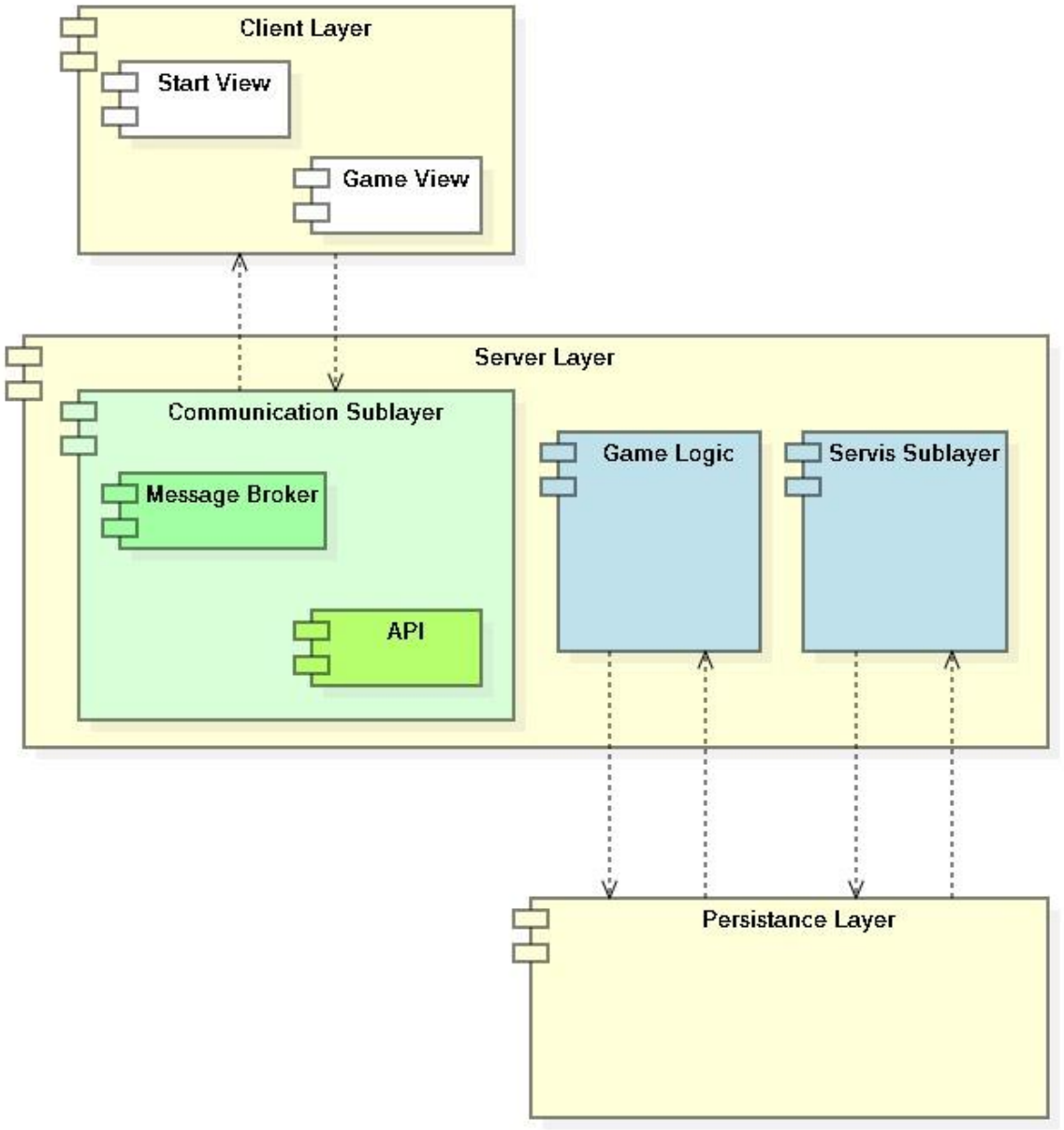
Arhitektura sistema podrazumeva postojanje klijenta, servera i baze podataka u kojoj će se čuvati informacije o korisnicima i njihovim igrama.



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.

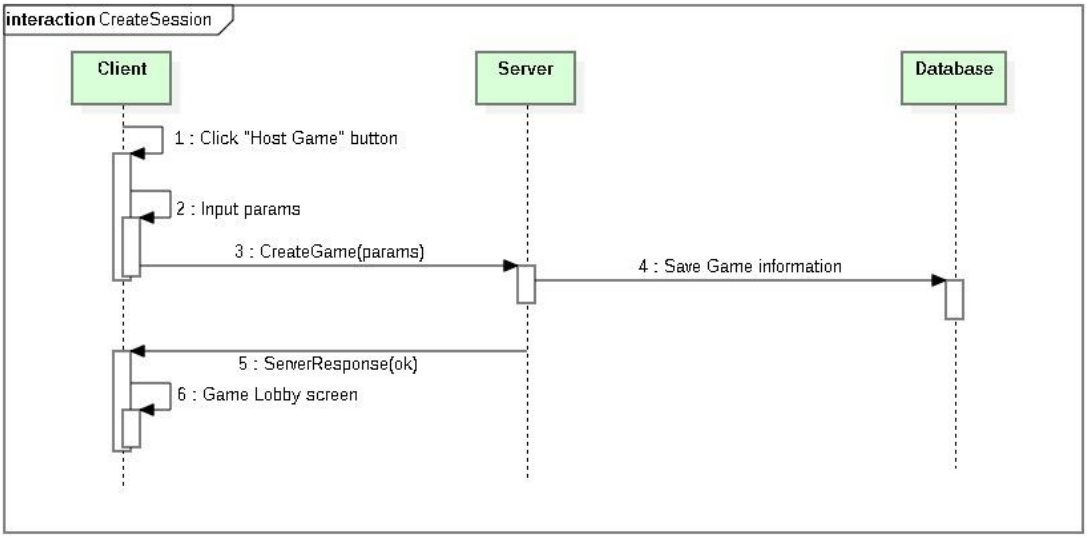
Klijentska aplikacija je implementirana kao Web aplikacija i predviđena je za rad na svim računarima sa pristupom Internetu. Serverska aplikacija se realizuje kao .NET *Core Web* API, dok se kao sistem za upravljanje bazom podataka koristi MS SQL. Za asinhronu komunikaciju između klijenata se koristi *SignalR message broker*.



3.4 BIHEJVIORALNI POGLEDI

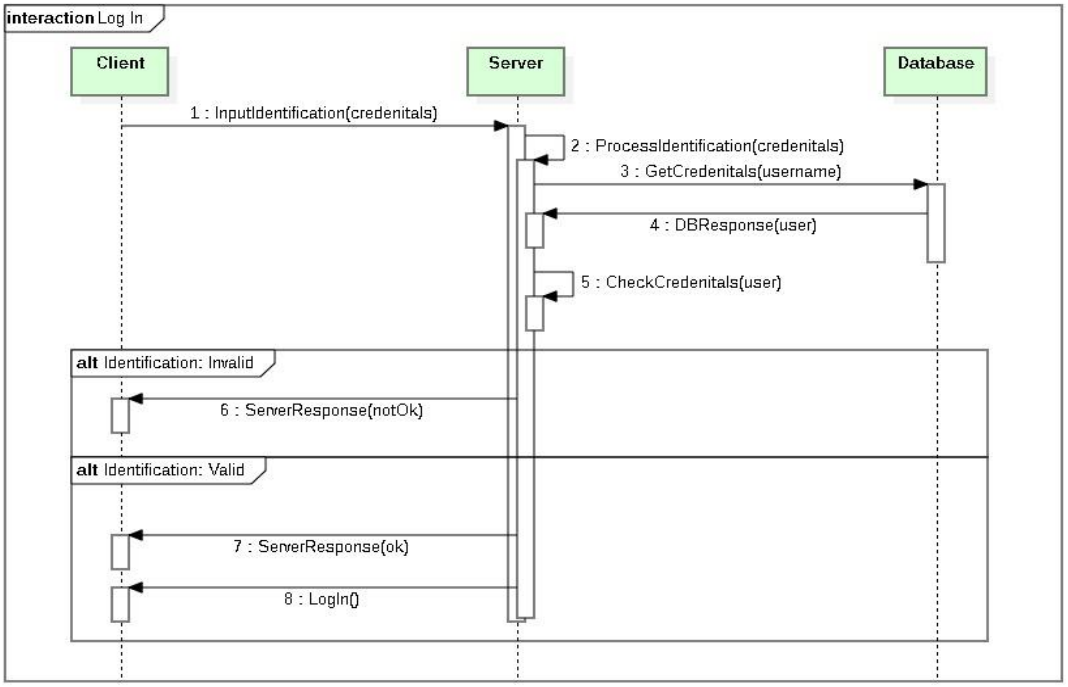
3.4.1 Registracija korisnika

Nakon što unese odgovarajuće podatke u prazna polja, 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.



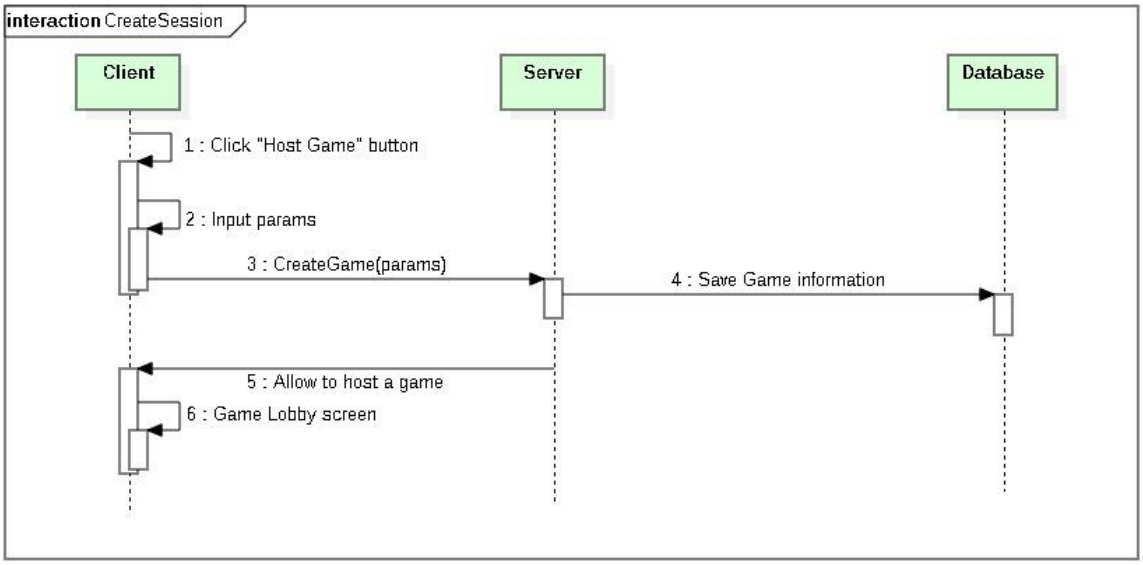
3.4.2 Prijava korisnika

Klijent unosi podatke i šalje zahtev za prijavljivanje 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.



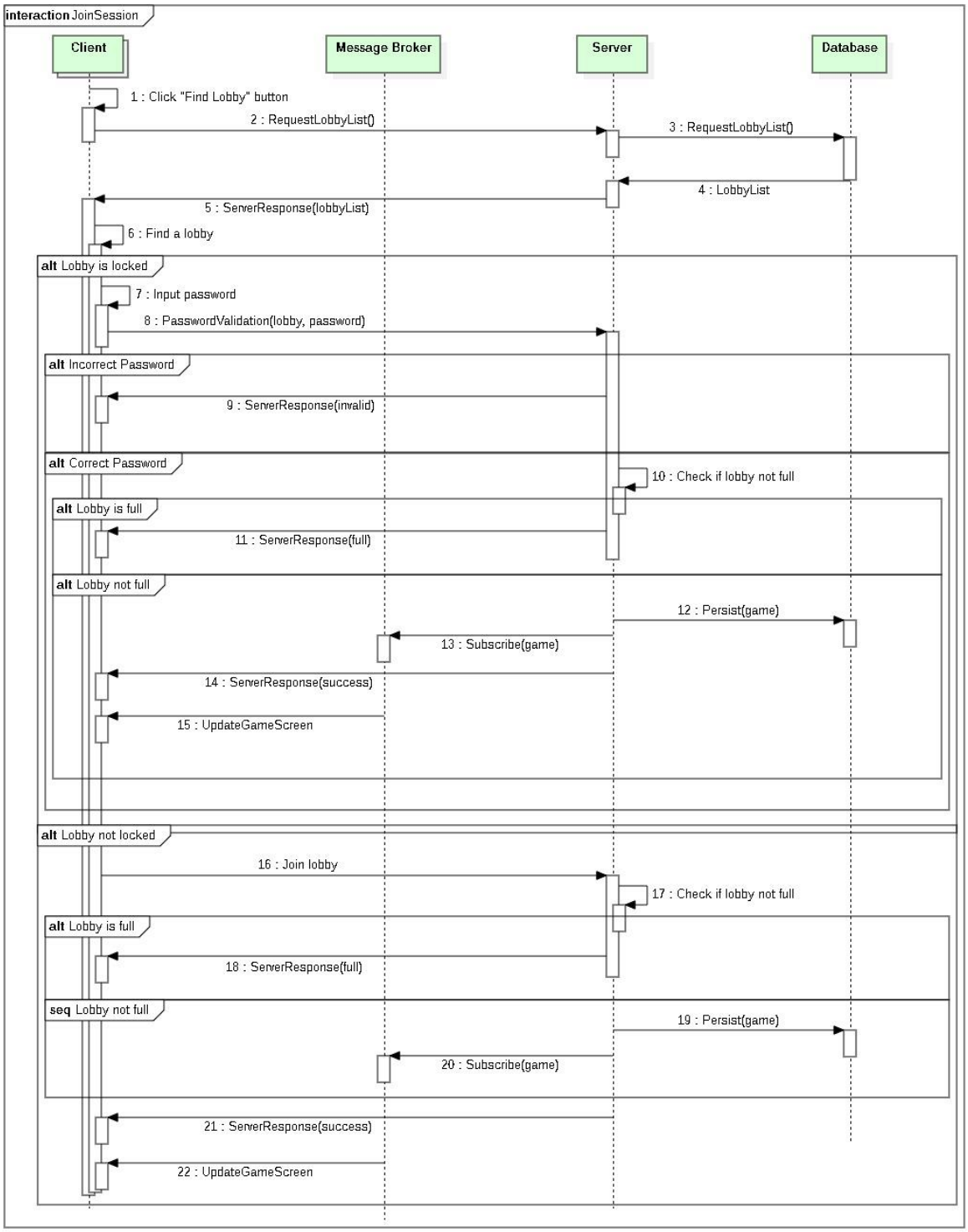
3.4.3 Kreiranje sesije

Klikom na dugme „*Host Game*“ klijent se opredeljuje za kreiranje nove sesije i prikazuje se strana 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 prihvatanju zahteva. Klijent zatim pristupa *lobby* stranici gde čeka da se ostali igrači priključe.



3.4.4 Pristup sesiji

Klikom na dugme „*Find Lobby*“ klijent se opredeljuje za traženje već kreirane igre kojoj se može pridružiti. Zahtev se šalje serveru, i server od baze podataka traži listu svih *lobby*-ja kojima se igrači mogu priključiti. Server vraća listu*lobby*-ja koji se zatim prikazuju klijentu nakon čega klijent bira željeni *lobby*. Ukoliko je *lobby* zaključan, potrebno je uneti lozinku. Ako je lozinka netačna, klijent se vraća nazad na unos. U slučaju da je lozinka tačna, ili da *lobby* nije zaključan, server proverava da li je pun, i ako jeste, klijent se obaveštava o tome i vraća se nazad na osvežen izbor. U slučaju da *lobby* nije pun, baza se osvežava, klijent se obaveštava o uspehu, a server šalje informaciju *message broker*-u koji obaveštava sve klijente iz *lobby*-ja o novom igraču, i svim igračima se osvežava strana.



3.4.5 Odigravanje poteza

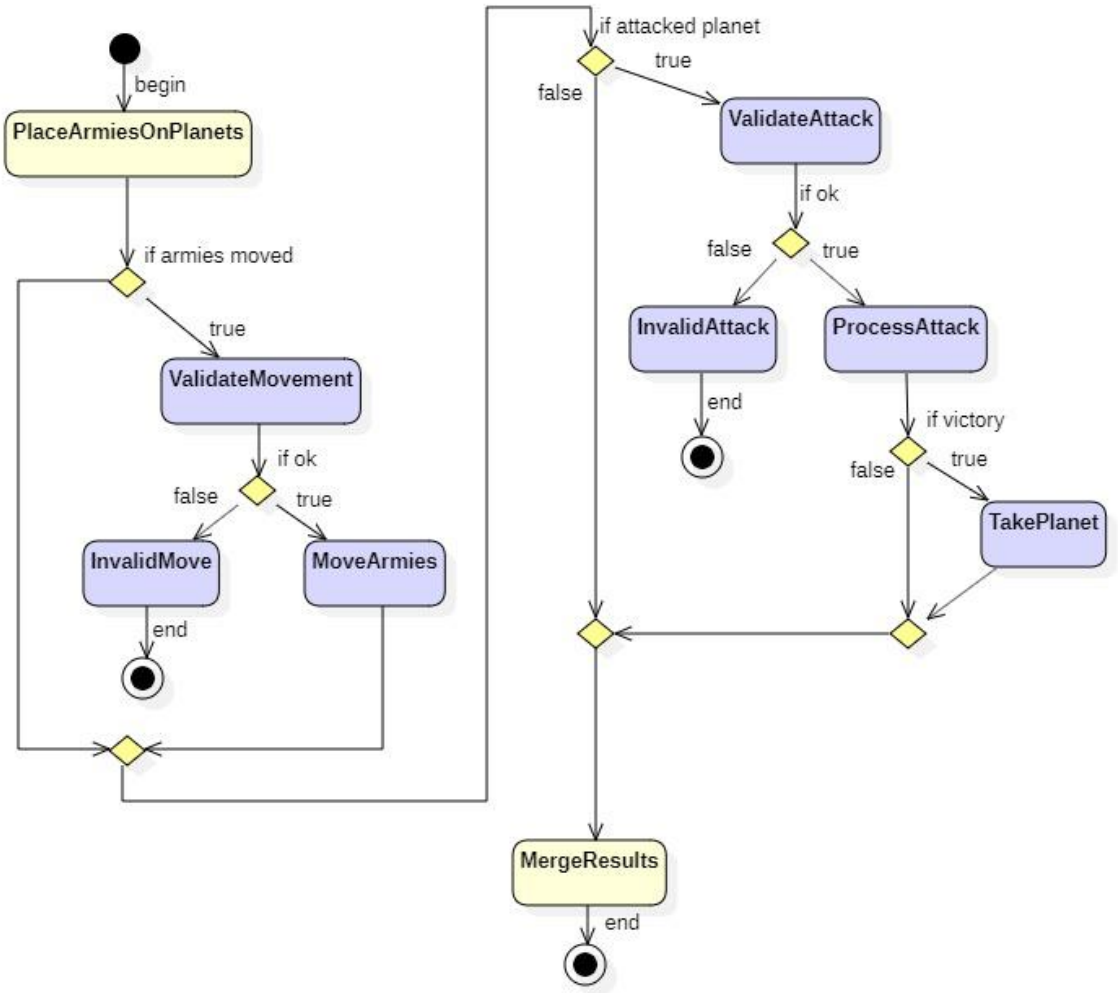
Igrač na početku poteza dobija određeni broj armija. Na samom početku igre, svaki igrač dobija po 3 armije, a u svakom sledećem potezu broj armija zavisi od broja planeta koje taj igrač poseduje. Igrač u okviru poteza ima obavezu da dobijene armije rasporedi, i da odigra maksimalno dve akcije. Akcije mogu biti pomeranje određenog broja armija sa jedne planete na susednu, i napad neke planete koju poseduje neki drugi igrač. Igrač ima pravo da izabere željenu kombinaciju akcija. Neke planete mogu imati određene resurse, koji utiču na snagu armija koje

napadaju s te planete, snagu armija koje brane planetu, i razdaljinu koju armije s te planete mogu da pređu, a koja je podrazumevano do susedne planete.

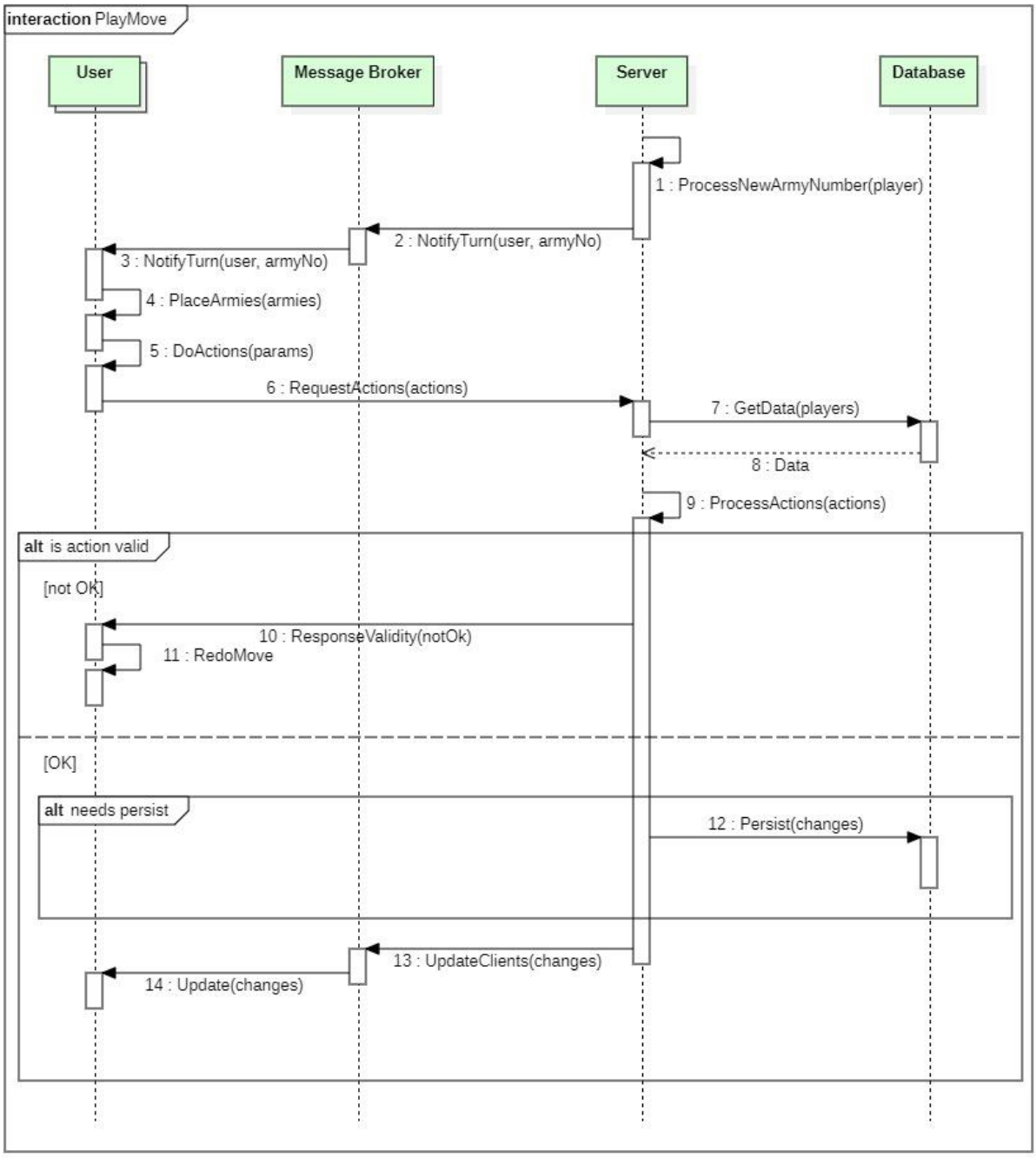
Napad se odvija tako što igrač izabere planetu sa koje napada, izabere broj armija kojim napada, i koju planetu će napasti. Drugi igrač se brani pasivno, i ako je njegova odbrana veća od moći napada prvog igrača, pobeđuje. Pri napadanju se gubi onaj broj armija koliko je potrebno (na primer, ako prvi igrač napada sa 3 armije, a drugi se brani sa 2, prvi će pobetiti ali će mu ostati samo jedna armija da zauzme planetu).

Zauzimanje planeta se dešava u dva slučaja: prvi ako planeta nema vlasnika, što je situacija na početku igre, gde je dovoljno samo pomeriti armiju na datu planetu, i drugi, kada se planeta zauzima napadom i u tom slučaju će biti preoteta samo ako napadaču pretekne neka armija iz napada. U suprotnom, planeta ostaje u vlasništvu drugog igrača. Ako se napad vrši na planetu koja je zauzeta ali nema armije, napadač ne gubi nijednu armiju i samo zauzima planetu.

Dijagram aktivnosti koji opisuje ponašanje servera pri obradi jednog poteza na najvišem nivou dat je na sledećoj slici:



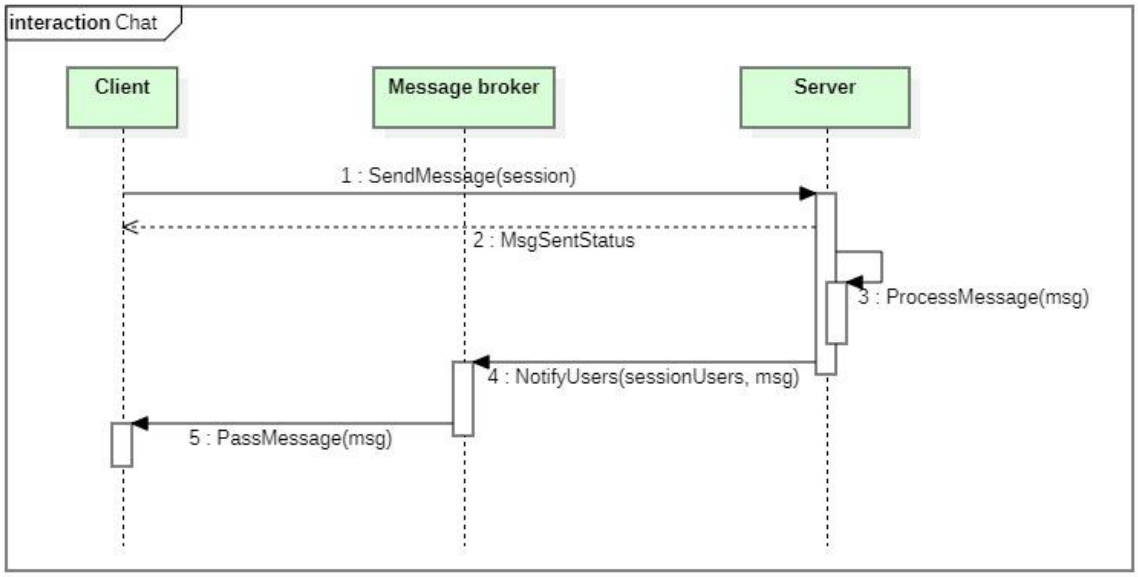
Dijagram interakcije koji opisuje odigravanje poteza dat je na sledećoj slici:



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 *message broker* 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:

- **Angular** – JavaScript frontend framework za pisanje Web aplikacija,
- **SignalR** – Message broker, omogućava API za real-time klijent-server komunikaciju,
- **.NET Core Web API** – Serverska aplikacija,
- **Microsoft SQL DBMS** – Baza podataka

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.