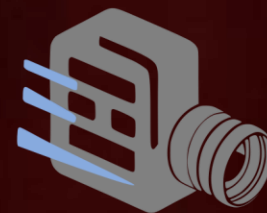




Univerzitet u Sarajevu
Elektrotehnički fakultet
Sarajevo



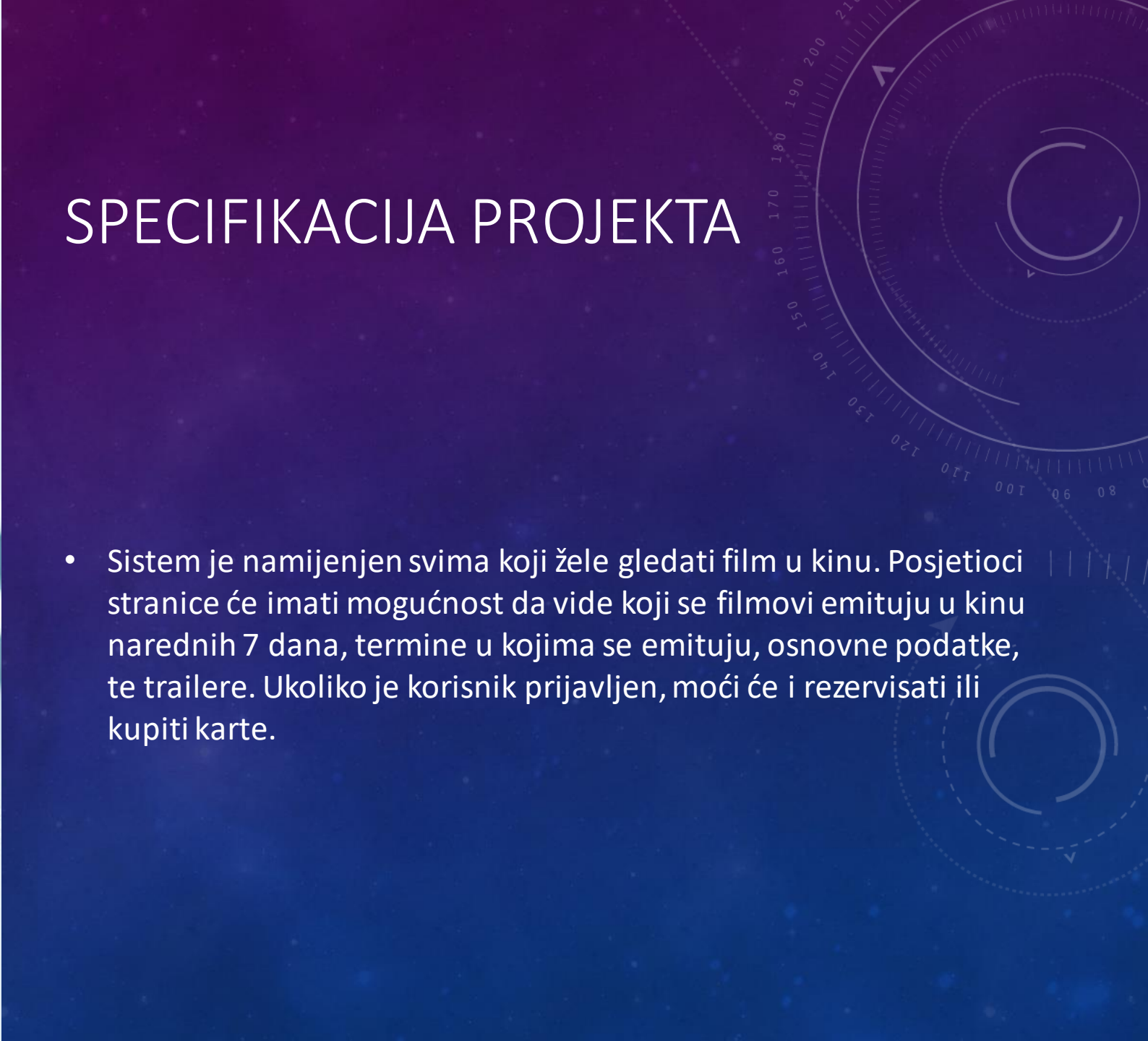
CINETECH


Članovi ekipe:

- Adi Drakovac
- Adnan Dervišević
- Arman Bašović
- Benjamin Hadžihasa



SPECIFIKACIJA PROJEKTA

- Sistem je namijenjen svima koji žele gledati film u kinu. Posjetioci stranice će imati mogućnost da vide koji se filmovi emituju u kinu narednih 7 dana, termine u kojima se emituju, osnovne podatke, te trailere. Ukoliko je korisnik prijavljen, moći će i rezervirati ili kupiti karte.
- 



FUNKCIONALNOSTI SISTEMA

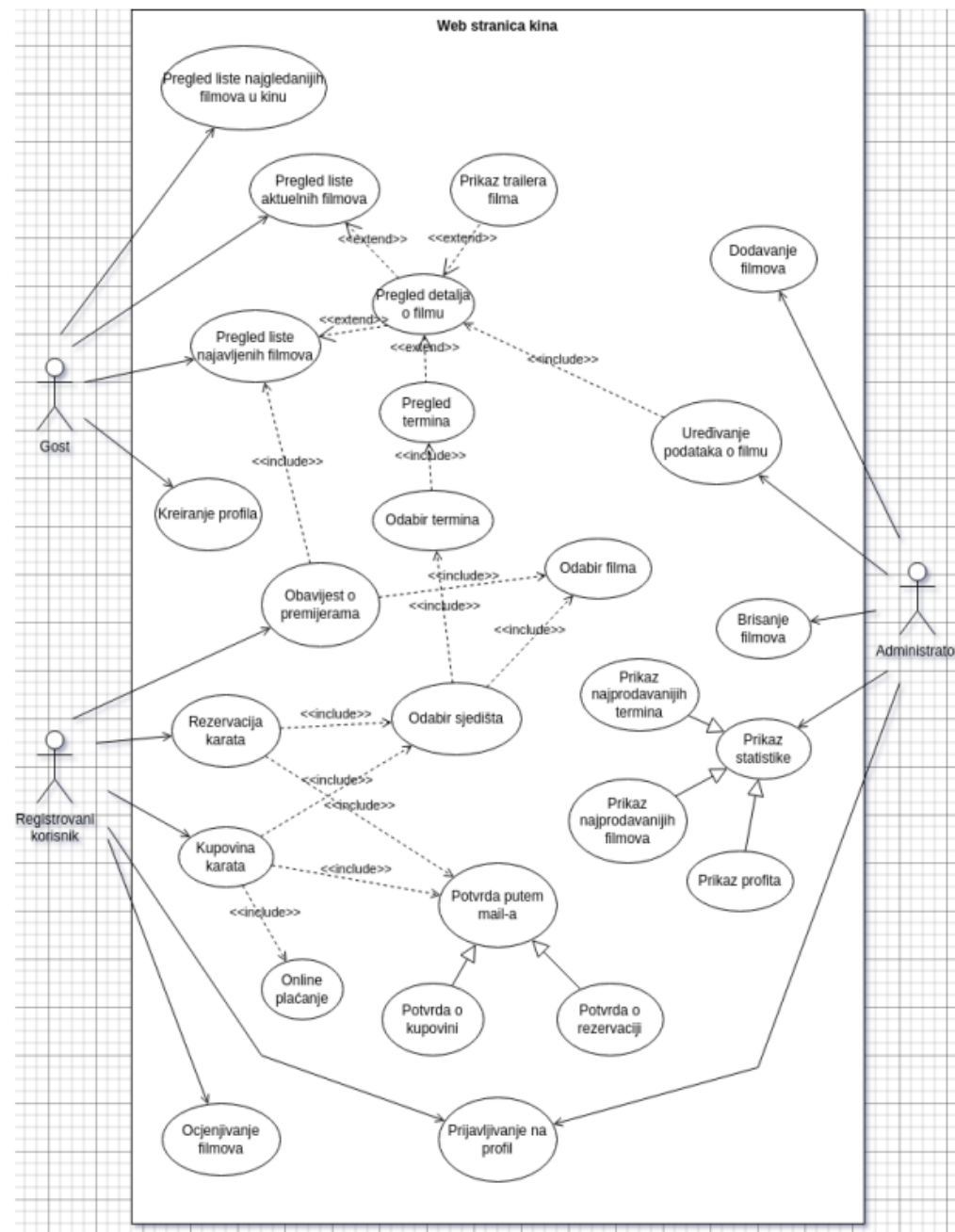
- **Korisnici sistema imaju mogućnosti:**
 - Pregled aktuelnih filmova i informacija o njima
 - Ocjenjivanje i komentarisanje filmova
 - Pregled najgledanijih filmova u kinu pa i svijetu
 - Pregled najavljenih filmova koji još nisu izašli

AKTERI

- U našem sistemu postoje sljedeći akteri:
- -Gost
- -Registriran Korisnik
- -Administrator

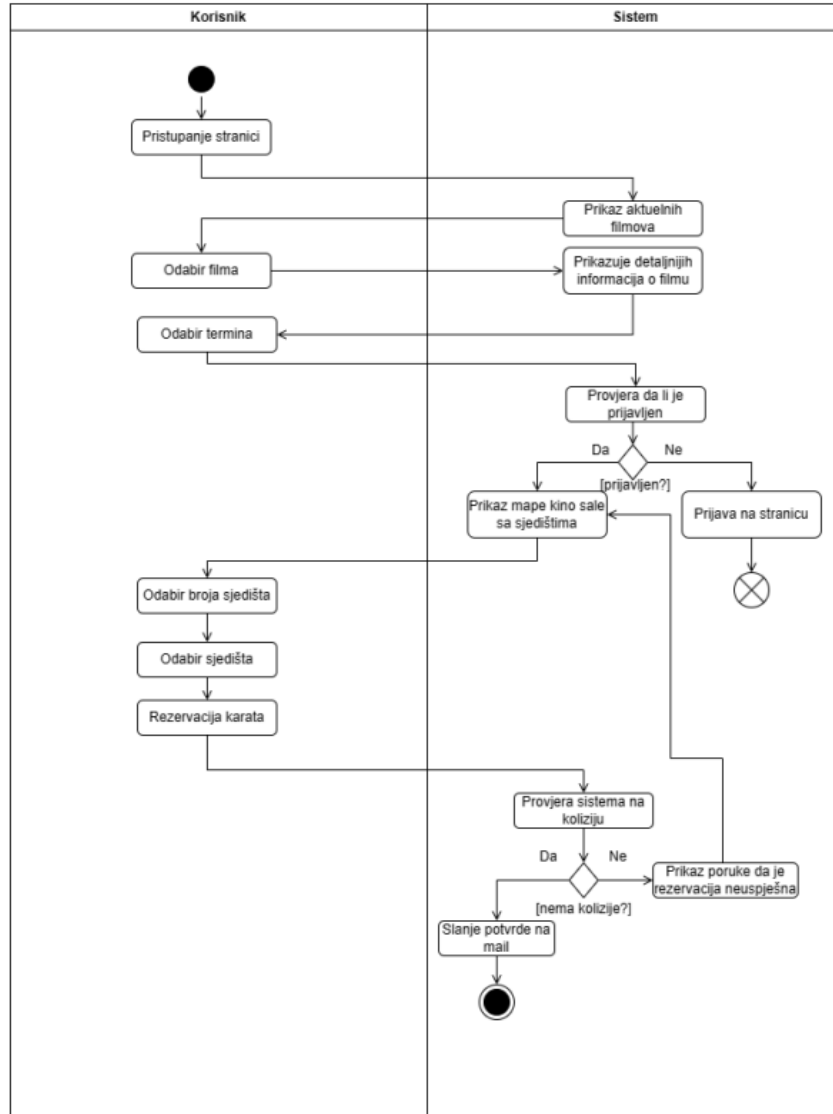


USE-CASE DIJAGRAM

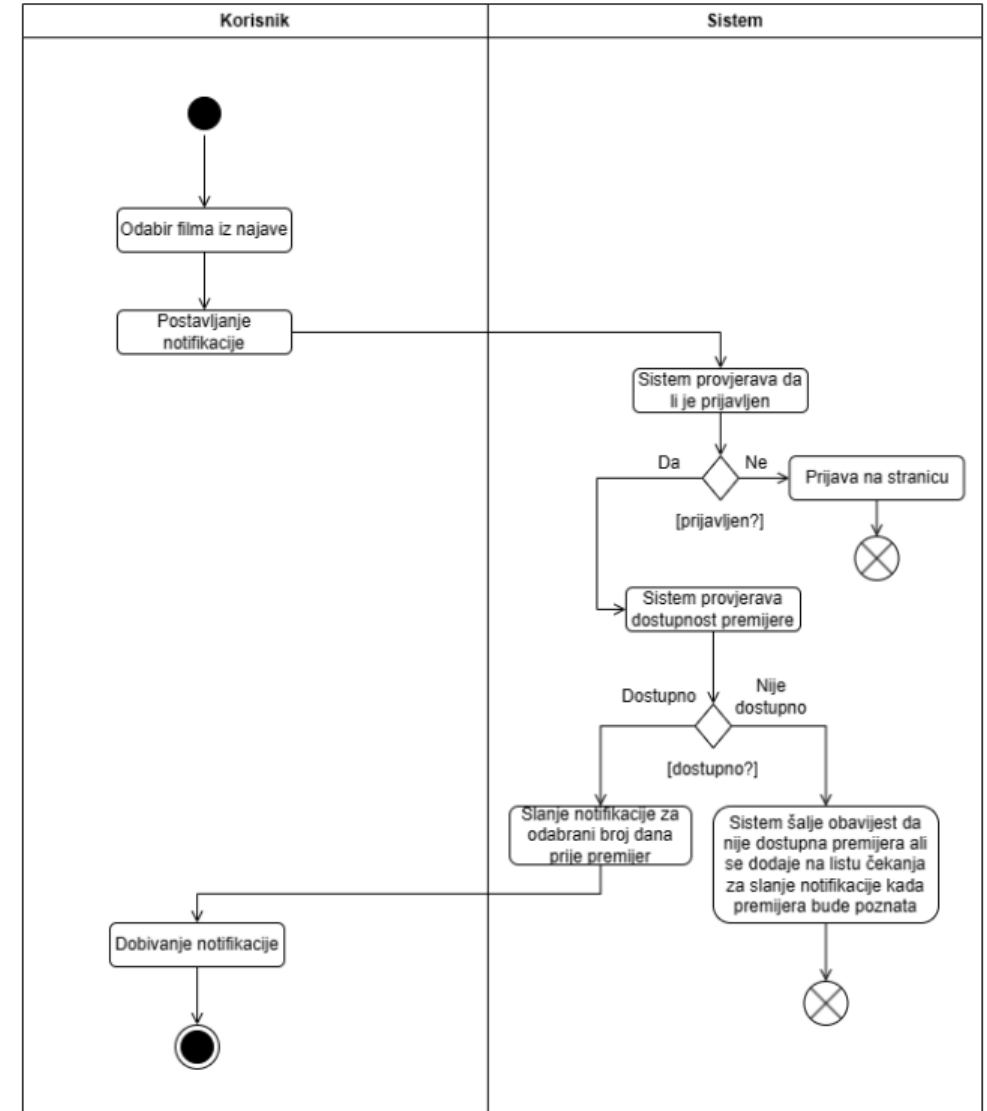


DIJAGRAM AKTIVNOSTI

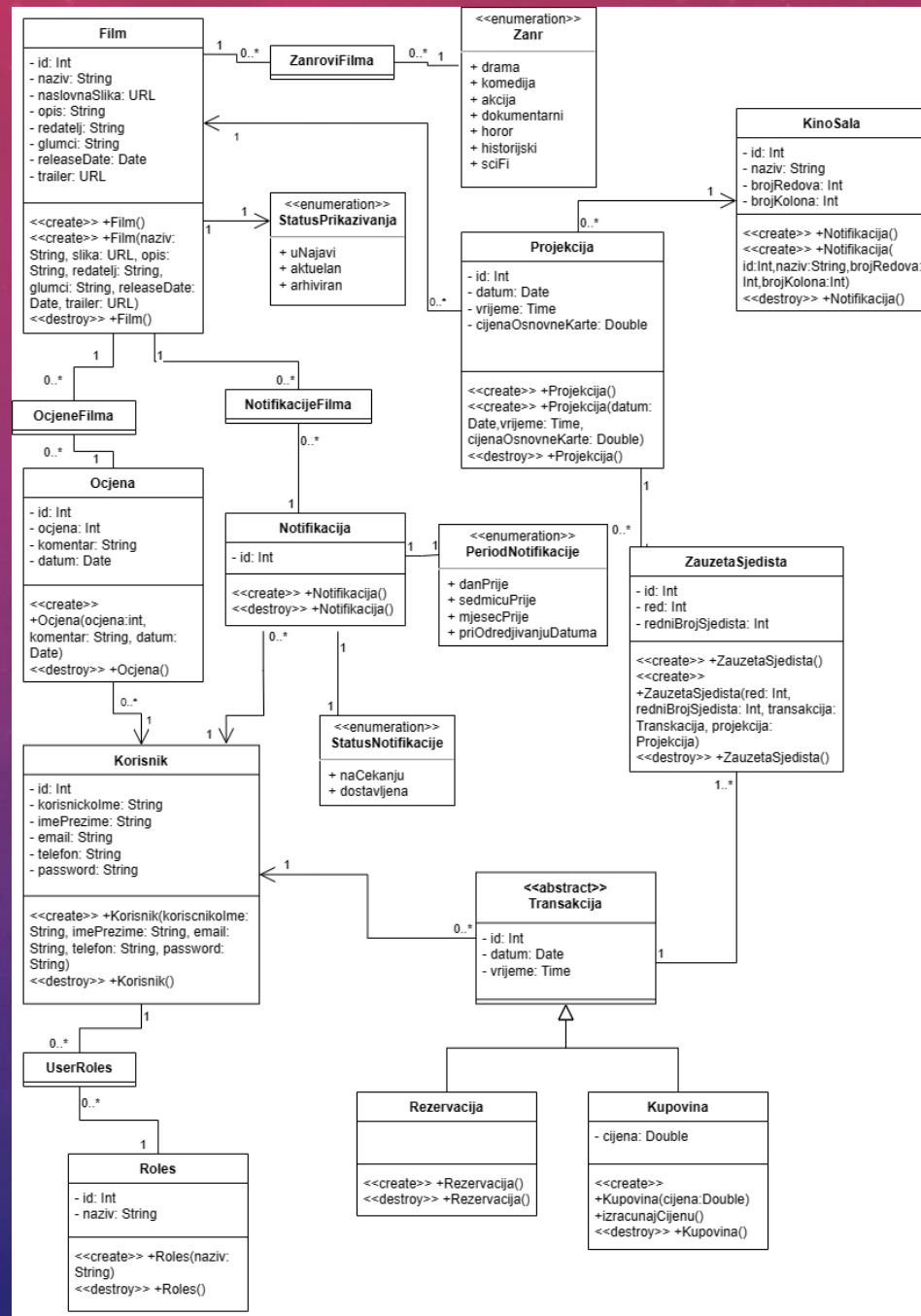
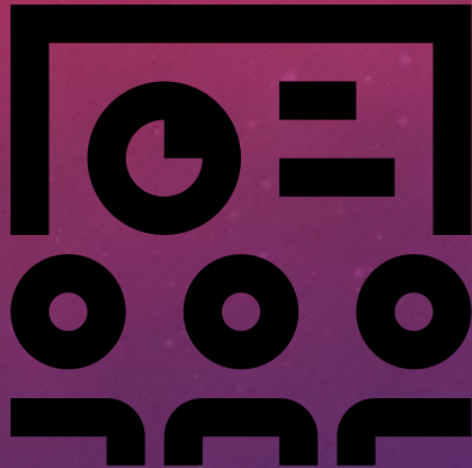
Scenario 3: Rezervacija karata



Scenario 8: Notifikacije o premijeri



DIJAGRAM KLASA



SOLID PRINCIPI

- **1. Single Responsibility Principle** - Princip pojedinačne odgovornosti Svaka klasa bi trebala imati samo jednu odgovornost Ovaj princip je ispoštovan jer sve klase imaju pojedinačne odgovornosti, odnosno svaki posao se veže za jednu klasu. Ako posmatramo klasu Korisnik, možemo primijetiti da ona posjeduje samo one atribute koje svaki korisnik mora imati, te će klasa posjedovati samo one metode koje će moći vraćati i modifikovati te atribute, klasa Film također samo nosi informacije o filmu, kao i metode za pristup istima, a tako i sve ostale klase.
- **2. Open Closed Principle** - Otvoreno zatvoren princip Klase bi trebale biti otvorene za proširenje, ali zatvorene za modifikaciju, odnosno da dodavanje novih atributa i funkcionalnosti jednoj klasi ne izaziva potrebu za izmjenama na nekim drugim klasama s kojima je prva klasa povezana

SOLID PRINCIPI

- **3. Liskov Substitution Principle** -Liskov princip zamjene Podtipovi moraju biti zamjenljivi svojim osnovnim tipovima bez uticaja na korektnost programa, Ovaj princip je zadovoljen jer nema nasljeđivanja
- **4. Interface Segregation Principle** -Princip izoliranja interfejsa Princip izoliranja interfejsa naglašava da korisnici ne bi trebali biti prisiljeni da zavise o interfejsima koje ne koriste. Princip nije narušen jer u modelu nemamo interfejsa

SOLID PRINCIPI

- **5. Dependency Inversion Principle** - Princip inverzije ovisnosti Princip inverzije ovisnosti kaže da ne treba ovisiti od konkretnih klasa. Princip je zadovoljen jer klase zavise od apstraktnih klasa. Imamo apstraktnu klasu transakcija koja veže korisnika sa rezervacijom ili kupovinom u ovisnosti od zauzetosti sjedišta.

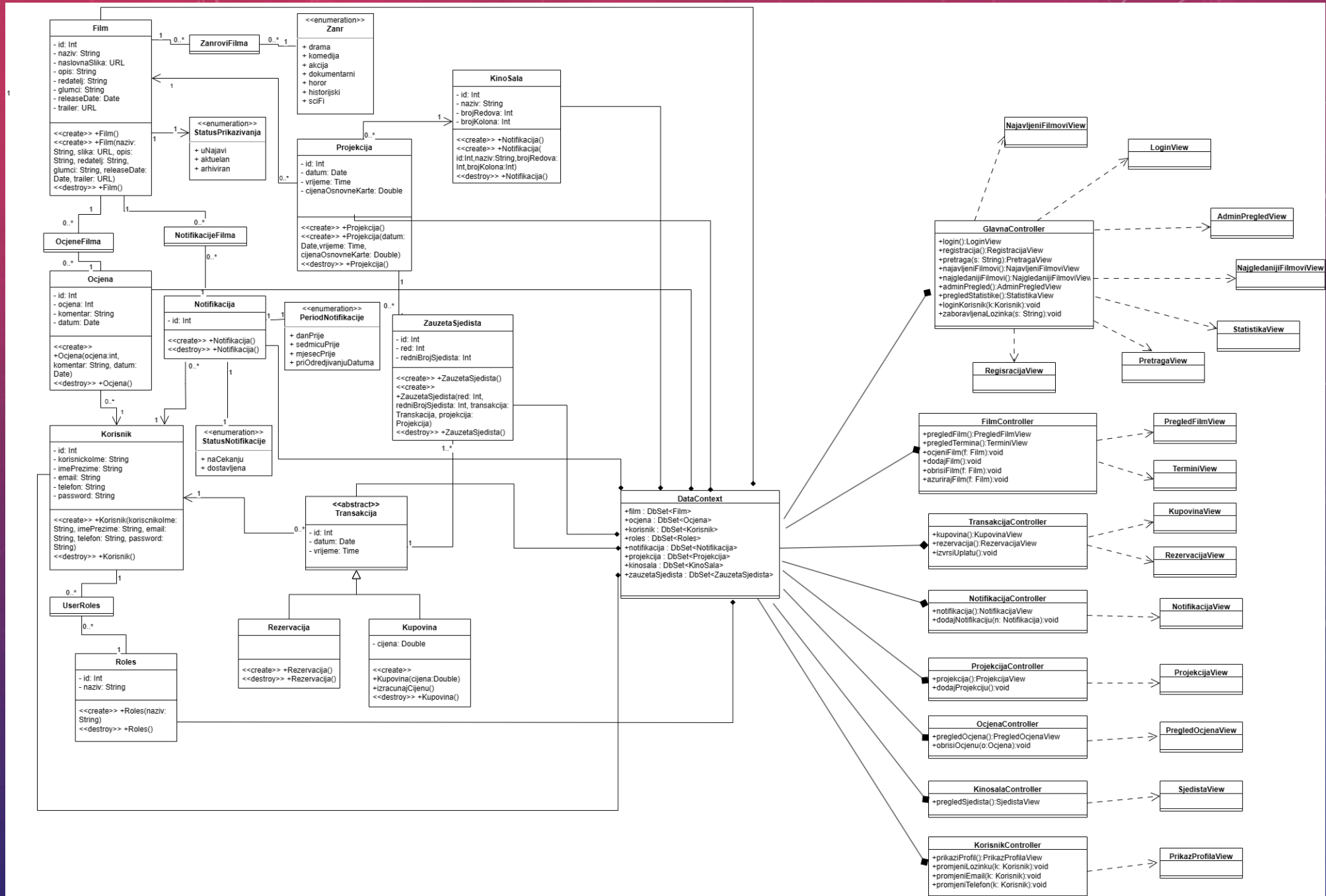
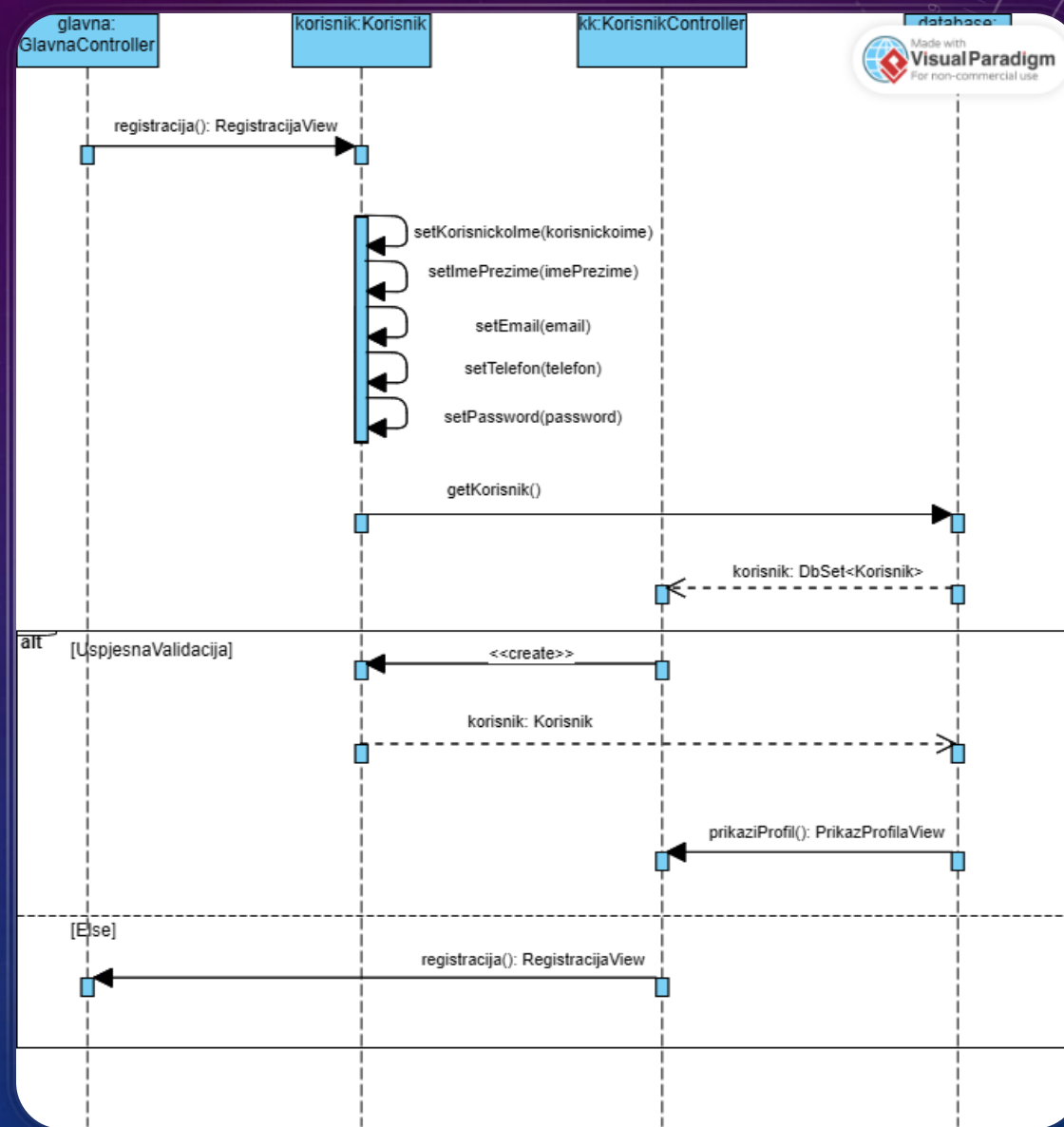
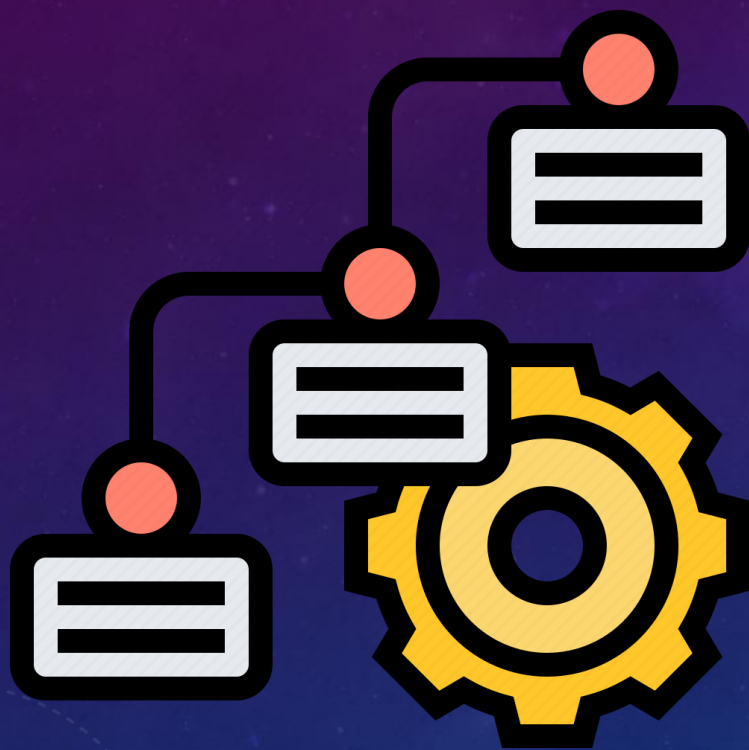


DIAGRAM SEKVENCI

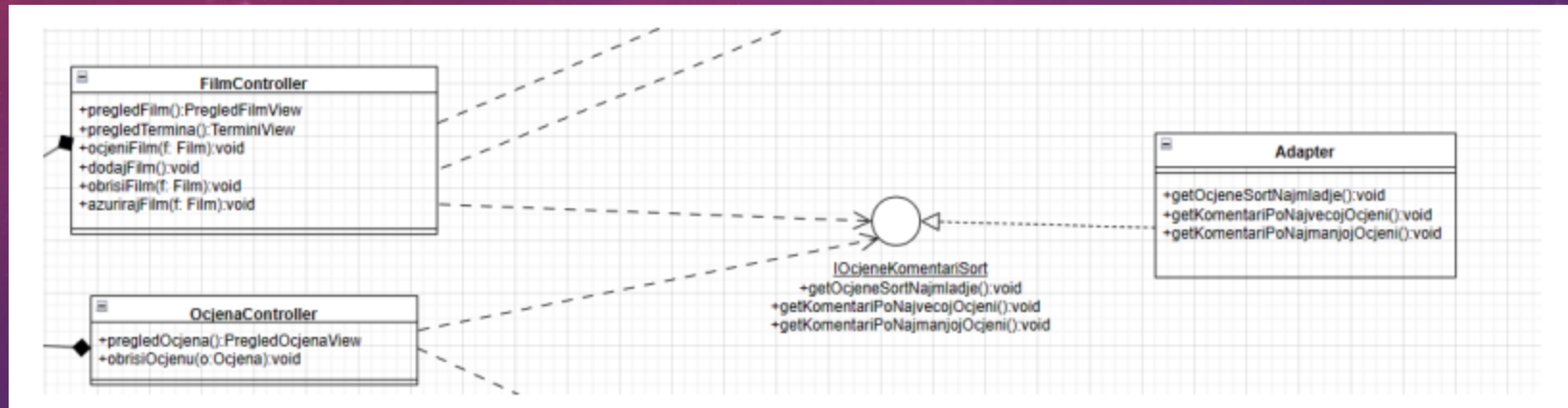


STRUKTURALNI PATTERNI



Adapter pattern:

- Svrha Adapter patern-a je da omogući širu upotrebu već postojećih klasa. U našem sistemu možemo nadograditi da korisniku omogućimo sortiranje ocjena po visini ocjena i u nekim slučajevima ovisnosti i od komentara. Kreirat ćemo interfejs “IOcjeneKomentariSort” koji će posjedovati metode: `getOcjeneSort Najmladje()`, `getKomentariPoNajvecojOcjeni()`, `getKomentariPoNajmanjojOcjeni()`. Klasa Adapter implementira sve nabrojanje metode iznad



Facade pattern:

- Fasadni pattern služi kako bi se korisnicima pojednostavilo korištenje kompleksnih sistema, odnosno koristimo kada imamo neki sistem koji ne želimo da razumijemo kako funkcioniše “ispod haube”. Ovaj pattern nećemo implementirati, ali ako bi smo htjeli to bi smo mogli na način da pretpostavimo da u našem sistemu imamo jasno definisane podsisteme, npr podsisteme za ocjenjivanje i komentarisanje filmova, za pretraživanje filmova, za kupovinu karata... U slučaju da imamo veliki broj ovakvih podsistema, kao i dovoljnu raznolikost među korisnicima da im omogućimo različite poglede na sistem, mogli bismo napraviti klasu Fasada koja bi objedinila sve ove podsisteme i koja bi pružala jedinstven interfejs u ovisnosti o kojem se klijentu radi. Međutim mi nemamo dovoljno funkcionalnosti da bismo bili u potrebi da koristimo ovaj pattern

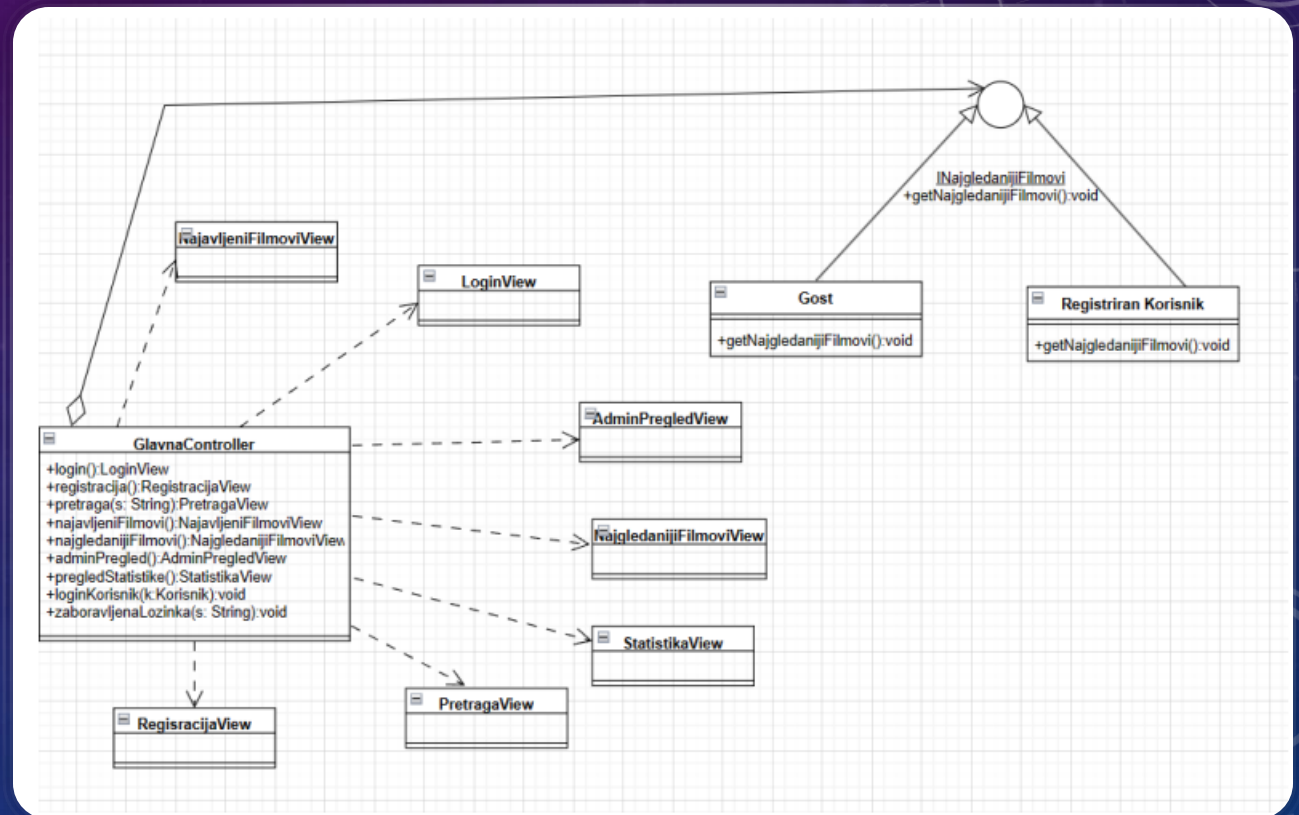
Decorator pattern:

- Decorator pattern služi za omogućavanja različitih nadogradnji objektima koji svi u osnovi predstavljaju jednu vrstu objekta. U našem sistemu mogli bismo primijeniti decorator pattern na sljedeći način: - Pretpostavimo da smo uveli atribut slika u klasi Korisnik. To znači da svi prijavljeni imaju sliku na svom profilu. - Uvedimo interfejs ISlika sa metodom edituj() - Uvest ćemo također dvije klase: SlikaFilteri i SlikeOsnovnePromjene. Ove klase će posjedovati atribut tipa Korisnik i svaka će na drugačiji način uređivati njegovu sliku pozivajući metodu interfejsa. Obavili smo dinamičko dodavanje novog elementa i ponašanja postojećem objektu dijagrama klasa. Objekat pri tome ne zna da je urađena dekoracija što je veoma korisno za ponovnu upotrebu komponenti softverskog sistema.



Bridge pattern:

- Osnovna namjena Bridge paterna je da omogući odvajanje apstrakcije i implementacije neke klase tako da ta klasa može posjedovati više različitih apstrakcija i više različitih implementacija za pojedine apstrakcije.
- Ovaj pattern bi mogao biti iskorišten ako bismo htjeli da realizujemo različit način prikazivanja filmova. Na primjer, korisnik i gost na isti način pretražuju filmove. Međutim, razlika u načinu prikazivanja najgledanijih filmova je u tome što korisniku filmovi budu sortirani i uz to prikazani sitni note koji naglašava ukoliko je on pogledao taj film (kupio kartu ili rezervisao kartu za taj film), dok kod gosta to ne može.
- Ovo bismo implementovali na sljedeći način: - Uveli interfejs `INajgledanijiFilmovi` sa metodom `getFilmovi()` - Dodali klasu Bridge koja vraća listu Filmova u formatu koji je isti i za gosta i korisnika - Interfejs pozivaju dvije klase koje na različite načine sortiraju filmove. Jedna klasa se odnosi na način sortiranja filmova kod korisnika, a druga kod gost



Composite pattern:

- Composite pattern koristimo da omogućimo formiranje strukture stabla pomoću klasa, u kojoj se individualni objekti (listovi stabla) i kompozicije individualnih objekata (korijeni stabla) jednako tretiraju. U našem slučaju to bi mogli biti obični korisnik i VIP korisnik kada bi smo ga implementirali, koji bi imali posebne pogodnosti kao što je popust prilikom kupovine karata, neki pokloni... Iako su ova dva korisnika na “različitim nivoima”, pristupalo bi im se na isti način i implementacija bi bila pojednostavljena. Tada je potrebno napraviti interfejs IPristup koji definira zajedničke operacije za objekte, te Composite klasu koja će sadržavati listu objekata tipa IPristup. Klase korisnik i VIP korisnik će implementirati na drugačiji način metodu interfejsa.

Proxy pattern:

- Svrha Proxy patern-a je da omogući pristup i kontrolu pristupa stvarnim objektima. Proxy je obično mali javni surogat objekat koji predstavlja kompleksni objekat čija aktivizacija se postiže na osnovu postavljenih pravila. Ovaj pattern možemo primijeniti u našem sistemu tako što ćemo postaviti kontrolu pisanja komentara. Jedino korisnik aplikacije može pisati komentare, dok osoba koja je prijavljena ko gost nema tu mogućnost.





Flyweight pattern:


- Ovaj pattern se koristi kada kreiramo objekte samo po potrebi kada imaju različito specifično stanje, a osnovno stanje je isto za sve objekte. Naš sistem je moguće proširiti na način da se omogući svim korisnicima aplikacije da imaju svoju profilnu sliku. U tom slučaju bilo bi potrebno koristiti unaprijed zadanu sliku, ako korisnik ne želi da prikazuje vlastitu. S obzirom da je moguće da više korisnika zadrži tu default-nu sliku potrebno je implementirati flyweight pattern kako bi ti korisnici koristili jedan zajednički resurs. To možemo postići kreiranjem interfejsa ISlika sa metodom `dajSliku()`, koji će nam vratiti sliku korisnika. Taj interfejs će implementirati klase `Slika` (čuva individualnu sliku korisnika) i `DefaultSlika` (čuva defaultnu sliku i potrebna je samo jedna njena instanca)

PROTOTIP KORISNIČKOG INTERFEJSA



PROTOTIP KORISNIČKOG INTERFEJSA

**CINETECH** [My profile](#)



O vama

Ime
Adi

Prezime
Drakovac

Broj telefona
062009537

E-mail
adrakovac1@etf.unsa.ba

Uredi profil

Unesite sve podatke koje želite da promijenite
(ukoliko ne želite da promijenite ostavite prazno)

Nova email adresa

Novi broj telefona

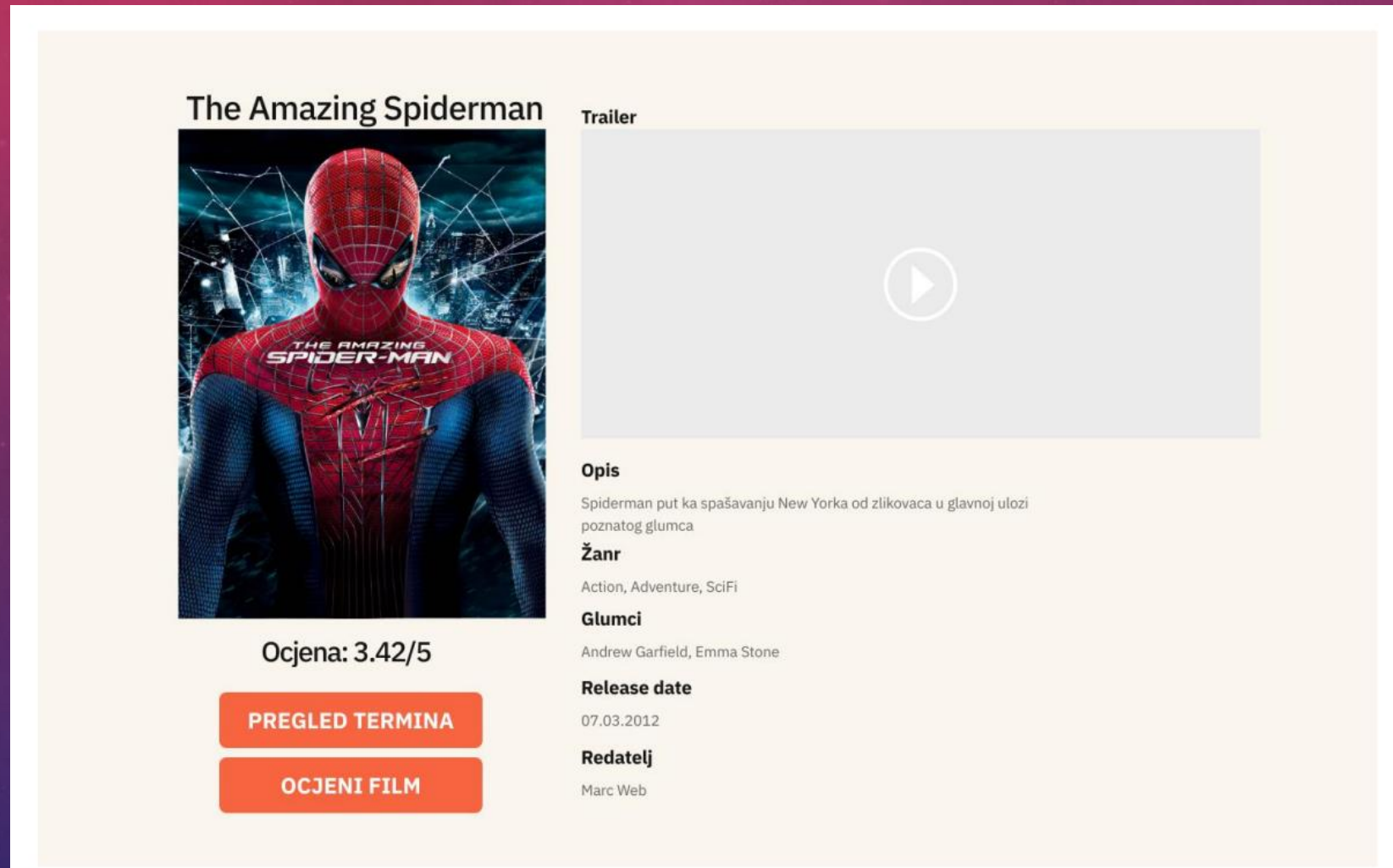
Novi password

Potvrdite da ste vi unesite trenutnu šifru

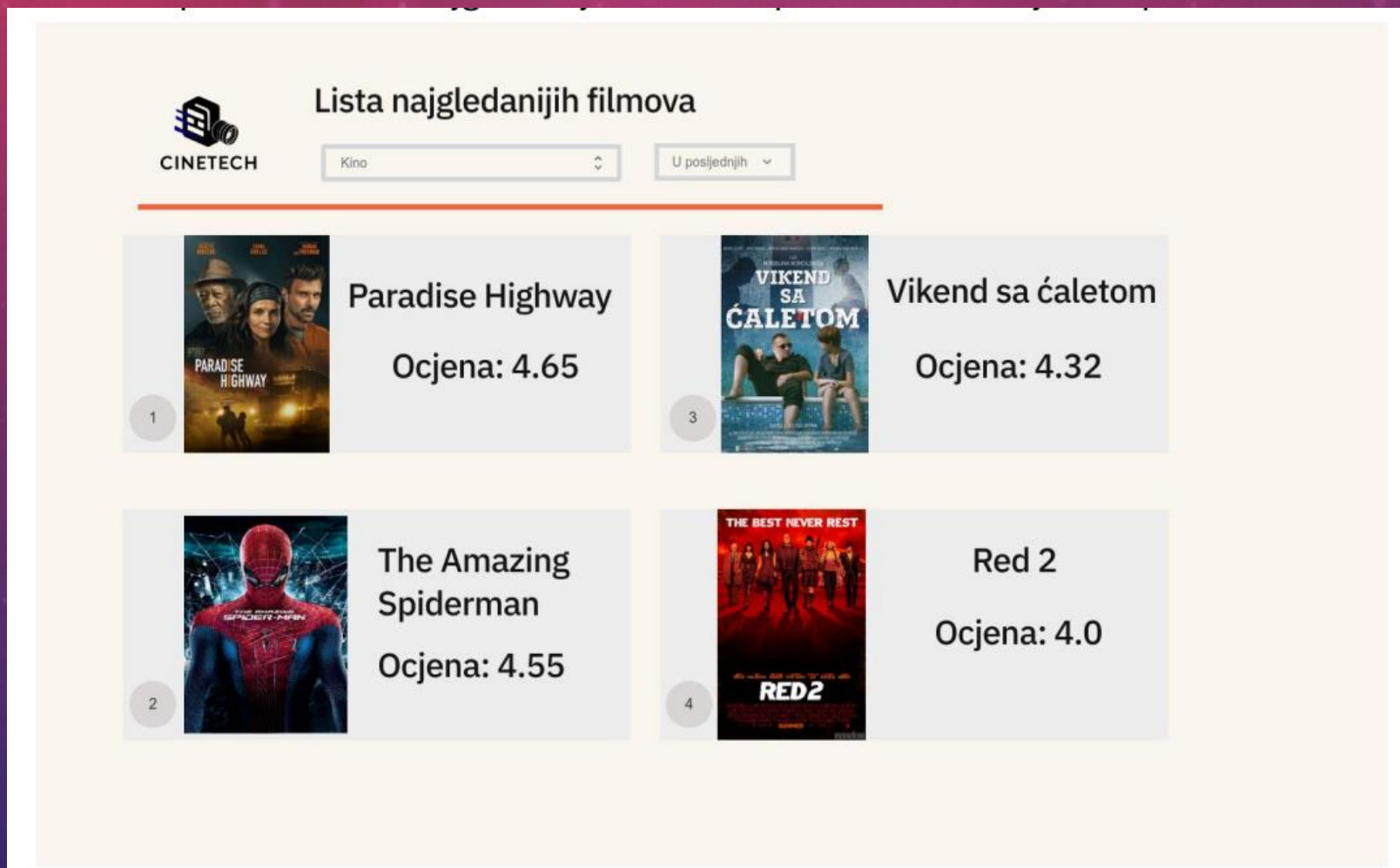
Password

Promijeni podatke

PROTOTIP KORISNIČKOG INTERFEJSA




PROTOTIP KORISNIČKOG INTERFEJSA








PROTOTIP KORISNIČKOG INTERFEJSA

OCJENE I KOMENTARI DRUGI KORISNIKA

The Amazing Spiderman




VašeKorisničkoIme
Ocjena:




Komentar...


POŠALJI



Username1
Ostavljena ocjena: 5
Komentar: Nije ostavljen komentar

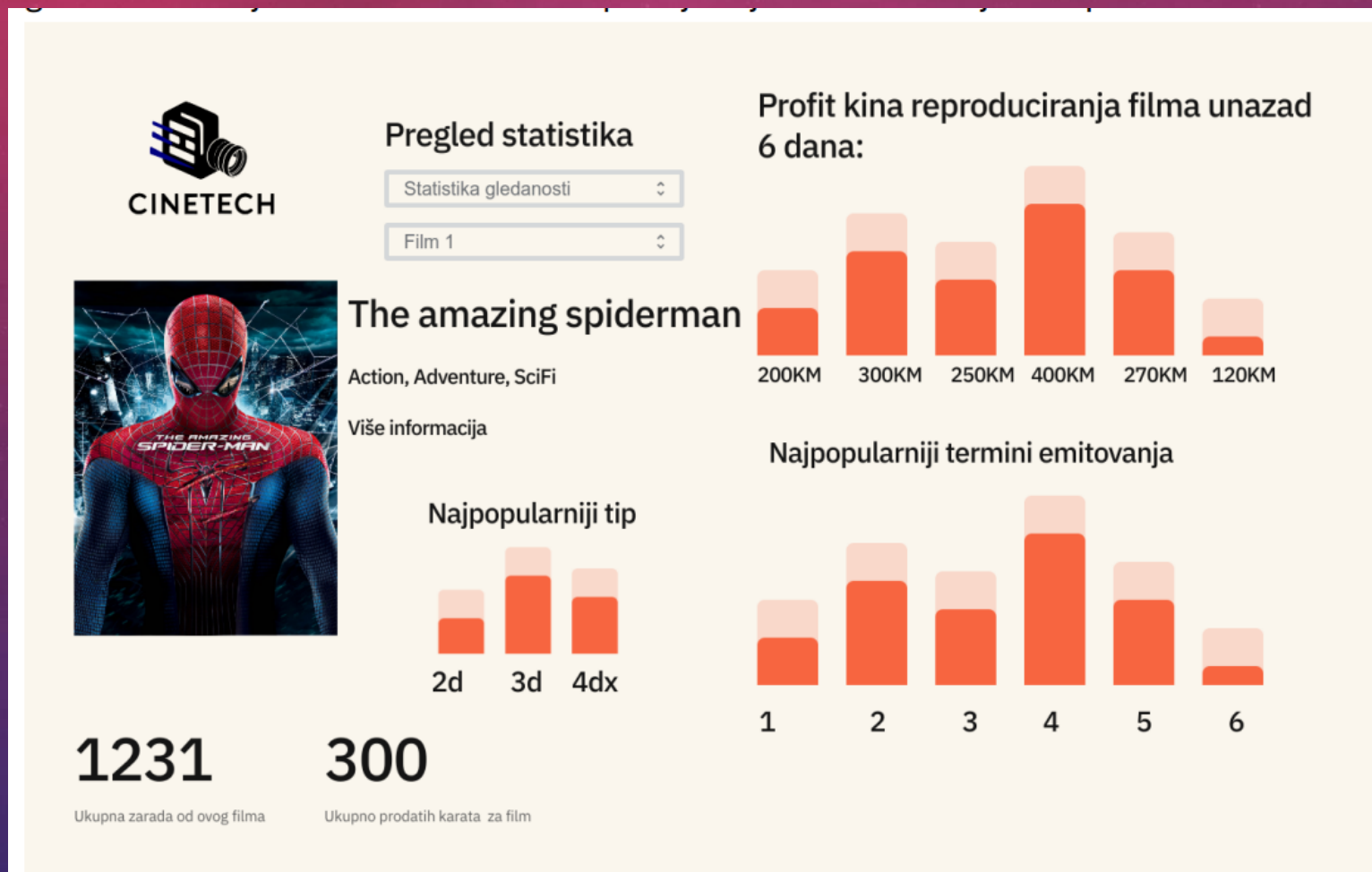


Username2
Ostavljena ocjena: 3
Komentar: Onako, moglo je bolje




Username3
Ostavljena ocjena: 5
Komentar: Sve top

PROTOTIP KORISNIČKOG INTERFEJSA





PROTOTIP KORISNIČKOG INTERFEJSA



CINETECH

DODAVANJE FILMA

Naziv filma <input type="text" value="Naziv"/>	Glumci <input type="text" value="Žanr"/>
Redatelj <input type="text" value="Redatelj"/>	Opis filma <input type="text" value="Opis filma"/>
Žanr <input type="text" value="Žanr"/>	
Release date <input type="text" value="Release date"/>	
Ubaci naslovnu sliku 	Ubaci trailer 

[Dodaj film](#)

KREACIJSKI PATTERN

Singleton pattern:

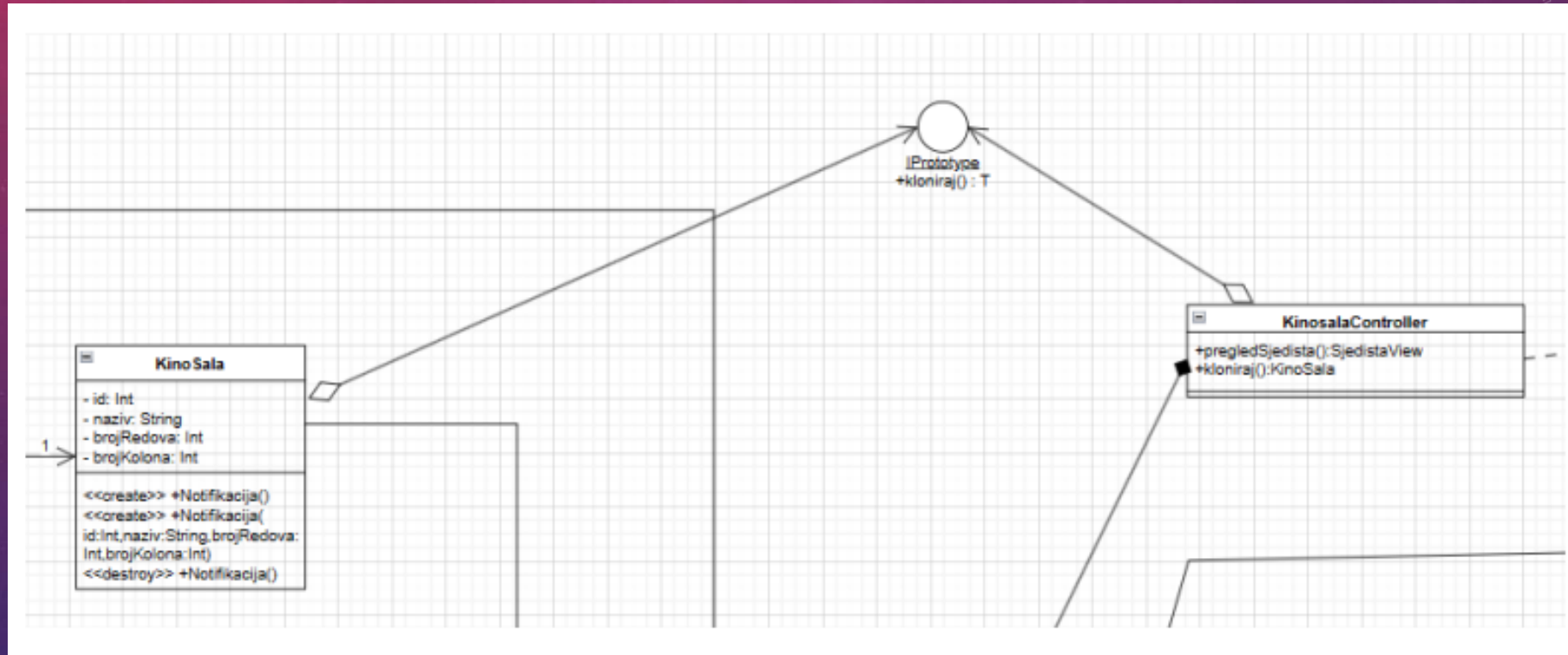
- Uloga Singleton patterna je da osigura da se klasa može instancirati samo jednom i da osigura globalni pristup kreiranoj instanci klase. Ovo bi mogli izvesti kada bi posjedovali klasu Log u kojoj bi smjestili sve ocjene, komentare, informacije o dodanim filmovima, maknutim filmovima, administratorske bilješke te registracije korisnika ujedno i bilježenje login korisnika. Ovo bi se smještalo u neku tekstualnu datoteku te istu datoteku bi mogli očistiti nakon nekog perioda.

Factory method pattern:

- Factory method pattern omogućava kreiranje objekata na način da podklase odluče koju klasu instancirati. Ovo se radi preko interfejsa sa jednom metodom koju različite podklase mogu implementirati drugačije. Kada bi smo u našoj aplikaciji imali neku vrstu mogućnosti oglasa, kojih bi mogli generisati. Imali bi smo klasu kao Kreator koja bi imala opciju PokreniOglase() onda bi imali klase više tipova oglasa primjerice PremijereFilma, Akcije onda imamo klasu IVrstaOglasa gdje će biti implementirana metoda kreirajOglas().

Prototype pattern:

- Osnovna funkcija ovog patterna je da olakša kreiranje objekata koristeći postojeću instancu, koja se ponaša kao prototip. Novokreiranom objektu možemo promijeniti određene osobine po potrebi. Ovaj pattern smo primjenili nad klasom KinoSala kako bi olaksali bilježenje sala te izvršimo promjene naziva ili broja redova/kolona ukoliko je to potrebno.

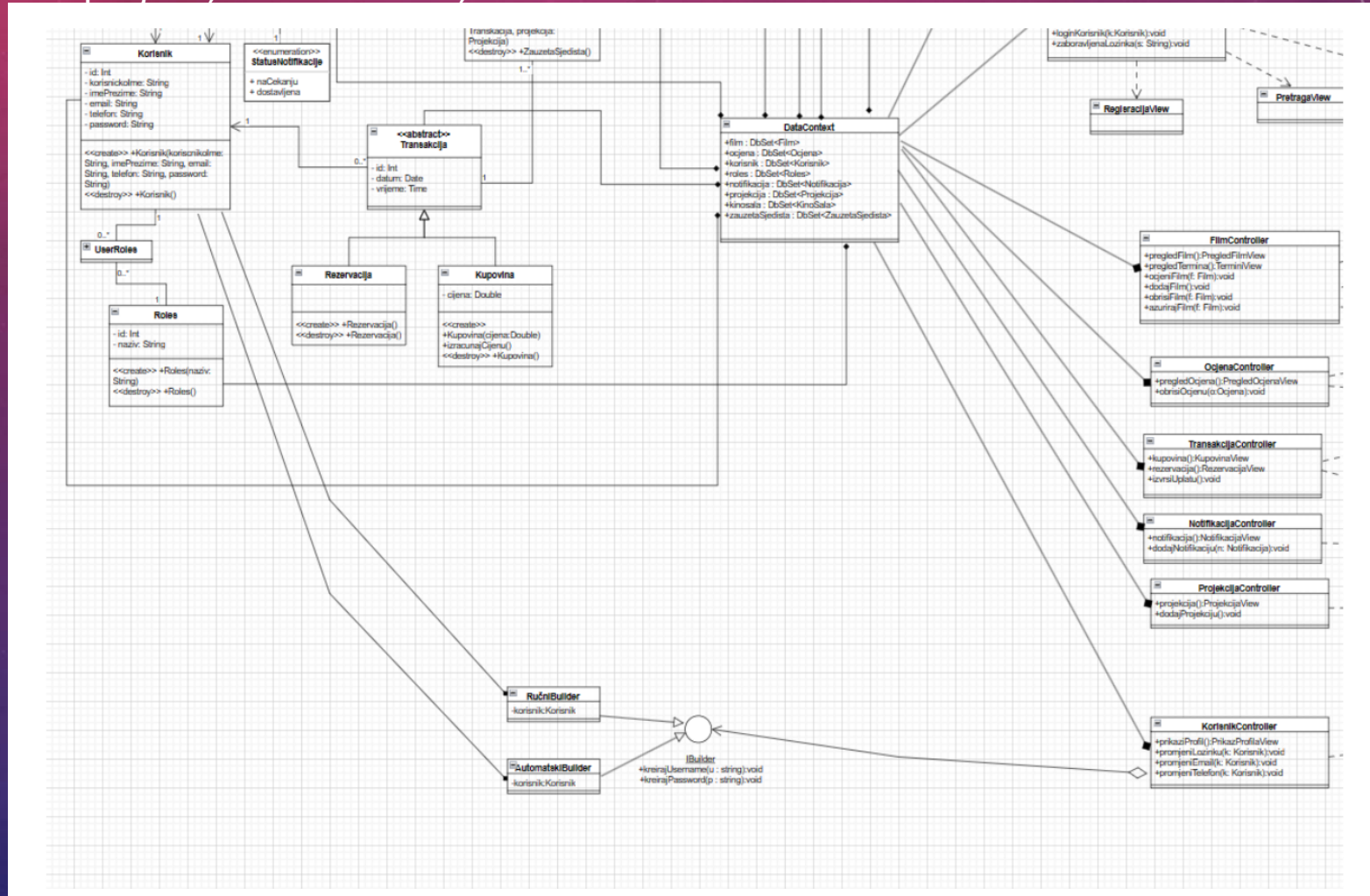


Abstract factory pattern :

- Ovaj patern nam omogućava kreiranje više familija objekata sa mogućnošću dodavanja novih u budućnosti. Kada bi smo željeli implementirati ovaj pattern logčnije bi bilo da posjedujemo još jednu vrstu korisnika a to bi bio premium korisnik. Ono što bi smo omogućili jeste da Korisnik i PremiumKorisnik imaju mogu kupovati karte samo za svoje kategorije. Ovo bi dovelo do toga da PremiumKorisnik bi dobio mogućnost da kupuje posebne karte sa nekim povlasticama

Builder pattern:

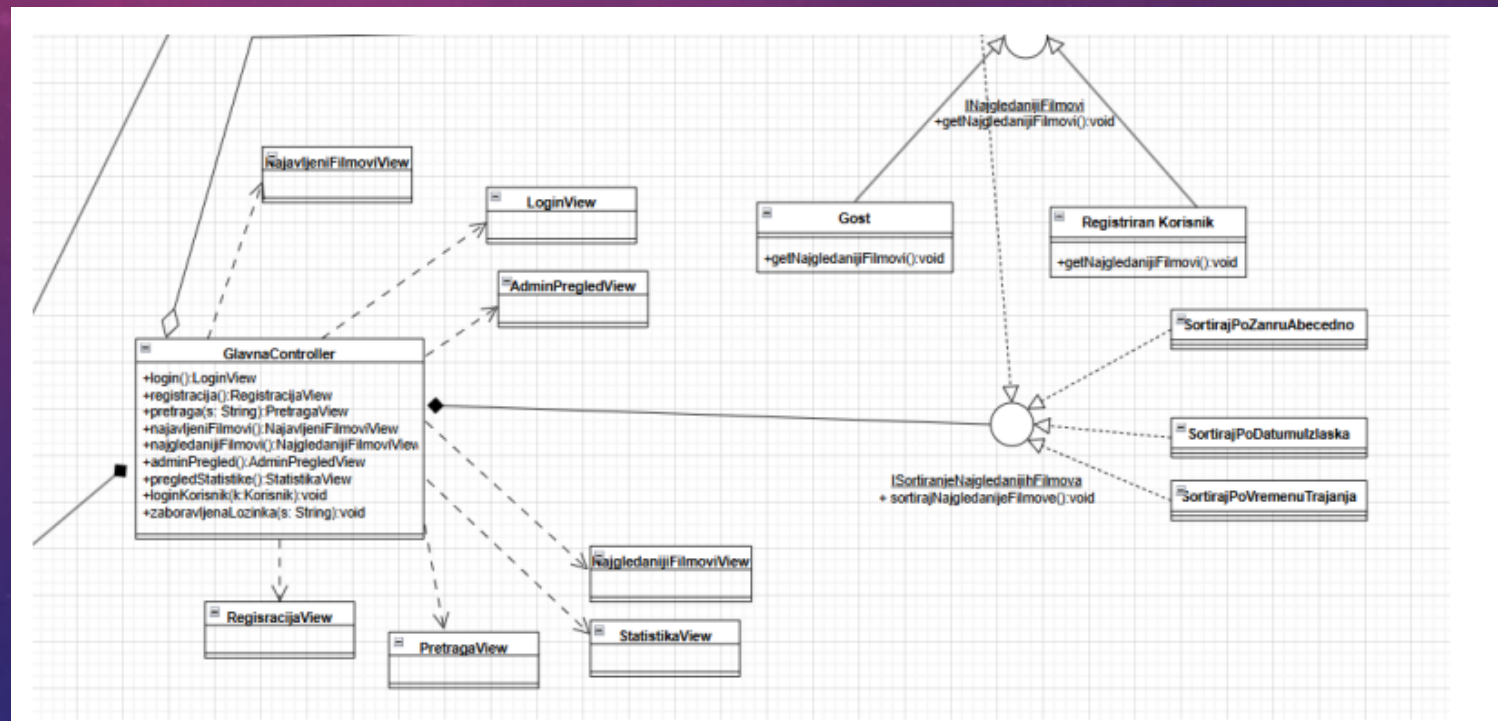
- Builder pattern koristimo kada nam je potrebno primijeniti različite postupke za konstrukciju istog objekta. Ovaj patern ćemo implementirati tako da omogućiti korisniku koji se prvi put registruje na sistem da bira između dva načina registrovanja. On bi mogao odabrati opciju za automatsko generiranje username i password polja ili, ukoliko to želi, da ih sam unese.



PATERNI PONAŠANJA

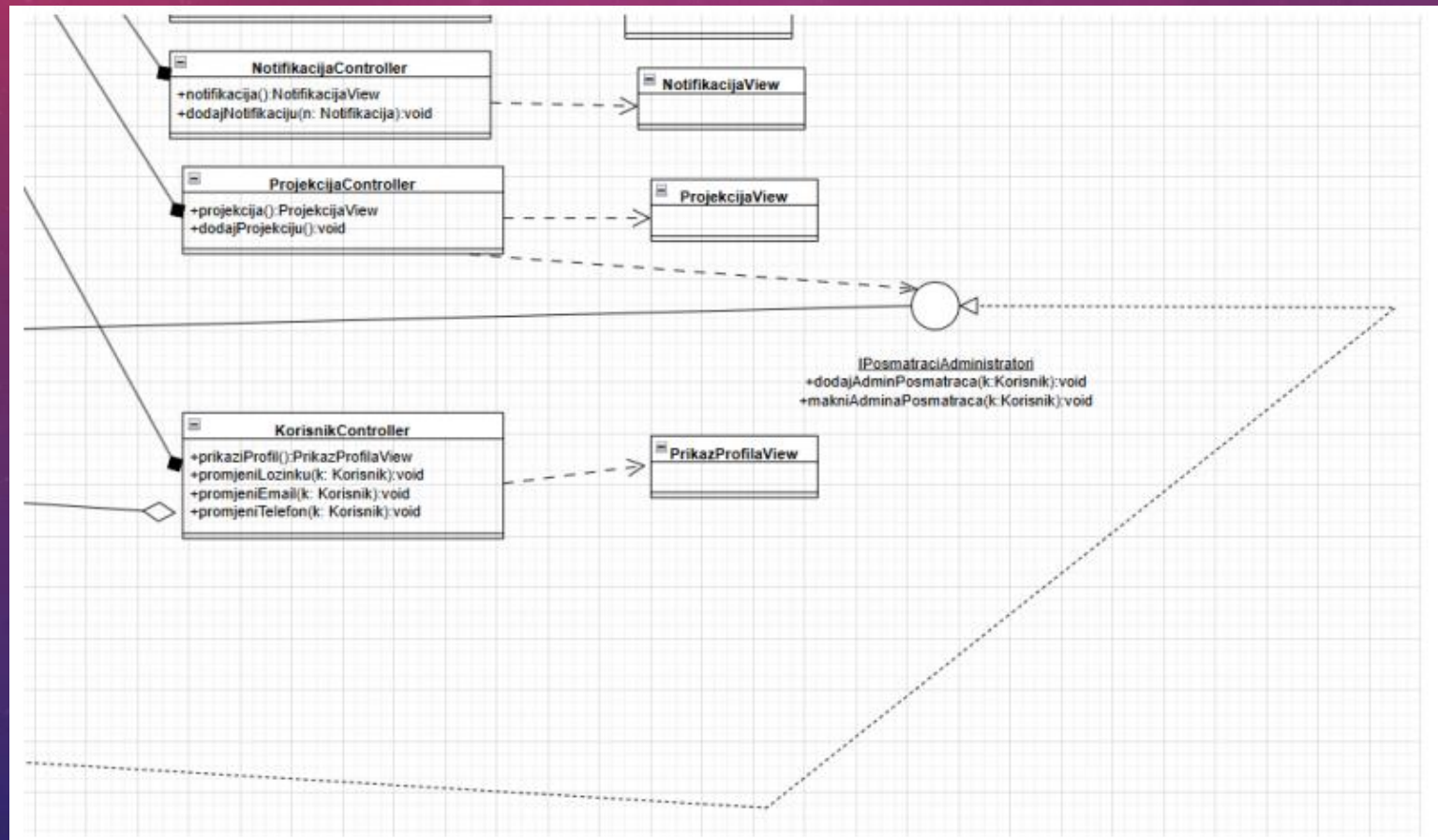
Strategy pattern:

- Strategy patern izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Pogodan je kada postoje različiti primijenjivi algoritmi(strategije) za neki problem i omogućava klijentu izbog jedne od strategija za korištenje. Korištenje ovog paterna nam olakšava brisanje ili dodavanje novih algoritama koji se po želji mogu koristiti. Ovaj patern smo implementirali na način da smo uveli tri nova algoritma za sortiranje najgledanijih filmova. Korisnici mogu da biraju da sortiraju najgledanije filmove po zanru abecedno, po vremenu trajanja filma i po datum izlaska.



Observer pattern:

- Uloga Observer patterna je da uspostavi relaciju između objekata tako da kada jedan objekat promijeni stanje, drugi zainteresirani objekti se obavještavaju. Jednu od mogućnosti što smo dodali jeste da Zaduženi ljudi mogu obavijestiti posmatrace o desavanjima tokom projekcije i promjenama u tijeku projekcije te i mogućnost da obavijeste sve korisnike koji gledaju taj film u datoj projekciji.
- Povezali smo s klasom Korisnici i Projekcije.



State pattern:

- State pattern omogućava objektu da mijenja svoja stanja, od kojih zavisi njegovo ponašanje. Sa promjenom stanja objekat se počinje ponašati kao da je promijenio klasu. Stanja se ne mijenjaju po želji klijenta, već automatski, kada se za to steknu uslovi. State Pattern je dinamička verzija Strategy patterna. Ovaj pattern bismo mogli iskoristiti npr da u sistemu imamo klase VIP i NonVIP zajedno s interfejsom IStanje. Ideja je sljedeća da Vip korisnici imaju mogućnost pregleda više trailer jednog filma dok NonVIP korisnici imaju mogućnost jednog osnovnog trailera. Nalazila bi se metoda pregledTrailera koja bi na osnovu tipa korisnika bila drugačije realizovana.

Template method pattern:

- Template method pattern služi za omogućavanje izmjene ponašanja u jednom ili više dijelova. Najčešće se primjenjuje kada se za neki kompleksni algoritam uvijek trebaju izvršiti isti koraci, ali pojedinačne korake moguće je izvršiti na različite načine. U našoj aplikaciji to bi mogli uraditi na način da prilikom logina na aplikaciju korisnik koji je administrator ako bi želio da koristi svoje administratorske ovlasti morao bi unijeti pin koji bi mu dodjeli mogućnost korištenja u suprotnom dok ne unese on će se smatrati običnim korisnikom. Realizovali bismo neku metodu adminPin() koja bi vršila provjeru pina i dodjelila role ako je taj pin tačan.

Iterator:

- Ovaj patern omogućava sekvencijalni pristup elementima kolekcije bez poznavanja kako je kolekcija struktuirana. Ovo bi smo mogli npr implementirati tako što bi osmisliti način na koji bi se iterativno prolazilo kroz kolekciju filmova tj listu filmova koji su dostupni za gledanje u kinu. To bi smo mogli uraditi tako što bi dodali tipa klase VremenskiIterator i tipa OcjenalIterator gdje bi birali između toga na koji način bi se prolazilo kroz filmove da li putem vremenski predloženi termina (poredani od manjeg ka većem) ili putem Ocjena za dati film.

Chain of responsibility patern:

- Chain of responsibility patern namijenjen je tome kako bi se jedan kompleksni proces obrade razdvojio na način da više objekata na različite načine procesiraju primljene podatke. Ovo bismo mogli realizovati tako da smo dali tipa mogućnost administratoru da kreira obavijesti. Imali bismo dvije vrste to su random (za korisnike i goste) i komentarfilm (samo za korisnike). Administrator kreira nesto od ovoga, za random unosi neki tekst dok za komentarfilm unosi neki komentar za određeni film. Stvari koje nemaju se automatski nalaze na internetu.

Medijator patern:

- Mediator patern namijenjen je za smanjenje broja veza između objekata. Umjesto direktnog međusobnog povezivanja velikog broja objekata, objekti se povezuju sa međubjektom medijatorom, koji je zadužen za njihovu komunikaciju. Kada neki objekt želi poslati poruku drugom objektu, on šalje poruku medijatoru, a medijator prosljeđuje tu poruku namijenjenom objektu ukoliko je isto dozvoljeno. Ovo bismo mogli implementirati tako da kao kreiramo nekog tipa “bota” koji bi da imamo log svih komentara koji su postavljeni u sekcijama za filmove provjeravao te logove i pronalazio riječi koje nisu adekvatne za našu kino stranicu te bi poslao obavijest administratorima da uklone datu riječ i sankcionišu korisnika

Command:

- Command je patern ponašanja koji pretvara zahtjev u samostalni objekt koji sadrži sve informacije o zahtjevu. Ova transformacija vam omogućava da prosljeđujete zahtjeve kao argumente metode, odlažete ili stavljate u red izvršenja zahtjeva i podržavate operacije koje se ne mogu izvršiti. Ovaj patern možemo iskoristiti za neke komande, koje se nalaze na različitim mjestima a proizvode isti kod.

DIJAGRAM PAKETA

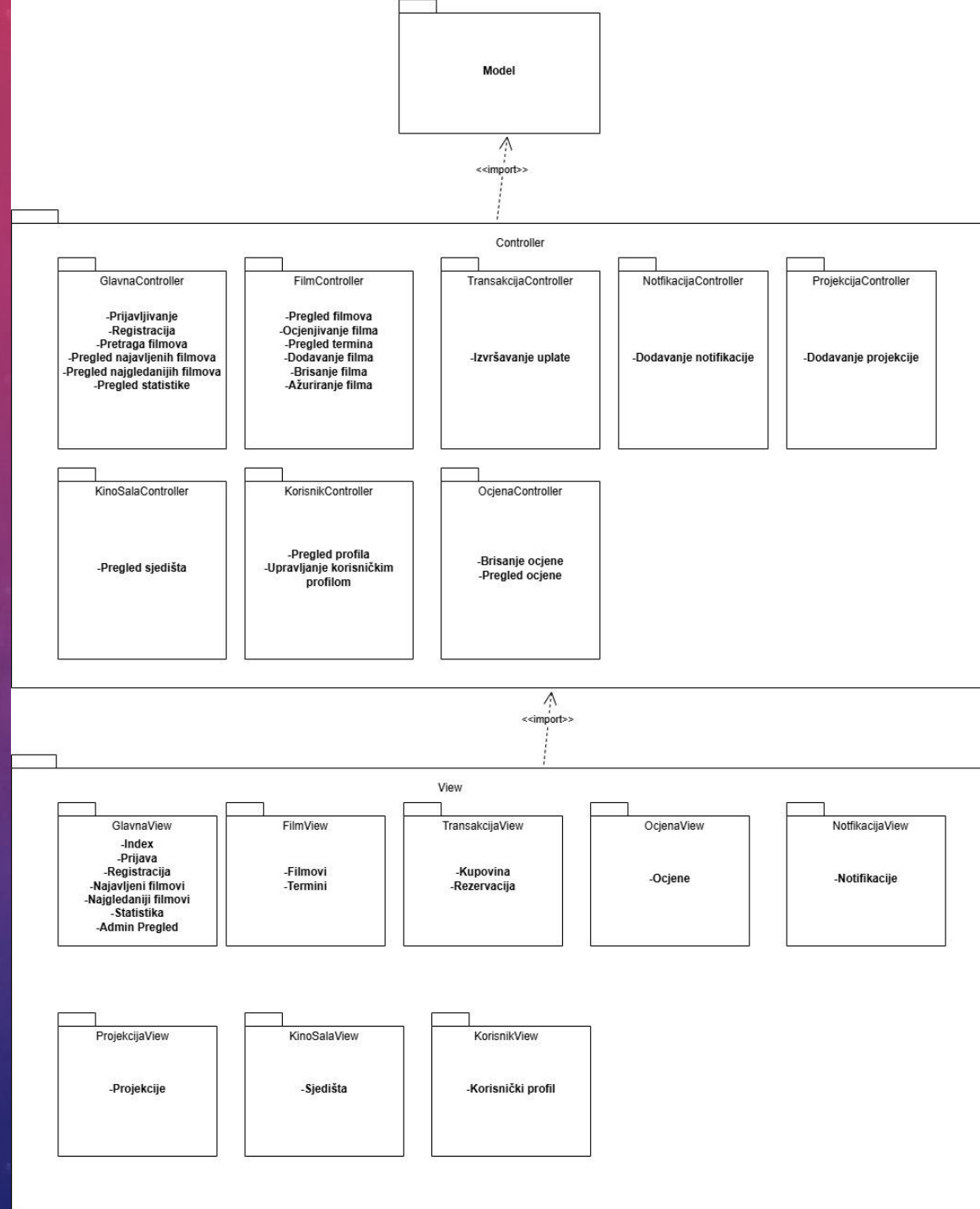


DIAGRAM KOMPONENTI

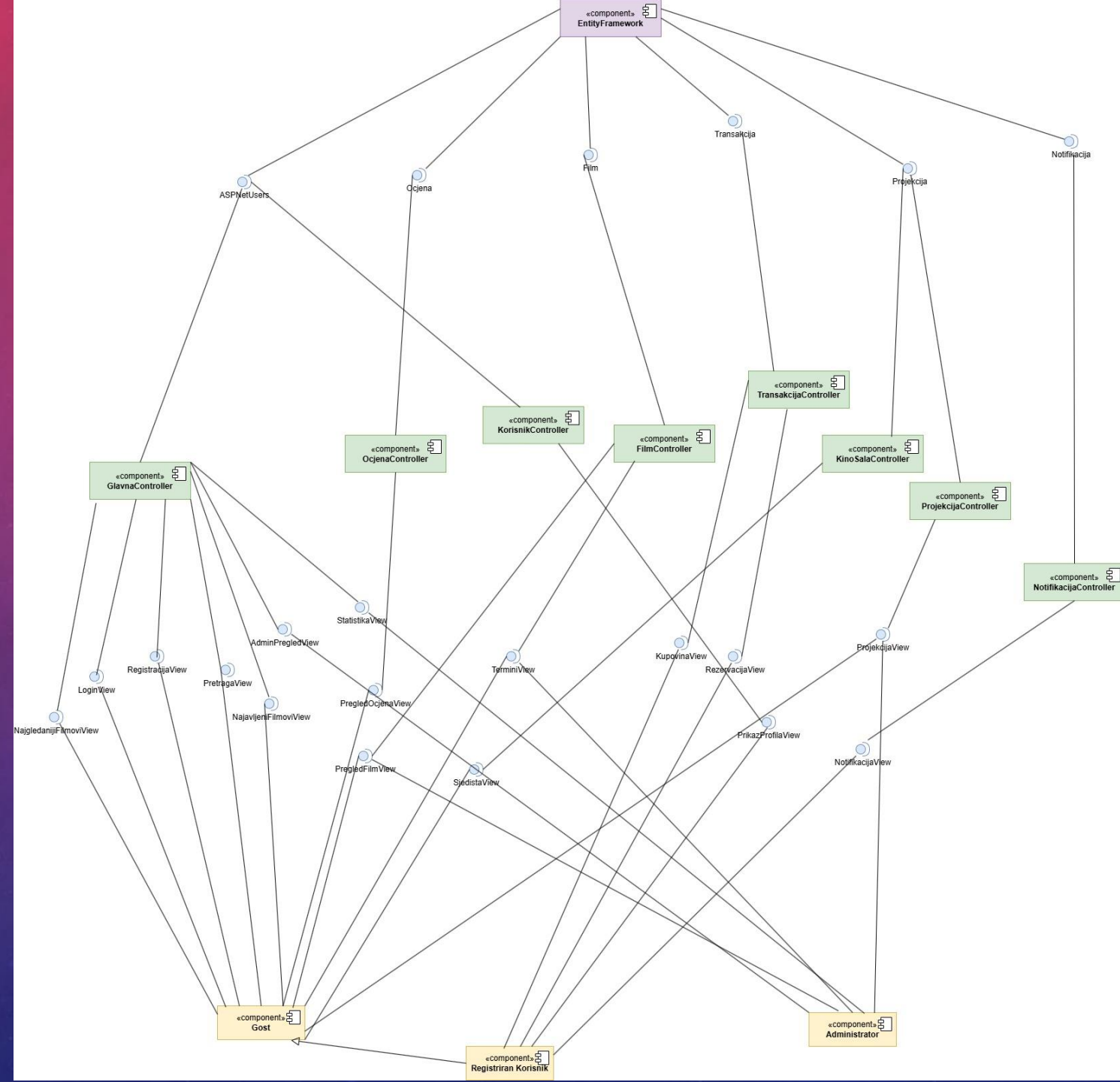
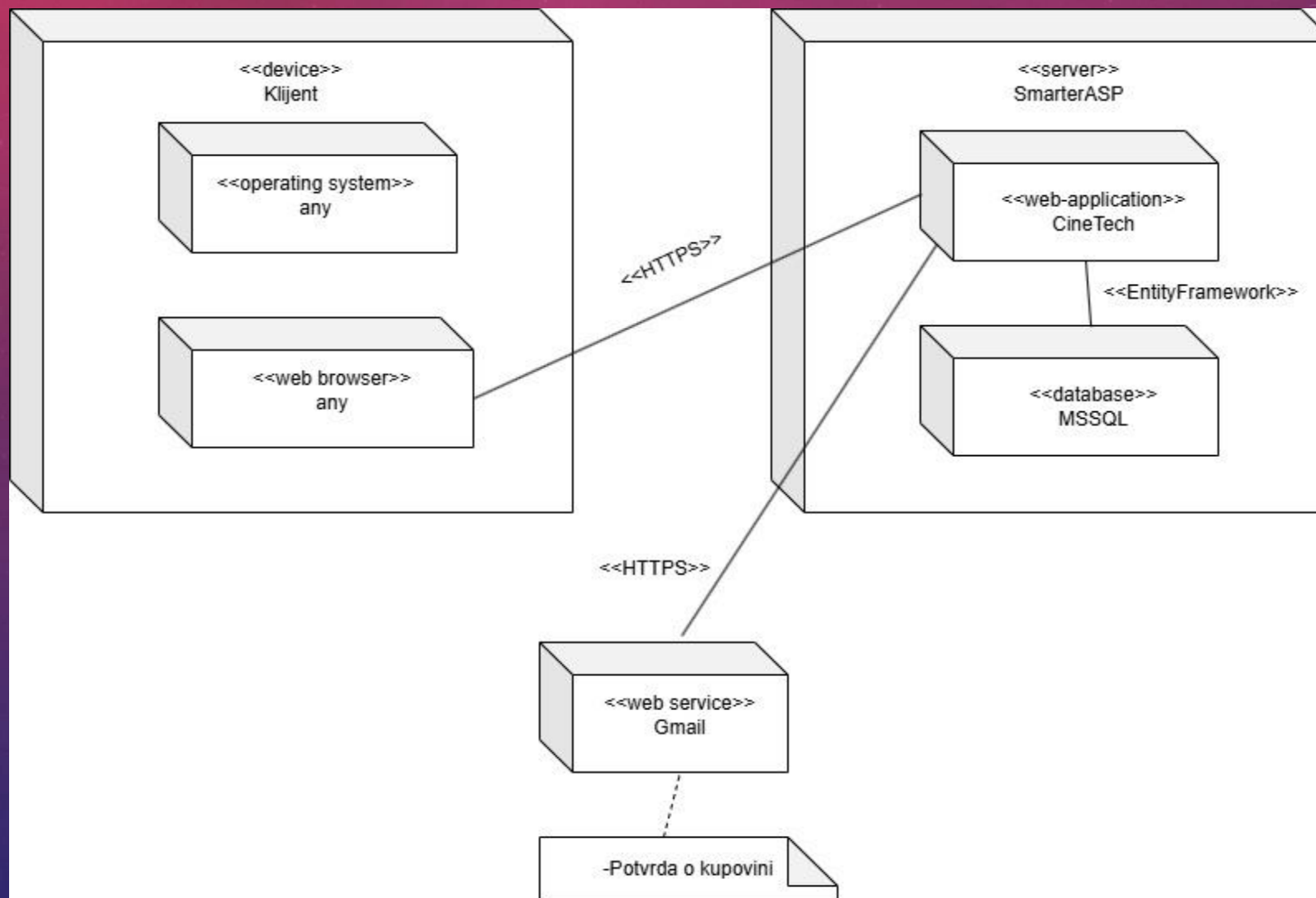


DIAGRAM RASPOREĐIVANJA





HVALA NA PAŽNJI

THANK YOU!

