Arhitektura i projektovanje softvera

DON'TGETSPICY

Web aplikacija za kolaborativno realtime igranje društvene igre Ludo (Ne ljuti se čoveče)

Faza 1: Arhitekturni dizajn softverskog sistema

Ognjen Damnjanović 16528

Mila Mirović 16742

Sadržaj

Kontekst i cilj projekta

Aplikacija *Don'tGetSpicy* je kolaborativna realtime web aplikacija koja prati pravila poznate društvene igre Ludo ili "Ne ljuti se čoveče". Generalna funkcionalnost aplikacije je pružanje podrške za više igrača po igri u skladu sa pravilima, dok će sami igrači imati mogućnost komunikacije preko chat-a. Svaki igrač će imati pregled statistike na osnovu svojih odigranih igara. Cilj projekta je omogućavanje: grupnog igranja društvene igre, komunikacije između učesnika igre i jednostavnog pregleda sopstvene statistike.

Arhitekturni zahtevi

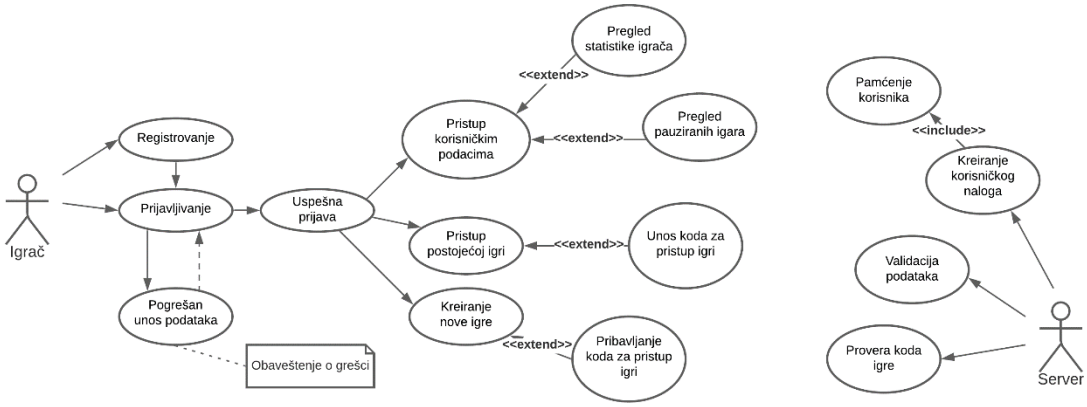
U ovom odeljku su definisane osnovne funkcionalnosti softverskog sistema *Don'tGetSpicy*, arhitekturno značajni funkcionalni zahtevi i slučajevi korišćenja, ne-funkcionalni zahtevi tj. atributi kvaliteta koje treba zadovoljiti i tehnička i poslovna ograničenja.

Arhitekturno značajni slučajevi korišćenja (glavni funkcionalni zahtevi)

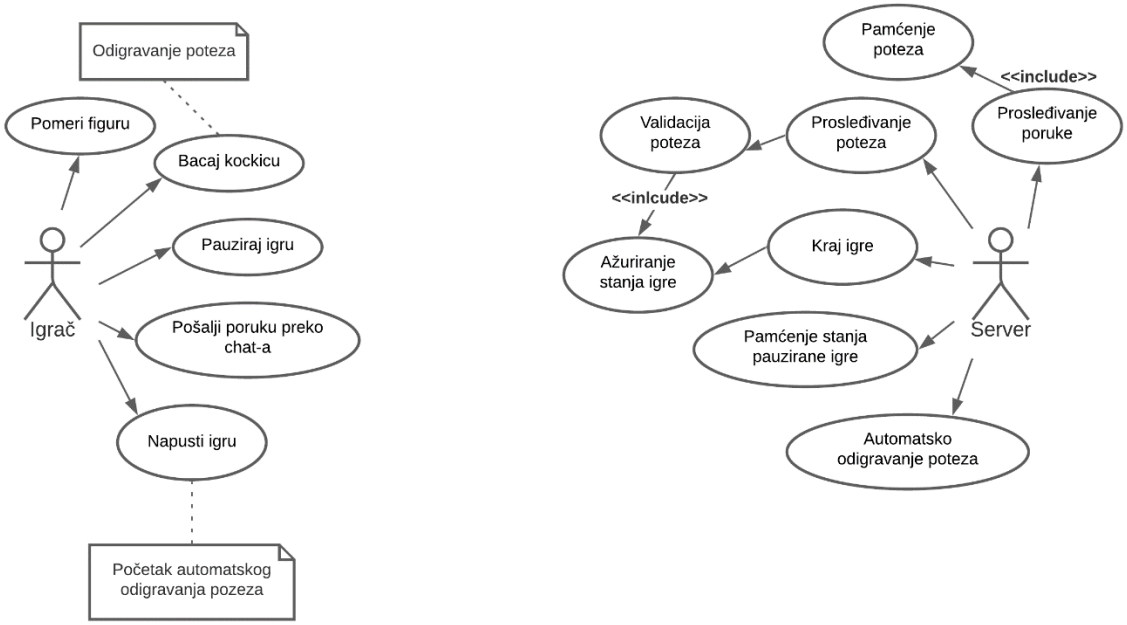
1. **Registracija igrača** – Registracija novih korisnika se vrši zadavanjem korisničkog imena i lozinke. Obavlja se provera ispravnosti korisničkog imena.
2. **Prijavljivanje na nalog igrača** – Prijavljivanje na postojeći nalog obavlja se unosom korisničkog imena i lozinke. Proverom unetih podataka korisnik pristupa svom nalogu ili biva obavešten o eventualnoj grešci.
3. **Startni prozor** – Predstavlja inicijalni prozor nakon prijavljivanja. Omogućava igraču da pristupi svojim podacima, pridruži se partijii koja je u toku pomoću koda ili kreira novu igru.
4. **Odigravanje poteza** – Promena stanja aplikacije na osnovu odigranog poteza jednog od igrača. Ta promena je vidljiva svim igračima.
5. **Pauziranje igre** – Igrač koji je kreirao igru može da pauzira trenutnu igru, pri čemu svi učesnici bivaju vraćeni na startni prozor. Igrači mogu da nastave igru uvidom u svoje prethodno pauzirane igre.
6. **Validacija poteza** – Provera da li je moguće odigrati potez koji je pristigao od korisnika koji je bio na potezu na strani servera.
7. **Automatsko odigravanje poteza** – U slučaju napuštanja igre od strane nekog od igrača sistem odigrava poteze umesto njega.
8. **Kraj igre** – Provera da li su stečeni uslovi za završetak igre odnosno da li je neki od učesnika igre popunio sva ciljna polja. Po završetku igre podaci o istoj se brišu, a statistike njenih učesnika se ažuriraju.
9. **Mogućnost komunikacije preko chat-a** – Razmenjivanje poruka između učesnika igre sa mogućnošću isključivanja pristizanja poruka.
10. **Prikaz podataka o igraču** – Prikaz statistike igrača, ličnih podataka i pauziranih igara.

U nastavku su dati dijagrami za gore navedene funkcionalne zahteve aplikacije *Don'tGetSpicy*.

Pre početka same igre



Tokom igre



Nakon igre

Ne-funkcionalni zahtevi (atributi kvaliteta)

Tokom projektovanja sistema cilj je ostvarati sledeće ne-funkcionalne zahteve (atribute kvaliteta):

1. **Pristupačnost** – aplikacija ne treba da zavisi od platforme na kojoj se izvršava.
2. **Upotrebljivost** – aplikacija treba da pruža intuitivan korisnički interfejs i koji rezultuje lakoćom korišćenja.
3. **Dostupnost** – aplikacija treba da je dostupna putem Interneta 24/7.
4. **Sigurnost i bezbednost** – sistem treba poštovati bazične principe kontrole (autorizacije i autentifikacije) naloga igrača, što se odnosi na lične podatke korisnika kojima može pristupiti samo vlasnik istih.
5. **Performanse** – sistem treba obezbediti brzo odigravanje poteza, ažuriranje prozora ostalih igrača pri promenama u toku igre i minimalno vreme odziva.
6. **Pouzdanost** – sistem treba omogućiti određen nivo perzistencije i trajno skladištenje podataka o partijama koje nisu završene i o igračima.

Tehnička i poslovna ograničenja

U dizajnu svake softverske arhitekture evidentno je da postoje određena tehnička i poslovna ograničenja koja predstavljaju restrikcije (ne obavezno loše) za dizajn sistema i/ili proces kojim se sistem razvija.

Tehnička ograničenja ogledaju se u izboru programskog jezika, baze podataka, operativnog sistema/platforme i korišćenja određene biblioteke ili framework-a. U pogledu web aplikacije *Don'tGetSpicy*, framework na frontend-u Vue.js je prikladan za manje projekte sa bogatim korisničkim interfesjom (što odgovara trenutnoj prirodi ovakve aplikacije), ali ukoliko bi se javila želja za proširivanjem sistema gotovim komponentama (plugin-ovima) izbor istih bi bio ograničen. Zbog prirode aplikacije, a u cilju efikasnog iskorisćavanja resursa, moraju se odabrati framework-ci (kako na klijentskoj, tako i na serverskoj strani) koji podrzavaju neki vid asinhronog slanja poruka direktno sa servera ka klijentu, a koji su mežusobno kompatibilni.

Intuitivnost i jednostavnost korišćenja aplikacije – realizuje se prilagođavanjem korisničkog iskustva i interfejsa (User Expirience i User Interface) za opšti tip korisnika (pošto je ova društvena igra po prirodi namenjena svim tipovima korisnika);

Transparentnost i izolacija podataka – potrebno je odvojiti šemu baze podataka od korisničkog prikaza, jer je korisncima potrebno prikazati podatke na specifičan način;

Poslovna ograničenja u pogledu razvoja same aplikacije, javljaju se indirektno kroz poslovne odluke i to su najčešće raspored ili rok završetka, budžet, sastav tima, i slično. Pošto se radi o studentskom projektu, ovi elementi nisu toliko veliki ograničavajući faktor. Poslovno ograničenje koje se nameće izborom tehnologije na frontend-u je to što je Vue.js je relativno nov, pa je samim tim podložan promenama, zahteva redovno praćenje ažuriranja i njihovu primenu u aplikaciji, te ona može postati neisplativa za održavanje. Iz istog razloga je teško naći developer-e koji su dobro upoznati sa ovim framework-om tako da ne mogu svi koji su angažovani na projektu da imaju iste nivoe znanja o Vue.JS-u.

Arhitekturni dizajn

Arhitekturni obrasci

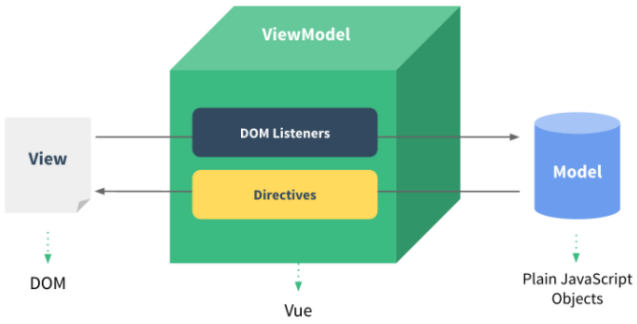
Aplikacija Don'tGetSpicy će koristiti sledeće obrasce:

- ***Layered obrazac*** – komponente aplikacije Don'tGetSpicy biće podeljene u 3 sloja – klijentski sloj, serverski sloj i sloj baze podataka (kao što je prikazano na slici generalne arhitekture). U nastavku je dat detaljniji opis.
- ***Publish/Subscribe obrazac*** – za komunikaciju izmedju servera i klijenta kada je u pitanju ažuriranje stanja igre. Svaki aktivan učesnik igre mora biti u toku sa promenama od interesa nastalim u igri.
- ***MVC obrazac*** na strani klijenta

Sistem se sastoji od sledećih slojeva:

- **Klijentski sloj:**

  Ovaj sloj predstavlja sadrži korisnički interfejs koji će se korisnicima prikazati u browser-u i čini direktnu vezu sa korisnicima. Koristićemo ***Vue.js*** framework za kreiranje interaktivnih web interfejsa, na klijentskoj strani. Ovaj framework koristi varijantu MVC obrasca – **MVVM** (*Model-View-ViewModel*) koja bidirekciono povezuje **View** i **Model**, te omogućava njihovu uzajamnu sinhronizaciju.



(Pregled koncepta MVVM obrasca u Vue.js-u)

ViewModel je objekat koji sinhronizuje Model i View i u Vue.js-u svaka instanca je *ViewModel*. Instanciranje se vrši pomoću Vue konstruktora. U MVVM obrascu Controller (iz MVC) je zamenjen sa ViewModel.

View je konkretni DOM kojim upravljaju Vue instance, a svaka Vue instanca je povezana sa odgovarajućim DOM elementom.

Model je neznatno modifikovan plain JavaScript objekat ili data objekat. Kada se objekat koristi unutar Vue instance on postaje *reaktivan* i može se manipulisati njegovim svojstvima, pri čemu će Vue instance koje ih posmatraju biti obaveštene o tim promenama.

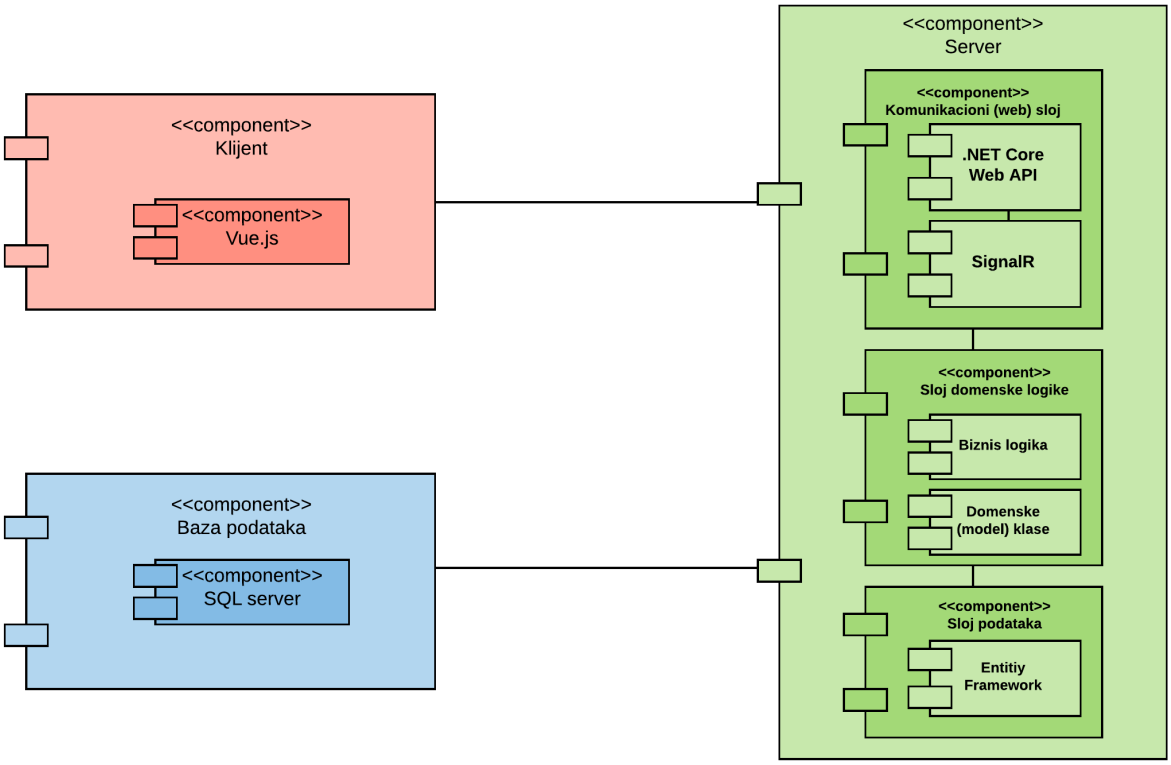
- **Serverski sloj:**

Ovaj sloj je veza između klijentskog sloja i sloja baze podataka. Komunikacija klijenta sa serverom obavljaće se pomoću API-ja i to ***ASP.NET Core WebAP***I-ja. Biblioteka .NET-a ***SignalR*** omogućava asinhrono obaveštavanje od servera ka klijentu i na taj način omogućava komunikaciju iniciranu od strane servera. Sloj domenske logike sadrži biznis logiku, servise i domenske klase, ovaj sloj će vršiti komunikaciju sa slojem podataka i na taj način generisati odgovor na pozive API-ja i/ili obaveštavati klijente o dostupnim ažuriranjima. Na sloju podataka je EntityFramework Core koji služi kao ORM za konverziju domenskih objekata u odgovarajuće entitete u realcionoj bazi podataka koju koristimo.

- **Sloj baze podataka:**

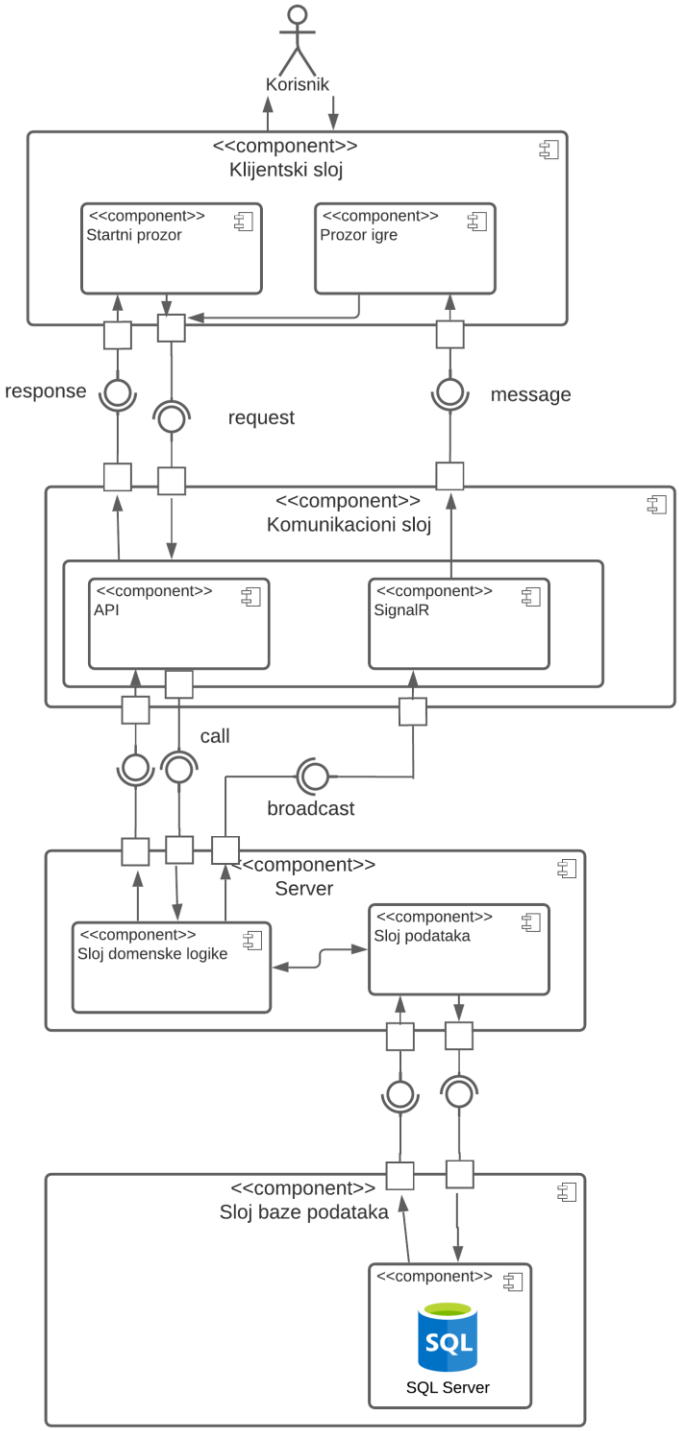
Čuvanje podataka o igračima (korisnicima) i igrama pružaće sloj baze podataka. Za ovaj sloj koristimo Microsoft SQL Server koji predstavlja relacionu bazu podataka. Azure Data Studio koristimo za vizuelnu manipulaciju bazom podataka.

Generalna arhitektura (box-line)

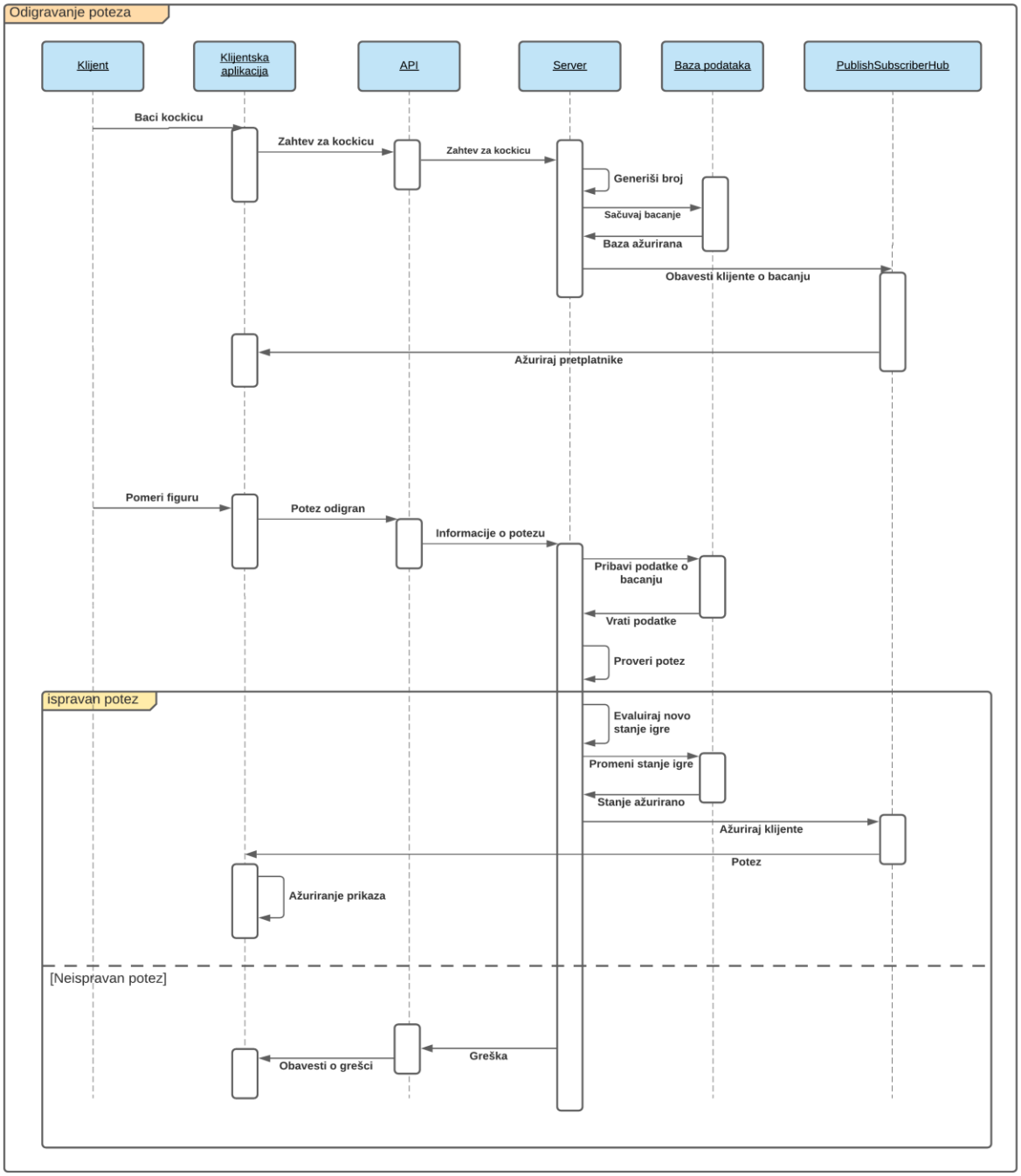


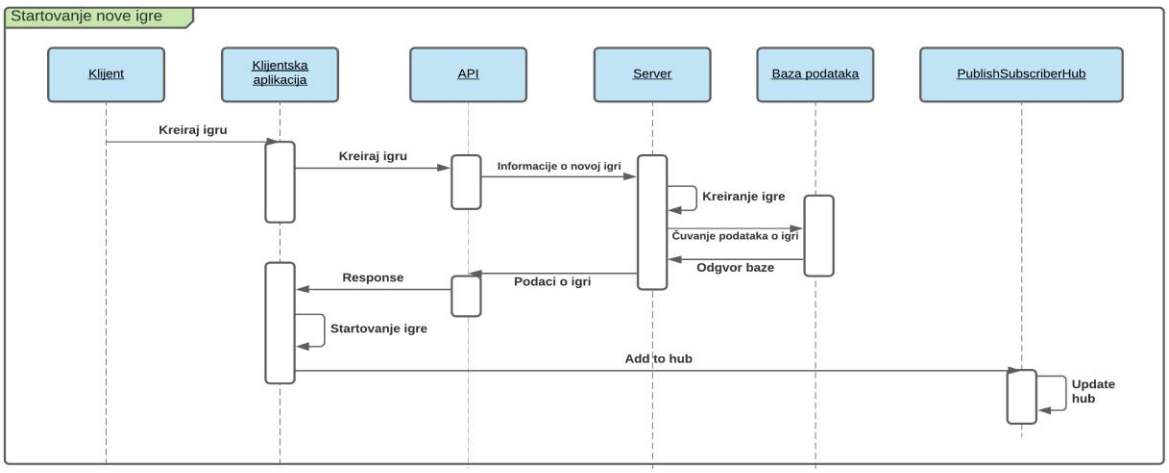
Slika generalne arhitekture

Strukturni pogledi



Bihevioralni pogledi





Implementaciona pitanja – biblioteke, komponente i okviri (framework-ci) koji će biti korišćeni za implementaciju

Od tehnologija koristiće se sledeće tehnologije:

- ***Vue.js*** na klijentskoj strani,
- ***ASP.NET Core WebAPI*** na strani servera,
- biblioteka ***SignalR*** za komunikaciju
- ***MS SQL Server*** kao relaciona baza podataka
- ***EntityFramework*** – ORM framewrok

Analiza arhitekture

Potencijalni rizici u implementaciji i strategije prevazilaženja

Neki od potencijalnih rizika i njihovih strategija prevazilaženja su:

- Zbog potrebe sistema za korišćenjem Publish-Subscribe arhitekturnog obrasca mora se koristiti neki sistem razmene poruka, pre svega od servera ka klijentu. Neki ovakvi sistemi nemaju podršku u starijim pretraživačima sa kojih se pristup aplikaciji mora predvideti. Da se ne bi javio problem nekompatibilnosti ovakvog sistema (koji najčešće koristi *WebSocket*-e) i da bi se sprečilo uvođenje dodatnih tehničkih ograničenja (u vidu obavezne upotrebe nekog operativnog sistema na serverskom računaru) koristiće se SignalR za asinhronu razmenu poruka koji podrazumevano koristi *WebSocket*-e, a ukoliko oni nisu dostupni koristi se uzastopno slanje zahteva serveru (*long polling*) čime se simulira server push funkcionalnost.

- Pošto aplikacija implementira igru, koja zahteva brzu interakciju sa serverom i može dovesti do povećanog broja korisnika, baza podataka i server koji opslužuje pozive API-ja mogu doprineti lošem korisničkom iskustvu. Ovakav problem bi mogao da se reši prelaskom na neku distribuiranu bazu podataka, kao i hostovanjem aplikacije na više čvorova.