

# **OOAD Projekat**

Dokumentacija vezana za projekat-**Grupa4-Kino**



# **CINEMAGIC**

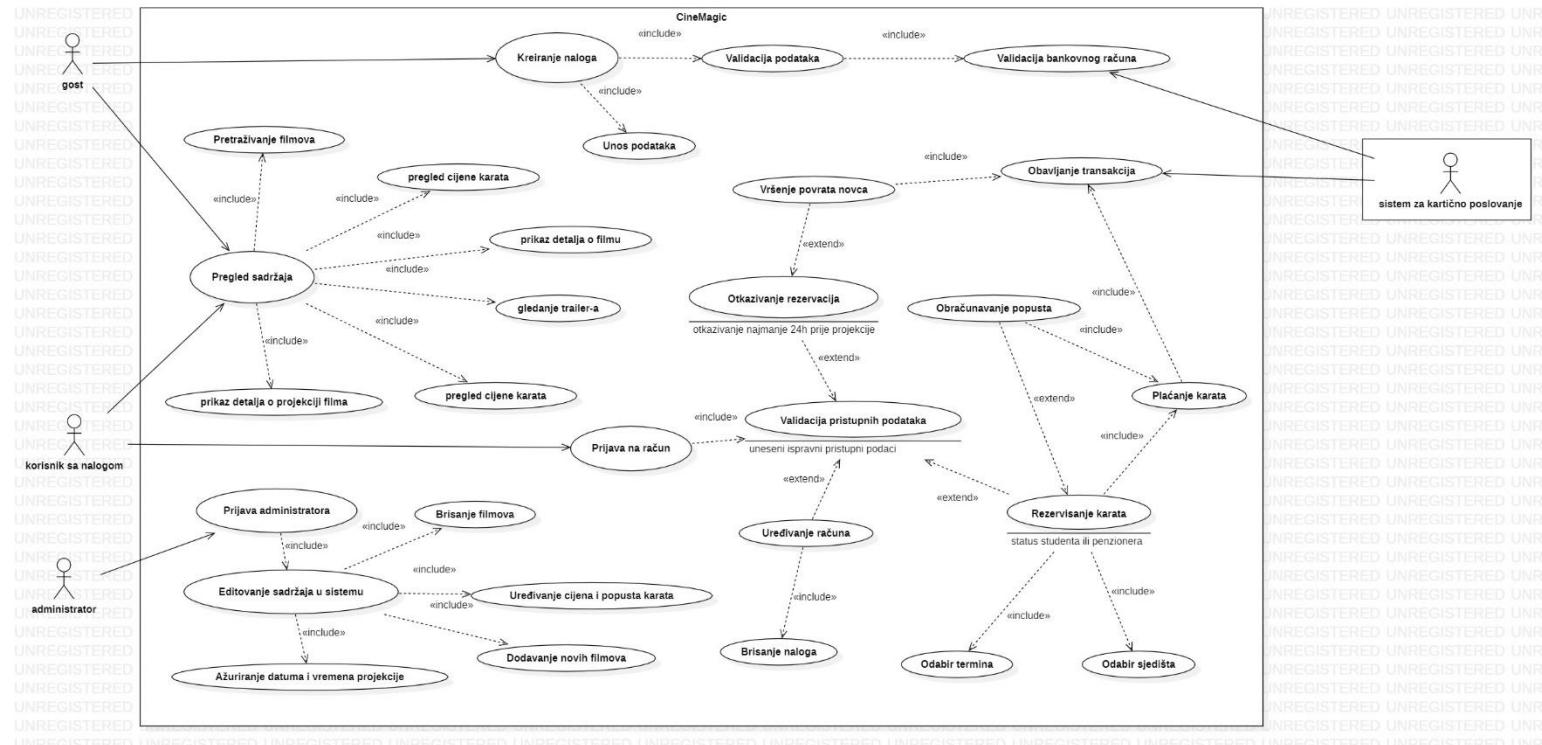
Radili:

Kanita Đokić

Selma Ćelosmanović

Tarik Šahović

## 1. Use Case Diagram



Dijagrami slučajeva upotrebe definišu se na osnovu opisa sistema na visokom nivou apstrakcije. Uloga Use Case Diagrama je da prikaže osnovne funkcionalne zahtjeve i aktere sistema. Njegova namjena nije da detaljno prikaže tok izvršavanja određenih aktivnosti niti poziciju sistema u okruženju

## 2. Scenario

Ovdje imamo prikazane scenarije za naš sistem

## **Scenarij br. 1**

Naziv slučaja upotrebe:	Pregled sadržaja
Opis:	Omogućen pregled postojećeg sadržaja i prikaz informacija o istom
Preduslov:	Korisnik ima pristup internetu
Posljedice:	Korisnik je pregledao sve što ga je interesovalo
Primarni akteri:	Gost i korisnik sa nalogom
Glavni tok:	Korisnici na glavnoj stranici imaju prikaz 7 filmova 7 dana unaprijed. Omogućeno je i pretraživanje filmova po određenim

### **Tok događaja**

Korisnik	Sistem
1. Ulaskom u aplikaciju prikazan je spisak svih filmova	2. Prikazani su filmovi sortirani po datumima; omogućena je pretraga po određenim parametrima
3. Korisnik bira film ili ga pretražuje	4. Film se pronalazi u bazi
	5. Sistem prikazuje film
6. Korisniku se prikazuje naziv filma, trailer, glumci i opis	
7. Korisnik pritisne na dugme za informacije o projekciji	8. Sistem prikazuje informacije o prikazivanju filma
9. Korisniku su prikazani termini projekcije, cijene karata i broj raspoloživih sjedišta	
10. Korisnik pritisne na dugme za rezervaciju	11. Sistem proslijeđuje korisnika na dio za prijavu na račun ili na dio za kreiranje novog računa

### **Alternativni tok događaja 1: Sadržaj pretrage nije pronađen**

Korisnik	Sistem
1. Korisnik unosi pogrešne podatke za pretragu	2. Sistem pokušava pronaći podatke u bazi
	3. Sistem ne pronalazi ništa
	4. Sistem prikazuje poruku upozorenja te mogućnost povratka na početnu stranicu
5. Korisnik se vraća na početnu stranicu ili završava korištenje aplikacije	

### **Scenarij br. 2**

Naziv slučaja upotrebe:	Kreiranje naloga
Opis:	Omogućeno kreiranje naloga kako bi korisnik mogao rezervisati karte i platiti iste
Preduslov:	Korisnik ima pristup internetu i pristupio je sistemu
Posljedice:	Korisnik je kreirao nalog
Primarni akteri:	Gost
Glavni tok:	Korisnik ima status gosta i potreban mu je nalog, unosi svoje pristupne podatke, uključujući i broj bankovnog računa. Nakon toga se vrši validacija podataka i korisnik uspješno kreira

### **Tok događaja**

Korisnik	Sistem
1. Korisnik pritisne dugme za rezervaciju karte	2. Sistem javlja da korisnik nema kreiran nalog
	3. Sistem nudi opciju za kreiranje naloga
	4. Sistem prikazuje prozor za kreiranje naloga
5. Korisnik unosi svoje podatke	6. Sistem validira unesene podatke, te bankovni sistem provjerava validnost broja računa
7. Korisnik je uspješno kreirao svoj nalog	

Preduslov:	Uneseni su podaci za pretragu koji se ne nalaze u bazi
Posljedice:	Pojavljuje se poruka upozorenja

### **Alternativni tok događaja 1: Korisnik nije uspio kreirati nalog**

Preduslov:	Uneseni su pristupni podaci koji nisu validni
Posljedice:	Pojavljuje se poruka upozorenja
Korisnik	Sistem
1. Korisnik unosi pogrešne pristupne podatke (nevalidan e-mail, nevalidan broj bankovnog računa ..)	2. Sistem javlja da podaci nisu validni
	3. Sistem prikazuje poruku upozorenja
4. Korisnik nije uspio kreirati nalog	
5. Korisnik se vraća na korak unošenja podataka	

### **Scenarij br. 3**

Naziv slučaja upotrebe:	Prijava administratora i editovanje sadržaja u sistemu
Opis:	Administrator se prijavljuje u sistem i edituje trenutni sadržaj koji se nalazi u sistemu
Preduslov:	Administrator unosi ispravne pristupne podatke
Posljedice:	Administrator je izmijenio sadržaj
Primarni akteri:	Administrator
Glavni tok:	Administrator unosi svoje pristupne podatke, pristupa sistemu i edituje sadržaj , briše trenutne filmove, uređuje cijene i popuste, dodaje nove filmove po potrebi i ažurira datum i .. . . . ..

**Tok događaja:**

Korisnik	Sistem
1. Administrator unosi pristupne podatke	2. Sistem provjerava pristupne podatke
	3. Sistem otvara prozor za administratora
4. Administrator briše jedan film koji se prikazuje	
5. Administrator dodaje novi film umjesto onog koji je obrisao	
7. Administrator je uspješno izvršio izmjene	

**Scenarij br. 4**

Naziv slučaja upotrebe:	Prijava na račun i rezervisanje karte
Opis:	Korisnik sa nalogom se prijavljuje u sistem i rezerviše kartu
Preduslov:	Korisnik posjeduje nalog
Posljedice:	Korisnik je rezervisao kartu
Primarni akteri:	Korisnik sa nalogom
Glavni tok:	Korisnik se uspješno prijavljuje na račun i rezerviše kartu

**Tok događaja:**

Korisnik	Aplikacija	Sistem za kartično plaćanje
1.Unos podataka potrebnih za prijavu	2. Verifikacija datih podataka	
3. Odabir opcije za rezervaciju karata		
4. Odabir termina prikazivanja sadržaja		
5. Odabir sjedišta		
6. Klik na dugme Plati		
		7. Obavlja transakciju vezanu za plaćanje pojedine karte
	8. Vrši rezervaciju termina za korisnika koji je zahtjevao rezervaciju	
	9. Ažuriranje sistema	

	10. Vraćanje na početnu stranicu	
--	----------------------------------	--

### Alternativni tok 1: Rezervisanje karte sa popustom

Korisnik	Aplikacija	Sistem za kartično plaćanje
1.Unos podataka potrebnih za prijavu	2. Verifikacija datih podataka	
3. Odabir opcije za rezervaciju karata		
4. Odabir termina prikazivanja sadržaja		
5. Odabir sjedišta		
6. Klik na dugme Plati		
	7.Obračunava popust za korisnike koji imaju status studenta ili penzionera	
		8. Obavlja transakciju vezanu za plaćanje pojedine karte
	9. Vrši rezervaciju termina za korisnika koji je zahtjevao rezervaciju	
	10. Ažuriranje sistema	
	11. Vraćanje na početnu stranicu	

### Scenarij br. 5

Naziv slučaja upotrebe:	Prijava na račun i uređivanje profila
Opis:	Korisnik sa nalogom se prijavljuje u sistem i uređuje profil
Preduslov:	Korisnik posjeduje nalog
Posljedice:	Korisnik je uredio podatke o profilu
Primarni akteri:	Korisnik sa nalogom
Glavni tok:	Korisnik se uspješno prijavljuje na račun i uređuje profil

**Tok događaja:**

Korisnik	Aplikacija
1.Unos podataka potrebnih za prijavu	2. Verifikacija datih podataka
3. Odabir opcije za uređivanje naloga	
4. Uređivanje podataka vezanih za nalog	
	5. Ažuriranje sistema
	6. Vraćanje na početnu stranicu

**Alternativni tok 1: Korisnik briše nalog(profil)**

Korisnik	Aplikacija
1.Unos podataka potrebnih za prijavu	2. Verifikacija datih podataka
3. Odabir opcije za uređivanje naloga	
4. Odabir opcije za brisanje naloga	
	5. Brisanje naloga iz baze podataka
	6. Ažuriranje sistema
	7. Vraćanje na početnu stranicu

**Scenarij br. 6**

Naziv slučaja upotrebe:	Prijava na račun i otkazivanje rezervacije
Opis:	Korisnik sa nalogom se prijavljuje u sistem i otkazivanje rezervacije
Preduslov:	Korisnik posjeduje nalog
Posljedice:	Korisnik je otkazao rezervaciju
Primarni akteri:	Korisnik sa nalogom
Glavni tok:	Korisnik se uspješno prijavljuje na račun i otkazivanje rezervacije

**Tok događaja:**

Korisnik	Aplikacija	Sistem za kartično plaćanje
1.Unos podataka potrebnih za prijavu	2. Verifikacija datih podataka	
3. Odabir opcije za otkazivanje rezervacije	4. Prikaz termina koje je korisnik rezervisao	
5. Odabir termina koji želimo otkazati		
6.Pritisak na dugme Otkaži		
	7.Određuje da je otkazivanje najkasnije 24h prije početka prikazivanja filma	
	8. Obračunava iznos povrata novca	
		9. Obavlja transakciju vezanu za povrat novca
	10. Otkazuje termin za koji se korisnik odlučio	
	11. Ažuriranje sistema	
	12. Vraćanje na početnu stranicu	

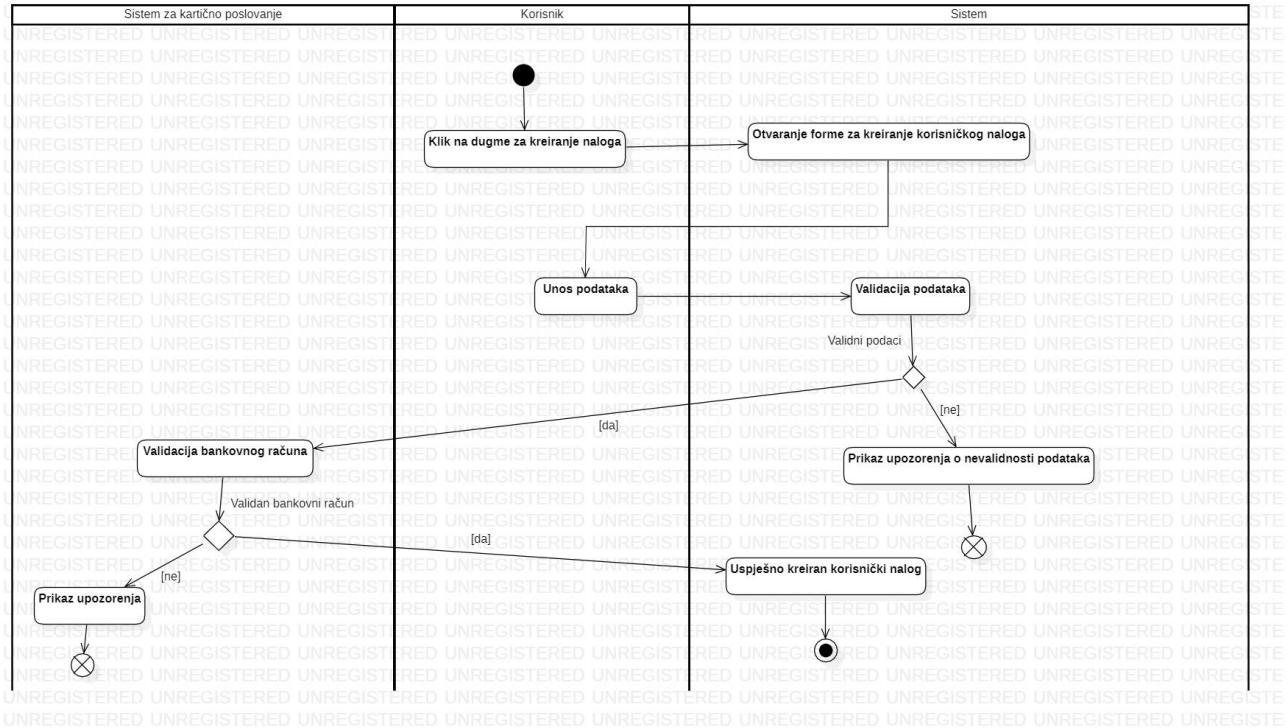
**Alternativni tok događaja: Otkazivanje bez povrata novca**

Korisnik	Aplikacija	Sistem za kartično plaćanje
1.Unos podataka potrebnih za prijavu	2. Verifikacija datih podataka	
3. Odabir opcije za otkazivanje rezervacije	4. Prikaz termina koje je korisnik rezervisao	
5. Odabir termina koji želimo otkazati		
6.Pritisak na dugme Otkaži		
	7.Određuje da otkazivanje nije najkasnije 24h prije početka prikazivanja filma	
	8. Otkazuje termin za koji se korisnik odlučio	
	9. Ažuriranje sistema	
	10. Vraćanje na početnu stranicu	

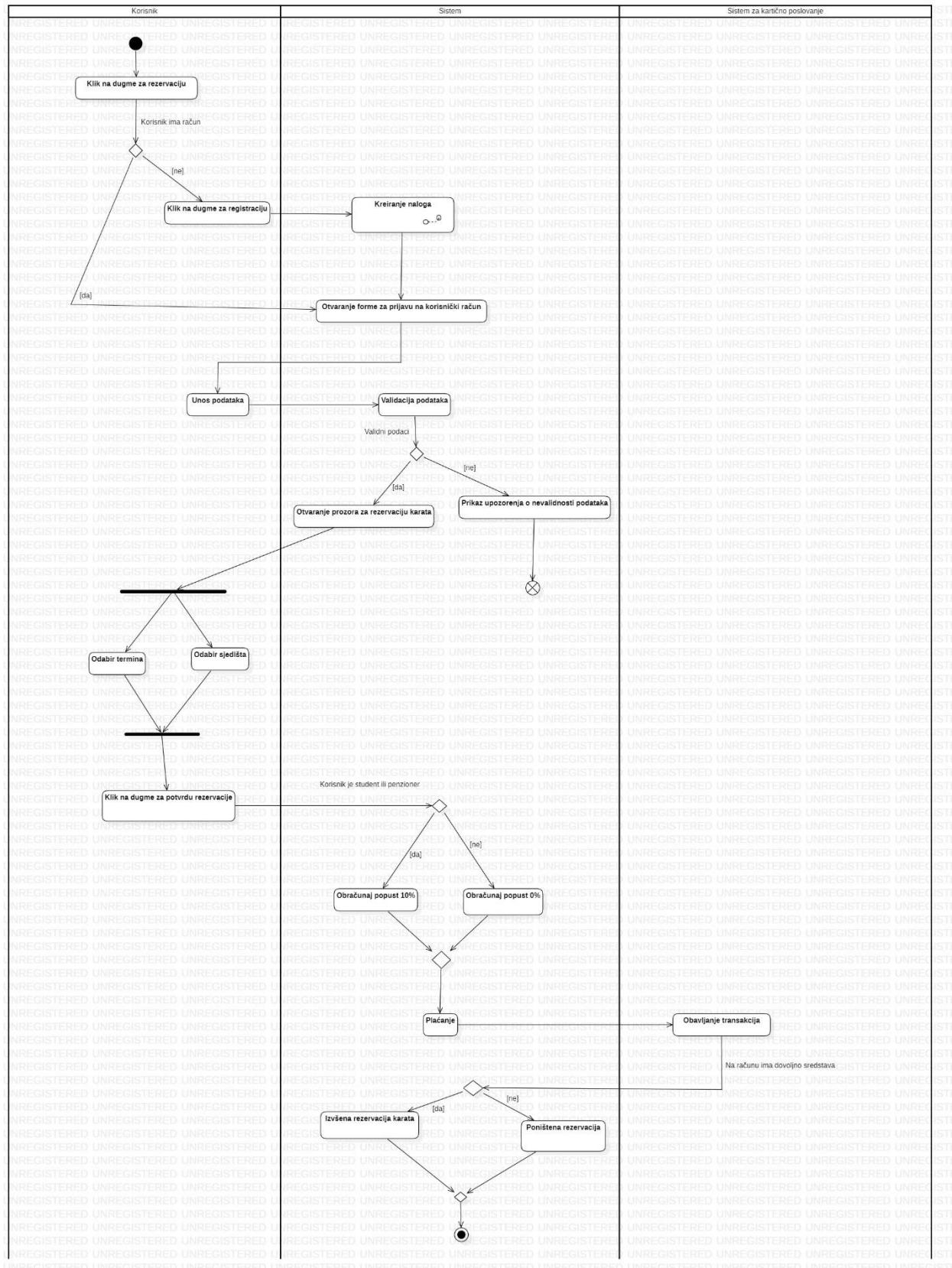
### 3. Activity Diagram

Dijagrami aktivnosti koriste se da opišu tok poslovnih procesa. Najčešće se povezuju sa slučajevima upotrebe i scenarijima.

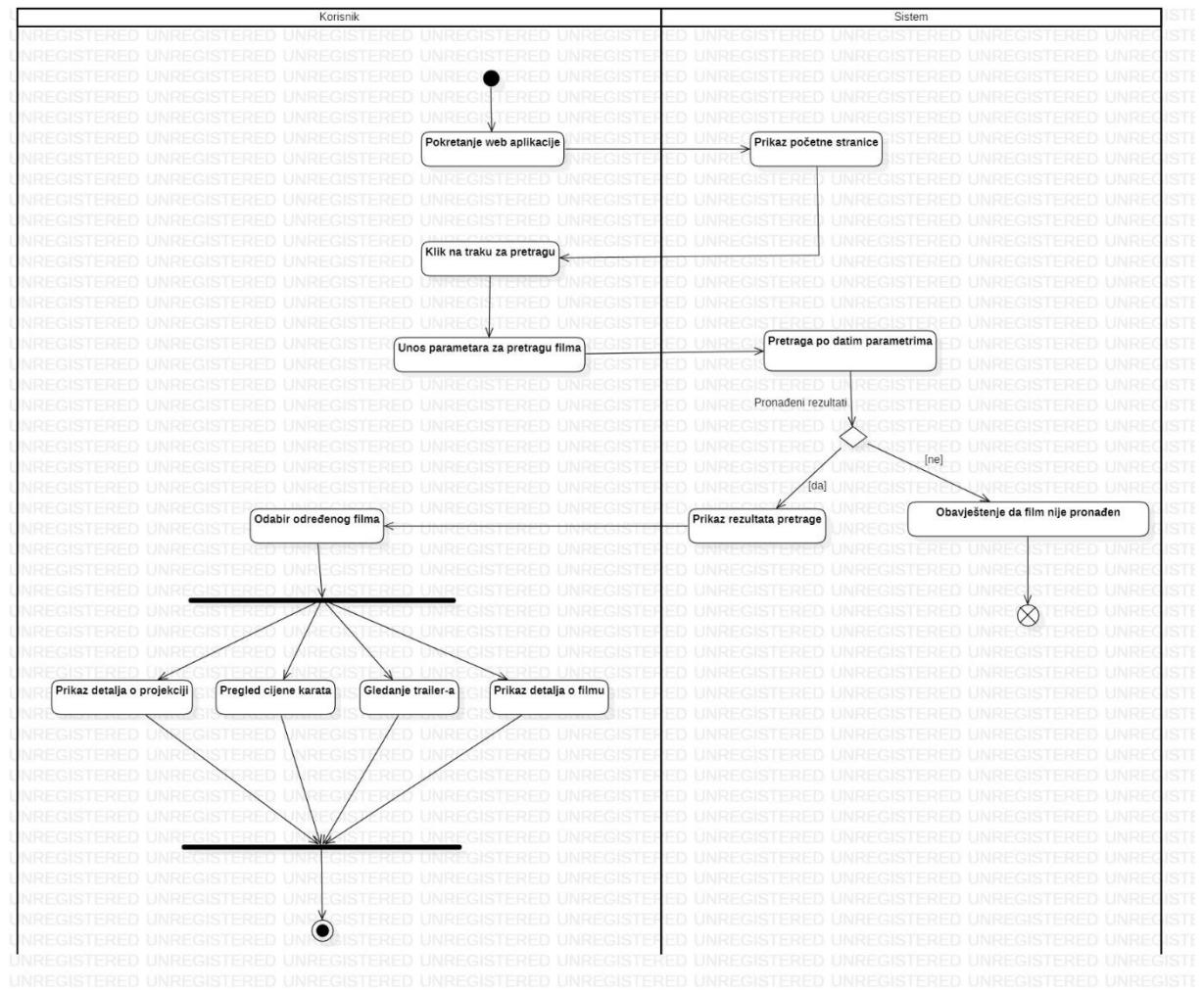
#### a) Kreiranje korisničkog računa



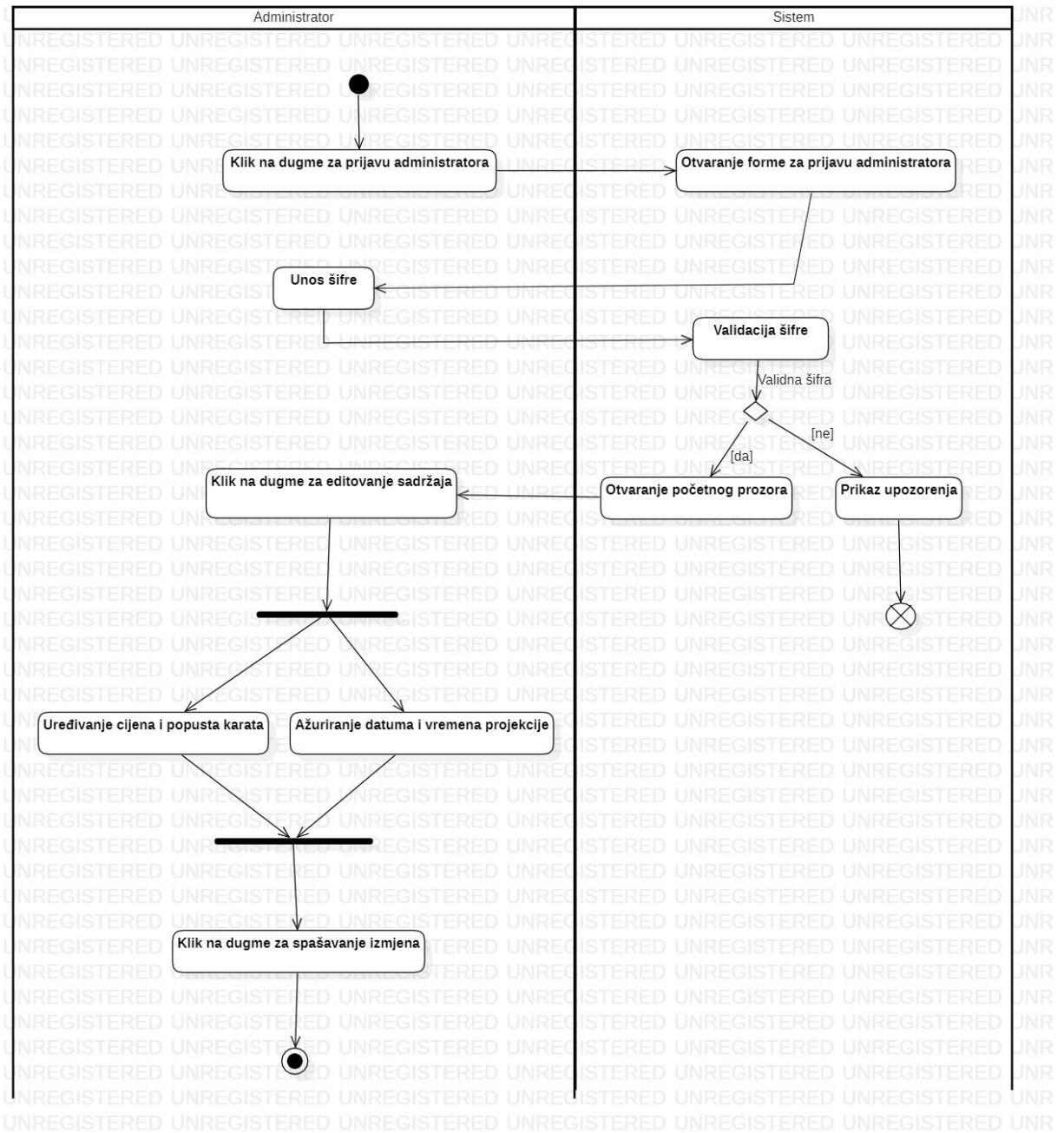
## b) Prijava i Rezervacija



### c)Pregled sadržaja



### d)Otvaranje Administratorskog naloga



#### 4. User Interface Prototype

Prototipovi za korisnički interfejs su prikazani na sljedećim slikama. Korisnički interfejs za našu početku stranicu je na sljedećoj slici:

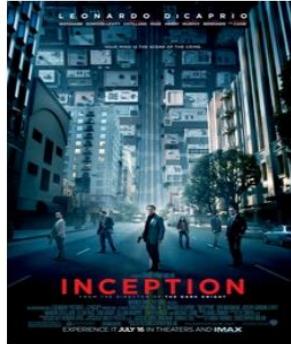
A Web Page

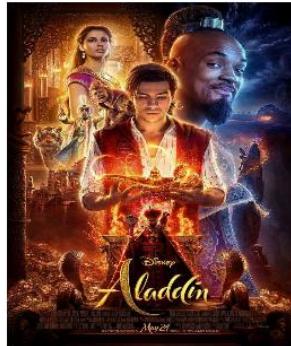
<https://cinemagic.ba>

  
**CINEMAGIC**

This is where magic happens







 Admin

2020 CineMagic All rights reserved

## Home Page

Pogled za rezervisanje filmova je prikazan ispod

## Reservation Site

## Pogled sa adminskog prozora

A Web Page

https://cinemagic.ba



CINEMAGIC

This is where magic happens

search movie

## All movies

Movie Title	Release Year	Genre	Director	Rating	Action
Movie 1	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie
Movie 2	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie
Movie 3	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie
Movie 4	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie
Movie 5	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie
Movie 6	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie
Movie 7	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie
Movie 8	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie
Movie 9	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie
Movie 10	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie
Movie 11	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie
Movie 12	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie
Movie 13	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie
Movie 14	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie
Movie 15	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie
Movie 16	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie
Movie 17	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie
Movie 18	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie
Movie 19	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie
Movie 20	2020	Science Fiction	Jane Doe	8.5	Edit movie  Remove movie

 Add movie

2020 CineMagic All rights reserved

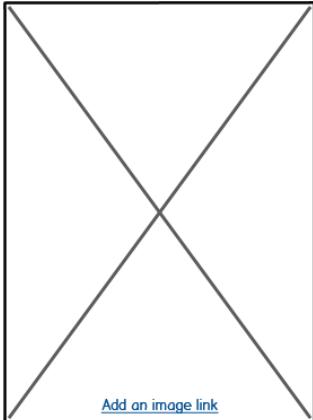
Admin View

## Dodavanje filmova od strane Administratora

A Web Page  
<https://cinemagic.ba>

This is where magic happens

**CINEMAGIC**

 Add an image link

**Set title**

**Synopsis**

**Cast**

Actor 1 \_\_\_\_\_  
Actor 2 \_\_\_\_\_  
Actor 3 \_\_\_\_\_  
Director \_\_\_\_\_

**Genre**  
Genre \_\_\_\_\_

**Duration(min)**  
Duration \_\_\_\_\_

**Ticket price:** \_\_\_\_\_

**Date & Time:**

<input type="checkbox"/>	Monday	<input checked="" type="checkbox"/>	12:00
<input type="checkbox"/>	Tuesday	<input type="checkbox"/>	13:00
<input type="checkbox"/>	Wednesday	<input type="checkbox"/>	14:00
<input checked="" type="checkbox"/>	Thursday	<input checked="" type="checkbox"/>	15:00
<input type="checkbox"/>	Friday	<input type="checkbox"/>	16:00
<input type="checkbox"/>	Saturday	<input type="checkbox"/>	17:00
<input type="checkbox"/>	Sunday	<input checked="" type="checkbox"/>	18:00
<input type="checkbox"/>		<input type="checkbox"/>	19:00
<input type="checkbox"/>		<input type="checkbox"/>	20:00
<input type="checkbox"/>		<input type="checkbox"/>	21:00
<input type="checkbox"/>		<input type="checkbox"/>	22:00
<input type="checkbox"/>		<input type="checkbox"/>	23:00
<input type="checkbox"/>		<input type="checkbox"/>	24:00

**Cinema hall:** \_\_\_\_\_

From: / /    To: / /

Daily projections: 1 | ▾



Add a link for trailer

**Add movie** **Cancel**

2020 CineMagic All rights reserved

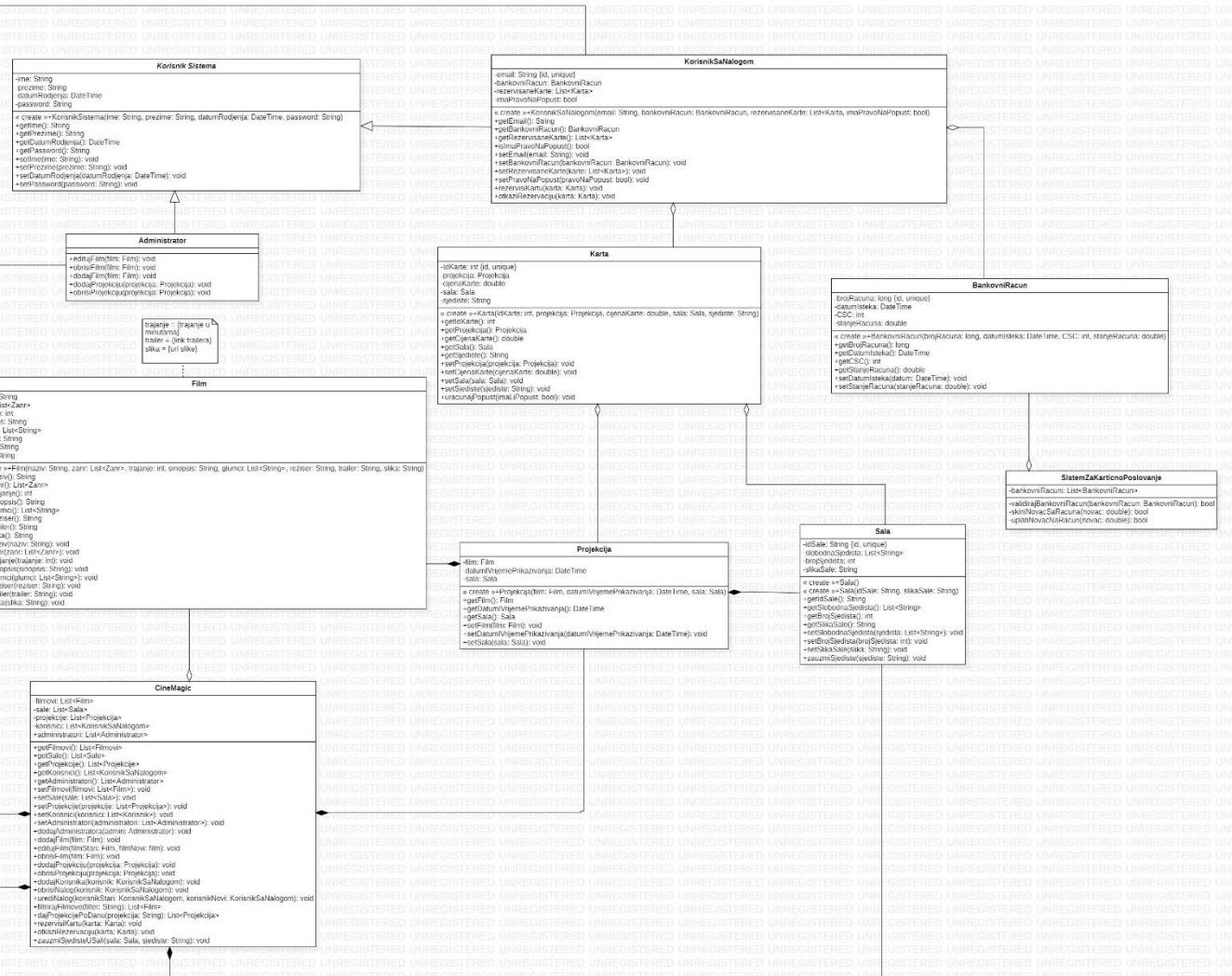
**Adding movies by Administrator**

## 5. Class Diagram

Dijagrami klase definišu se na osnovu opisa sistema, te je za njihovo uspješno kreiranje neophodno identificirati klase, njihove atribute, metode, interfejse i veze između klasa, što je najčešće moguće uraditi na više načina. Dijagram klase predstavlja kostur, odnosno osnovu za implementaciju samog sistema, te je važno ispravno definisati klase i njihove poveznice kako bi implementirani sistem mogao ispravno funkcionisati.

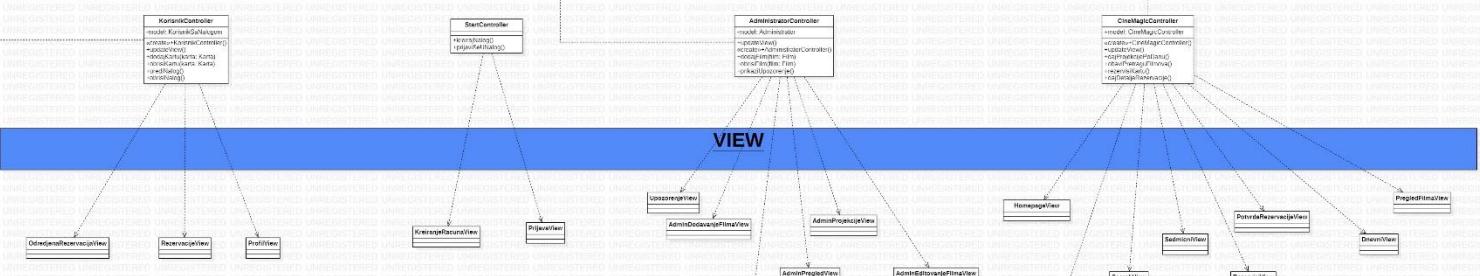
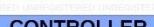
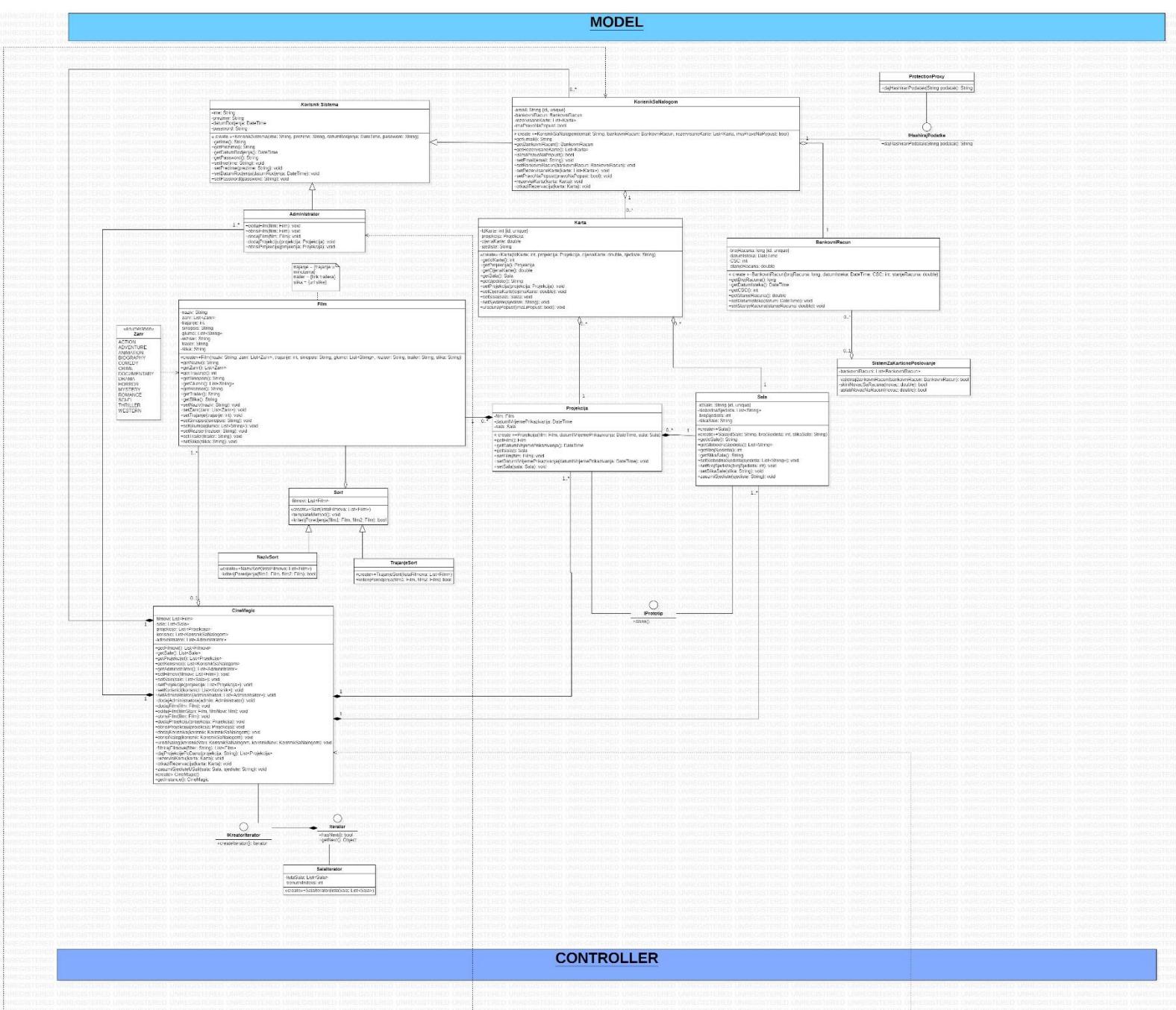
Imamo više verzija klasnog dijagrama koji se iz sedmice u sedmicu poboljšavao.

Prva verzija našeg dijagrama



Na kojoj su prikazane osnovne klase koje smo koristili u našem radu. Dalje imamo prikazanu posljednju verziju našeg dijagrama, znači izgled našeg dijagrama nakon dodavanja

strukturalnih, kreacijskih te paterna ponašanja. Te nakon primjene MVC-a imamo posljednju verziju koja izgleda ovako.

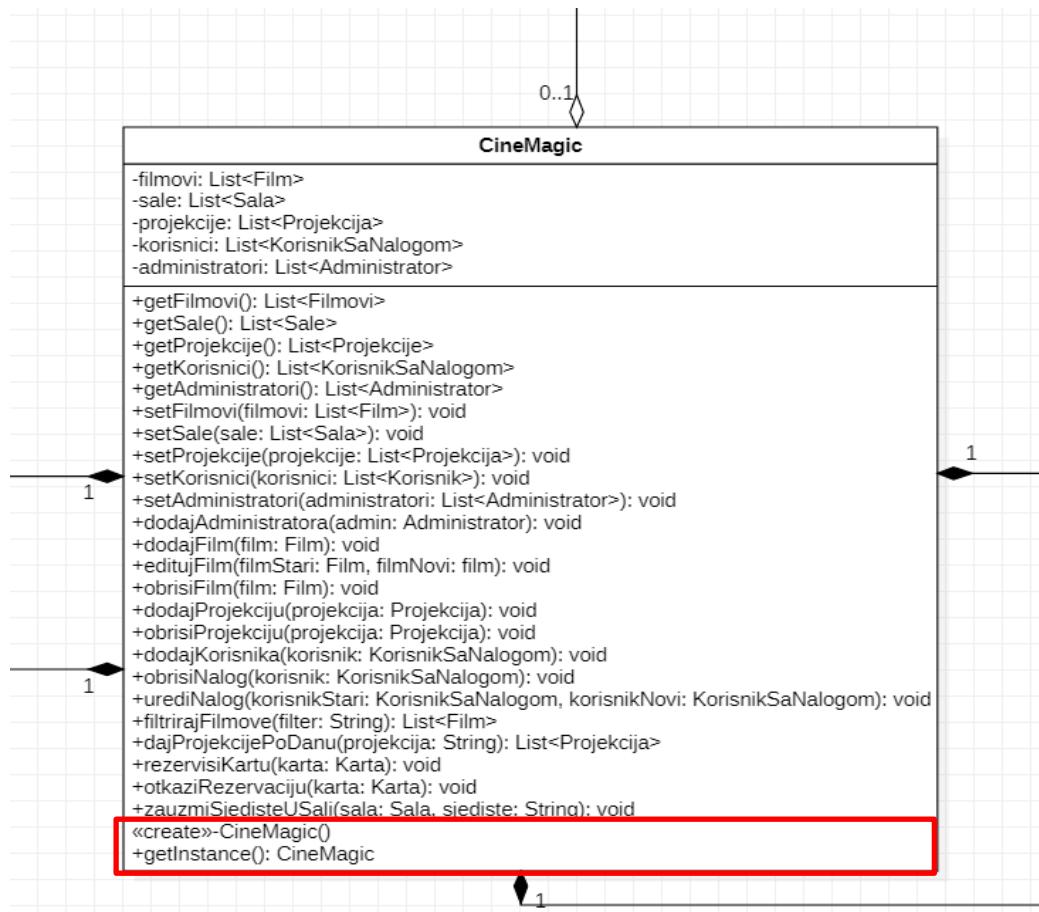


# 6. Patterni

## KREACIJSKI PATERNI

### 1. Singleton pattern

Singleton patern služi kako bi se neka klasa mogla instancirati samo jednom. Na ovaj način može se omogućiti i tzv. lazy initialization, odnosno instantacija klase tek onda kada se to prvi put traži. Osim toga, osigurava se i globalni pristup jedinstvenoj instanci - svaki put kada joj se pokuša pristupiti, dobiti će se ista instanca klase. Ovo olakšava i kontrolu pristupa u slučaju kada je neophodno da postoji samo jedan objekat određenog tipa. Primijenili smo ovaj patern na klasi CineMagic, jer će biti potrebno da imamo samo jednu instancu ove klase. Kada bi imali više instanci došlo bi do komplikacija.

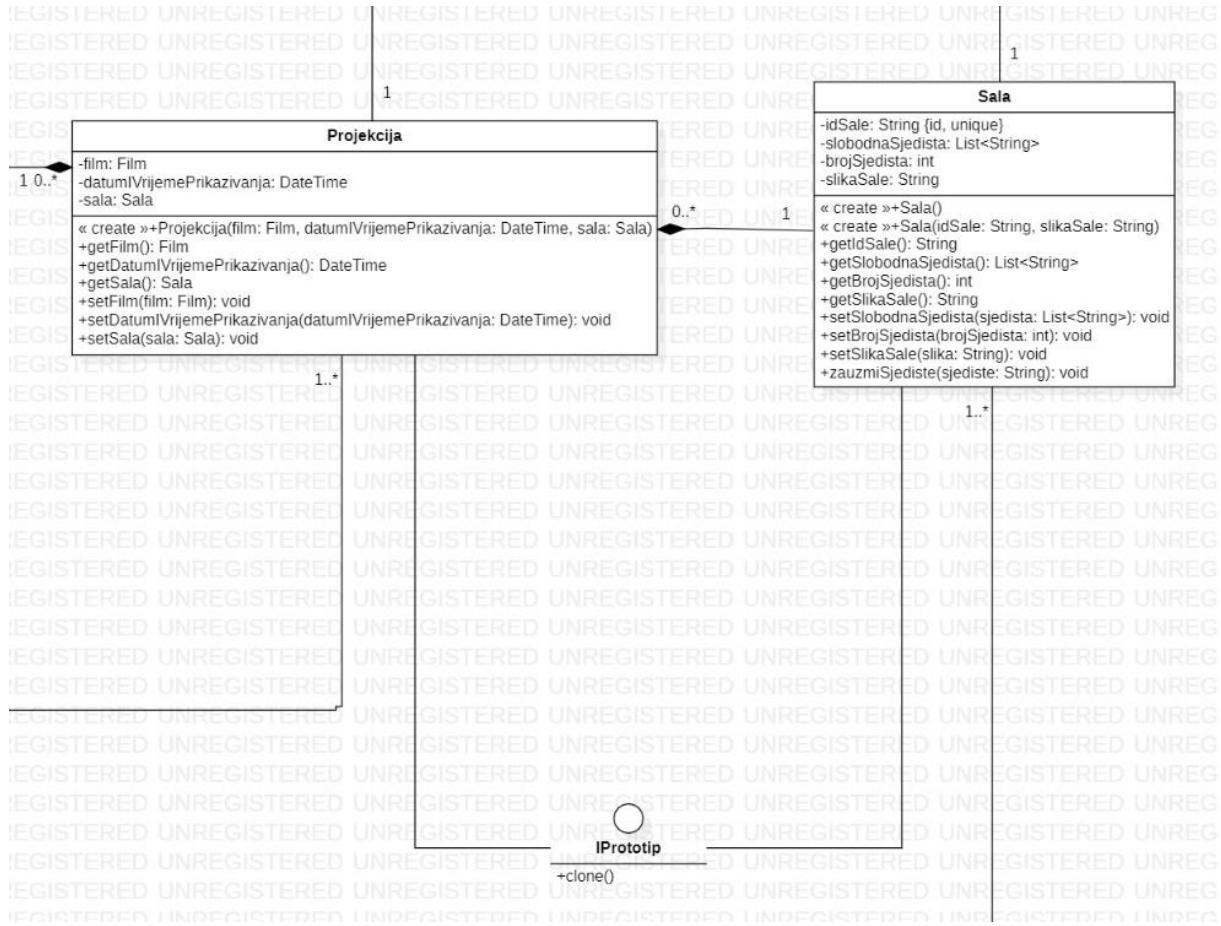


### 2. Prototype pattern

Prototype patern omogućava smanjenje kompleksnosti kreiranja novog objekta tako što se uvodi operacija kloniranja. Na taj način prave se prototipi objekata koje je moguće replicirati više puta a zatim naknadno promijeniti jednu ili više karakteristika, bez potrebe za kreiranjem novog objekta nanovo od početka. Ovime se osigurava

pojednostavljenje procesa kreiranja novih instanci, posebno kada objekti sadrže veliki broj atributa koji su za većinu instanci isti.

U našem projektu smo implementirali ovaj pattern, tako što smo dodali interfejs *IPrototip*, koji ima metodu *clone()*. Ovim interfejsom smo smanjili kompleksnost kreiranja novog objekta, a klase koje ga implementiraju su *Projekcija* i *Sala*.



### 3. Factory Method pattern

Factory method patern služi za omogućavanje instanciranje različitih vrsta podklasa koristeći factory metodu koja odlučuje koja će se podklasa instancirati i koja programska logika izvršiti. Na ovaj način osigurava se ispunjavanje O SOLID principa, jer se kod za kreiranje objekata različitih naslijedenih klasa ne smješta samo u jednu zajedničku metodu, već svaka podklasa ima svoju logiku za instanciranje željenih klasa, a samo instanciranje kontroliše factory metoda koju različite klase implementiraju na različit način. Ovaj pattern nismo implementirali. Njegova implementacija bi mogla biti moguća kod kreiranja korisnika. Dodali bi interfejs *IKorisnik*, koji bi imao metodu *dajUsera()*, dodali bi i *Kreator* klasu koja bi imala metodu *FactoryMethod()*. Ova metoda odlučuje koju klasu instancirati.

### 4. Abstract Factory pattern

Abstract factory patern služi kako bi se izbjeglo korištenje velikog broja if-else uslova pri kreiranju različitih hijerarhija objekata. Ukoliko postoji više tipova istih objekata te

različite klase koriste različite podtipove, te klase postaju fabrike za kreiranje objekata zadanog podtipa bez potrebe za specificiranjem pojedinačnih objekata. Na ovaj način se, korištenjem nasleđivanja, ukida potreba za postojanjem if-else uslova jer određeni tip fabrike sadrži određene tipove objekata i zna se tačno koju podklasu će instancirati. Nismo implementirali ovaj pattern, ali kada bi se odlučili da ga implementiramo, ne bi ga bilo teško uvesti. Iskoristili bi ga za smanjenje if-else uslova pri filtriranju po žanrovima, kreiranjem *IFactory* interfejsom kojim bismo olakšali postupak dobijanja instance potrebnog žanra. Međutim, mi smo ovo riješili na drugi način, pomoću enumeracije.

## 5. Builder pattern

Builder patern služi za apstrakciju procesa konstrukcije objekta, kako bi se kao rezultat moglo dobiti različite specifikacije objekta koristeći isti proces konstrukcije. Ovaj patern koristi se kako bi se izbjeglo kreiranje kompleksne hijerarhije klasa te kako bi se izbjegao kompleksni programski kod konstruktora jedne klase koja može imati različite konfiguracije atributa. Različiti dijelovi konstrukcije objekta izdvajaju se u posebne metode koje se zatim pozivaju različitim redoslijedom ili se poziv nekih dijelova izostavlja, kako bi se dobili željeni različiti podtipovi objekta bez potrebe za kreiranjem velikog broja podklasa. Nismo implementirali dati pattern, ali kada bi naša aplikacija bila dostupna u više gradova, mogli bismo implementirati ovaj pattern na sljedeći način. Pri ulasku na našu stranicu tražila bi se informacija o gradu u kojem se korisnik nalazi, i pomoću te informacije bi *KojiGradBuilder* klasa, koja implementira *IBuilder* interfejs, generisala početnu stranicu sa projekcijama i filmovima koji se projektuju u odabranom gradu.

# STRUKTURALNI PATERNI

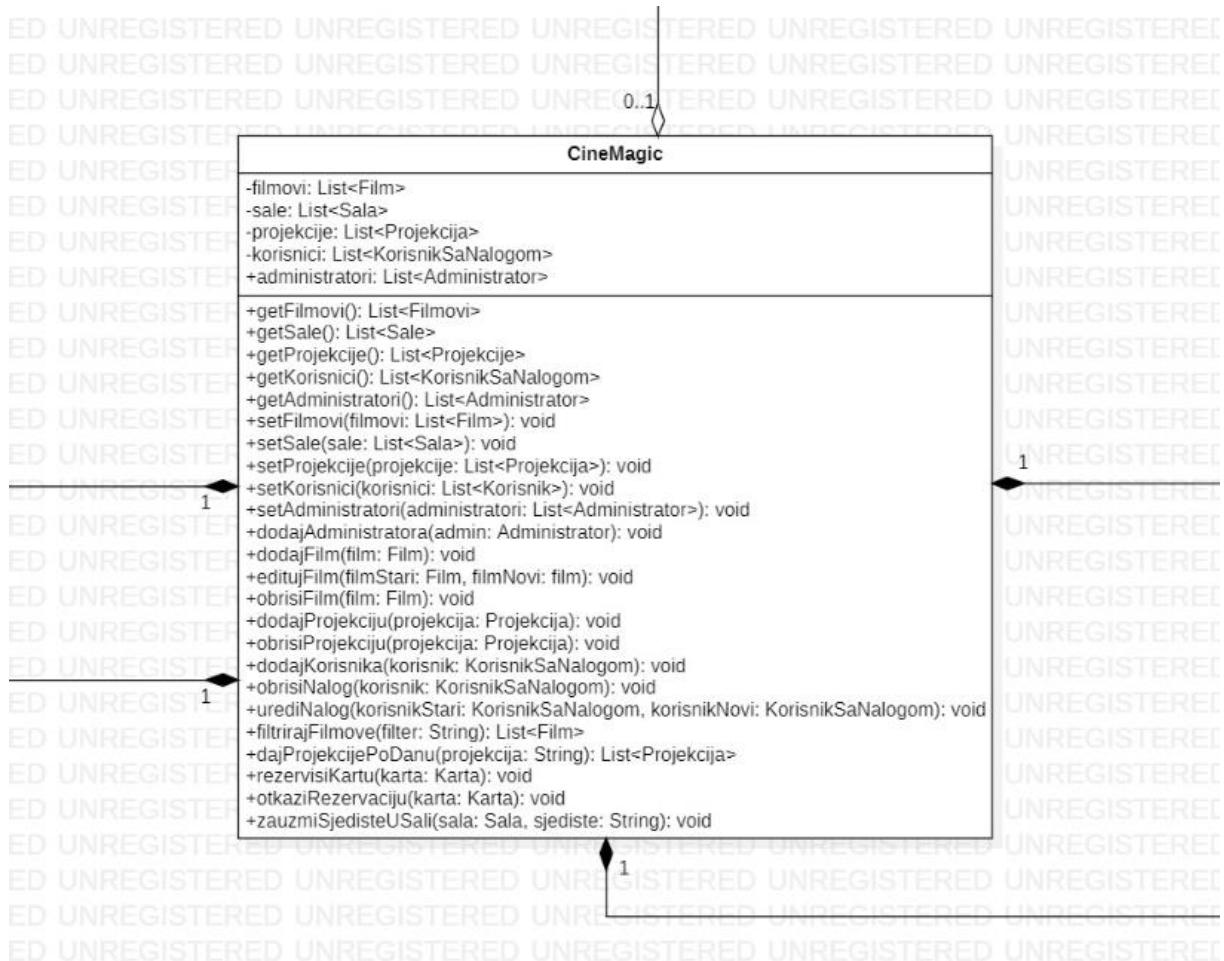
## 1. Adapter pattern

Adapter patern služi da se postojeći objekat prilagodi za korištenje na neki novi način u odnosu na postojeći rad, bez mijenjanja same definicije objekta. Na taj način obezbeđuje se da će se objekti i dalje moći upotrebljavati na način kako su se dosad upotrebljavali, a u isto vrijeme će se omogućiti njihovo prilagođavanje novim uslovima. U našem programu nismo implementirali ovaj pattern, ali kada bi se odlučili da proširimo aplikaciju novim funkcionalnostima, to bi bilo u vidu omogućavanja korisniku da ocjeni i komentariše film koji je prethodno rezervisao. Za dodavanje ove

funkcionalnosti bilo bi potrebno da napravimo Adapter klasu i da napravimo interfejs koji bi sadržavao metode potrebne za realizaciju ovih funkcionalnosti.

## 2. Facade pattern

Fasadni patern služi kako bi se klijentima pojednostavilo korištenje kompleksnih sistema. Klijenti vide samo fasadu, odnosno krajnji izgled objekta, dok je njegova unutrašnja struktura skrivena. Na ovaj način smanjuje se mogućnost pojavljivanja grešaka jer klijenti ne moraju dobro poznavati sistem kako bi ga mogli koristiti.



Ovaj pattern smo implementirali pomoću klase CineMagic, na način da u sebi sadrži druge klase kao atribute. Pozivom jedne metode iz ove klase se poziva više različitih metoda iz različitih klasa, te se time izvršava kompleksniji proces u koji korisnik nema uvid.

Primjer ovoga bi mogla biti metoda *rezervisiKartu* u kojoj se izvršava i plaćanje karte, zauzimanje sjedišta, te još neke dodatne metode potrebne za rezervisanje karte.

## 3. Decorator pattern

Decorator patern služi za omogućavanja različitih nadogradnji objektima koji svi u osnovi predstavljaju jednu vrstu objekta (odnosno, koji imaju istu osnovu). Umjesto da se definiše veliki broj izvedenih klasa, dovoljno je omogućiti različito dekoriranje

objekata (tj. dodavanje različitih detalja), te se na taj način pojednostavljuje i rukovanje objektima klijentima, i samo implementiranje modela objekata.

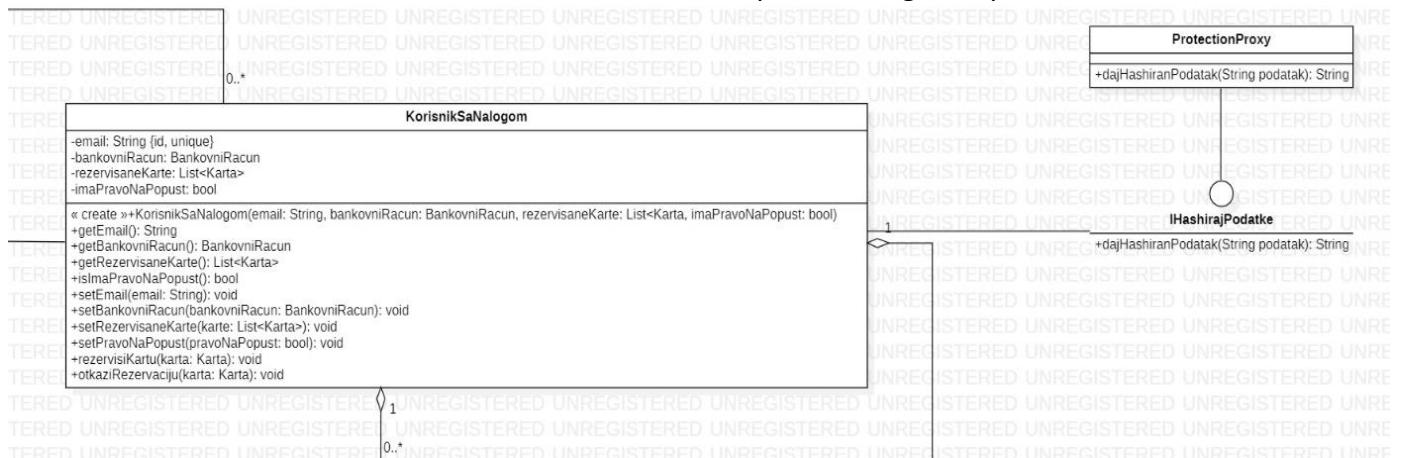
U našem sistemu trenutno nemamo potrebu za implementaciju ovog paterna. Ako bi dodali da cijena karte i izgled početne stranice zavise od određenih praznika, posebnih dana (godišnjica rada i sl.), iskoristili bi ovaj patern. Primjer za ovo je da bi za vrijeme novogodišnjih praznika cijene bile snižene za određeni procenat, te bi početna stranica imala temu prilagođenu ovim praznicima.

#### 4. Bridge pattern

Bridge patern služi kako bi se apstrakcija nekog objekta odvojila od njegove implementacije. Ovaj patern veoma je važan jer omogućava ispunjavanje Open-Closed SOLID principa, odnosno uz poštivanje ovog paterna omogućava se nadogradnja modela klase u budućnosti te osigurava da se neće morati vršiti određene promjene u postojećim klasama. U našem programu ovaj patern nije primijenjen, te nemamo mogućnost primijene istog.

#### 5. Proxy pattern

Proxy patern služi za dodatno osiguravanje objekata od pogrešne ili zlonamjerne upotrebe. Primjenom ovog paterna omogućava se kontrola pristupa objektima, te se onemogućava manipulacija objektima ukoliko neki uslov nije ispunjen, odnosno ukoliko korisnik nema prava pristupa traženom objektu. Iskoristit ćemo ovaj pattern tako što ćemo dodati interfejs *IHashirajPodatke* koji će sve osjetljive podatke heširati te ih tako ubaciti u bazu, kako bi se sačuvali ti podaci i osigurala privatnost korisnika.



#### 6. Composite pattern

Composite patern služi za kreiranje hijerarhije objekata. Koristi se kada svi objekti imaju različite implementacije nekih metoda, no potrebno im je svima pristupati na isti način, te se na taj način pojednostavljuje njihova implementacija. Ovaj patern nećemo implementirati, ali kada bi se odlučili da proširimo aplikaciju, omogućili bi posebnu vrstu korisnika *VIPKorisnik*, koji je uplatio godišnju kartu, te ima posebne pogodnosti. Iako bi ova dva korisnika bila na različitim nivoima, pristupalo bi im se na isti način i implementacija bi bila pojednostavljena.

## 7. Flyweight pattern

Flyweight patern koristi se kako bi se onemogućilo bespotrebno stvaranje velikog broja instanci objekata koji svi u suštini predstavljaju jedan objekat. Samo ukoliko postoji potreba za kreiranjem specifičnog objekta sa jedinstvenim karakteristikama (tzv. specifično stanje), vrši se njegova instantacija, dok se u svim ostalim slučajevima koristi postojeća opća instance objekta (tzv. bezlično stanje). Korištenje ovog paterna veoma je korisno u slučajevima kada je potrebno vršiti uštedu memorije. Nismo implementirali ovaj pattern, međutim mogli bi primijeniti ovaj patern kada bi kreirali klasu *Gost*. Kada bi se *Gost* prijavljivao na sistem, ne bi svaki put kreirali novu instance, već bi se instancirao samo jedan objekat ovog tipa, te bi se uštedila memorija.

# PATERNI PONAŠANJA

## 1. Strategy patern

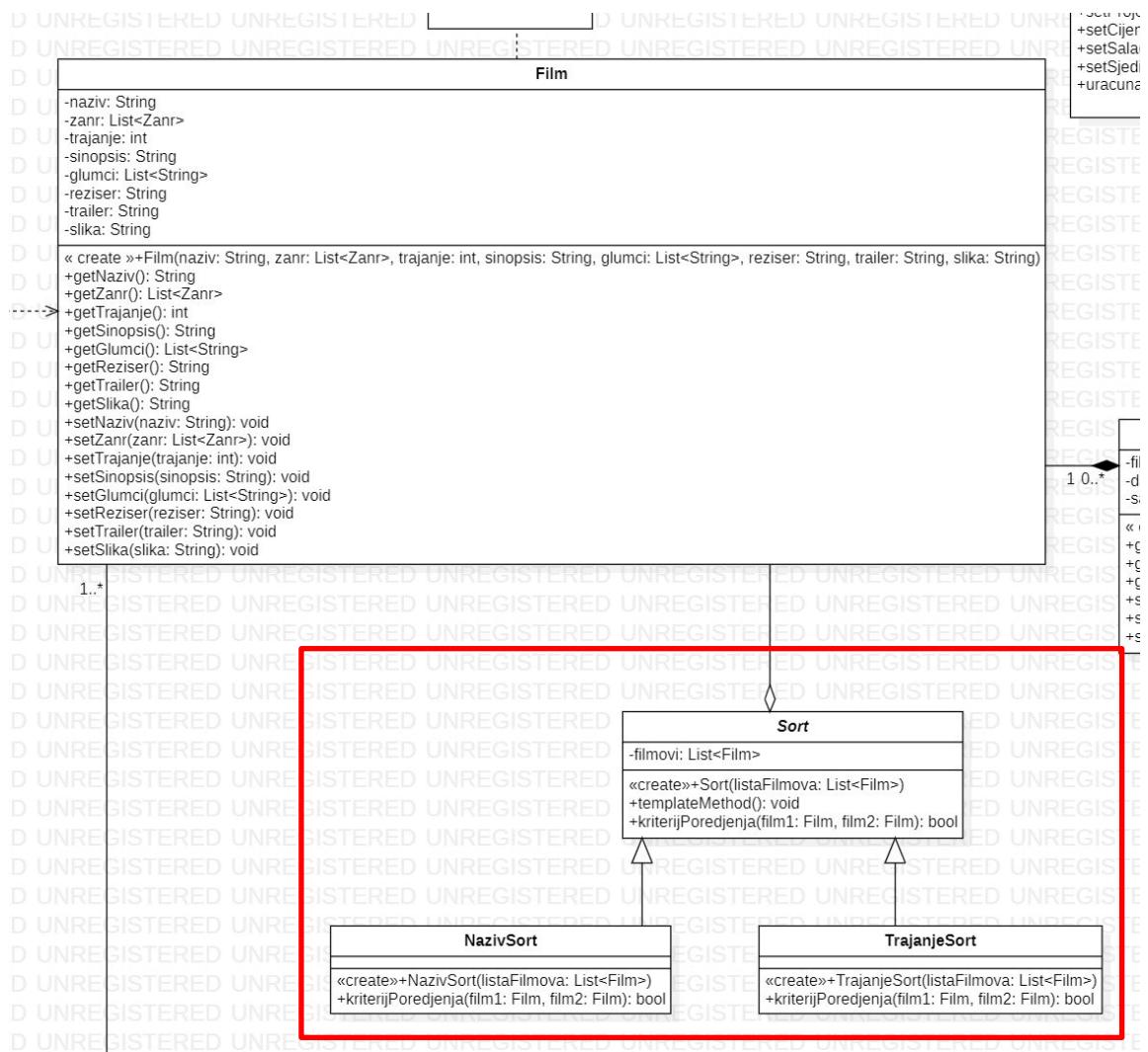
Strategy patern služi kako bi se različite implementacije istog algoritma izdvojile u različite klase, te kako bi se omogućila brza i jednostavna promjena implementacije koja se želi koristiti u bilo kojem trenutku. Na ovaj način omogućava se i jednostavno brisanje ili dodavanje novog algoritma koji se može koristiti po želji. Ovaj pattern nismo implementirali. Međutim kada bi željeli implementirati ovaj pattern to bi bilo kod plaćanja karte, tačnije kada bi omogućili drugačije načine plaćanja rezervisanja karte. Dakle dodali bi interfejs *IStrategy*, koji bi imao jednu metodu *placanjeKarte*, i napravili bi nove klase koje implementiraju ovaj interfejs, npr. klasa za plaćanje karticom, te klasa za plaćanje u kešu. Takođe bi morali napraviti posebnu klasu koja služi samo za plaćanje karte, u njoj bi imali atribut *IStrategy*, te metodu za promjenu načina plaćanja.

## 2. State patern

State patern omogućava objektu da mijenja svoja stanja, od kojih zavisi njegovo ponašanje. Sa promjenom stanja objekt se počinje ponašati kao da je promijenio klasu. Stanja se ne mijenjaju po želji klijenta, već automatski, kada se za to steknu uslovi. Nismo implementirali ovaj pattern. Imamo priliku za implementaciju ovog patterna kod rezervisanja karte, tj kada bi dodali interfejs *IStanje*, koji bi imao metode *rezervisiKartu*, i *otkaziRezervaciju*, te bi dodali *IStanje* interfejs kao atribut klase *Karta*. Također bi dodali i dvije klase koje bi implementirale interfejs *IStanje*, *Rezervisana* i *NijeRezervisana*, ove klase bi opisivale moguća stanja za pojedinu kartu. Funkcionisalo bi na način da kada je karta rezervisana, u klasi *Rezervisana* bi metodom *otkaziRezervaciju* mijenjali stanje karte u *NijeRezervisana*. Na isti način bi metodom *rezervisiKartu* u klasi *NijeRezervisana*, mijenjali stanje karte u *Rezervisana*.

## 3. Template Method patern

Template method patern služi za omogućavanje izmjene ponašanja u jednom ili više dijelova. Najčešće se primjenjuje kada se za neki kompleksni algoritam uvijek trebaju izvršiti isti koraci, no pojedinačne korake moguće je izvršiti na različite načine. Ovaj patern ćemo implementirati. Služi nam za sortiranje filmova po različitim parametrima. Moramo dodati apstraktnu klasu Sort sa listom filmova kao privatnim atributom, ona će imati metodu *templateMethod()*, te ćemo imati i metode koje će biti implementirane u izvedenim klasama tipa *kriterijPoredjenja()*. Imat ćemo izvedene klase pod nazivima NazivSort i TrajanjeSort koje će imati svoju metodu *kriterijPoredjenja()*.



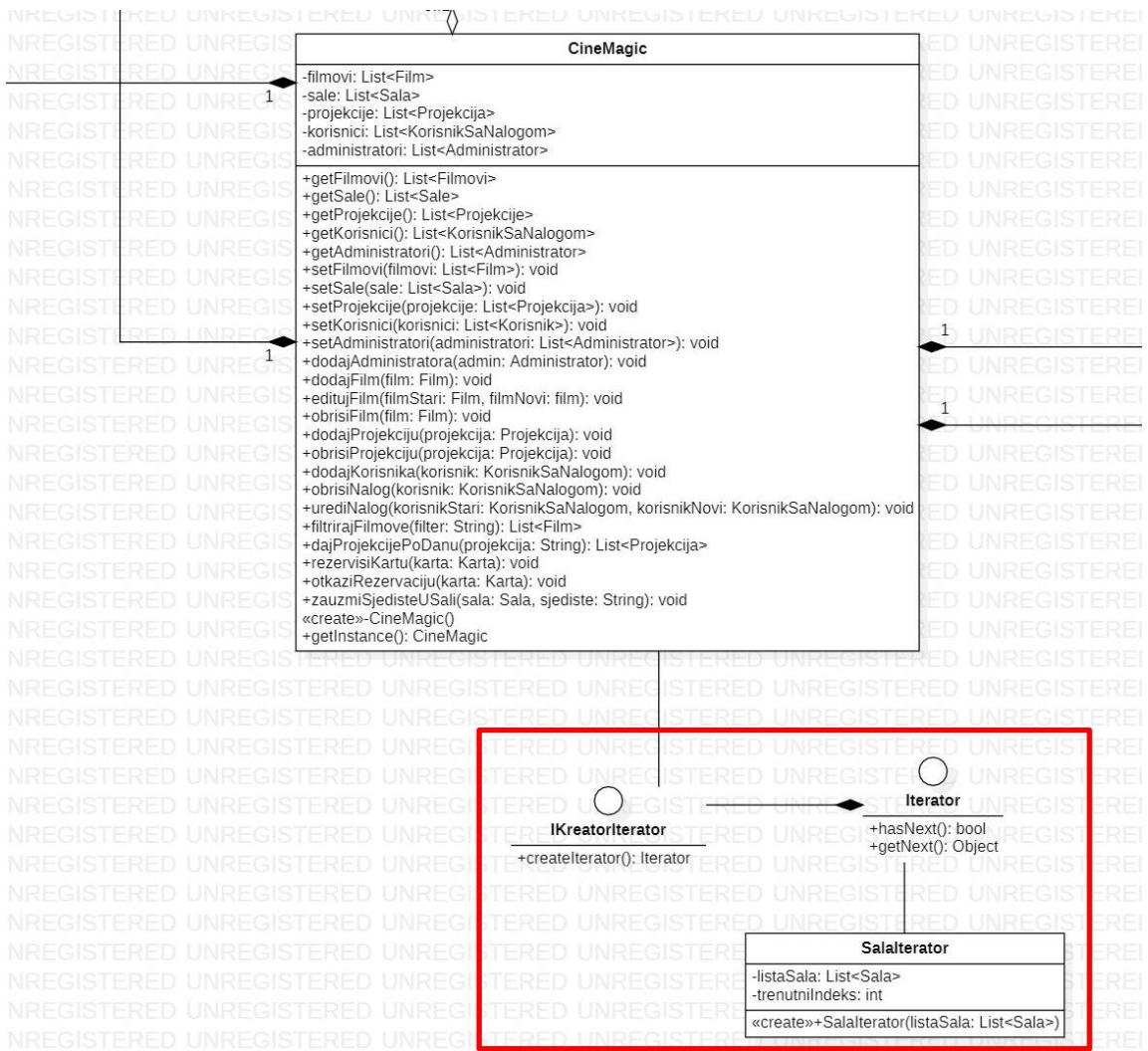
#### 4. Observer patern

Observer patern služi kako bi se na jednostavan način kreirao mehanizam pretplaćivanja. Pretplatnici dobivaju obavještenja o sadržajima na koje su pretplaćeni, a za slanje obavještenja zadužena je nadležna klasa. Na ovaj način uspostavlja se relacija između klasa kako bi se mogle prilagoditi međusobnim promjenama. Ovaj patern nismo implementirali, međutim prilika za implementaciju

ovog paterna bi bila kada bi imali nekog VIP korisnika, koji bi se mjesечно pretplaćivao na naš sajt. Pretplaćivanjem bi dobio obavijesti o svakom novom filmu koji je zakazan za prikazivanje. Za implementaciju nam je potreban interfejs *IObserver*, te klasa *VIPKorisnik* koja implementira taj interfejs. Interfejs će imati metodu *Update()* koja šalje obavijesti svim pretplaćenim tj VIP korisnicima. Dalje nam u klasi *CineMagic* treba lista VIP korisnika, zatim nam trebaju metode za dodavanje članova u tu listu i brisanje članova iz liste, te metoda *Notify()*. Također nam je potreban i privatni event *obavijestiVIPKorisnike()* u klasi *CineMagic*.

## 5. Iterator patern

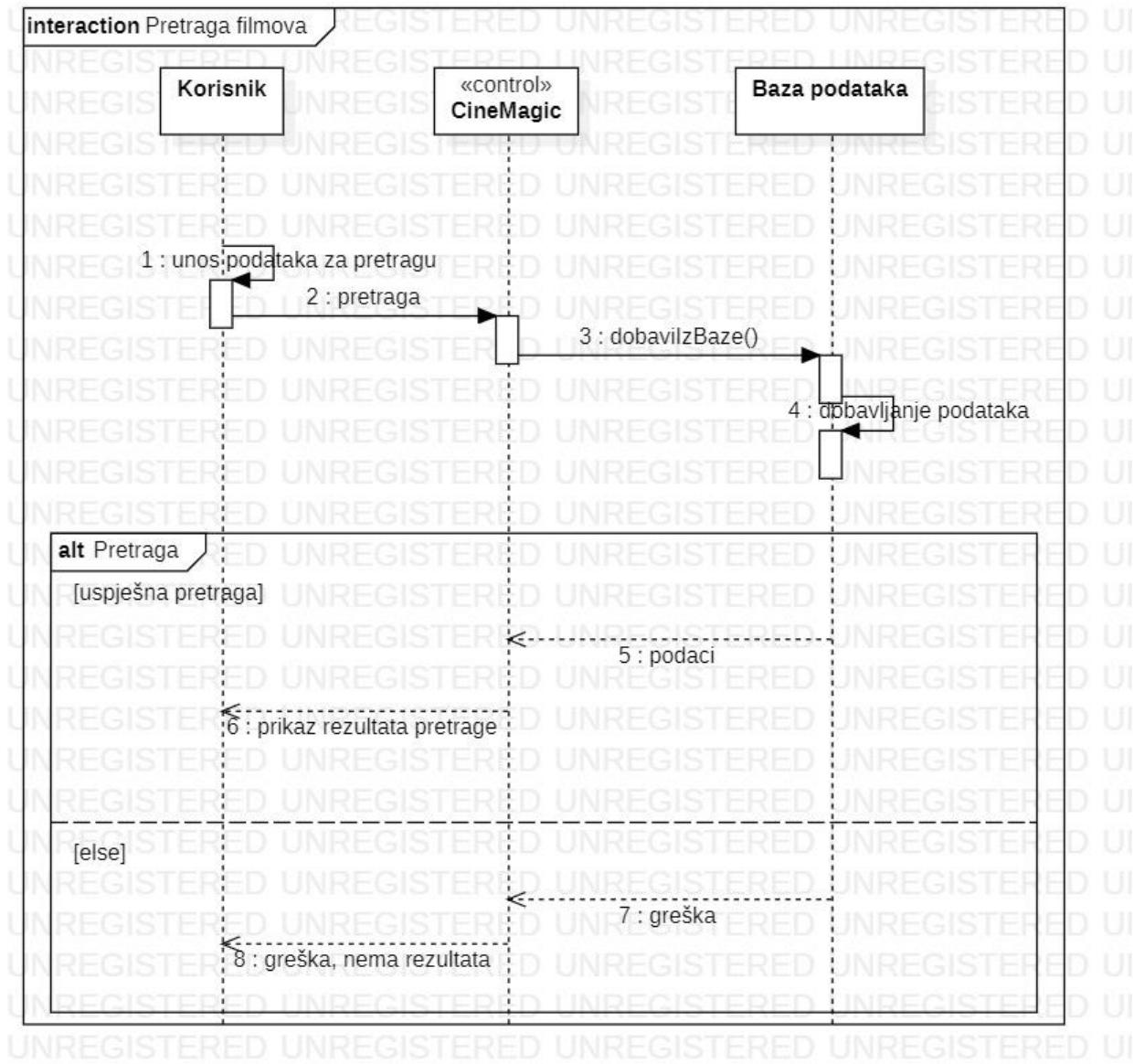
Iterator patern namijenjen je kako bi se omogućio prolazak kroz listu elemenata bez da je neophodno poznavati implementacijske detalje strukture u kojoj se čuvaju elementi liste. Izvedba liste može biti u obliku stabla, jednostrukih liste, niza i sl., no klijentu se omogućava da na jednostavan način dolazi do željenih elemenata. Osim toga, ovaj patern preporučljivo je iskoristiti kada se za iteriranje koristi kompleksna logika koja ovisi o više kriterija. Ovaj patern smo odlučili da implementiramo. Odlučili smo napraviti da pristupamo elementima kolekcije sala u klasi *CineMagic* preko iteratora. Da bi ovo postigli morali bi napraviti interfejs *IKreatorIterator*, koja kreira iterator, tj posjeduje metodu *createIterator()*, zatim bismo kreirali interfejs *Iterator* koji posjeduje dvije metode *hasNext()* i *getNext()*. Morali bismo i kreirati klasu *Salalterator*, koja kao atributima ima listu sala i atribut koji govori o trenutnom indeksu, ova klasa bi implementirala interfejs *Iterator*. Također bi u klasu *CineMagic* morali dodati još jedan atribut tipa *Iterator*, dok nam lista sala u klasi *CineMagic* ne bi bila potrebna.



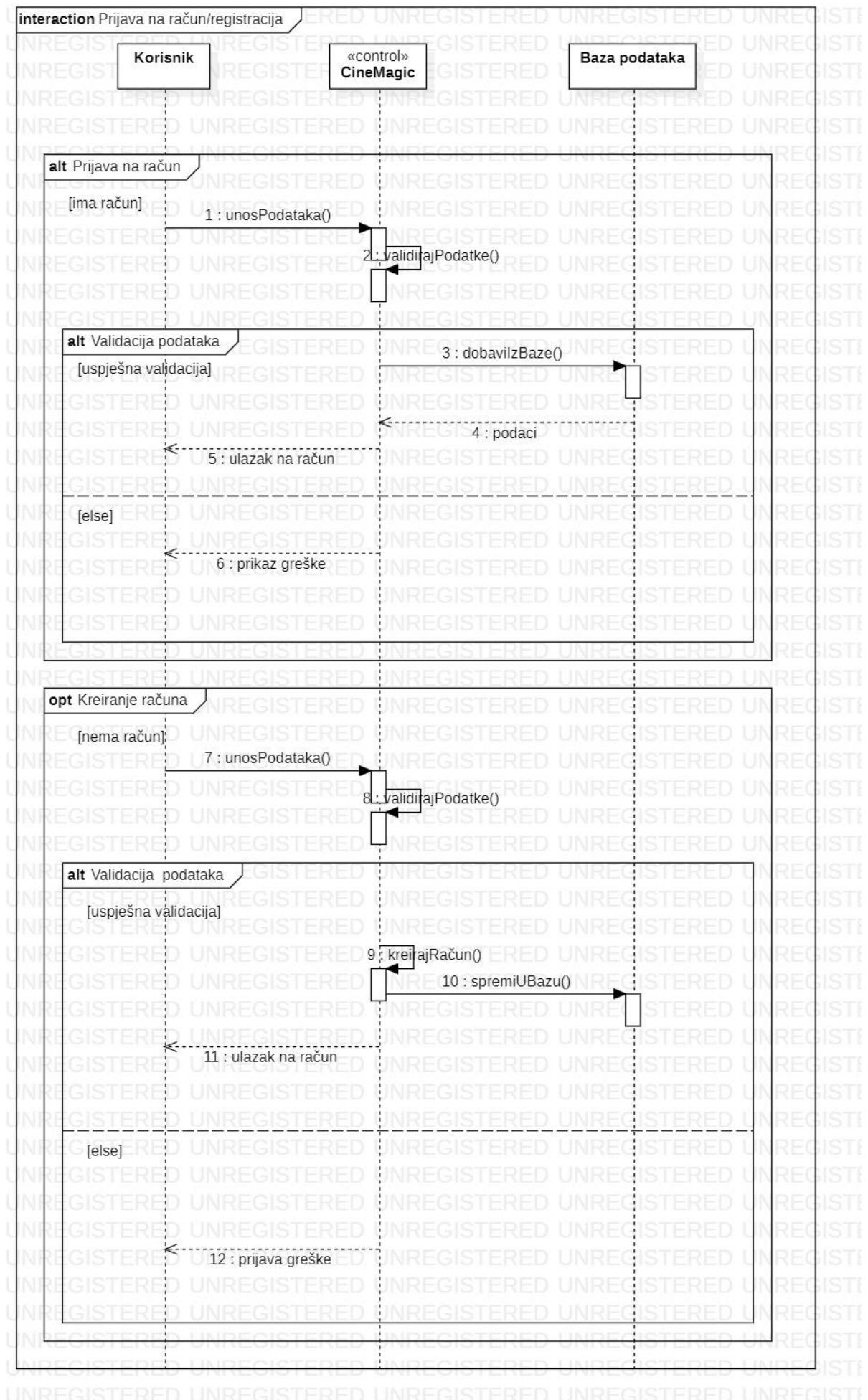
## 7.Sequence Diagram

Dijagrami sekvence definišu se na osnovu scenarija koji opisuje neki slučaj upotrebe (na osnovu čega se kreiraju i dijagrami aktivnosti). No, za razliku od dijagrama aktivnosti, dijagrami sekvence za cilj nemaju prikazivanje tokova događaja, već vremenskog izvršavanja akcija i poruka koje se šalju između učesnika u nekoj aktivnosti.

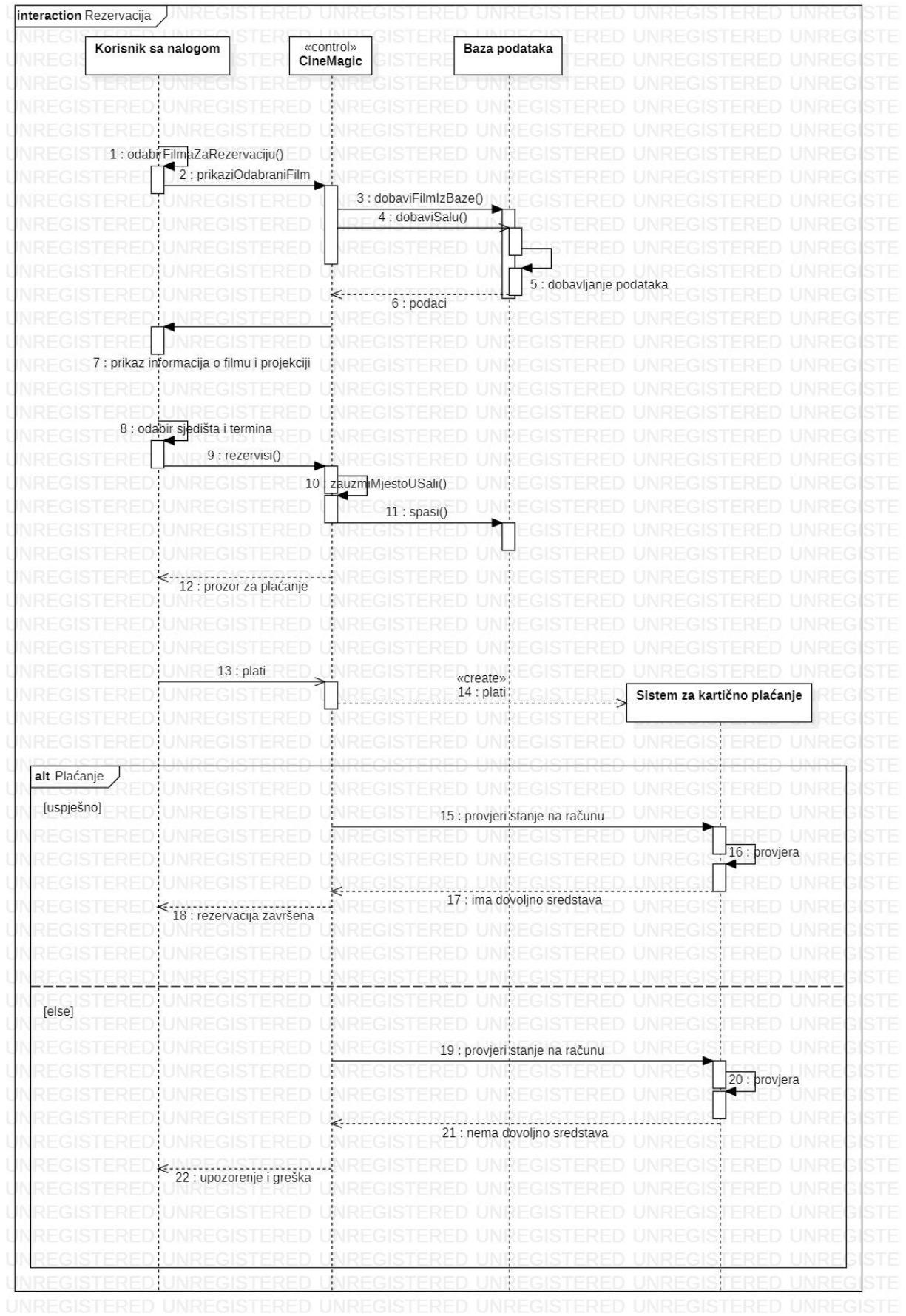
### a)Pretraga filmova



### b) Prijava na račun/registracija



**c) Rezervacija**



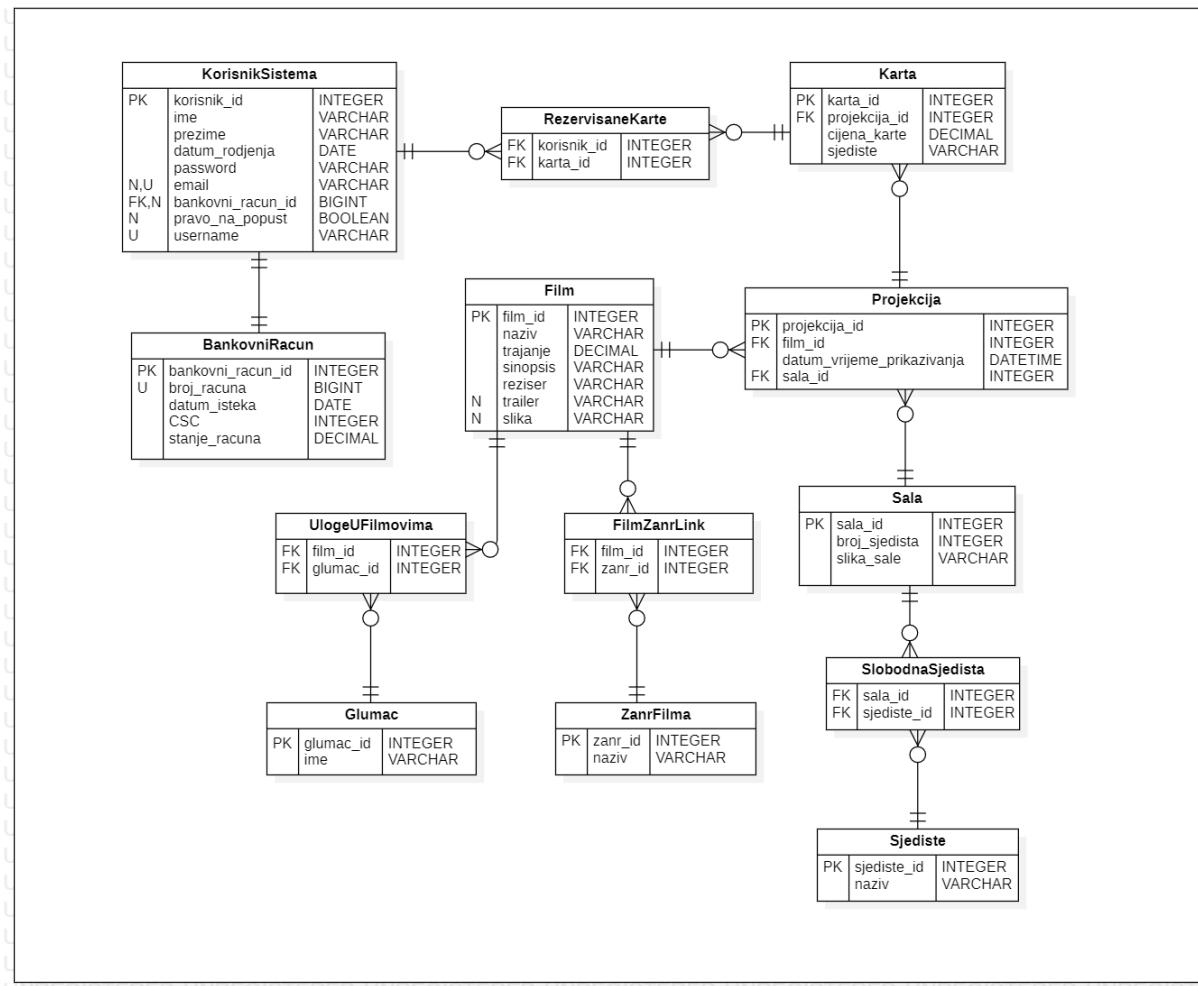
## 7.Entity Relationship Diagram

Detaljan logički prikaz podataka preko skupa entiteta, njihovih atributa i međusobnih veza.

Osnovni elementi modela su:

- entiteti (objekti)
- veze
- atributi

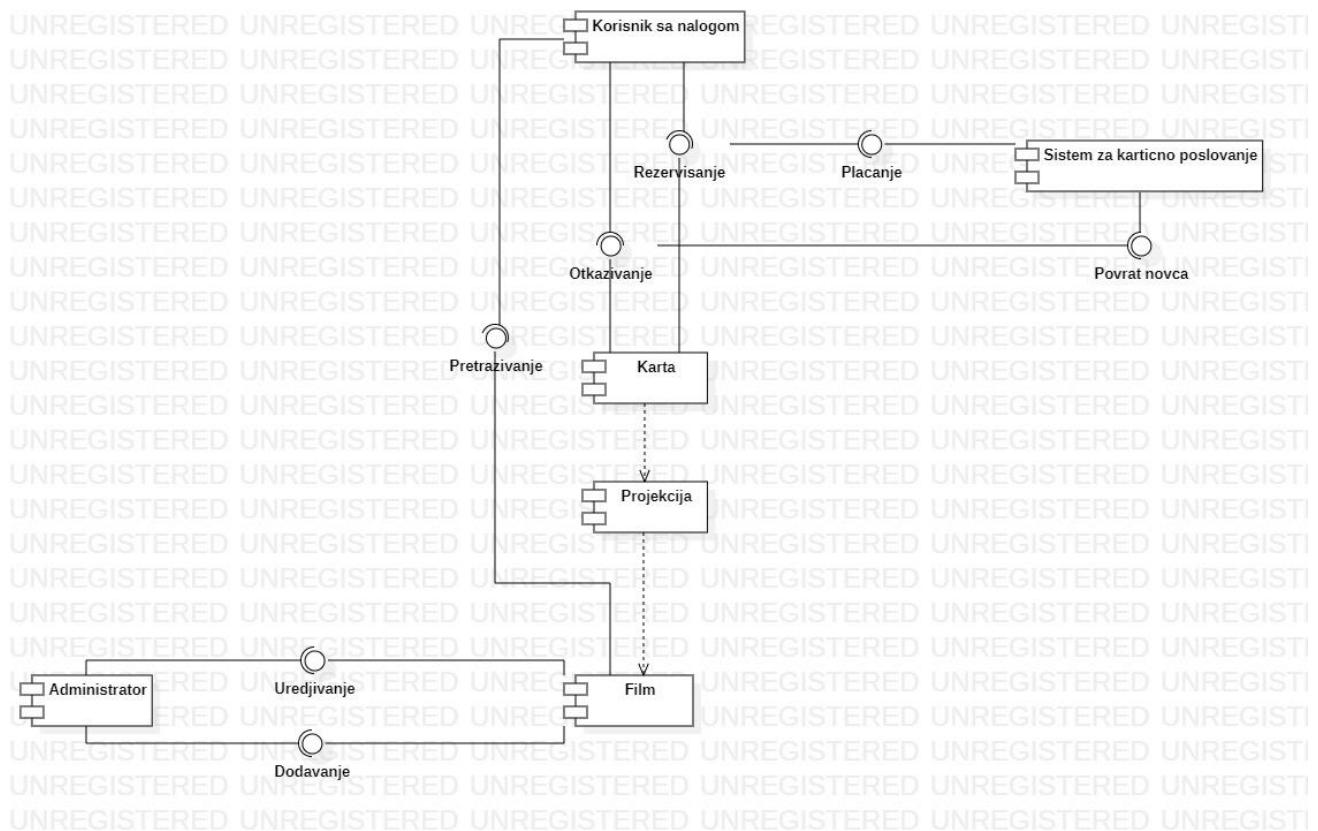
ER dijagram - grafički prikaz osnovnih elemenata ER modela.



## 8. Component diagram

Dijagrami komponenti služe kako bi se na višem nivou prikazao sam sistem i način na koji njegovi zasebni dijelovi komuniciraju. Zasebni dijelovi sistema nazivaju se komponentama i oni u pravilu trebaju biti atomični (odnosno nedjeljivi) kako bi, ukoliko dođe do potrebe njihovog mijenjanja, došlo do što manjeg utjecaja na druge dijelove sistema.

Na sljedećoj slici prikazan je dijagram komponenti za naš projekat



## 9. Composite Structure Diagram

Dijagram složene strukture se koristi za prikaz interne strukture neke klase, objekta ili slučaja upotrebe. Na ovoj slici je prikazan naš dijagram složene strukture.

