ZAVRŠNA FAZA

PuzzleStorm

by InnoStorm

Marija Stojković 15422

Dalibor Aleksić 15024

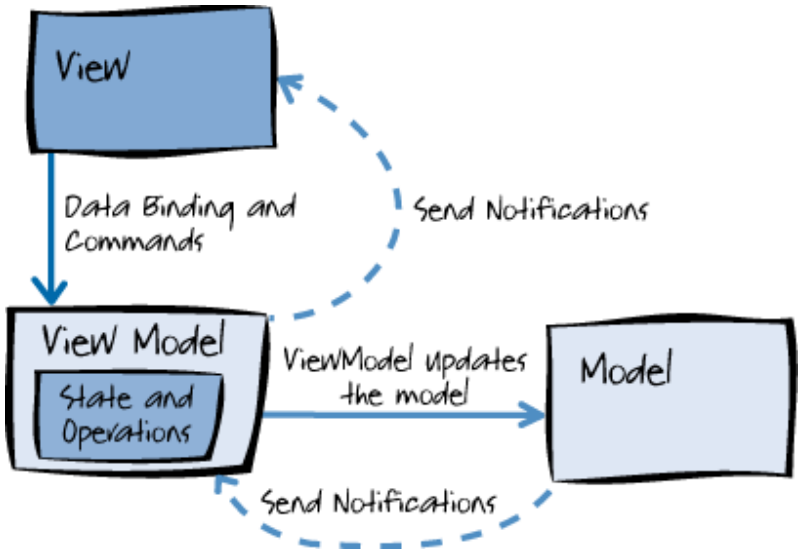
Nikola Savić 15355

Sadržaj

MVVM Obrazac

„Model-View-ViewModel“ patern se može koristiti na svim „XAML“ platformama kao što su *Windows Forms, **WPF**, Silverlight, Windows Phone*.. Njegova svrha je da obezbedi jasnu razdvojenost između kontrole korisničkog interfejsa i njegove logike.

Ovaj patern čine tri glavne komponente: model, view i view model. Sledeća slika ilustruje veze između ove tri komponente.



Da bi razumeli odgovornost svake komponente, bitno je da prvo razumemo kako komponente interaguju između sebe. Na nekom najvećem nivou, view „zna“ svoj view model, view model „zna“ model, dok obrnuto model „ne zna“ ništa o view modelima, kao što view modeli „ne znaju“ ništa o view-ovima.

Ovako razdvojene komponente nam omogućuju:

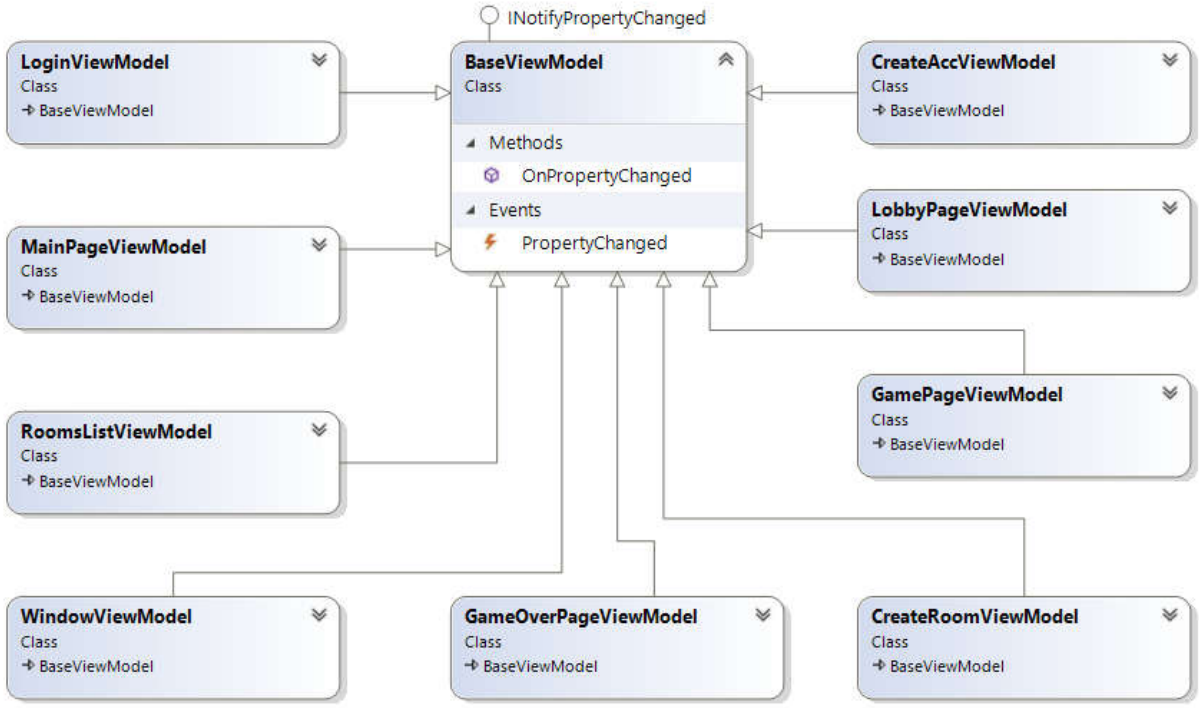
1. Rad na komponentama nezavisno (dizajner i koder mogu raditi nezavisno)
2. Interna izmena unutar jedne komponente ne utiče na ostale
3. Više view-ova može koristi jedan view model
4. Jednostavan redizajn aplikacije (nova verzija view-a ne zahteva novu verziju view modela)
5. Izolaciju komponenata prilikom testiranja (testiranje modela i view modela ne zahteva korištenje view-a)

View – Kao što se i da pretpostaviti, view je odgovoran za to šta korisnik vidi na ekranu. Po ovom paternu, idealno bi bilo da je view čist XAML, bez ikakvog koda iza. View ima svoj view model, od koga dobija podatke kroz „bindings“ (veza) ili poziva određene metode u tom view modelu. Određene metode se pozivaju setovanjem „Command property“-ja na određenu kontrolu.

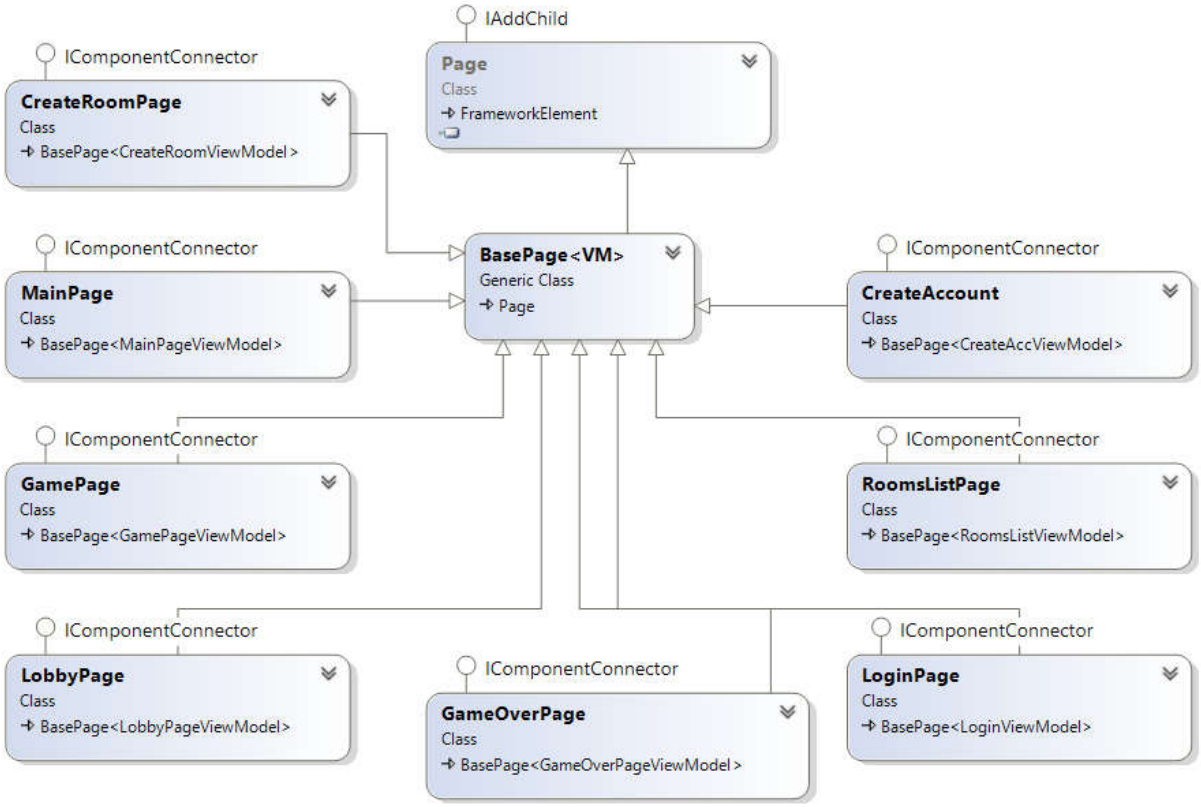
ViewModel – Ova komponenta pruža vezu između view-a i model-a. Obično, view model komunicira sa modelom tako što poziva njegove metode. Zatim, obezbeđuje da te podatke view može lako koristiti. „Property“ na koje se view „bindovao“ (povezao), mora posedovati „PropertyChanged event“ kako bi se preko njega obavestio view o promeni. U našem slučaju za „Properties“ u view modelima koristimo „ObservableCollection“.

Model – U MVVM paternu pod modelom se podrazumeva model podataka sa biznis logikom, kao i validacionom logikom. Kako se kod nas sva logika dešava na serveru, za model su korišćeni „data transfer“ objekti (DTOs) iz posebne biblioteke u projektu, pod nazivom „DTOLibrary“.

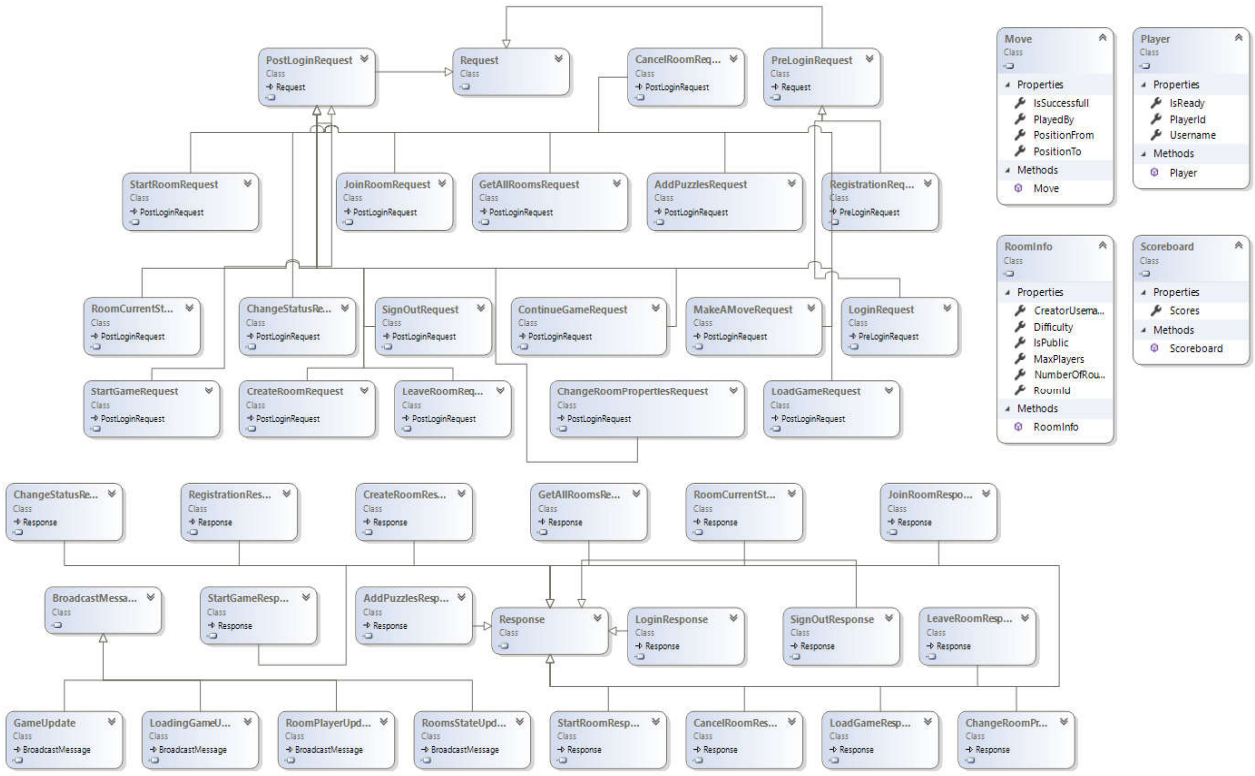
UML klasni dijagram koji ilustruje korišćenje MVVM paterna na našem projektu dat je na sledećim slikama.



ViewModels



Svaka View klasa (Page) se izvodi iz „BasePage“ gde se za „DataContext“ stavlja određen ViewModel (prosleđen generički kao VM)



Model iz DTOLibrary

Mikroservis struktura

Mikroservisi su varijanta servisno-orjentisano arhitekturnog stila koji struktuiraju aplikaciju kao kolekciju slabo spregnutih servisa. Prednost rastavljanja aplikacije na male servise je u tome što se povećava modularnost, aplikaciju je lakse razumeti, razvijati I testirati. Takođe omogućava paralelni razvoj dozvoljavajući malim autonomnim timovima da rade nezavisno. Refaktorisanje jednog od tih servisa i njegov razvoj nisu zavisni od ostalih servisa.

U našem projektu imamo tri tipa servisa:

1. Za autentifikaciju
2. Za upravljanje sobama i dešavanjima unutar njih
3. Za instanciranje igara i upravljanje istih

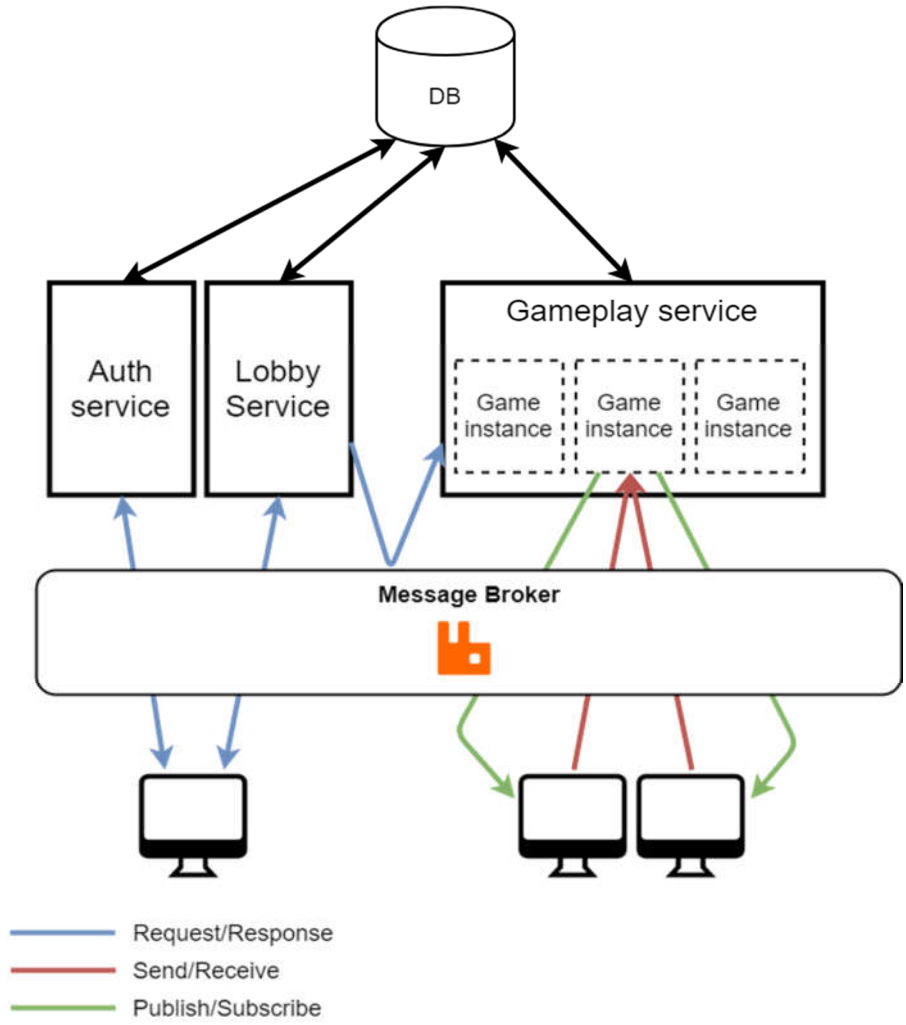
Autentifikacioni servis omogućava logovanje korisnika i kreiranje novih naloga.

Servis za upravljanje sobama se bavi kreiranjem, brisanjem i modifikacijom. Pored toga, bavi se i dešavanjima unutar istih (ulazak u sobu, spremnost za igru i napuštanje sobe).

Servis za igre se bavi instanciranjem nove igre, dodavanje igre sobi i tokom igre (povlačenjem poteza, obeveštavanjem da li je potez uspešan, menjanjem igrača..).

Svrha ovog paterna je da su ovi servisi potpuno nezavisni, tako da možemo instancirati više servisa istog tipa (na primer: više servisa za samu igru, a manje servisa za autentifikaciju). Time dobijamo da ukoliko svi servisi nekog tipa padnu, taj pad ne utiče na servise drugog tipa, yatim opterećenje jednog tipa servisa ne utiče na rad drugog itd..

Za autentifikaciju, upravljanje sobama i dešavanjima unutar njih komunikacija se vrši putem „Request-response“ obrasca, dok se tok igre odvija slanjem poteza Send (Send-receive obrazac). Nakon što je server primio poruku (potez), ostalim učesnicima igre dostavljaju se promene u igri slanjem Publish poruka na koje su oni Subscribe-ovani (Publish-subscribe obrazac).



Message broker

„Message broker“ je arhitekturni obrazac za validaciju, transformaciju i rutiranje podataka. Posreduje u komunikaciji između aplikacija smanjujući svesnost koju aplikacije treba da imaju jedna o drugoj kako bi razmenjivale poruke, implementirajući razdvojenost (nezavisnost između modula).

Svrha borkera je primanje dolaznih poruka od aplikacija i izvdršavanje akcije nad njima. Primer akcija koje mogu biti preuzete od strane brokera:

1. usmeravanje poruka sa jednog na jedno ili više odredišta
2. transformisanje poruka u alternativni oblik
3. agregacija poruka, dekomponovanje poruka na više poruka i njihovo slanje na odredište, zatim rekomponovanje odgovora u jednu poruku koja će biti vraćena korisniku
4. interakcija sa eksternim repozitorijumom radi čuvanja poruka
5. pozivanje web servisa radi preuzimanja podataka
6. odgovaranje na događaje i greške
7. obezbeđivanje sadržaja i rutiranje poruka zasnovano na redovima korišćenjem „publish-subscribe“ obrasca

Message broker koji se koristi u našem projektu je „RabbitMQ“. Hostovanje servera se vrši na „CloudAMQP“ platformi. Korišćenje message brokera se vrši preko .NET API-ja pod nazivom „EasyNetQ“.

Obrasci koji su pokriveni ovim API-jem su:

1. *Publish-subscribe*
2. *Request-response*
3. *Send-receive*

Publish-subscribe

Ovo je obrazac za manipulaciju porukama gde pošiljaoci poruka poznatiji kao „publishers“, nemaju znanje o načinu na koji će poruke biti poslate krajnjim primaocima poruka poznati kao „subscribe“-eri. Umesto toga poruke se pakuju u objekte određene klase bez znanja o primaocima ako ih uopšte ima. Slično, „subscribe“-eri su zainteresovani za objekte jedne ili više klasa i mogu da primaju poruke za koje su zainteresovani bez znanja o „publisher“-ima, ako ih uopšte ima.

Ovaj obrazac omogućava veliku skalabilnost i mnogo dinamičniju topologiju mreže.

Request-response

Obrazac za razmenu poruka u kome zahtevalac šalje zahtev sistemu za odgovore koji prima i procesira zahteve i vraća odgovor. Ovo je jednostavn ali moćan obrazac za razmenu poruka koji dozvoljava dvema aplikacijama da imaju dvosmernu komunikaciju jedan sa drugim kroz kanal.

Send-receive

U obrascima „Publish-subscribe“ i „Request-response“ nije potrebno specificirati gde se nalazi primalac poruke, u „Send-receive“ obrascu komunikacija se vrši kroz redove. Nije potrebno unapred znati tip poruke koja će se slati kroz red, što znači da se može slati više tipova poruka kroz isti red.

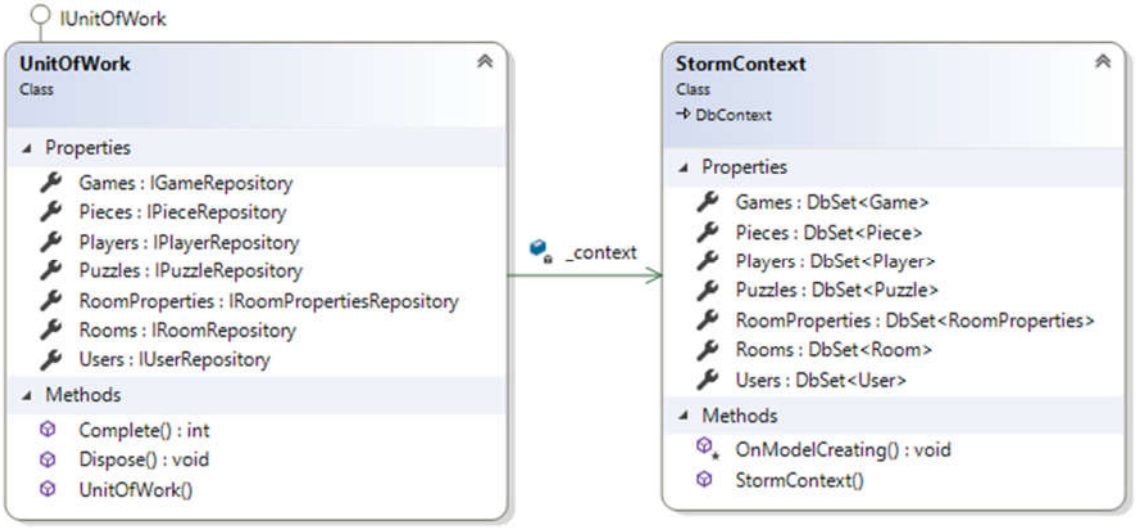
Repository i Unit of work obrasci

Unit of work obrazac se koristi da grupiše jednu ili više operacija (operacija za rad sa bazom) u jedinstvenu transakciju ili „unit of work“, tako da ili sve operacije budu uspešne ili nijedna.

Repository obrazac se koristi da upravlja CRUD operacijama kroz apstraktni interfejs koji u prvi plan dovodi domenske klase, a sakriva detalje implementacije koda koji pristupa bazi.

Za pristup podacima su napravljene Repository klase date na sledećoj slici.

Ove klase zajedno sa kontekstom baze koristi UnitOfWork, pa se preko njega pristupa podacima.



Ostali obrasci

Na kraju, ali ne i najmanje važni, u projektu se koriste „Observer“ i „Singleton“ obrasci koji su uobičajeni u ovakvim višekorisničkim kolaborativnim distribuiranim aplikacijama, stoga se neće detaljno razmatrati.