**RTOG**

***Risk Type Of a Game***

# Strateška igra osvajanja

Arhitekturni Dizajn SistemaIstorija izmena

| **Datum** | **Izmene** | **Odgovorni** |
| --- | --- | --- |
| 16.12.2022. | Inicijalna verzija | Nikola Rašić, Ivan Bogosavljević |
| 10.01.2023. | Izbačen React.js iz radnih okvira | Nikola Rašić |
|  |  |  |
|  |  |  |

# 

# 

# Opis Sistema

Igra je predviđena da omogućava borbu između 2 do 4 igrača. Svaki igrač tokom svog poteza može da kreira, vidi i unapređuje svoje borbene figure. Borbene figure dolaze u nekoliko rasa, sa različitim unapređenjima. Svaki igrač ima za cilj osvajanje svih polja na mapi igre.

Funkcionalni zahtevi:

* Omogućiti igračima kreiranje sobe za igru bez potrebe prijavljivanja ili pravljenja naloga
* Omogućiti vlasnicima soba generisanje jedinstvenog koda svake sobe.
* Omogućiti igračima pridruživanje sobama tuđih igrača pomoću jedinstvenog koda.
* Omogućiti vlasnicima soba pokretanje igre sa prisutnim igračima.
* Omogućiti igračima manipulisanje svojih trupa i strukture mape. Ovo uključuje osvajanje novih neokupiranih polja, mogućnost napadanja polja koja su zauzeta od strane neprijatelja, dodavanje novih trupa na osvojena polja (moguće je postaviti više trupa na jedno polje), unapređivanje postojećih trupa i pomeranje istih sa jednog na drugo polje.
* Dodavanje i unapređivanje trupa će se obavljati pomoću izmišljene valute, koja je dostupna samo unutar igre. Omogućiti igračima da dobiju određenu sumu ove valute na početku svakog svog poteza.
* Omogućiti kvantizaciju jačine jedne figure (na primer, poeni snage figure). Pomoću ovih brojki omogućiti određivanje jačine jednog polja.
* Omogućiti određivanje ishoda napada na neprijateljsko polje. Ovo uključuje simuliranje gubitaka figura koje su “poginule” u bitci.
* Omogućiti ažuriranje mape u realnom vremenu.
* Omogućiti određivanje kraja igre i prikazivanje ishoda iste svakom igraču.
* Omogućiti više instanci različitih igara istovremeno.

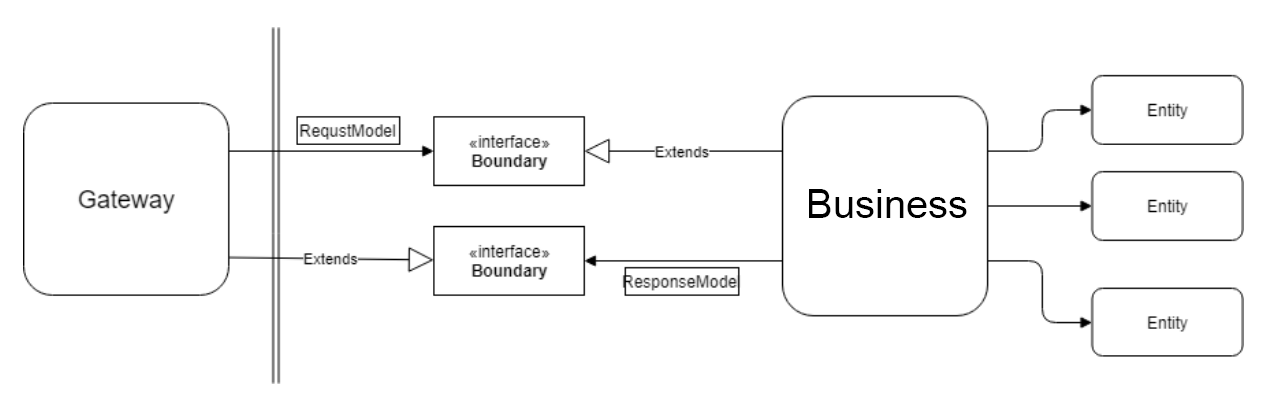
Nefunkcionalni zahtevi:

* Performanse
  + Promene na mapi moraju biti vidljive ostalim igračima u najviše 10s. Iako je igra spora i performanse nisu najveći prioritet, smatramo da će čekanja preko ovog vremenskog intervala ostavljati loš utisak na korisničko iskustvo.
* Pouzdanost
  + Igra treba da funkcioniše i biti dostupna 99% vremena godišnje. Ovo ne uključuje prekid rada potreban za deploy novih verzija igre.
* Portabilnost
  + Treba omogućiti jednostavnu instalaciju i deployment igre na nove servere.
  + Osigurati saglasnost sa zakonom, tj. ne koristiti tuđi kod, slike i ostalo bez dozvole tj. licence i/ili reference, ukoliko je to traženo.
* Lakoća održavanja
  + Igra treba biti realizovana sa razdvojenim servisima tj. komponentama na kojima se može raditi odvojeno.
  + Igra treba čuvati dnevnik grešaka i dnevnik upita od strane klijenata, kako bi analiziranje raznih situacija bilo što efikasnije.
  + Igra treba biti realizovana na način koji omogućava lako dodavanje novih funkcionalnosti.
* Kompatibilnost
  + Osigurati se da igra adekvatno funkcioniše na svim popularnim web pretraživačima.
* Lakoća korišćenja
  + Igra mora biti intuitivna i laka za navigaciju i korišćenje čak i za korisnike koji nisu ranije igrali igre.
  + Interfejs igre mora biti realizovan na način koji ne dozvoljava odstupanje pravilima igre od strane korisnika.
* Lakoća testiranja
  + Izbegavati direktne povezanosti komponenata i omogućiti lako praćenje izvršenja koda.

# Prikaz Arhitekture Sistema:

Clean Architecture

Arhitektura koriscena u sistemu je slojevita po ugledu na Clean Architecture dizajn koji je prikazan na slici ispod.

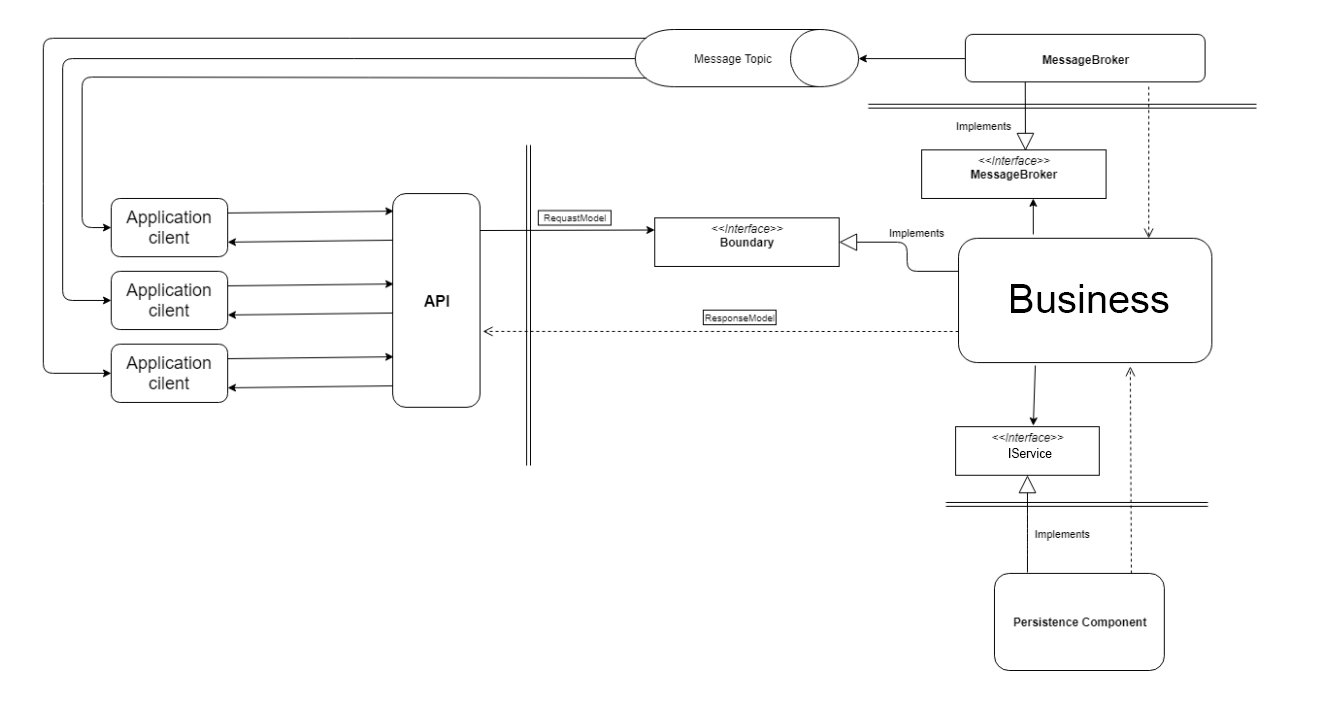


Slojevi u ovoj arhitekturi su Gateway, Boundary, Business i Entities.

* **Business** - najbitniji sloj arhitekture, u njemu se nalazi sva poslovna logika specificna za aplikaciju. On se sastoji od vise businessObjekta a svaki od njih predstavlja jednu atomicnu akciju koju korisnik moze da zatrazi od sistema
* Entity sloj je sloj u aplikaciji koji sadrži objekte koji predstavljaju stvari u sistemu (npr. korisnike, proizvode, porudžbine itd.), kao i logiku koja se koristi za rad sistema. Ovi objekti obično izvode neka generička računanja ili opšte zadatke česte u ovakvim vrstama aplikacija. Entity sloj se često koristi za upravljanje skladištenjem podataka i izvršavanjem osnovnih radnji nad njima (npr. čitanje, ažuriranje, brisanje itd.). Entity sloj se takođe može koristiti kao modul koji se može koristiti u drugim aplikacijama, jer sadrži logiku koja nije specifična za samu ovu aplikaciju već se može primeniti u širem kontekstu.
* Boundary - sloj koji predstavlja granicu izmedju Gateway i Business sloja. Obezbedjuje da ta dva sloja nisu cvrsto Povezana.
* Gateway - zadatak ovog sloja je da prikuplja zahteve od spoljasnjeg sveta, u najcescem Slucaju je to web, i da ih prosledi dalje Business sloju preko boundary Sloja

Nasa Implementacija Arhitekture

Nasu implementaciju Clean Architecture dizajn obrasca mozete videti na slici ispod



Gateway sloj: Ovaj sloj se odnosi na sve interfejse koji omogućavaju komunikaciju sa spoljnim sistemima, kao što su baza podataka, mreža itd. Gateway sloj se obično sastoji od adaptera koji omogućavaju pristup podacima i servisima iz spoljnih sistema.

Boundary sloj: Ovaj sloj se odnosi na interfejse koji se koriste za komunikaciju sa korisnicima aplikacije. To uključuje interfejse za kreiranje sobe za igru, unos koda za pridruživanje sobi, unos poteza itd. Boundary sloj se obično sastoji od kontrolera koji se koriste za rukovanje zahtevima korisnika i za slanje odgovora.

Business sloj: Ovaj sloj se odnosi na funkcionalnosti koje aplikacija pruža. To uključuje logiku za kreiranje i unapređivanje trupa, određivanje jačine polja, određivanje ishoda napada itd. Business sloj se obično sastoji od interaktora koji se koriste za izvršavanje određenih akcija i za vraćanje rezultata.

Entities sloj: Ovaj sloj se odnosi na podatke koje aplikacija čuva i manipuliše. To uključuje informacije o trupama, poljima, igrama itd. Entities sloj se obično sastoji od objekata koji predstavljaju podatke i koji se koriste za rad sa tim podacima.

Entity objekti u nasoj implementaciji, izmedju ostalih, cine Service i MessageBroker.

Zadaci ovih entiteta su:

* Message Broker se koristi da obavesti korisnike o tome da je stanje mape promenjeno i salje odgovarajucu poruku u MessageTopic. Svaka soba ima svoj poseban MessageTopic.
* Service su entiteti koji vrse osnovne CRUD operacije nad nekom bazom podataka. Obezbedjuju perzistenciju podataka u sistemu.

# Izbor Radnih Okvira

Za izradu sistema smo izabrali sledeće radne okvire koji će biti korišćeni u datim komponentama:

* **ASP.NET MVC**
  + API
  + Client Application
* **Entity Framework Core**
  + Database ORM
* **SignalR**
  + MessageBroker

# Arhitektura Klijentske Aplikacije

ASP.NET MVC implementira MVC obrazac korišćenjem .cshtml fajlova, gde se html nadograđuje pomoću C#, koristeći podatke iz Modela, da bi generisao View. Controller od koga je zatražen ovaj View ga potom vraća korisniku.

MVC Arhitektura

MVC (Model-View-Controller) arhitektura je jedan od najčešće korišćenih modela u razvoju softvera, koji se koristi za razdvajanje logike aplikacije od njenog korisničkog interfejsa.

Model je deo aplikacije koji se bavi upravljanjem podacima i logikom poslovnog procesa. To je deo aplikacije koji se bavi manipulacijom podacima i komunikacijom sa bazom podataka.

Prikaz (View) je deo aplikacije koji se bavi prikazivanjem podataka korisniku. To je ono što korisnik vidi i sa čime interaguje.

Kontroler (Controller) je deo aplikacije koji povezuje Model i View. On prima zahteve korisnika preko View-a i obrađuje ih koristeći Model, a zatim vraća odgovarajući rezultat korisniku preko View-a.

Ovo olakšava razvoj, održavanje i skaliranje aplikacije.