# Dizajn sistema: Multipleks

# Sadržaj

# 1. Uvod

## 1.1. Namjena sistema

Softverski proizvod je namijenjen za upravljanje radom Multipleksa, kao i za vođenje statistike o važnijim dokumentima u istom. Predstavljen je korisnicima putem desktop aplikacije koja se adaptira datom korisniku. Samim tim svaka grupa zaposlenih ima logički razdvojen softverski podsistem. Pristup podsistemu je realizovan putem korisničkog imena i lozinke.

Kako je ovaj sistem prvenstveno namijenjen uslugama koje pruža jedan Multipleks, on ne sadrži usluge koje pruža knjigovodstveni/bankarski softver. Kao takav, omogućuje samo pomoć pri vođenju evidencije o transakcijama, računima, fakturama i ponudama.

Fiskalizacija računa se radi naknadno i realizovana je od strane ovlaštenog lica koje bi trebalo ovaj softverski sistem prilagoditi funkcionalnosti fiskalne obrade.

Detaljniji opis namjene sistema, te grupa korisnika detaljno je opisan u Specifikaciji softverskih zahtjeva.

## 1.2. Projektni ciljevi

Osnovni cilj realizacije softverskog sistema, odnosno projekta, jeste olakšavanje rada i ubrzanje radnih aktivnosti Multipleksa kojem je softverski proizvod i namijenjen. Imajući ovo u vidu, jasno je da softverski proizvod mora biti jednostavan za korišćenje ali i dovoljno brz i efikasan pri izvođenju akcija. Takođe, kako Multipleks ima različite vrste, kako uslužnih tako i administrativnih radnika, sigurnost i razdvajanje informacija dostupnih radnicima se postavljaju kao veoma važan aspekt pri projektovanju.

Oslanjajući se na najvažnije zahtjeve datog sistema izdvajaju se najvažniji, tj. osnovni, ciljevi za realizaciju softverskog proizvoda, i to:

- Jednostavnost korišćenja
- Sigurnost
- Skrivanje informacija od neovlaštenih lica
- Brzina (efikasnost)

Pored gore navedenih ciljeva važno je navesti još neke ciljeve koji ne mogu ostati nepomenuti, kao što su jednostavnost održavanja te izmjene softverskog proizvoda, otpornost na otkaze, nedvosmislenost podataka (konzistentnost), lakoća nadogradnje, jednostavno prilagođenje novom hardveru itd.

## 1.4. Referentni dokumenti


- D. Đurić, A. Vujinović, O. Ristić, S. Lekanić, M. Danilović: *Specifikacija softverskih zahtjeva – Softver za Multipleks*
- B. Bruegge, A.H. Dutoit: *Object-Oriented Software Engineering Using UML, Patterns, and Java*
- E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Design Patterns: Elements of Reusable Object-Oriented Software*
- T. Stahl, et al.: *Model-Driven Software Development*
- P. Clements, L. Northrop: *Software Product Lines: Practices and Patterns*

# 2. Arhitektura postojećeg sistema

Budući da nemamo postojeći sistem (eng. *system as is*), u ovom poglavlju razmatramo alternativne arhitekture, kao i potencijalnu arhitekturu sistema u budućnosti (eng. *system to be next*). Alternativna arhitektura sistema za Multipleks, u odnosu na arhitekturu Model–View–Controller ( razmotrena u poglavlju 3.1.) bi bila troslojna arhitektura. Razlozi zbog kojeg je odabrana arhitektura Model–View–Controller (dalje kao MVC), kao i neke od razlika između arhitektura su sljedeće:

- Namjena ovih arhitektura se razlikuje. Troslojna arhitektura je namijenjena za velike (eng. *enterprise*) aplikacije, dok MVC arhitektura za male aplikacije, gdje je vrijeme razvoja ključan faktor. Budući da je naša aplikacija malog obima, MVC arhitektura ima prednost.
- Troslojna arhitektura je više orijentisana na podjelu komponenata po hardverskim čvorovima (svaki sloj ove arhitekture je raspoređen na jedan hardverski čvor), dok je MVC arhitektura više orijentisana na podjelu programskog koda na cjeline čija se implementacija može dodijeliti jednom programeru ili jednom razvojnom timu. Budući da je nama u interesu, podjela programskog koda na način da svaki član tima razvija podsistem koji je on identifikovao tokom specifikacije zahtjeva, MVC arhitektura ima prednost. Takođe, MVC arhitektura dozvoljava da više članova tima može da razvija komponente jednog podsistema, ako za tim ima potrebe (npr. za dati podsistem, jedna osoba bi mogla da razvije pogled, druga kontroler, dok treća model), što je dodatni plus.
- MVC arhitektura podrazumijeva će se, tokom izdavanja novih verzija aplikacije, interfejs mnogo češće mijenjati nego aplikativna logika, dok aplikativna logika češće nego domenski objekti, što odgovara softveru za Multipleks. Kod troslojne arhitekture svaki sloj se posmatra ravnopravno.
- MVC arhitektura omogućava zavisnost između *modela* i *pogleda*, tj. promjena u modelu, se reflektuje na korespodentnom pogledu. Kod troslojne arhitekture, za isti mehanizam, bi smo morali imati dodatnu komponentu da drugom sloju, koja bi služila kao medijator.
- MVC arhitektura sama po svojoj prirodi pruža nisku *spregu* i visoku *povezanost*, dok kod troslojne arhitekture to ne mora biti slučaj.

Takođe, bitno je napomenuti i domensko znanje razvojnog tima, koji se već susreo MVC arhitekturom, što dodatno smanjuje vrijeme implementacije sistema.

Razvoj arhitekture u budućnosti bi vjerovatno išao ka klijent–server arhitekturi, iz razloga što je distribuirani sistem (trenutno centralizovan) pogodniji za domen problema. U tom slučaju bi klijentu bio dostupan samo korisnički interfejs, dok bi sva aplikativna logika, kao i baza podataka bila implementirana na strani servera. Benefiti ovakvog pristupa su višestruki.

# 3. Predložena arhitektura

## 3.1 Kratak pregled arhitekture i funkcionalnosti podsistema

Sistem je podijeljen na šest podstistema:

1. Podsistem za rad sa posjetiocem, koji omogudava jednostavnu interakciju sa posjetiocem, u vidu prodaje i rezervacije karata, te prodaje prehrambenih proizvoda.

2. Podsistem za upravljanje artiklima, koji omogudava olakšano vođenje evidencije o artiklima u skladištu.

3. Podsistem za menadžment, koji pruža usluge ažuriranja, pregleda i dodavanja projekcija, filmova, sala i ostalih poslovnih jedinica vezanih za menadžment.

4. Podsistem za pravne poslove, koji uključuje administraciju zaposlenih u vidu pružanja osnovnih informacija o zaposlenom, koje se mogu mijenjati.

5. Podsistem za računovodstvo, koji uključuje pomod pri evidenciji faktura, transakcija i plata zaposlenih u Multipleksu.

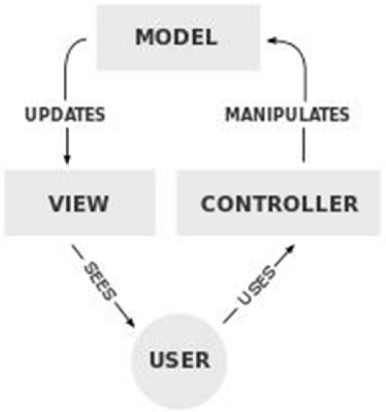
6. Podsistem za rad sa filmovima i kinoopremom, koji pruža usluge ažuriranja, pregleda i dodavanja opreme kao i pregled filmova i projekcija.

Detaljniji opis funkcionalnosti podsistema se može naći u Specifikaciji softverskih zahtjeva.

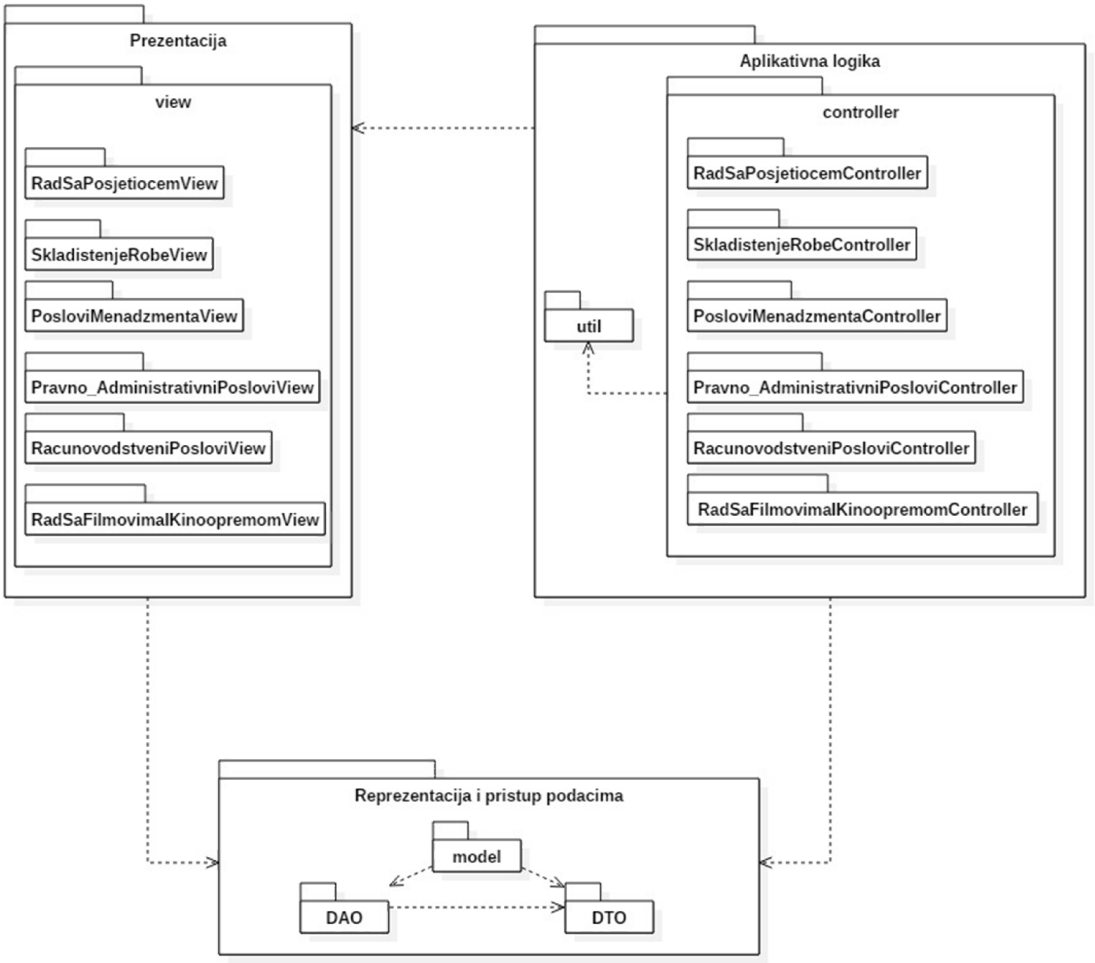
Arhitektura svih podsistema, kao i cijelog sistema, je zamišljena u MVC arhitekturnom stilu. MVC (Model-View-Controller) je arhitekturni stil koji razdvaja aplikativnu logiku, odnosno način rada sistema, od prezentacije sistema korisniku te načina na koji su predstavljeni podaci u sistemu.

Model predstavlja način na koji su organizovani podaci u sistemu, odnosno organizaciju i definiciju klasa tj. domenskih objekata u sistemu. View (pogled) predstavlja način prikaza funkcionalnosti sistema korisniku. Ovo je realizovano putem korisničkog interfejsa (UI - User Interface) čiji je najčešći predstavnik grafički korisnički interfejs (GUI - Graphical User Interface) koji predstavlja funkcionalnosti sistema putem formi za prikaz, te nizovima labela, tekstualnih polja, tebela i dugmadi. Controller (kontrolor) predstavlja aplikativnu logiku koja se izvršava u zavisnosti od pritisnutog dugmeta, unešenog teksta ili sličnih akcija nad pogledom. Kontrolor u suštini predstavlja vezu između pogleda i modela, odnosno vrši manipulaciju objekata u modelu na osnovu akcije izabrane iz pogleda (korisničkog interfejsa).

Način na koji korisnik interaguje sa sistemom koji ima MVC arhitekturu dat je na sljedećoj slici:




## 3.2. Dekompozicija sistema (eng. System decomposition)



Dekompozicija sistema je prikazana na slici. Osnovni šablon Model-View-Controller arhitekturnog stila je proširen podsistemima koji se nalaze unutar *view* i *controller* podsistema, u čijim nazivima imamo prefikse koji označavaju podsisteme[1] na koje se odnose dati pogledi i kontroleri. Podsistem *model*, nije dekomponovan na dodatne podsisteme, a sadrži klase navedene u dijagramu klasa[2], kao i okružujuće klase (eng. wrapper class) koje se koriste za pristup bazi podataka.

*Sprega* (eng. coupling) između podisistema je niska, tj. imamo nizak nivo sprege između modela, kontrolera i pogleda, dok veze između novouvedenih podsistema ne postoje, što predstavlja olakšicu pri implementaciji jednog podsistema – implementacija jednog podsistema može biti dodijeljena članu tima koji je prethodno radio na specifikaciji zahtjeva za dati podsistem.

*Povezanost* (eng. coherence) nije prikazana ovom slikom, ali možemo reći da je visoka jer sve klase koje nisu posljedica arhitekturnog stila su grupisane prema odgovarajućim podsistemima sa već prethodno određenim asocijacijama, izvršavaju slične poslove, i nemaju interakciju s drugim podsistemima, dok visoku povezanost klasa koje će nastati kao rezultat arhitekturnog stila, obezbeđuje sam arhitekturni stil.

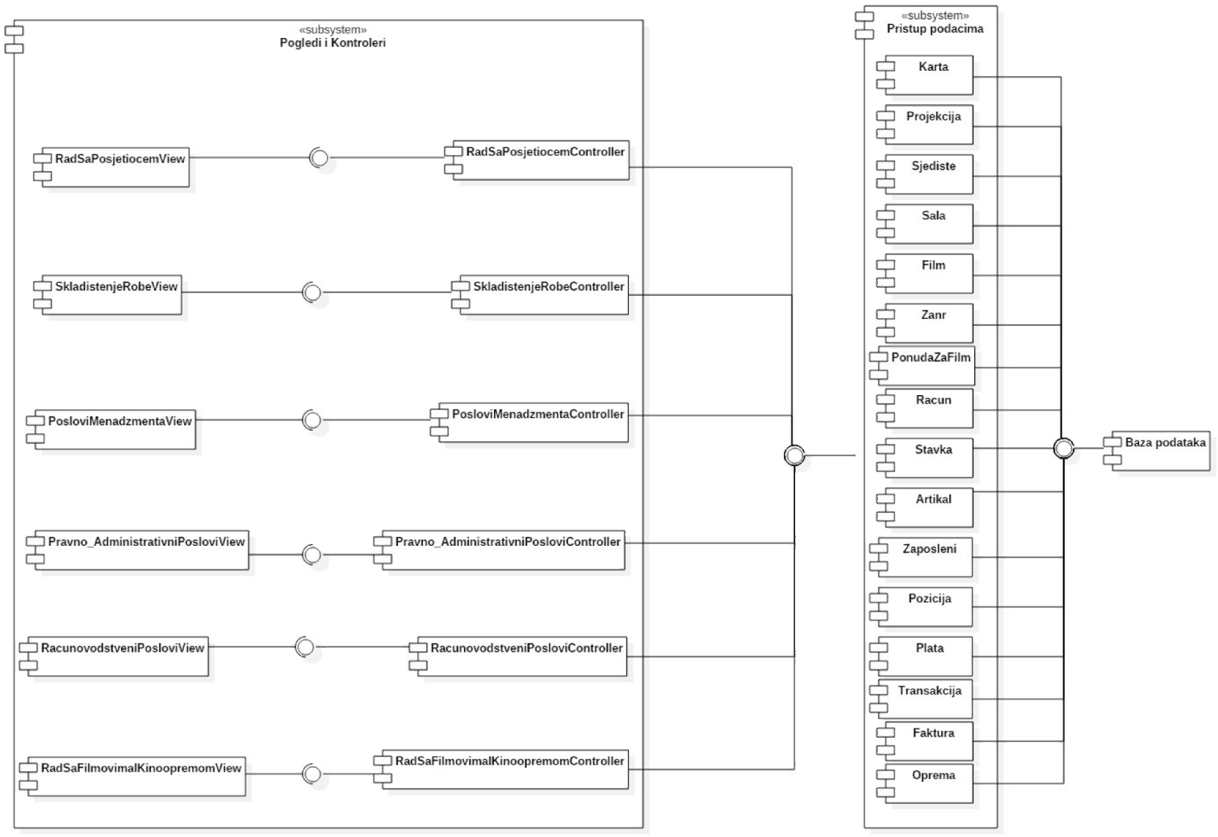
Budući da smo ostvarili nisku spregu i visoku povezanost, osnovni cilj dekompozicije je ostvaren, tj. redukcija složenosti.

[1] Odnosi se na podsisteme navedene u SRS dokumentu (Specifikaciji softverskih zahtjeva: Softver za Multipleks)

[2] Dijagram klasa se nalazi u SRS dokumentu

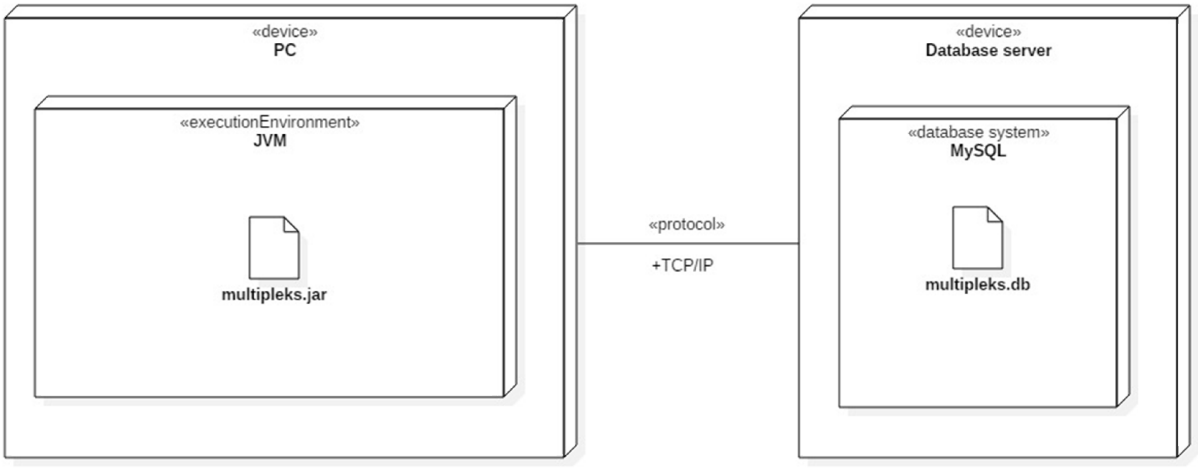
## 3.3. HW/SW mapiranje

### 3.3.1. Dijagram komponenti (eng. Component diagram)




Na osnovu dijagrama dekompozicije, navedenom u prethodnom poglavlju, prikazani podsistemi se mapiraju u komponente softverskog sistema. Ovaj prikaz i dalje odražava arhitekturni stil *Model–Pogled–Kontroler*. Dijagram komponenti je prikazan na slici. Svaka komponenta koja predstavlja kontroler pruža interfejs prema njemu korespodentnom pogledu, kao i što komponente podsistema *Pristup podacima* pružaju odgovorajući interfejs pogledima i kontrolerima. Dalje, baza podataka pruža odgovarajući interfejs komponentama koje se nalaze u podsistemu *Pristup podacima.*

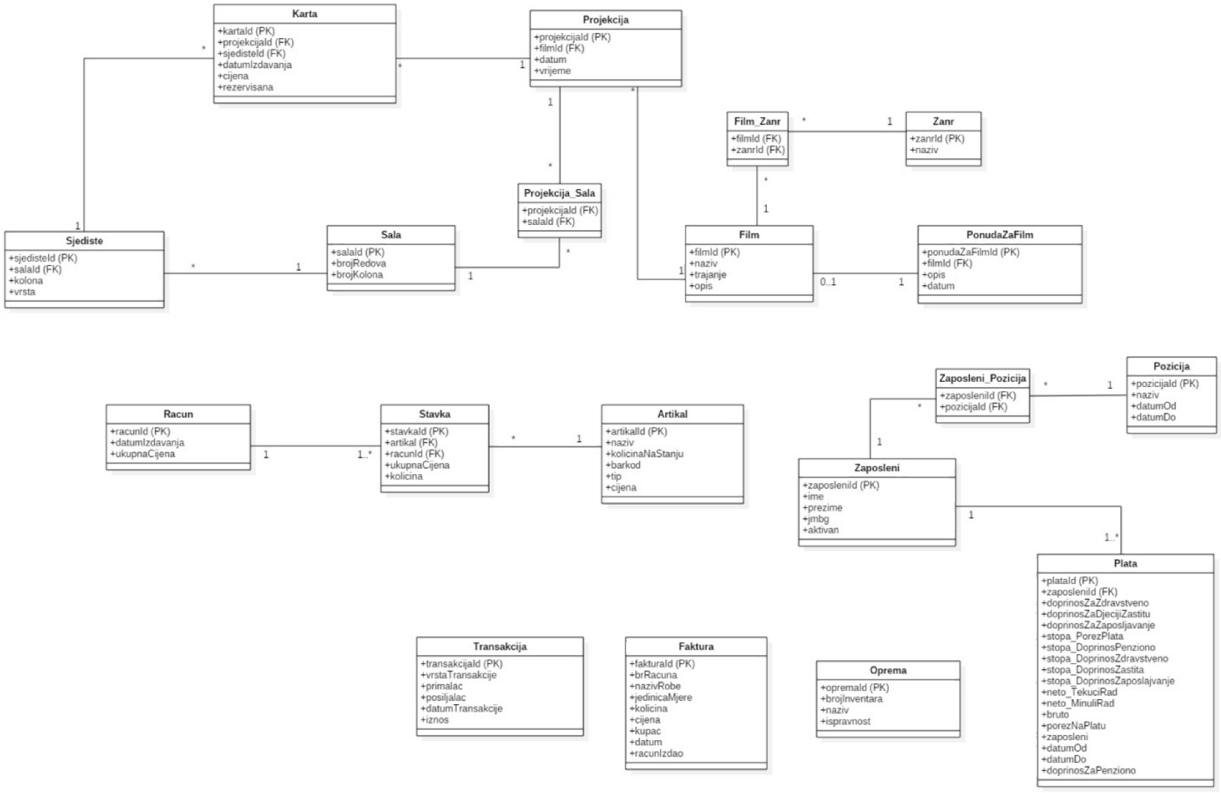
### 3.3.2. Dijagram razmještaja (eng. deployment diagram)




Dijagram razmještaja komponenti prikazan na slici 1 (ubaciti sliku) je zamišljen na sljedeći način: Baza podataka će se nalaziti na jednom (centralnom) uređaju – serveru baze podataka. Odabrani sistem za upravljanje bazom podataka je *MySQL*, i njegova je uloga je da opslužuje zahtjeve za podacima koji će dolaziti sa drugih uređaja (personalnih računara), na kojima očekujemo postojanje *Java virtuelne mašine*, radi izvršavanja naše aplikacije. Komunikacija između navedenih uređaja će se odvijati u lokalnoj računarskoj mreži, najvjerovatnije korištenjem *TCP/IP* protokola koji obezbjeđuje pouzdan prenos podataka preko mreže.

## 3.4. Perzistentni sloj

Zamišljeno je da perzistentni sloj softverskog proizvoda bude realizovan preko baze podataka, odnosno da se svi objekti (podaci) čuvaju pomoću sistema za upravljanje bazom podataka. Konceptualna šema baze podataka je data na sljedećoj slici:



## 3.5. Prava pristupa

Prava pristupa su prikazana po klasama, i razdvojena su po listama prava pristupa, tako da se zna koji korisnici imaju pravo pristupa kojim operacijama u datoj klasi.

***Klasa KartaController:***

| **Učesnik** | **Operacija** |
|---|---|
| Prodavac karata | dodaj(projekcija: Projekcija): Boolean |
| Prodavac karata | dodaj(sala: Sala): Boolean |
| Prodavac karata | pretraziRezervaciju(unos: String): Boolean |
| Prodavac karata | sacuvaj(): Boolean |

***Klasa ProjekcijaController:***

| **Učesnik** | **Operacija** |
|---|---|
| Menadžer | pretrazi (unos: String): Projekcija |
| Menadžer | getListaProjekcija(): Projekcija [*] |
| Menadžer | sacuvaj(): Boolean |
| Kinooperater | getListaProjekcija(): Projekcija [*] |

***Klasa SalaController:***

| **Učesnik** | **Operacija** |
|---|---|
| Menadžer | pretrazi (unos: String): Sala |
| Menadžer | getListaSala(): Sala [*] |
| Menadžer | sacuvaj(): Boolean |

***Klasa FilmController:***

| **Učesnik** | **Operacija** |
|---|---|
| Menadžer | pretrazi (unos: String): Film |
| Menadžer | getListaFilmova(): Film [*] |
| Menadžer | sacuvaj(): Boolean |
| Kinooperater | getListaFilmova(): Film [*] |

***Klasa PonudaZaFilmController:***

| **Učesnik** | **Operacija** |
|---|---|
| Menadžer | getListaPonuda(): PonudaZaFilm [*] |
| Menadžer | sacuvaj(): Boolean |

***Klasa FilmController:***

| **Učesnik** | **Operacija** |
|---|---|
| Prodavac pića i hrane | dodaj (artikal: Artikal): Boolean |
| Prodavac pića i hrane | sacuvaj(): Boolean |

***Klasa ArtikalController:***

| **Učesnik** | **Operacija** |
|---|---|
| Menadžer | pretrazi (unos: String): Artikal |
| Menadžer | getListaArtikala(): Artikal [*] |
| Menadžer | sacuvaj(): Boolean |
| Prodavac pića i hrane | getListaArtikala(): Artikal [*] |
| Prodavac pića i hrane | pretrazi (unos: String): Artikal |

***Klasa ZaposleniController:***

| **Učesnik** | **Operacija** |
|---|---|
| Menadžer | pretrazi (unos: String): Zaposleni |
| Menadžer | getListaZaposlenih (): Zaposleni [*] |
| Direktor | pretrazi (unos: String): Zaposleni |
| Direktor | getListaZaposlenih (): Zaposleni [*] |
| Računovođa | pretrazi (unos: String): Zaposleni |
| Računovođa | getListaZaposlenih (): Zaposleni [*] |
| Pravnik | getListaZaposlenih (): Zaposleni [*] |
| Pravnik | pretrazi (unos: String): Zaposleni |
| Pravnik | sacuvaj(): Boolean |

***Klasa PlataController:***

| **Učesnik** | **Operacija** |
|---|---|
| Računovođa | getListaPlata (): Plata [*] |
| Računovođa | pretrazi (unos: String):Plata |
| Računovođa | sacuvaj(): Boolean |

***Klasa TransakcijaController:***

| **Učesnik** | **Operacija** |
|---|---|
| Računovođa | getListaTransakcija (): Transakcija [*] |
| Računovođa | pretrazi (unos: String): Transakcija |
| Računovođa | sacuvaj(): Boolean |

***Klasa FakturaController:***

| **Učesnik** | **Operacija** |
|---|---|
| Računovođa | getListaFaktura (): Faktura [*] |
| Računovođa | pretrazi (unos: String): Faktura |
| Računovođa | sacuvaj(): Boolean |

***Klasa OpremaController:***

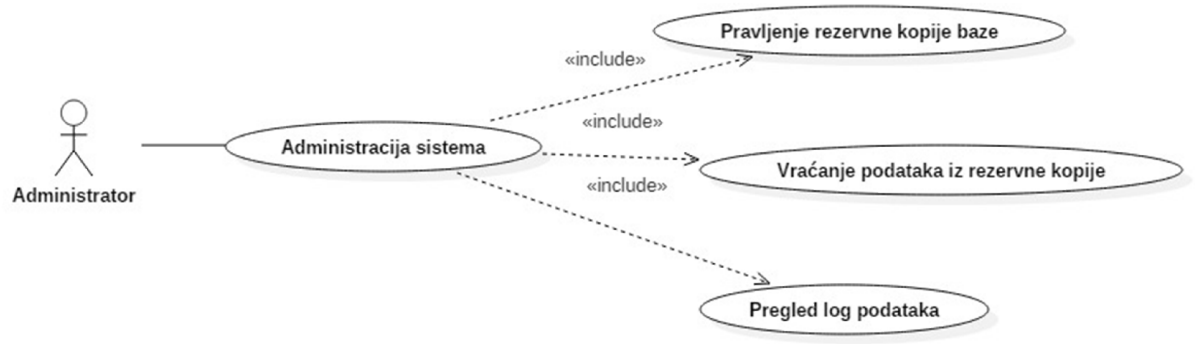
| **Učesnik** | **Operacija** |
|---|---|
| Kinooperater | getListaOpreme (): Oprema [*] |
| Kinooperater | pretrazi (unos: String): Oprema |
| Kinooperater | sacuvaj(): Boolean |

## 3.6. Konkurentnost

Kako svaki korisnik treba da posjeduje posebnu mašinu na kojoj će se izvršavati softverski proizvod, on nema konkurentnost u pravom smislu te riječi. Jedini problem koji se može pojaviti prilikom realizacije softvera jeste istovremeni pristup podacima koji su smješteni na centralizovanoj mašini koja posjeduje sistem za upravljanje bazom podataka. Imajući u vidu da je sistem za upravljanje bazom podataka realizovan sa mogućnošću istovremenog opsluživanja većeg broja korisnika ovaj problem se veoma lako riješava na strani baze podataka. Samim tim "sva" konkurentnost koju sadrži softver je realizovana na strani sistema za upravljanje bazom podataka koju proizvod sadrži.

## 3.7. Granični slučajevi upotrebe

Ispod je prikazan dijagram graničnih slučajeva upotrebe:




***Administrator*** ima ulogu administracije sistema, u vidu mogućnosti da iz samog sistema uradi sljedeće:

1. ***Pravljenje rezervne kopije podataka***, gdje administrator koji je trenutno spojen na sistem ima mogućnost da napravi lokalnu rezervnu kopiju podataka
2. ***Vraćanje podataka iz rezervne kopije***, gdje administrator koji je trenutno spojen na sistem ima mogućnost ažuriranja ili postavljanja rezervne kopije podataka na sistem
3. ***Pregled log podataka***, gdje administrator ima uvid u greške koje su nastale u sistemu, zarad izrade izvještaja. Uvidom u izvještaj, u slučaju greške u sistemu, koja može da se interpretira samo kao greška sistema, a ne greška korisnika, može se napraviti ispravka softvera.

## 3.8 Kontrola toka i organizacija sistema

Softverski sistem koji će biti realizovan karakteriše se centralizovanim oblikom kontrole toka. Kontrola toka predstavlja redoslijed izvršavanja akcija. Kod centralizovanog oblika kontrole toka samo jedan upravljački objekat ima kontrolu. Mehanizam koji koristi dati centralizovani sistem jeste mehanizam upravljanja događajima (event-driven). Ovaj mehanizam se zasniva na korisničkim akcijama, odnosno izvršavanjem kojim upravljaju događaji. Taj događaj može biti pomjeranje miša, klik miša, pritisak tipke na tastaturi, poruke drugih programa I tako dalje. Sva konkurentnost datog sistema je realizovana na nivou baze podataka kako je opisano u tački 3.6. Razvoj softverskog proizvoda je zamišljen korištenjem MVC (Model – View – Controller) šablona.