*Arhitekturni projekat*

**RizzyCo**

*Članovi tima: Naziv tima:* La BUGette

Jovan Dimitrijević 16540

Ivana Milivojević 16704

Damjan Ćupić 16959

*Sadržaj*

1. **Kontekst** **i cilj projekta3**
2. Arhitekturni zahtevi3

2.1 Arhitekturno značajni slučajevi korišćenja3

2.2 Nefunkcionalni zahtevi5

2.3 Tehnička i poslovna ograničenja5

1. Arhitekturni dizajn5

3.1 Arhitekturni obrasci5

3.2 Generalna arhitektura8

3.3 Strukturni pogledi8

3.4 Bihevioralni pogledi10

3.5 Implementaciona pitanja12

1. Analiza arhitekture12

4.1 Potencijalni rizici u implementaciji i strategije prevazilaženja12

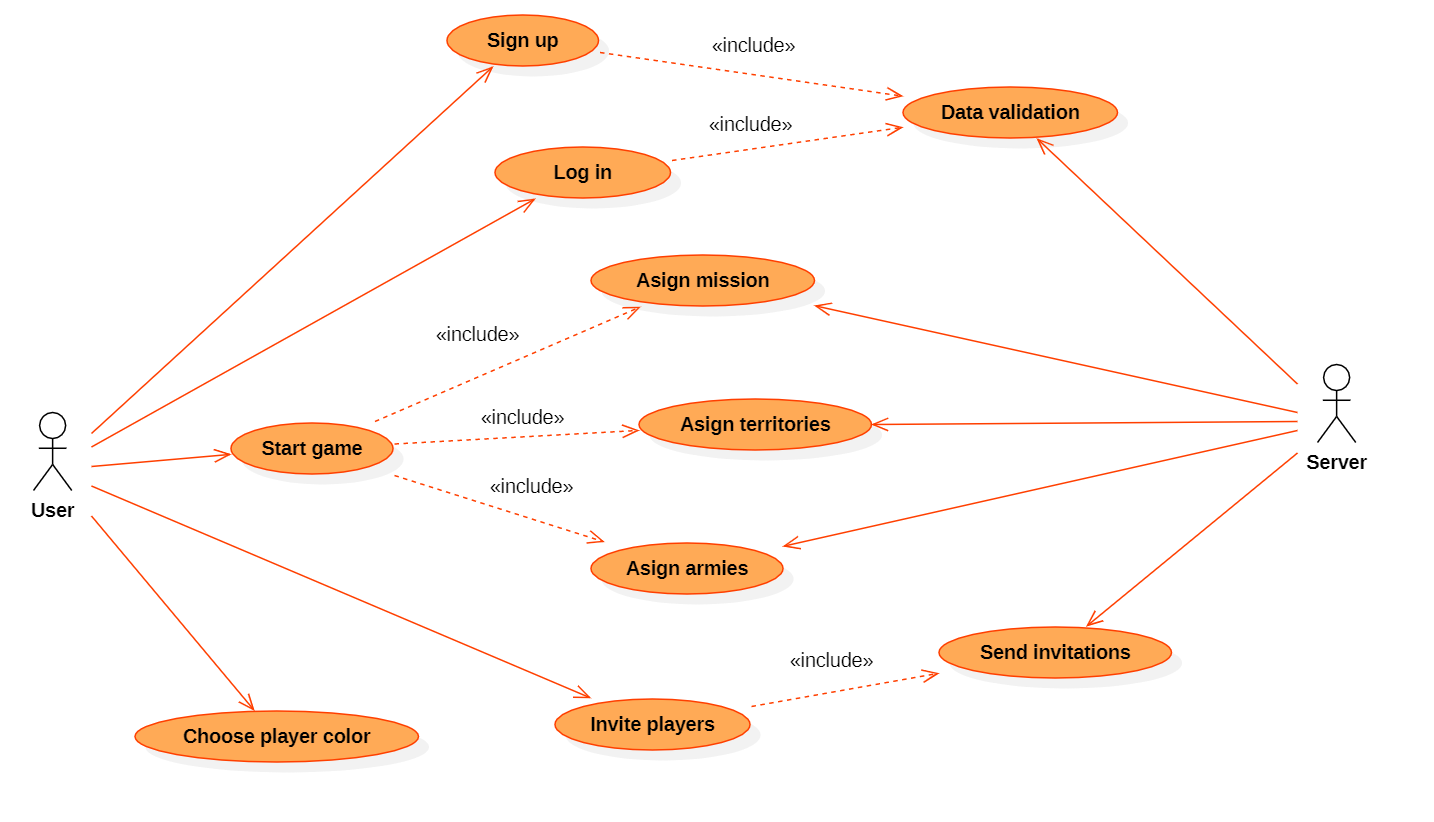
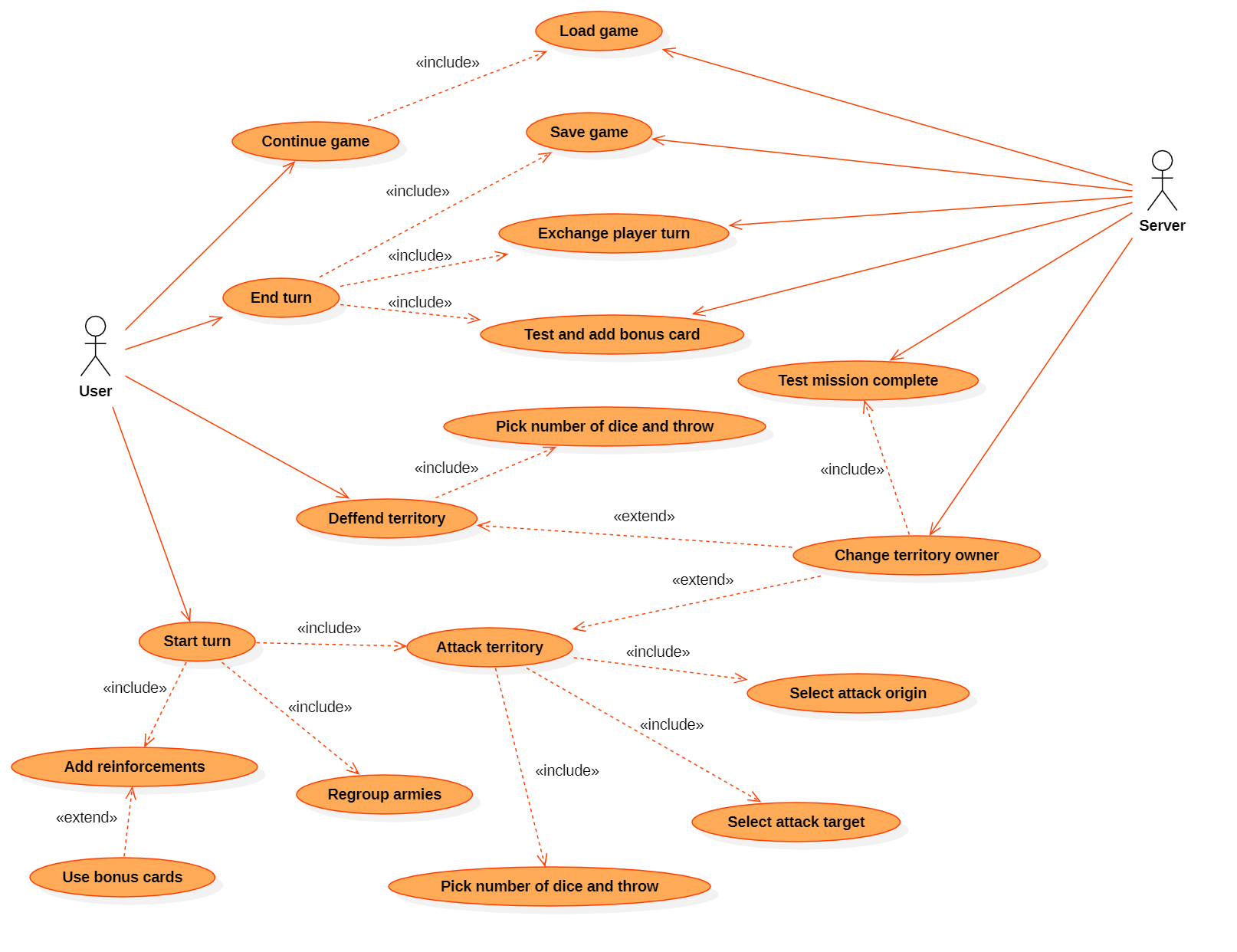
1. **Kontekst i cilj projekta**

*RizzyCo* je Web aplikacija namenjena za online multiplayer igranje popularne društvene igre *Riziko.* Aplikacija će obezbediti igranje u realnom vremenu. Igrači imaju mogućnost da započnu novu igru i pozovu svoje prijatelje da se toj igri pridruže. Nakon pridruživanja igrača tok igre se odvija u skladu sa postojećim pravilima. Na početku igre igrači biraju boje svojih armija i dobijaju misiju koju treba da ispune. Nakon toga se vrši podela svih teritorija u zavisnosti od broja igrača. Pre početka svakog poteza igrač ima mogućnost da pojača svoju vojnu moć dodavanjem određenog broja armija u zavisnosti od broja teritorija koje su u njegovom posedu. Dodatne armije dobija ukoliko poseduje sve teritorije na nekom kontinentu ili na osnovu bonus kartica. U toku svog poteza igrač može da izvrši više napada na teritorije susedne teritorijama koje poseduje. Ishod napada određuje se na osnovu brojeva na kockicama koje su bacili igrač napadač i igrač koji se brani. Ukoliko je igrač na kraju svog poteza osvojio bar jednu teritoriju, dobija jednu bonus karticu. Kraj igre je kada neki od igrača ispuni svoju misiju. Stanje igre može biti sačuvano, tako da igra može da se nastavi kada svi igrači ponovo budu spremni.

1. **Arhitekturni zahtevi**

U ovom odeljku su prikazani arhitekturni zahtevi *RizzyCo* sistema, koji obuhvataju arhitekturno značajne slučajeve korišćenja, nefunkcionalne zahteve i tehnička i poslovna ograničenja.

* 1. Arhitekturno značajni slučajevi korišćenja
* Registracija i prijava korisnika (korisniku je neophodan nalog kako bi učestvovao u igri)
* Kreiranje igre
* Dodavanje drugih korisnika kao igrača u svoju igru
* Dodeljivanje misije svakom igraču
* Inicijalna raspodela teritorija igračima
* Omogućavanje napadanja i osvajanja teritorija
* Bacanje kockica i određivanje ishoda
* Funkcionalnost završetka poteza i izvlačenja bonus kartice
* Ispitivanje uslova kraja igre (ispunjenja misije)
* Sinhronizacija stanja igre između svih igrača u igri
* Skladištenje podataka (čuvanje stanja nezavršene igre i učitavanje igre)



Use case dijagram sistema u toku igre

Use case dijagram sistema na početku igre

* 1. Nefunkcionalni zahtevi

• Pouzdanost – sistem treba da omogući perzistenciju na početku/kraju poteza igrača

• Performanse – aplikacija treba da obezbedi što manje vreme odziva i najbolje performanse u zavisnosti od trenutnog broja korisnika

• Dostupnost – potrebno je da je aplikacija bude dostupna 24/7

• Modifikabilnost – potrebno je omogućiti relativno laku promenu sistema

• Skalabilnost – potrebno je da aplikacija može da podrži povećanje broja korisnika

• Upotrebljivost – potrebno je da aplikacija bude intuitivna i jednostavna za korišćenje

* 1. Tehnička i poslovna ograničenja

Tehnička ograničenja:

• Pristup preko web-a – Neophodno je koristiti web tehnologije koje omogućavaju potrebnu komunikaciju i interakciju između sistema i korisnika

• Komunikacija – Sistem treba da podrži dva različita tipa komunikacije (sinhronu komunikaciju između klijentskog i serverskog dela sistema i asinhronu komunikaciju prilikom propagacije izmena od jednog klijenta ka ostalima)

• Skrivenost baze podataka – Korisnicima su dostupni samo podaci predviđeni za prikaz, a od njih je sakriven način reprezentacije tih 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. Takođe, jedno od ograničenja je i budžet samog projekta koji ne dozvoljava iznajmljivanje jačeg hardvera kojim bi se omogućio istovremeni pristup sistemu velikom broju korisnika.

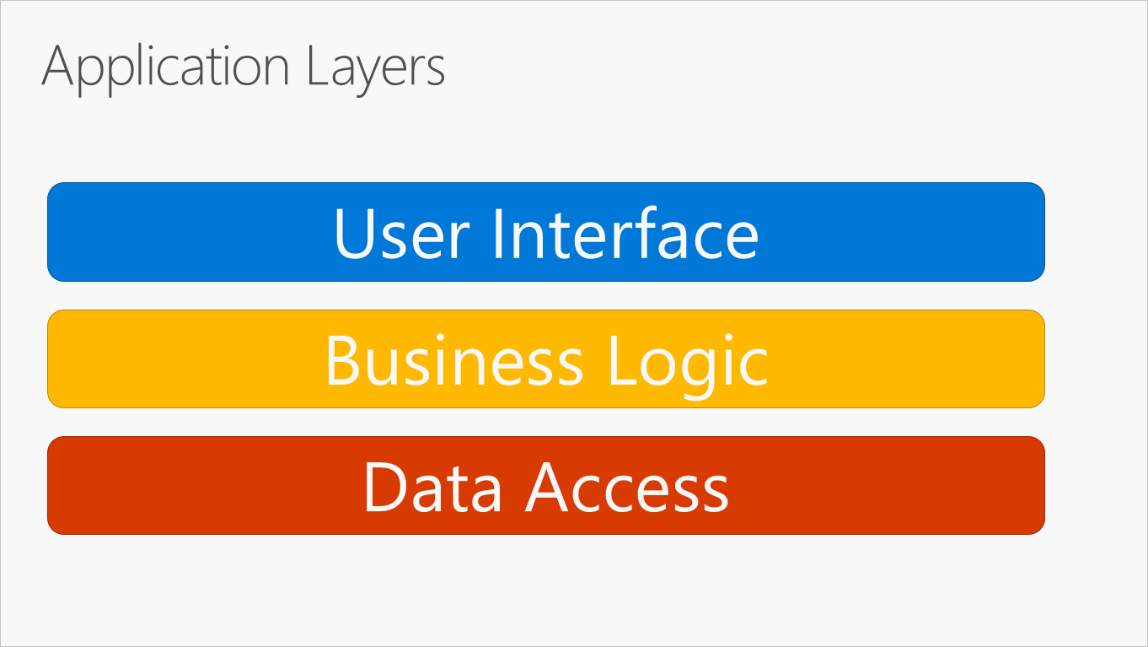
1. **Arhitekturni dizajn**

U ovom poglavlju arhitekturni dizajn prikazan je kroz arhitekturne obrasce, generalnu strukturu arhitekture, strukturne poglede, bihevioralne poglede i implementaciona pitanja.

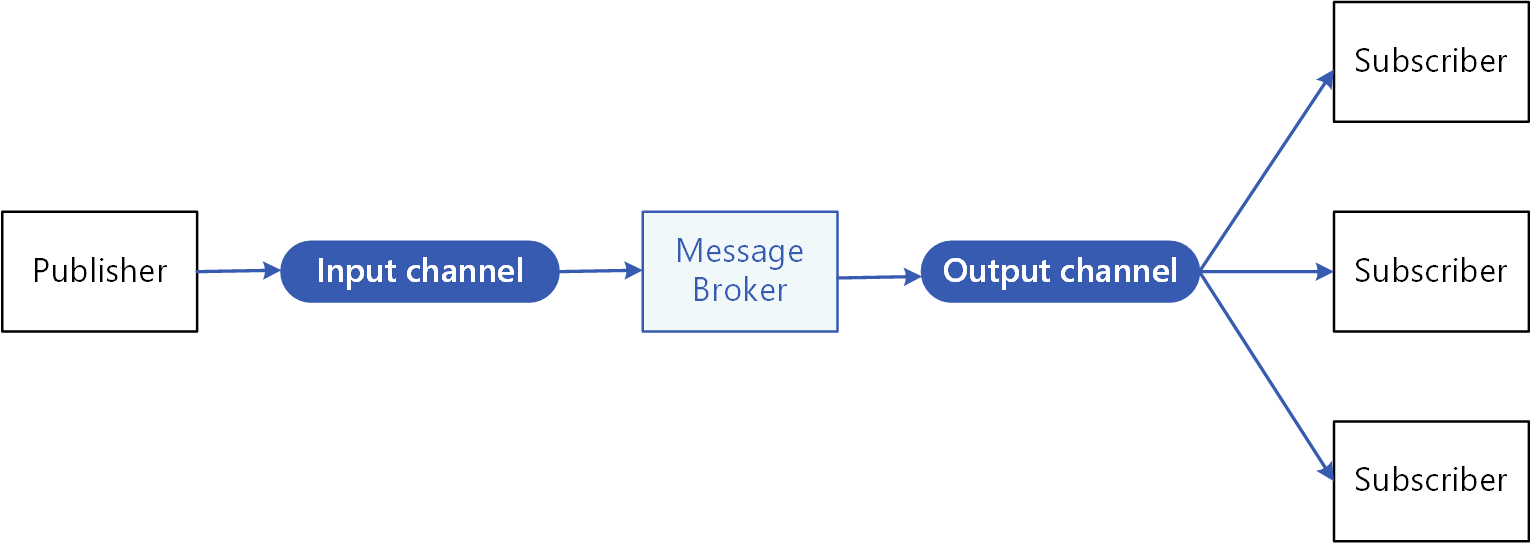
* 1. Arhitekturni obrasci

*Layered*

*RizzyCo* sistem će implementirati troslojni *Layered* (Slojeviti) arhitekturni obrazac. Prvi sloj predstavlja klijentska aplikacija, drugi sloj se sastoji od message broker-a i serverske aplikacije koji treba da obezbede aplikacionu logiku, dok treći sloj treba da obezbedi logiku za komunikaciju sa bazom podataka – implementira *Entity Framework* kao alat za perzistenciju podataka iz relacione baze podataka. Između ovih slojeva postoje još dva međusloja: broker između klijenta i servera i ORM alat između servera i baze podataka. Serverska aplikacija ostvaruje sinhronu komunikaciju sa klijentskom aplikacijom korišćenjem REST-ful API-ja, dok message broker služi za asinhronu komunikaciju sa klijentom. Serverska aplikacija ostvaruje komunikaciju sa bazom koristeći ORM alat za mapiranje domenskih entiteta.

*Publish-subscribe*

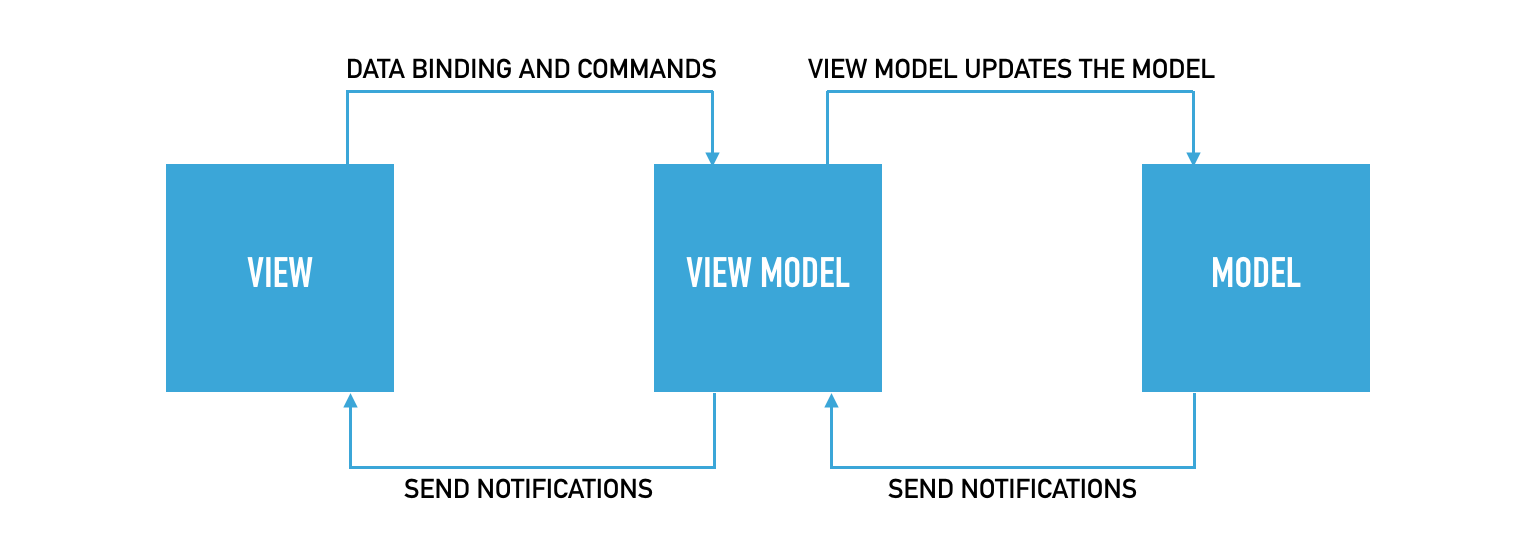
Troslojna slojevita (Layered) arhitektura

*RizzyCo* sistem će implementirati *Publish-subscribe* arhitekturni obrazac (ovaj obrazac će biti implementiran kroz *Message Broker* koji predstavlja sponu između klijenta i servera). Svaka igra ima određeni broj igrača. Kada dođe do bilo koje promene na mapi, svi igrači vide da je došlo do promene. Igrači su automatski subcribe-ovani na promene u igri u kojoj učestvuju, pa će svaku promenu videti u realnom vremenu.

*Dijagram publish-subscribe*

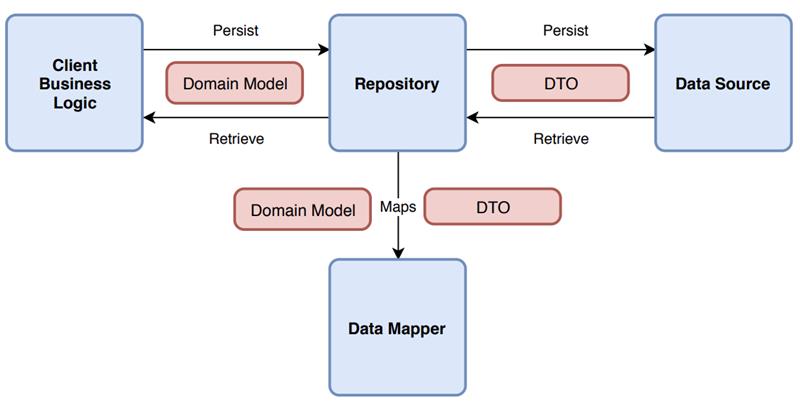
*MVVM (Model – View – View Model)*

*RizzyCo* sistem će implementirati *MVVM* obrazac na strani klijentske aplikacije. *MVVM* obrazac je modifikacija standardnog *MVC* obrasca. *MVVM* razdvaja aplikaciju na više komponenti tako da svaka komponenta ima svoje specifične odgovornosti.

**

Model – View – View Model dijagram

*Repository*

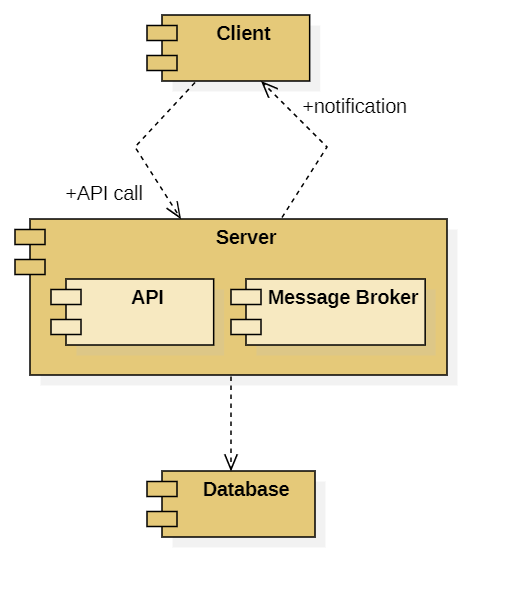
*RizzyCo* sistem će sadržati centralizovanu bazu podataka. Na sloju podataka koristićemo *Entity Framework Core* kao ORM alat za MS Azure SQL relacionu bazu. Ovaj framework implementira UnitOfWork obrazac u obliku DbContext klase, pri čemu se on sastoji od više DbSet-ova koji čine *Repository* obrazac.

Repository dijagram

* 1. Generalna arhitektura

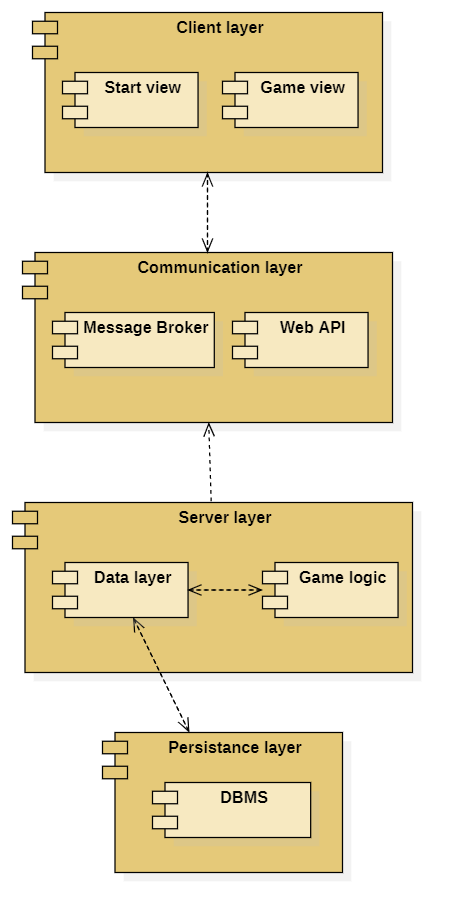
Arhitektura sistema podrazumeva postojanje klijenta, servera i baze podataka u kojoj će se čuvatiinformacije o korisnicima i njihovim igrama.

Generalna arhitektura

* 1. Strukturni pogledi

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

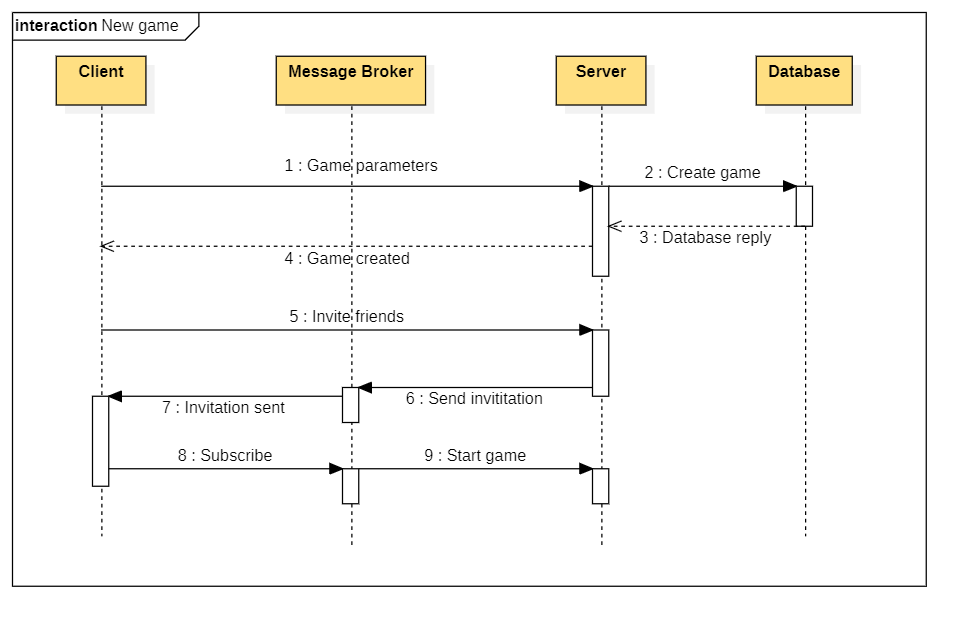
Klijentska aplikacija je implementirana kao *Web aplikacija* i predviđena je za računare svih operativnih sistema. Serverska aplikacija realizuje se kao *ASP.NET Core Web API*, dok se kao server baze podataka koristi *Azure DBMS.* Za asinhronu komunikaciju između klijenta i servera, koristi se *SignalR* message broker.



*Dijagram strukture sistema*

* 1. Bihevioralni pogledi

Bihevioralni pogled opisuje interakciju između komponenti sistema prilikom nastanka događaja. Sledeći dijagram predstavlja bihevioralni pogled sistema na primeru kreiranje nove igre. Klijent šalje sve parametre igre koje je odabrao serveru, koji nakon toga kreira igru i čuva je u bazi. Zatim, klijent bira koje će saigrače pozvati da se pridruže kreiranoj igri. Svi pridruženi igrači se subscribe-uju na obaveštenja o promenama u datoj igri, nakon čega igra počinje.

**

*Sekvencijalni dijagram za kreiranje nove igre*

Naredni dijagram predstavlja bihevioralni pogled sistema na primeru toka jednog poteza. Server proziva igrača koji je na redu, nakon čega se igrač priprema za napad regrupisanjem svojih armija. U toku svog poteza igrač može da napada proizvoljan broj puta. Nakon svakog napada server proverava stanje igre i po potrebi komunicira sa bazom kako bi se podaci o igri ažurirali. Zatim, server ostvaruje komunikaciju sa message broker-om putem “publish” operacije. Svi igrači koji su u datoj igri (“subscribers“) bivaju obavešteni ako je došlo do izmena.

**

*Sekvencijalni dijagram za odigravanje poteza*

* 1. Implementaciona pitanja

Specifikacija biblioteka i programskihokvira**:**

• **React JS** – Java Script frontend framework za pisanje SPA Web aplikacija

• **SignalR**– Message broker, obezbeđuje API za real-time klijent-server komunikaciju

• **ASP.NET Core Web API** – Serverska aplikacija

• **Entity Framework** – Objektno-relacioni maper (ORM framework)

• **Microsoft Azure SQL DBMS** – Baza podataka

1. **Analiza arhitekture**
   1. Potencijalni rizici u implementaciji i strategije prevazilaženja

Potencijalni rizik pri implementaciji *RizzyCo* sistema je problem kapaciteta servera za veliki broj korisnika. Strategija za prevazilaženje ovog rizika može biti testiranje performansi i opterećenja kada se pokrene osnovni aplikacioni server. Ovakvom metodom bi se dobili konkretni pokazatelji neophodni za planiranje kapaciteta sistema.