***Battle of Mages***

*Arhitektura sistema*

*Jelena Firulović 17509*

*Teodora Stamenković 17414*

# Kontekst i cilj softverskog projekta

Battle of Mages je online, multiplayer, turn based kartaška igra. Cilj igre je eliminisati protivnike uz pomoć različitih tipova karata iz špila.

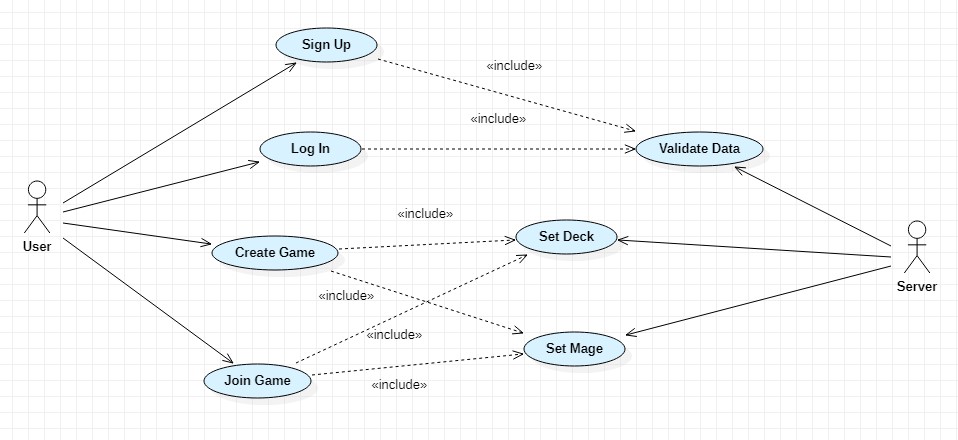
Registrovani igrači imaju opciju da kreiraju novu partiju, pridruže se ili određenoj partiji uz pomoć jedinstvenog koda koji se generiše prilikom kreiranja igre ili nekoj od partija sa slobodnim mestom, izabranom slučajnim izborom. Prilikom kreiranja partije igrač bira za koliko igrača je partija (2 – 4), a zatim podešava heroja i špil. Nakon uspešnog kreiranja partije i prijavljivanje dovoljnog broja igrača, igra započinje. Igra traje sve dok ne ostaje jedan igrač sa health points, a tokom same igre, igrači se smenjuju ko je na potezu. Svaki od igrača tokom svog poteza bira šta će odigrati, dok dostupne mogućnosti zavise od trenutnog stanja igre. Takođe tokom partije, učesnici mogu da komuniciraju međusobno.

# Arhitekturni zahtevi

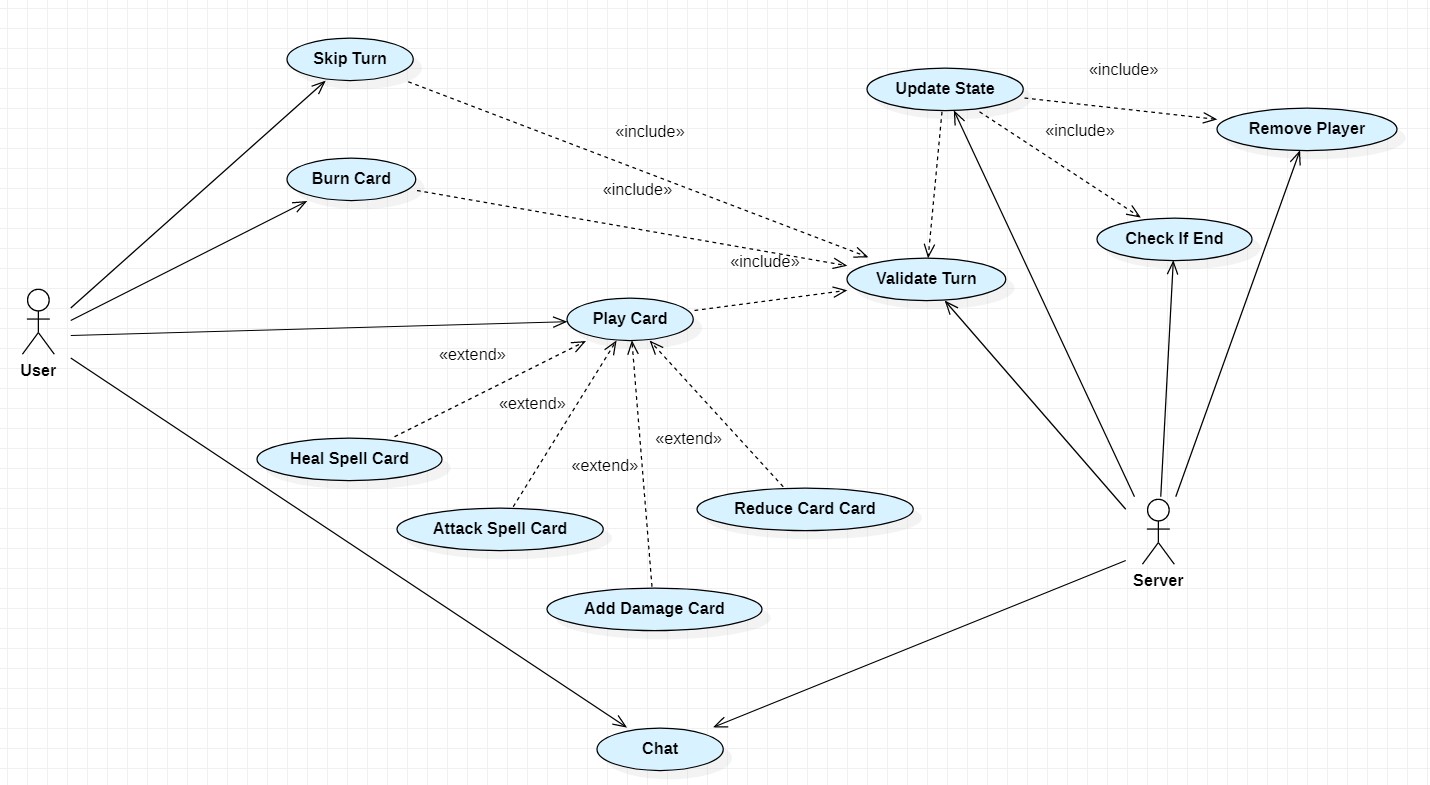
U ovom delu dokumenta biće navedeni arhitekturni zahtevi koje je potrebno da *Battle Of Mages* zadovolji, a uključuju glavne funkcionalne zahteve - arhitekturno značajne slučajeve korišćenja, nefunkcionalne zahteve – atribute kvaliteta, kao i tehnička i poslovna ograničenja.

## Glavni funkcionalni zahtevi

* **Registracija korisnika** – za kreiranje korisničkog naloga potrebne informacije su username i password. Nakon validacije zahteva, korisnik od sistema dobija jedinstveni tag koji je oblika #četvorocifreni\_broj.
* **Prijavljivanje korisnika**
* **Kreiranje partije ili pridruživanje partiji**
* **Podešavanje heroja pre partije** – Igrač pre partije bira jednu od 4 tipa magije (fire, ice, air, earth).
* **Podešavanje špila pre partije** – Izbor odnosa određenih tipova karata u špilu.
* **Generisanje špila** – Za svakog igrača se na osnovu podešavanja pre partije kreira špil karata.
* **Odigravanje poteza** – Igrač koji je na potezu bira jednu od dostupnih mogućnosti koje zavise od trenutnog stanja igre.
* **Eliminisanje protivnika** – Igrač je eliminisan kad njegovi *health points* dođu na nulu.
* **Kraj igre** – Igra je završena kada ostane samo jedan igrač sa *health* poenima.
* **Komunikacija između igrača**

****

*Use Case dijagram za početak igre*



*Use Case dijagram u toku igre*

## Nefunkcionalni zahtevi (atributi kvaliteta)

* **Performanse** – Težiti ka što manjem vremenu odziva.
* **Skalabilnost** – Potrebno je da aplikacija podrži veći broj korisnika i više zahteva istovremeno. Takođe, potrebno je obezbediti i efikasan rad sistema pri povećanju veličine skladišta.
* **Modifikabilnost** – Projektovanje sistema tako da modifikacija i evolucija bude laka.
* **Sigurnost** – Obezbediti autentifikaciju i autorizaciju, kao i enkripciju osetljivih podataka.
* **Upotrebljivost** – Obezbediti jednostavno i intuitivno korišćenje aplikacije.
* **Pouzdanost i dostupnost** – Potrebno je da konekcija sa serverom bude pouzdana tokom partije i da aplikacija bude dostupna korisnicima 24/7.

## Tehnička i poslovna ograničenja

Tehnička ograničenja koja se nameću su:

* **Komunikacija** – Treba obezbediti dva tipa komunikacije: sinhrona komunikacija (klijent-server) i asinhrona komunikacija (prosleđivanje izmene stanja svim igračima).
* **Skrivenost baze podataka** – Način reprezentacije podataka u bazi je skriven od korisnika.
* **Pokretanje igre na različitim konfiguracijama sistema**

Poslovna ograničenja se baziraju na osnovnim konceptima igre. Rezultat svakog poteza korisnika zavisi od tipa igrača i tipa karte. Akcije koje korisnik može da obavi zavise od trenutnog stanja partije. Implementacija igre će doprineti ispunjenju ovih ograničenja.

# Arhitekturni dizajn

## Arhitekturni obrasci

* **Layered pattern**

Igra će implementirati troslojni *Layered* obrazac – klijentski, serverski i sloj baze podataka.

Korisnik će interagovati sa aplikacijom preko korisničkog interfejsa koji predstavlja klijentski sloj.

Serverski sloj prestavlja spregu između klijentskog sloja i sloja baze podataka. Komunikacija između klijenta i servera može biti sinhrona i asinhrona. Sinhrona komunikacija se obavlja uz pomoć RESTful API-ja, a asinhrona preko Message Broker-a. Veza između servera i baze podataka biće ostvarena korišćenjem Entity Framework-a. Takođe, ovaj sloj je odgovoran za samu logiku igre.

Sloj baze podataka je zadužen za perzistenciju podataka i komunikaciju sa samom bazom podataka.

* **Publish/Subscribe pattern**

Arhitekturni obrazac *Publish/Subscribe* će biti implementiran preko message broker-a koji će biti zadužen za asinhronu komunikaciju između klijenta i servera. Svi igrači jedne partije su automatski *subscribe*-ovani za prijem informacija o izmenama stanja igre (odigrani potezi protivnika), kao i za prijem poruka od strane ostalih igrača u partiji. Rezultat ovoga biće vidljiva promena prikaza igre u realnom vremenu.

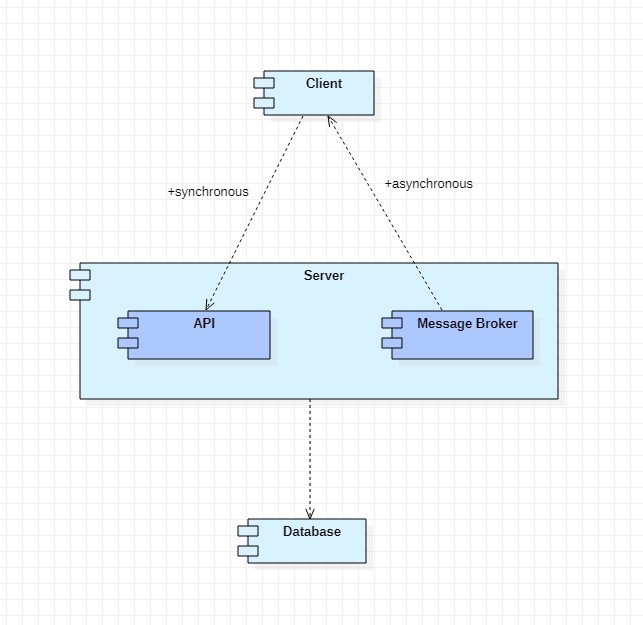
* **Repository**

Za perzistenciju podataka će se koristiti centralizovano skladište, odnosno centralizovana baza podataka. Serverska aplikacija pristupa i radi sa centralnim skladištem preko Entity Framework-a.

* **MVC**

MVC pattern će biti iskorišćen u sklopu ASP.NET Web API-ja. Uloga modela je definisanje domenskih klasa i njihova perzistencija. Controller-i će obezbediti korišćenje usluga samog API-ja. Odgovornost kontrolera je rad sa modelom i selektovanje odgovarajućeg prikaza (view). Informacije potrebne za prikaz će biti prosleđene Unity klasama, odnosno skriptama, na klijentskoj strani preko HTTP response-a. Ove skripte će biti zadužene za generisanje odgovarajućeg interfejsa.

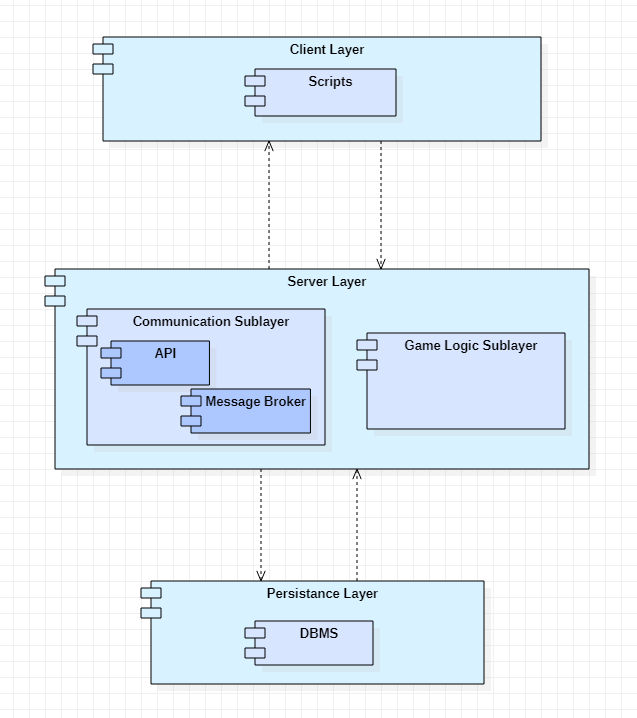
## Generalna arhitektura

****

*Generalna arhitektura*

## Strukturni pogled

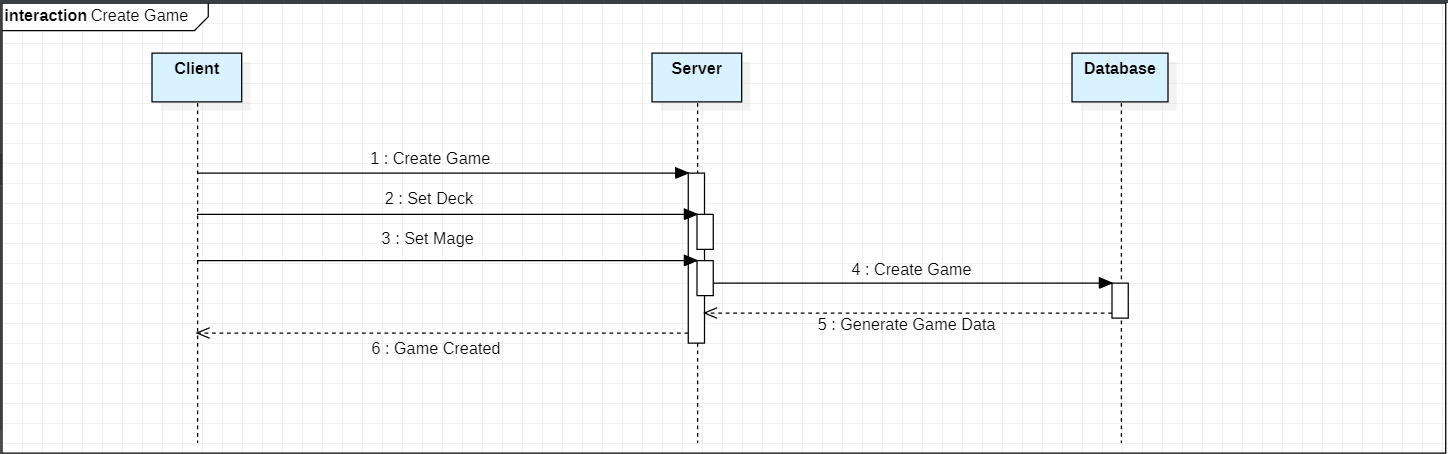
Dijagram prikazuje strukturu komponenti sistema i njihovu povezanost. Klijentski sloj, koji predstavlja deo aplikacije sa kojim korisnici interaguju, se sastoji od skripti, od kojih su neke zadužene za prikaz, dok neke ostvaruju komunikaciju sa serverom. Serverski sloj se sastoji od komunikacionog podsloja i podsloja za logiku igre. Komunikacioni podsloj obuhvata RESTful Web API za sinhronu komunikaciju i Message Broker za asinhronu komunikaciju sa klijentom. Na sloju perzistencije se nalazi DBMS kao konekcija sa bazom podataka.



## Bihevioralni pogledi

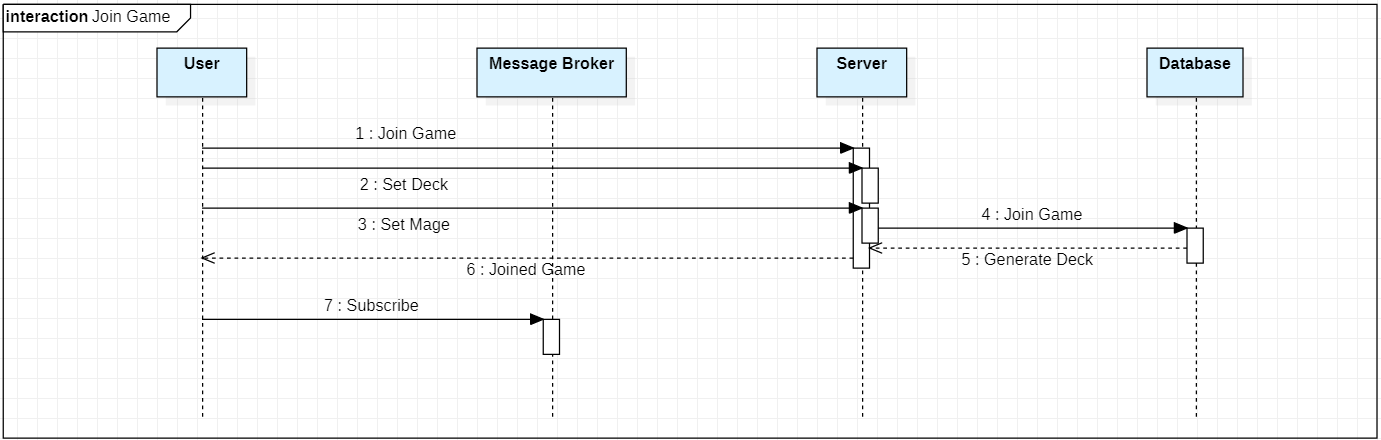
* **Kreiranje nove igre**

Klijent započinje kreiranje nove igre i šalje podešavanja špila i izbor magije serveru. Server nakon toga kreira igru, čuva je u bazi, generiše špil na osnovu podešavanja i generiše jedinstveni kod igre na osnovu kog će se drugi igrači pridružiti.

****

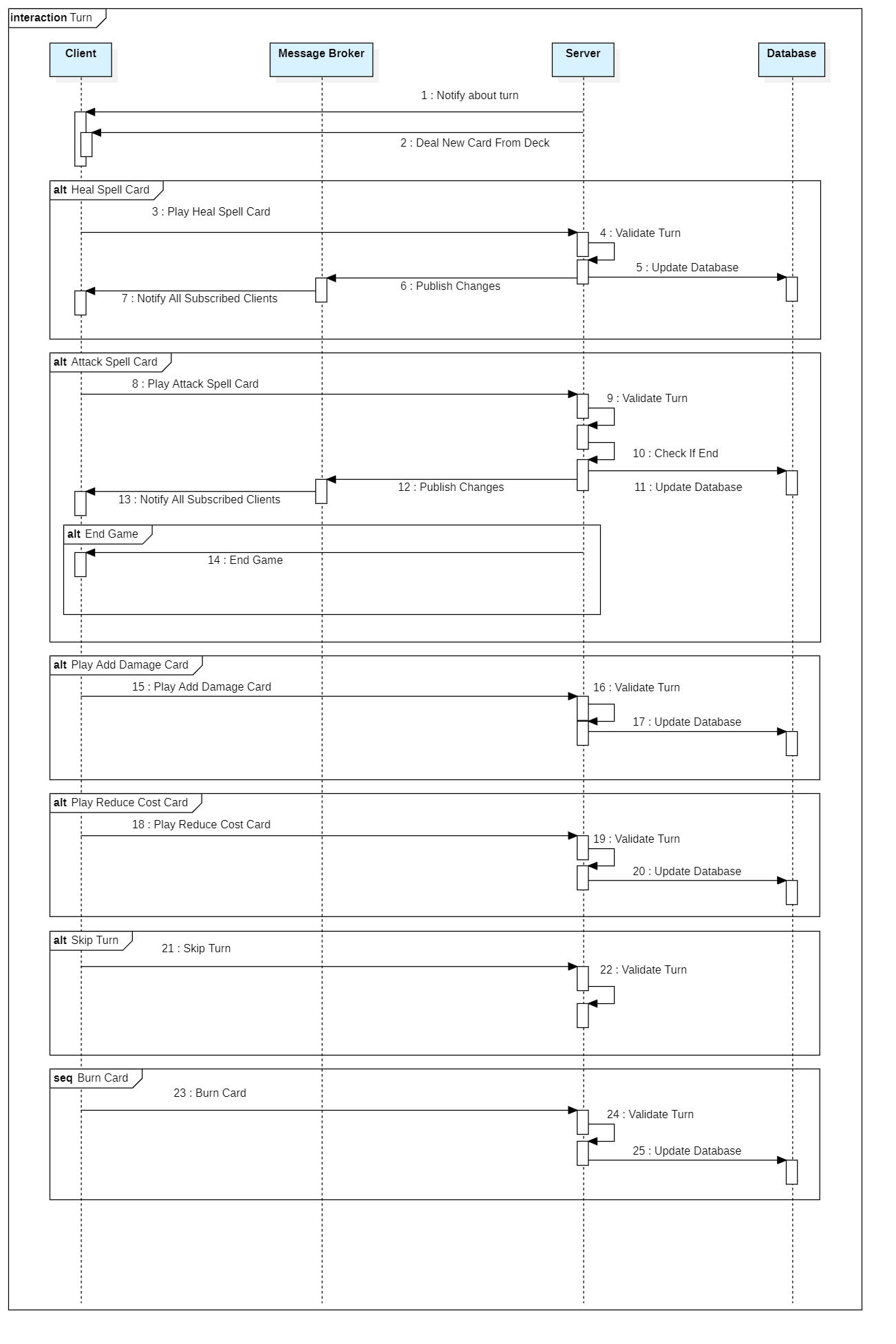
* **Pridruživanje igri**

Korisnik se pridružuje igri uz pomoć jedinstvenog koda koji se generiše prilikom kreiranja igre, ili izborom opcije *„Random Game“* kada se pridružuje nekoj od otvorenih igri. Nakon toga, korisnik šalje parametre špila i izbor magije serveru. Server validira zahtev, dodaje korisnika u igru i generiše špil karata za korisnika. Igrač se *subscribe*-uje na igru kako bi dobijao obaveštenja o promenama u toku igre.

****

* **Odigravanje poteza**

Server obaveštava igrača koji je na potezu i deli jednu kartu iz špila. Svaki potez se validira i ažurira se baza podataka. Tok poteza zavisi od tipa izabrane karte. Izborom neke od *Heal* karata, igrač povećava svoje health points i ostali igrači se obaveštavaju o ovoj promeni. Ukoliko igrač izabere kartu tipa *Attack*, ažuriranje podataka obuhvata i smanjenje *health* poena igrača koji je napadnut. Proverava se da li je kraj igre i igrači se obaveštavaju o stanju igre. Izborom *Add Damage* karte, igrač dodaje nekoj od svojih *Attack* karti veću moć. Karte tipa *Reduce Cost* smanjuju broj poena (*mana points*) koji je potreban da bi se karta odigrala. Promene koje su posledica izbora nekog od dva poslednja tipa su vidljive samo korisniku koji je na potezu, ostali korisnici se ne obaveštavaju o tome. Takođe, pored odigravanja karte, korisnik može da preskoči potez ili da izabere opciju *Burn Card. Burn Card* opcija povećava *mana* poene za onoliko poena koliko košta odigravanje karte.

****

# Implementacija

Za implementaciju klijentskog dela će se koristiti Unity Game Engine, dok će se za serversku stranu koristiti ASP.NET Web API. Za rad sa bazom podataka će se koristiti SQL Server, a kao ORM framework će se koristiti Entity Framework.

# Analiza arhitekture

Prilikom revizije načina implementacije sistema, potencijalni rizik koji je identifikovan je problem kapaciteta servera ukoliko dođe do porasta broja korisnika. Tokom rada sistema vrši će se čest monitoring rizika, testiranjem opterećenja i performansi, dok bi neke od strategija prevazilaženja bile zamena centralizovanog skladišta distribuiranim, ili rasterećivanje skladišta brisanjem ili prebacivanjem zastarelih podataka.