EBANK

Finalna dokumentacija modela

Sadržaj

Uvod

U modernom svijetu u kojem je novac pojam bez kojeg se ne može egzistirati, imamo potrebu da sve vezano za novac objedinimo u jedinstveni sistem – bankarski sistem. Aplikacija modernog bankarskog sistema omogućava obavljanje jednostavnih stvari kao što su operacije vezane za klijentovo bankovno stanje, podizanje sredstava u fizički novac (keš), depozit sredstava na račun, prebacivanje virtuelne vrijednosti stanja računa na drugi račun (transakcije), spremanje i čuvanje vrijednosti računa, itd. Osim osnovnih operacija, naša aplikacija ima veliki spektar dodatnih funkcionalnosti. Takve su one vezane za izdavanje kredita. Aplikacija implementira algoritam koji precizno određuje da li će klijent koji podiže kredit moći da ga vrati ili ne. Ta osobina klijenta se naziva kreditna sposobnost, a aplikacija posjeduje kalkulator koji je može izračunati na osnovu klijentovih podataka. Također, klijentima se omogućava da saznaju sve novosti banke, kao i promjene u sistemu koje se dešavaju jako često. Aplikacija je dostupna svim korisnicima uz prethodnu konsultaciju sa bankom, te registraciju. Ona olakšava komunikaciju sa bankom i omogućava klijentima da sa bilo kojeg mjesta, uz dostupnost interneta, mogu obaviti razne aktivnosti zbog kojih bi u suprotnom morali otići do poslovnice banke. Umjesto odlaska u poslovnicu i čekanja u redovima, koje je nekada otežano ili pak onemogućeno, klijenti sve mogu obaviti putem aplikacije, bez ikakve muke. Ovakva usluga omogućava jednostavno i efikasno upravljanje Vašim novcem, te siguran i brz pristup uz potpunu kontrolu nad sopstvenim finansijama. Doživite sve prednosti naših najmodernijih usluga!

Procesi

Klijent

- Konverzija deviza
- Vršenje uplata
- Slanje zahtjeva za kredit
- Pregled svih filijala i bankomata na mapi
- Pregled kontakt brojeva vezanih za banku
- Informisanje o novostima o banci

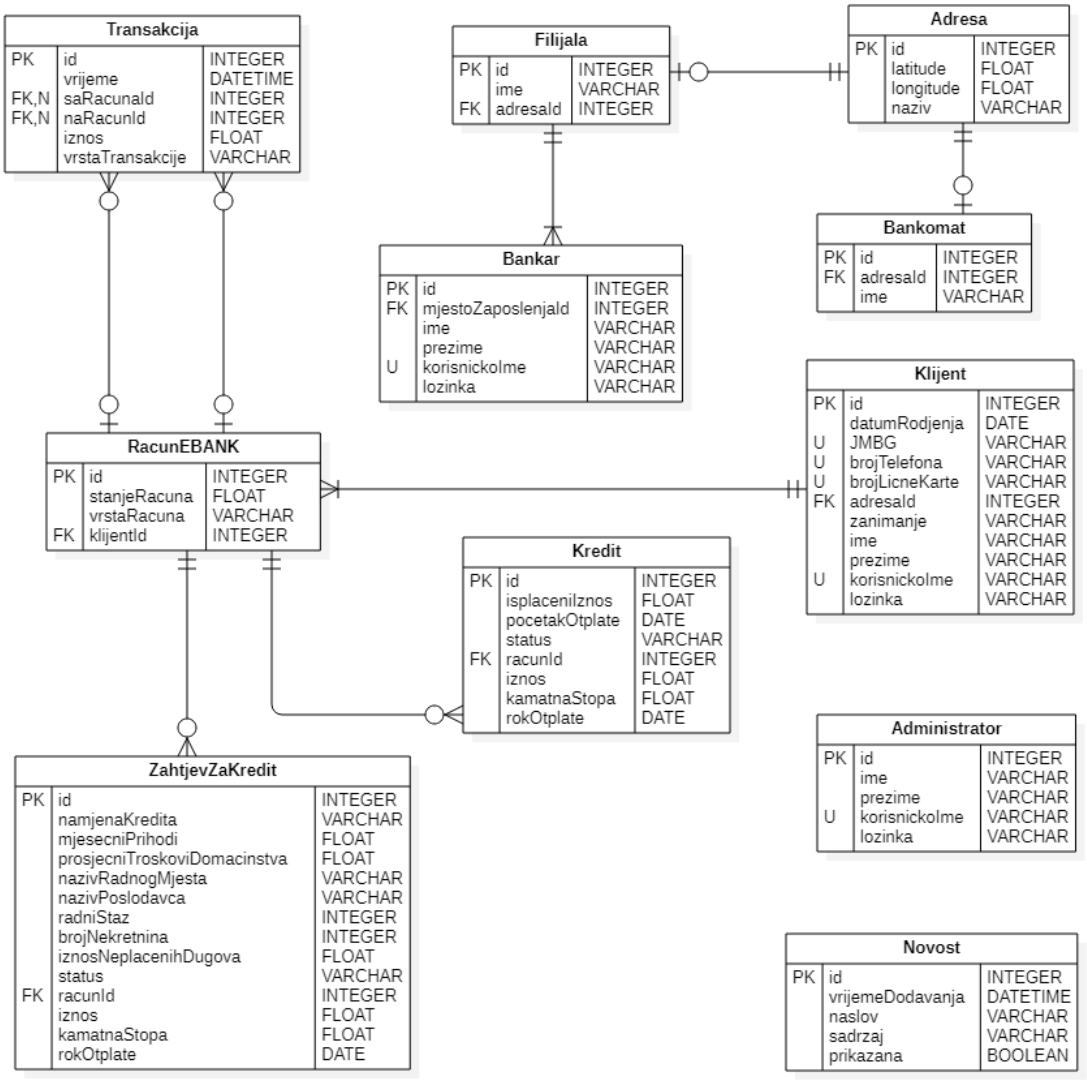
Bankar

- Dodavanje novih klijenata u sistem i brisanje starih
- Uređivanje podataka klijenata
- Otvaranje računa, pregledanje stanja, i zatvaranje istih
- Pregled transakcija
- Pregled, odobravanje i historiju kredita (uz algoritam za računanje kreditne sposobnosti)

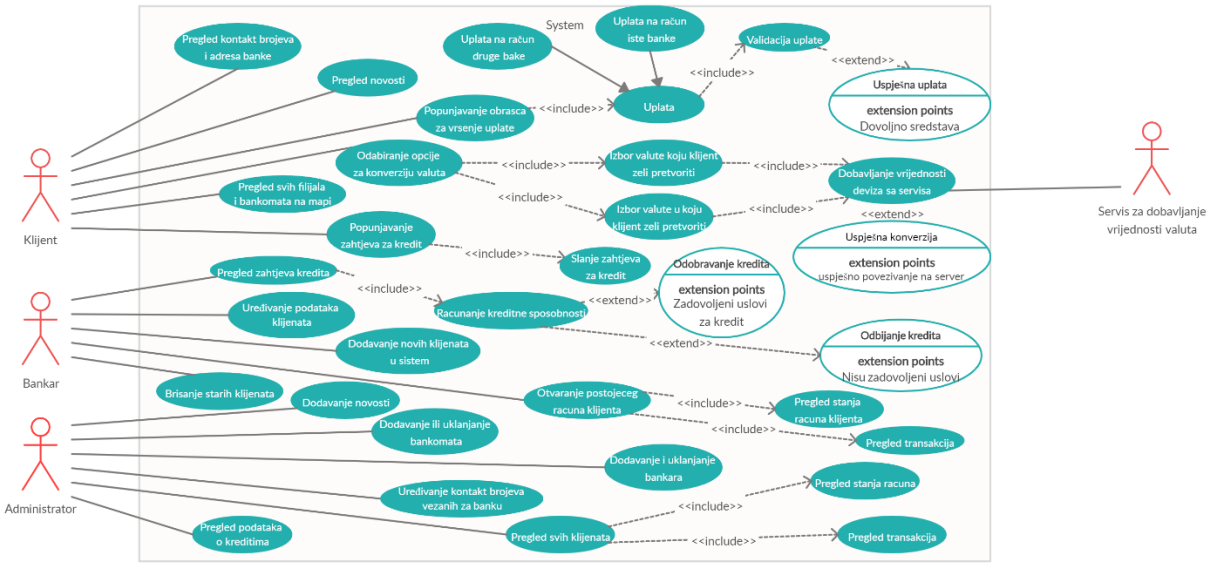
Administrator

- Dodavanje i uklanjanje uposlenika (bankara)
- Dodavanje, uklanjanje i uređivanje filijala i bankomata
- Pregled svih klijenata, njihovih računa, i transakcija između računa
- Pregled podataka o kreditima
- Uređivanje kontakt brojeva vezanih za banku
- Dodavanje novosti

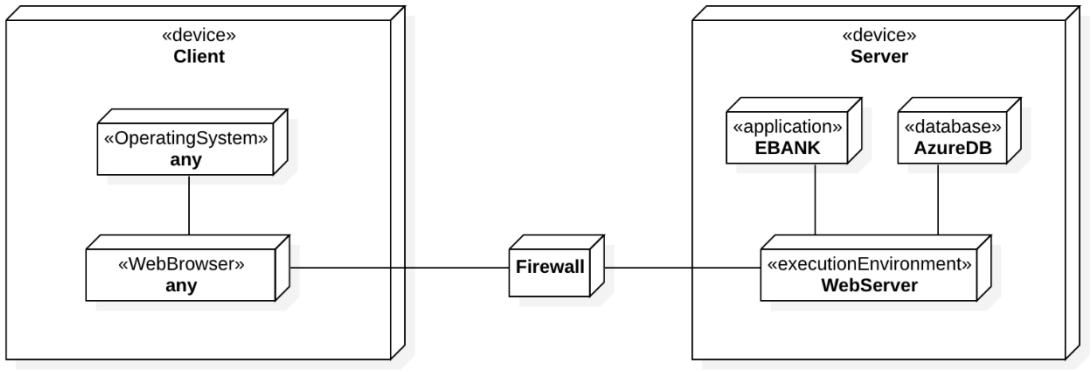
ER dijagram


Transakcija
PK id INTEGER
vrijeme DATETIME
FK,N saRacunaId INTEGER
FK,N naRacunId INTEGER
iznos FLOAT
vrstaTransakcije VARCHAR
Filijala
PK id INTEGER
ime VARCHAR
FK adresaId INTEGER
Adresa
PK id INTEGER
latitude FLOAT
longitude FLOAT
naziv VARCHAR
Bankomat
PK id INTEGER
FK adresaId INTEGER
ime VARCHAR
Bankar
PK id INTEGER
FK mjestoZaposlenjaId INTEGER
ime VARCHAR
prezime VARCHAR
U korisnickoIme VARCHAR
lozinka VARCHAR
Klijent
PK id INTEGER
datumRodjenja DATE
U JMBG VARCHAR
U brojTelefona VARCHAR
U brojLicneKarte VARCHAR
FK adresaId INTEGER
zanimanje VARCHAR
ime VARCHAR
prezime VARCHAR
U korisnickoIme VARCHAR
lozinka VARCHAR
RacunEBANK
PK id INTEGER
stanjeRacuna FLOAT
vrstaRacuna VARCHAR
FK klijentId INTEGER
Kredit
PK id INTEGER
isplaceniIznos FLOAT
pocetakOtplate DATE
status VARCHAR
FK racunId INTEGER
iznos FLOAT
kamatnaStopa FLOAT
rokOtplate DATE
Administrator
PK id INTEGER
ime VARCHAR
prezime VARCHAR
U korisnickoIme VARCHAR
lozinka VARCHAR
ZahtjevZaKredit
PK id INTEGER
namjenaKredita VARCHAR
mjesecniPrihodi FLOAT
prosjecniTroskoviDomacinstva FLOAT
nazivRadnogMjesta VARCHAR
nazivPoslodavca VARCHAR
radniStaz INTEGER
brojNekretnina INTEGER
iznosNeplacenihDugova FLOAT
status VARCHAR
FK racunId INTEGER
iznos FLOAT
kamatnaStopa FLOAT
rokOtplate DATE
Novost
PK id INTEGER
vrijemeDodavanja DATETIME
naslov VARCHAR
sadrzaj VARCHAR
prikazana BOOLEAN

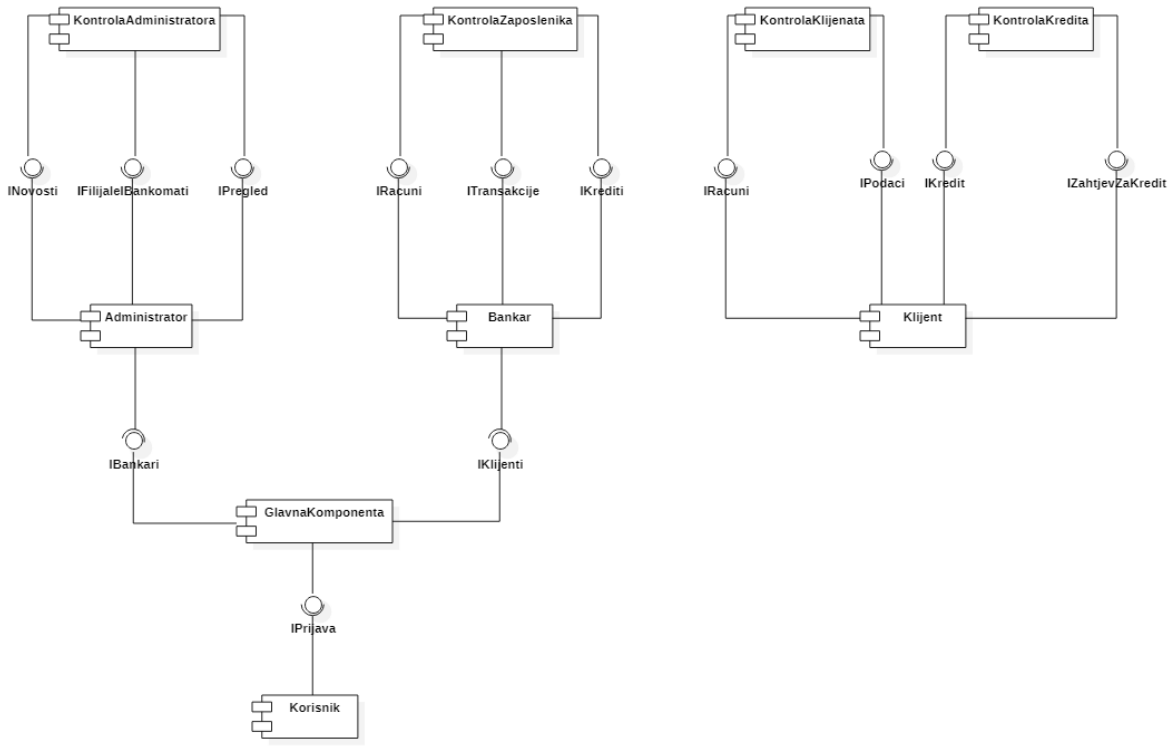
Use Case dijagram



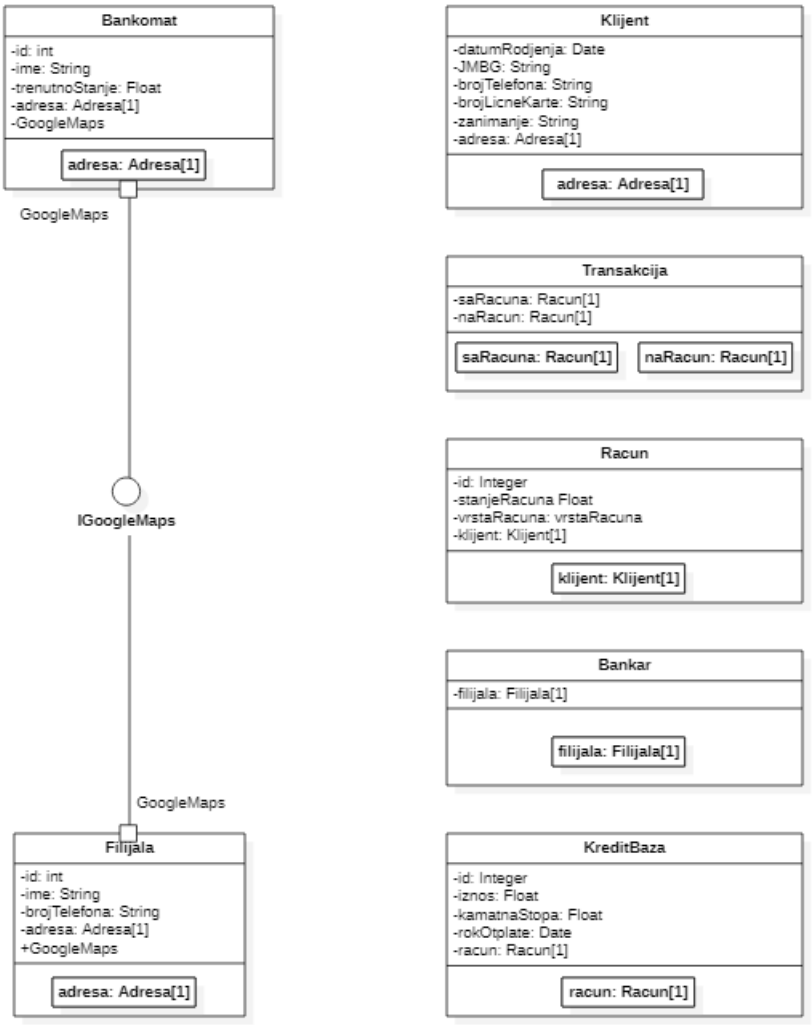
Dijagram raspoređivanja



Dijagram komponenti

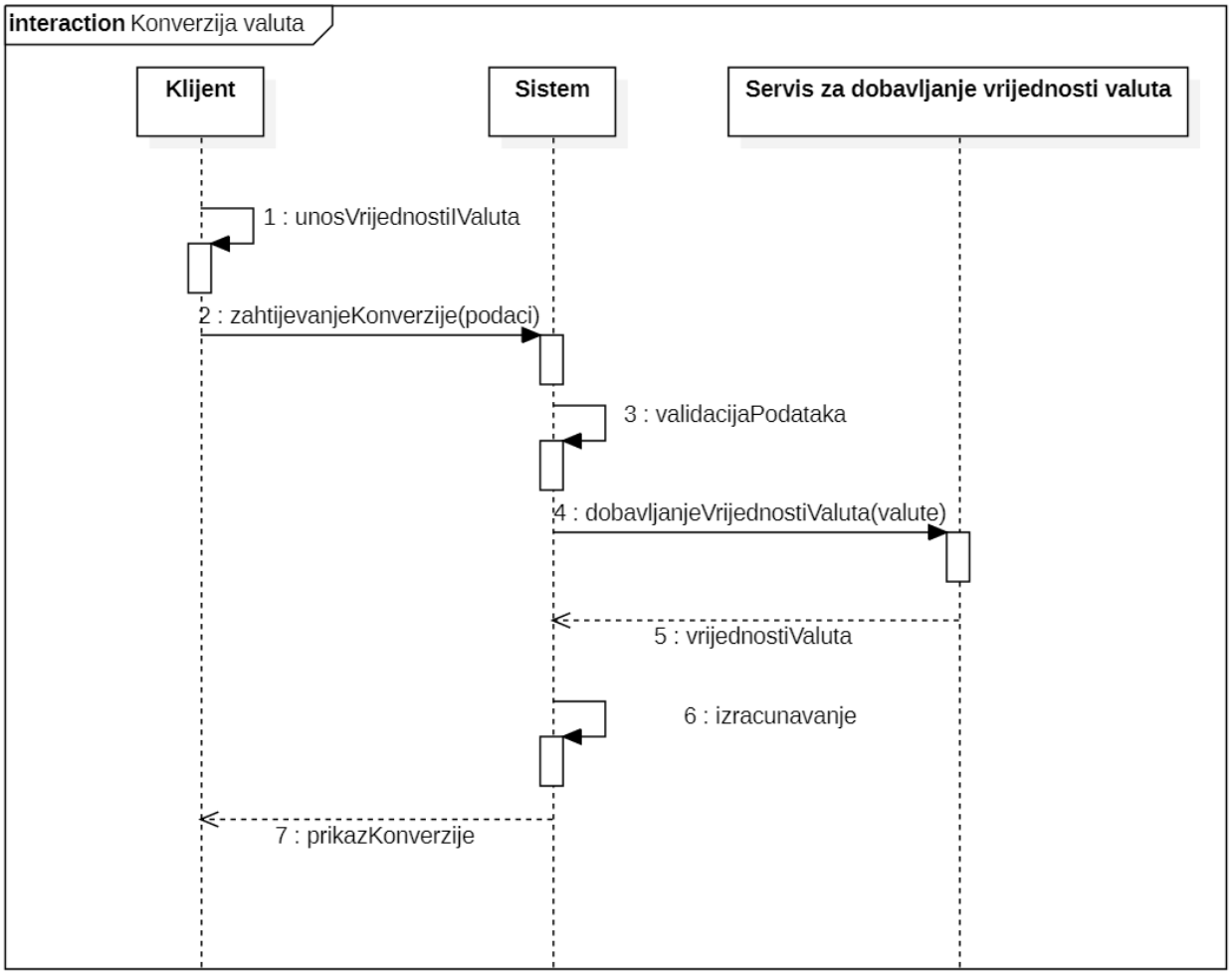


Dijagram složene strukture

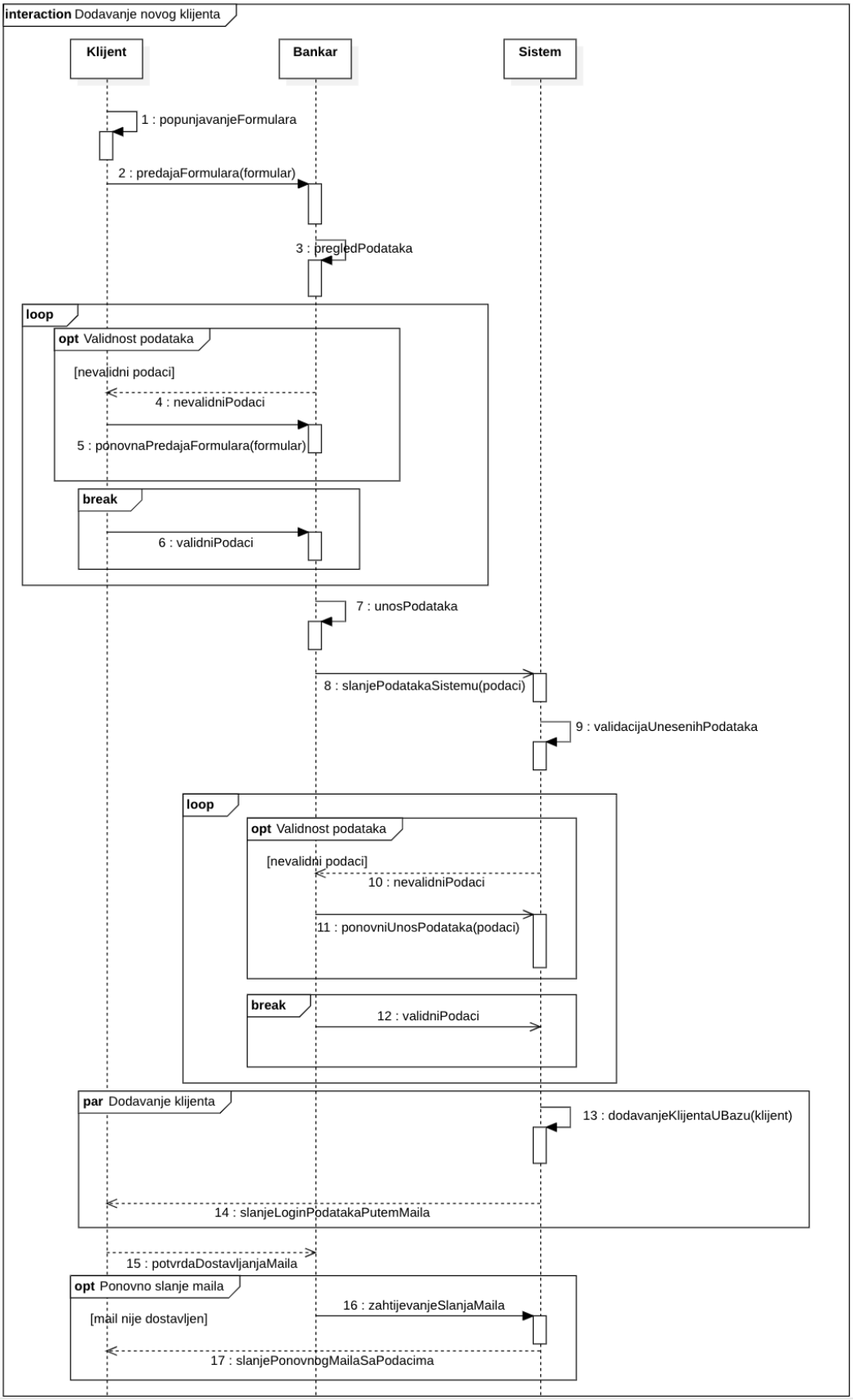


Dijagrami sekvenci

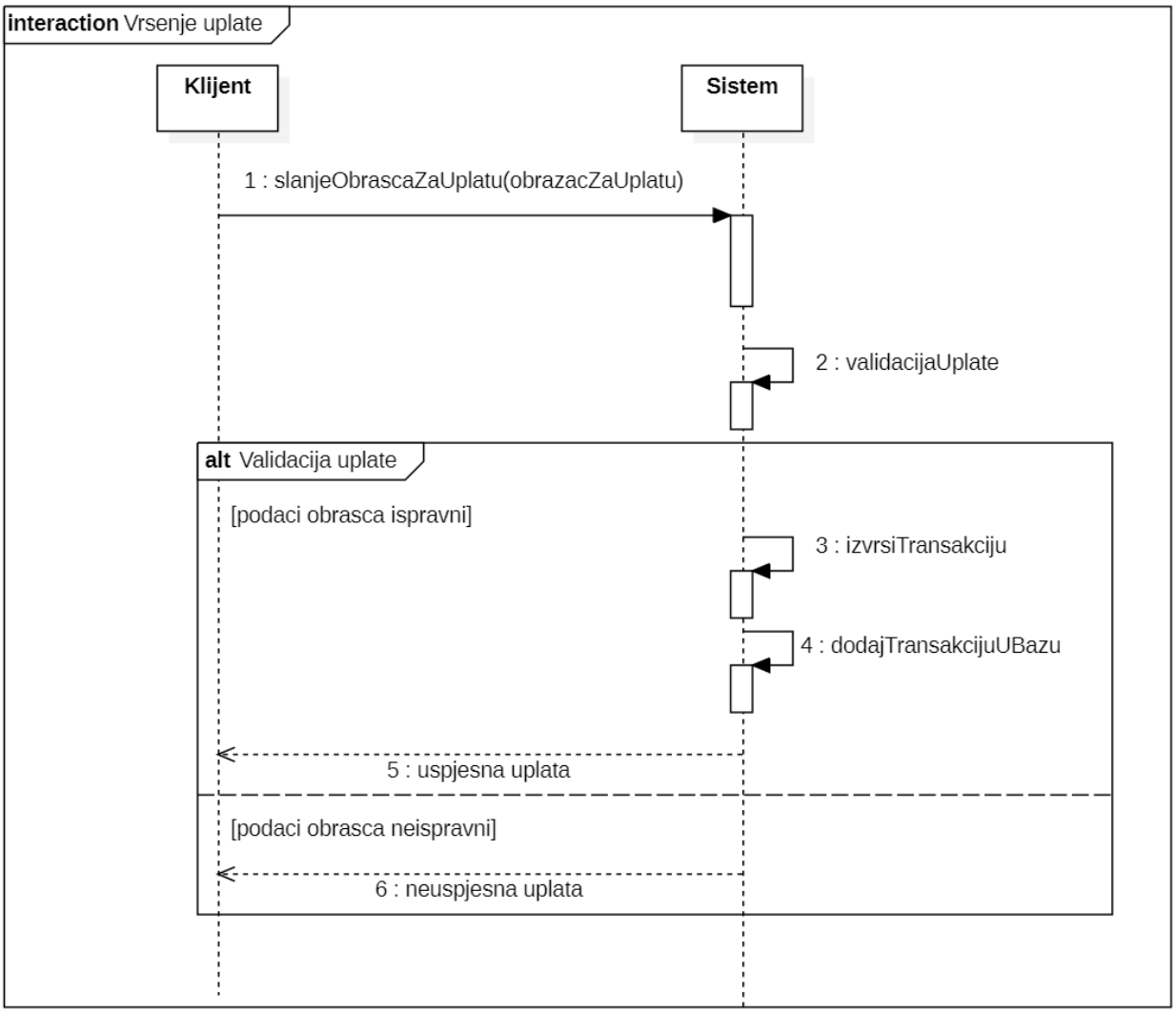
Konverzija valuta



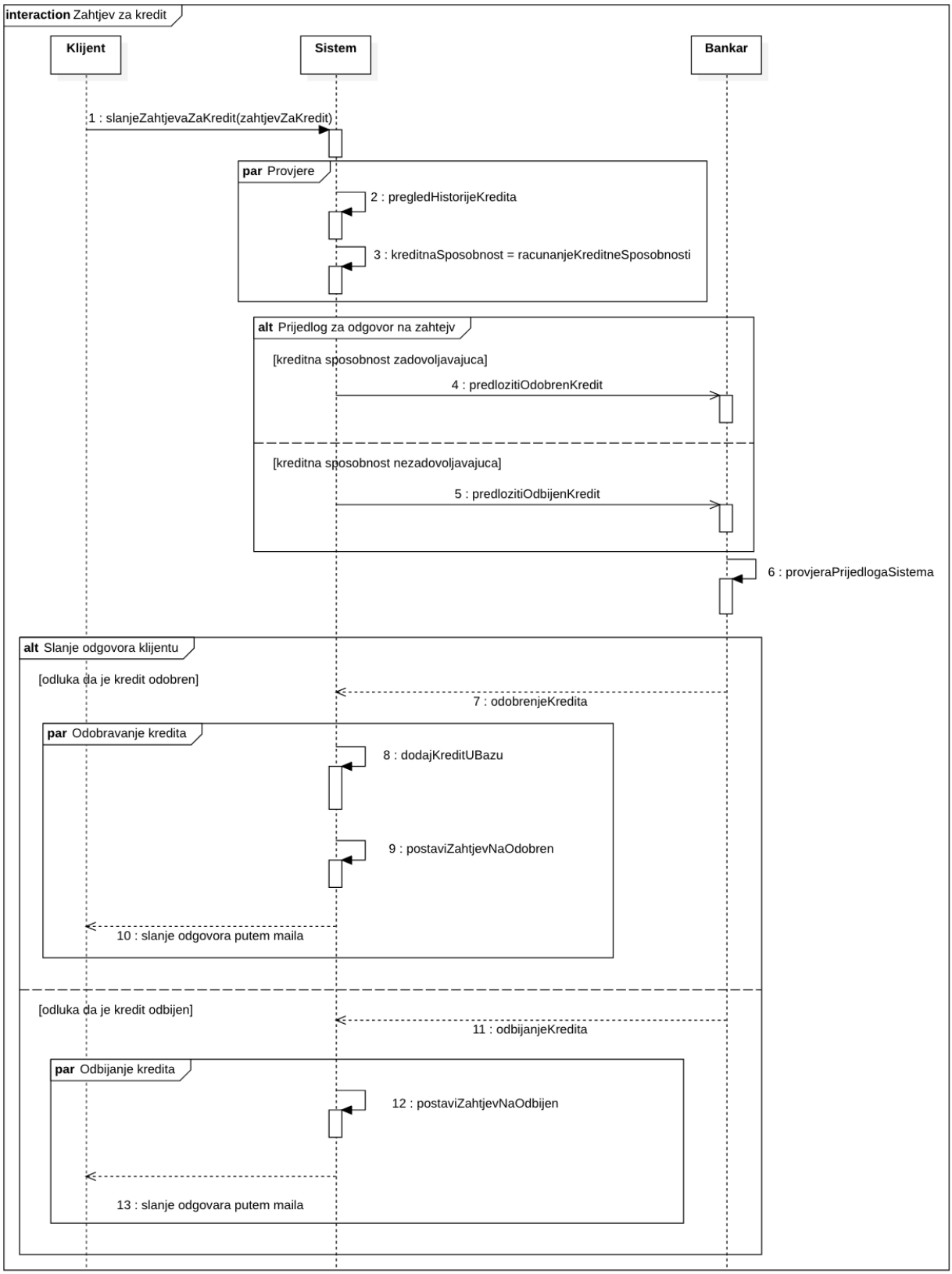
Dodavanje novog klijenta



Vršenje uplate

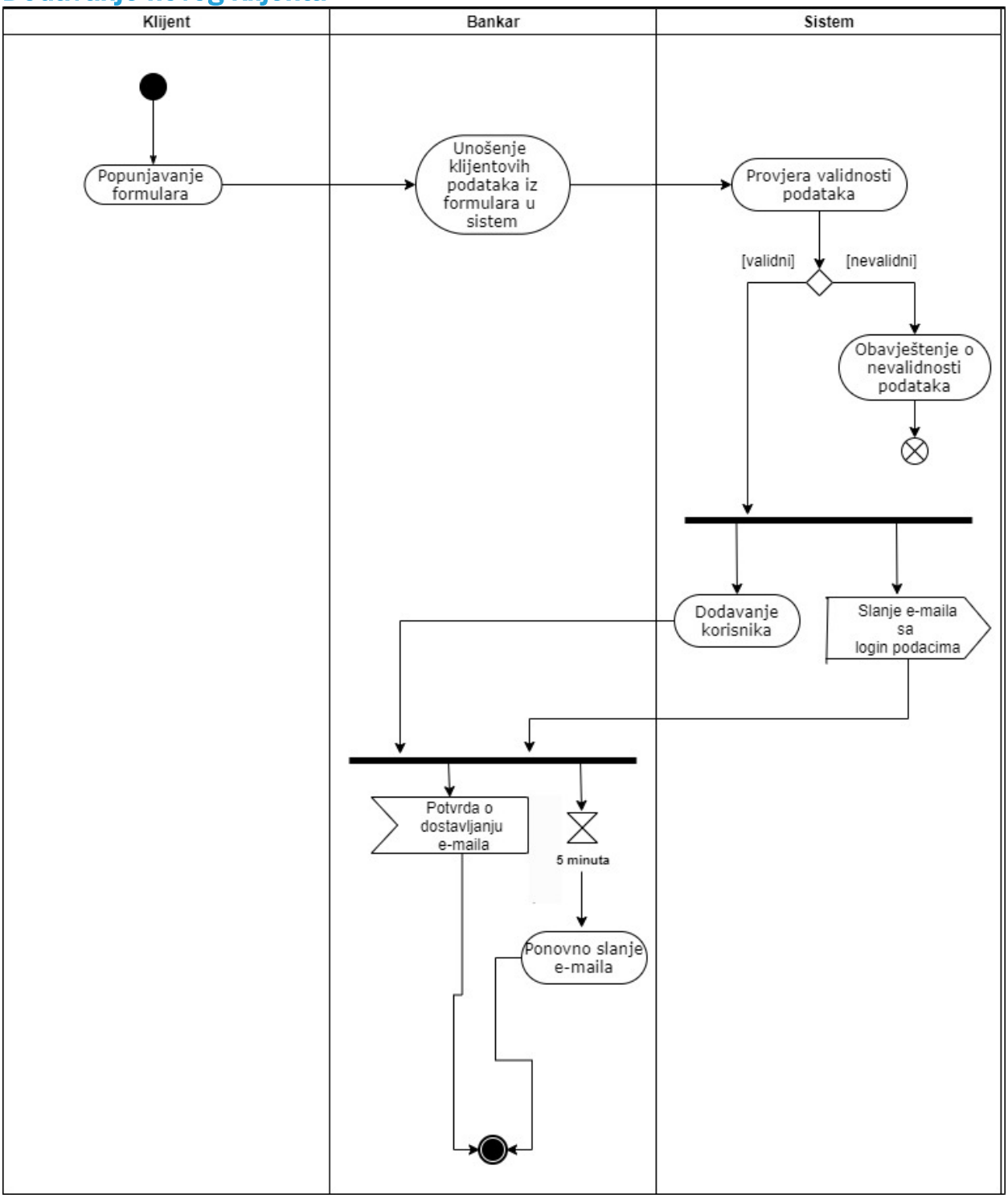


Zahtjev za kredit

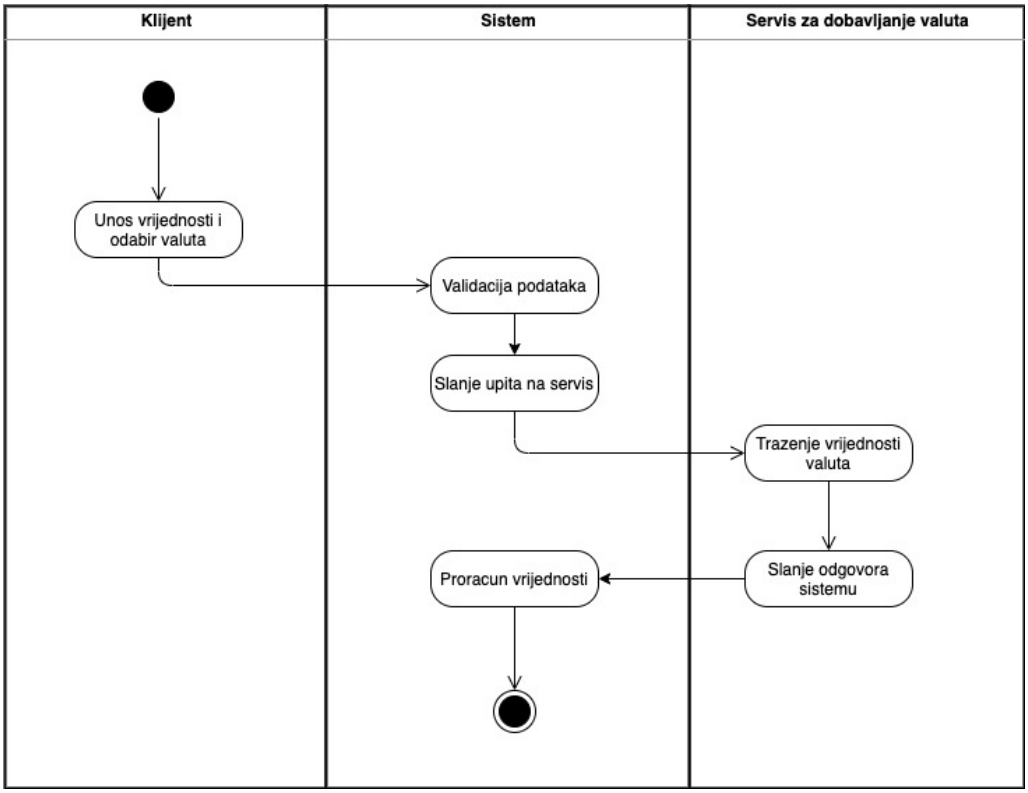


Dijagrami aktivnosti

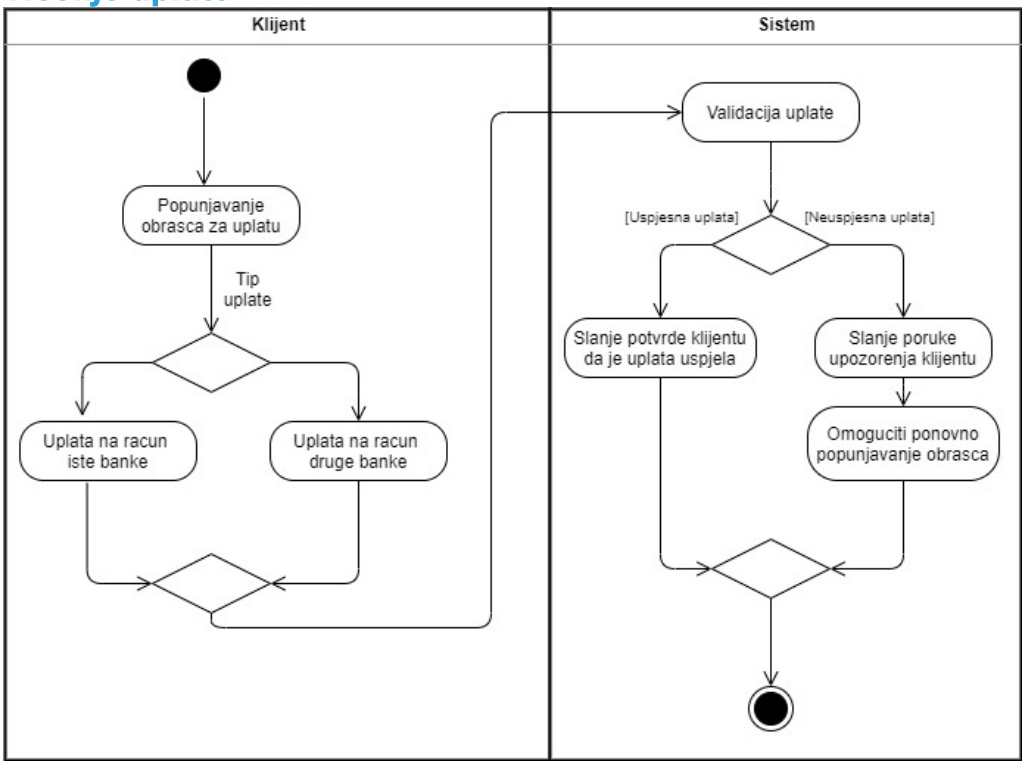
Dodavanje novog klijenta



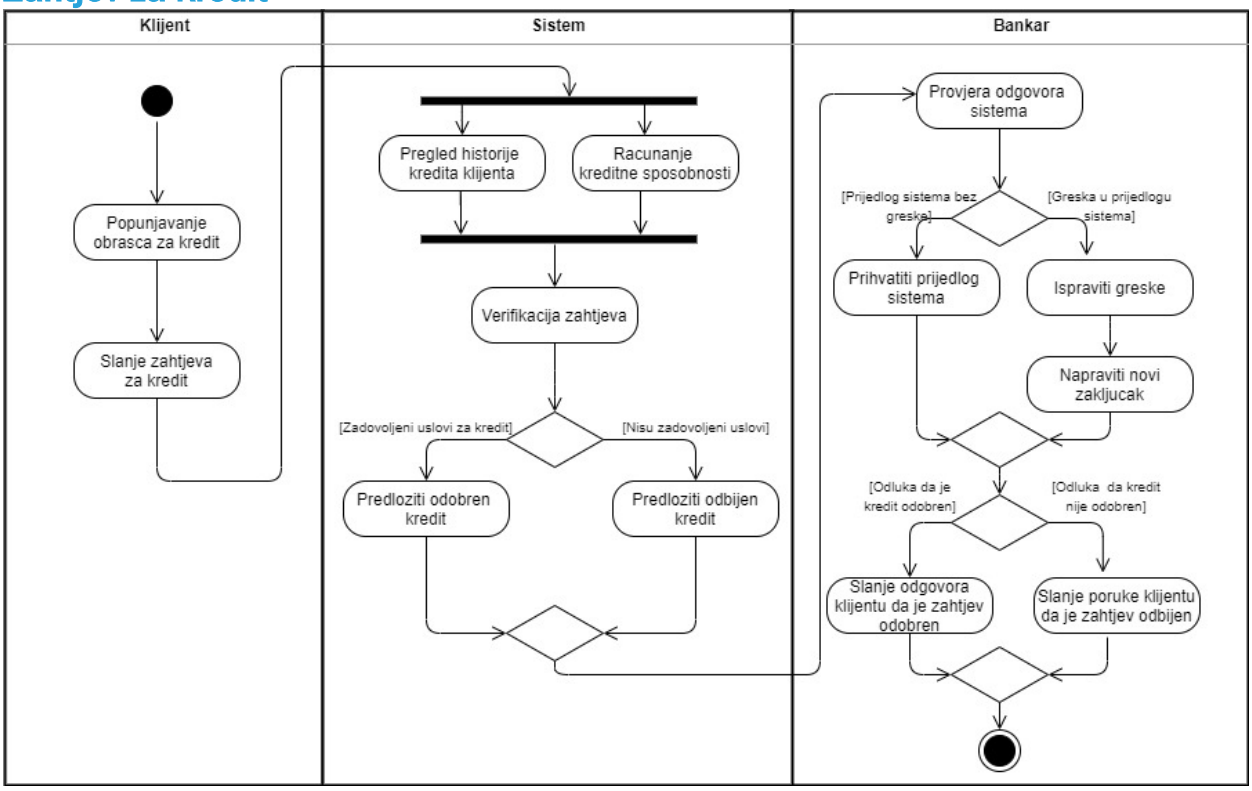
Konverzija valuta



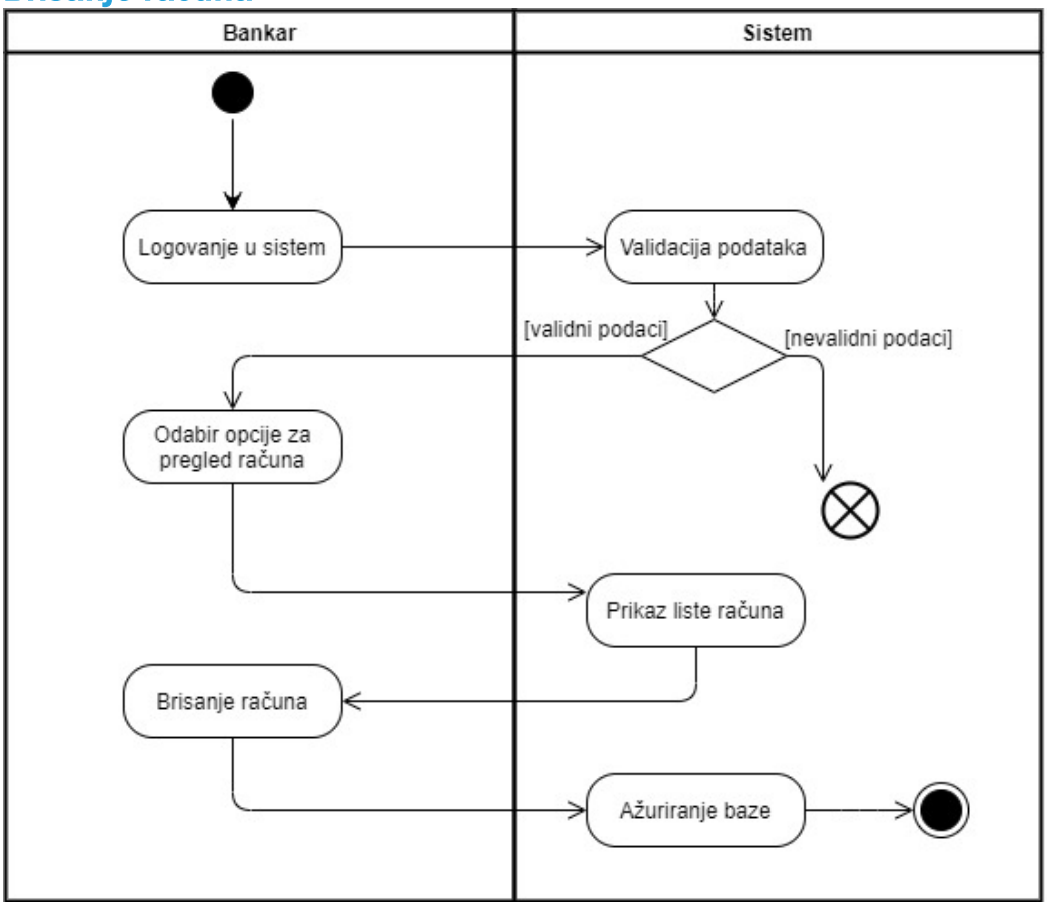
Vršenje uplate



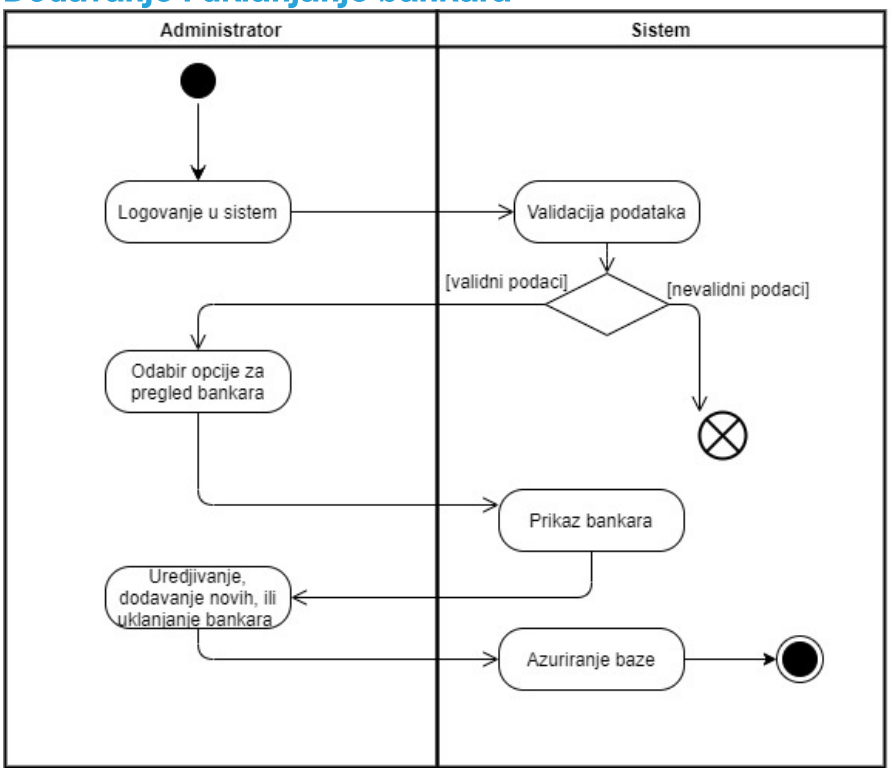
Zahtjev za kredit



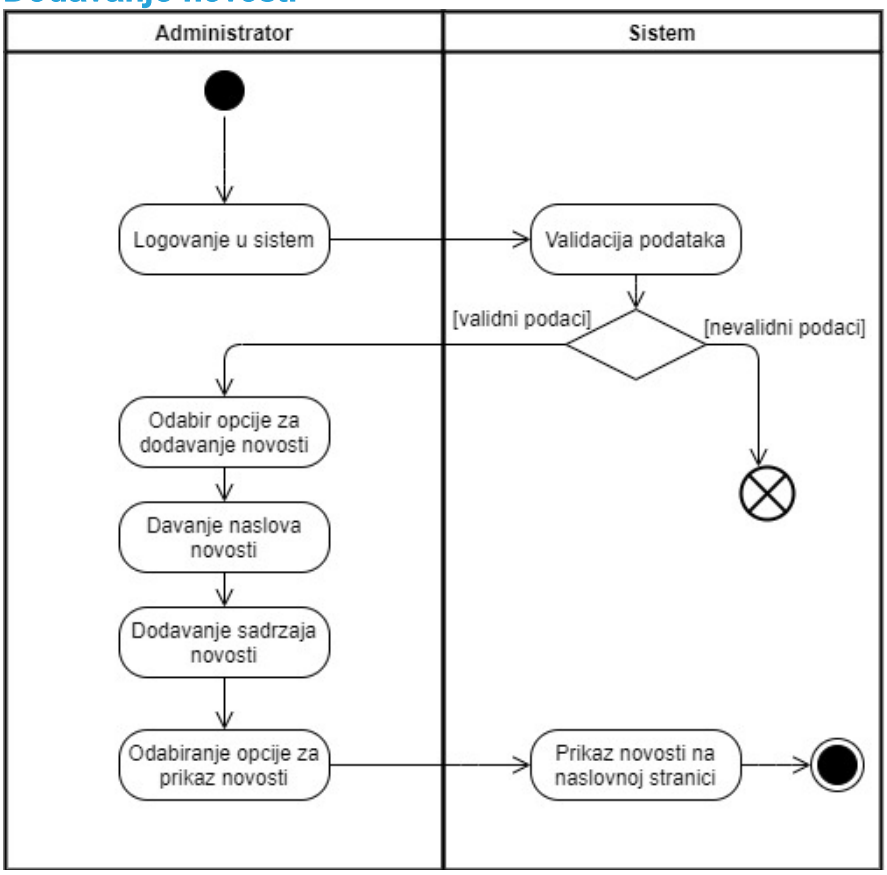
Brisanje računa



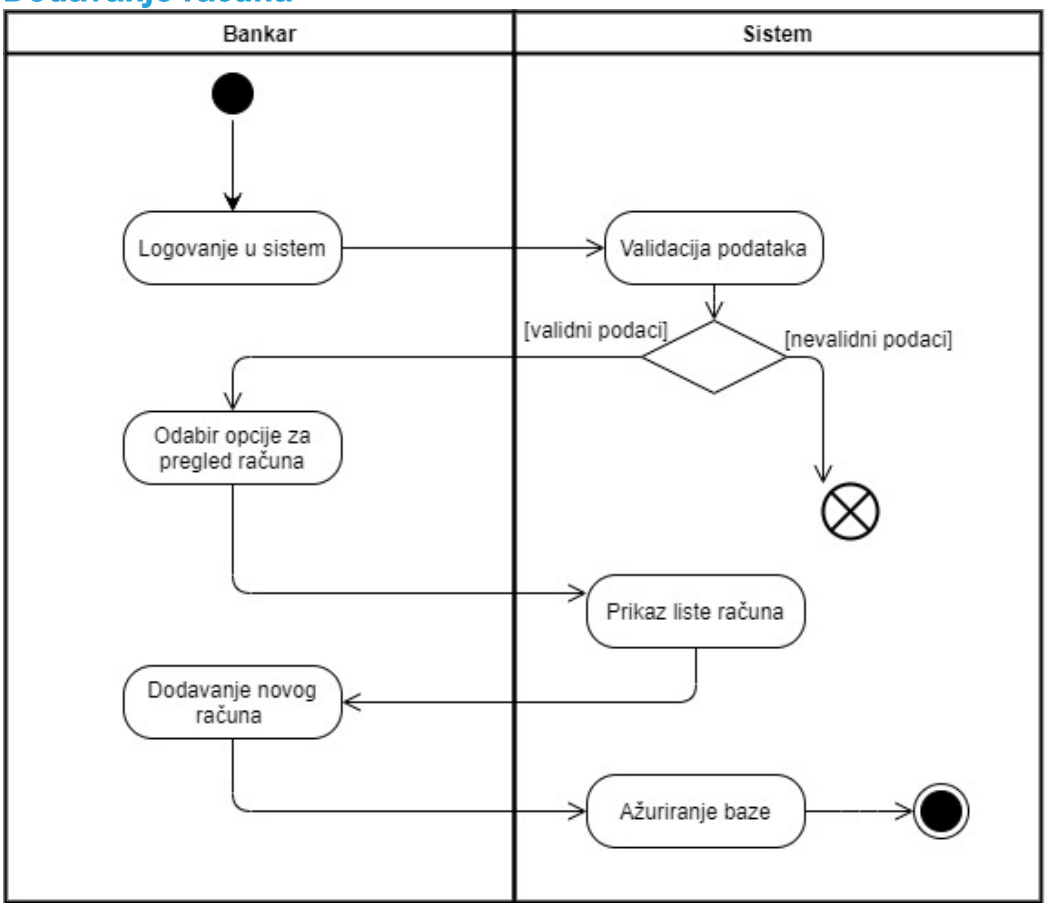
Dodavanje i uklanjanje bankara



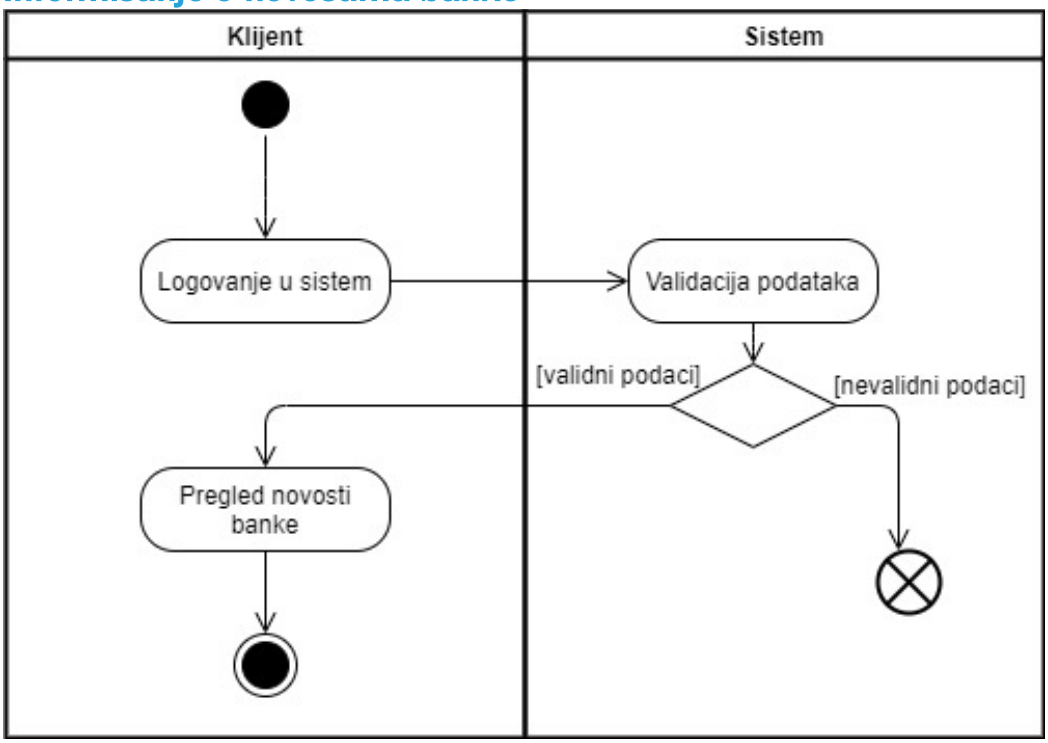
Dodavanje novosti



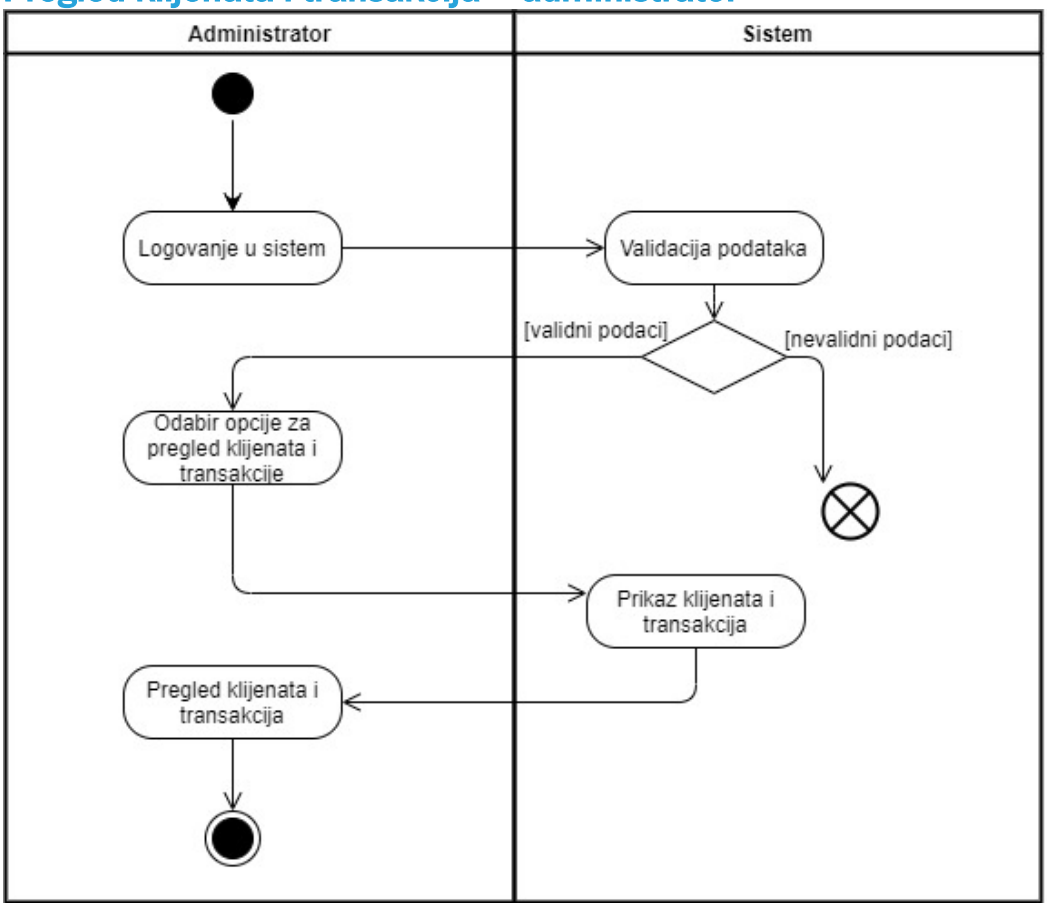
Dodavanje računa



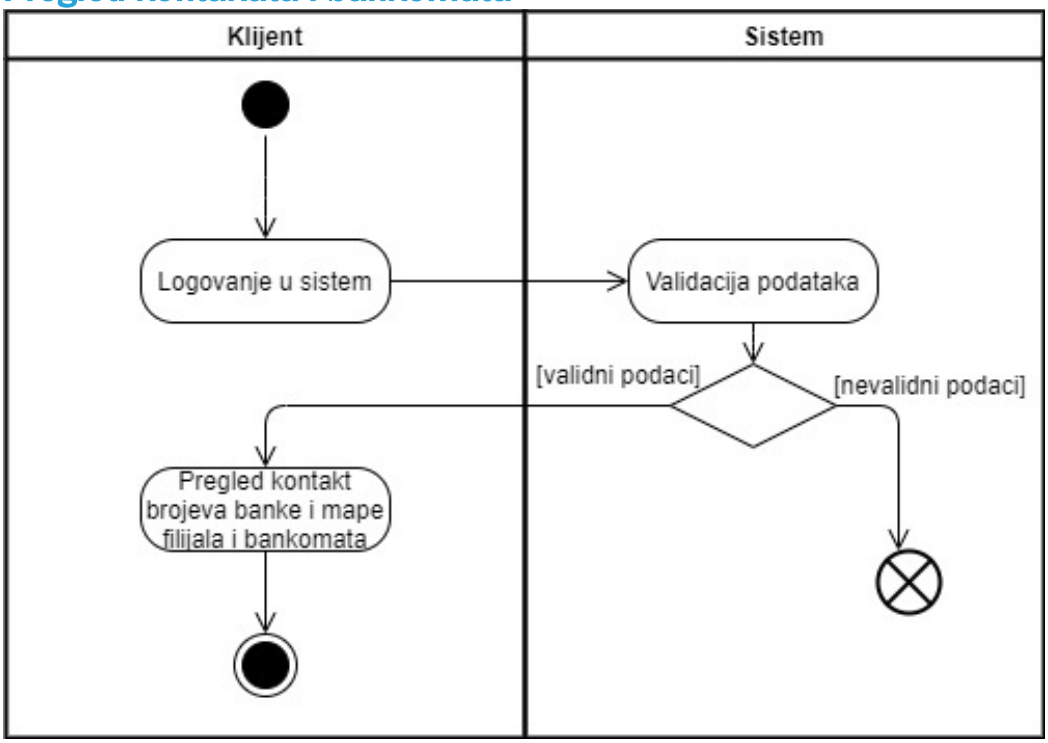
Informisanje o novostima banke



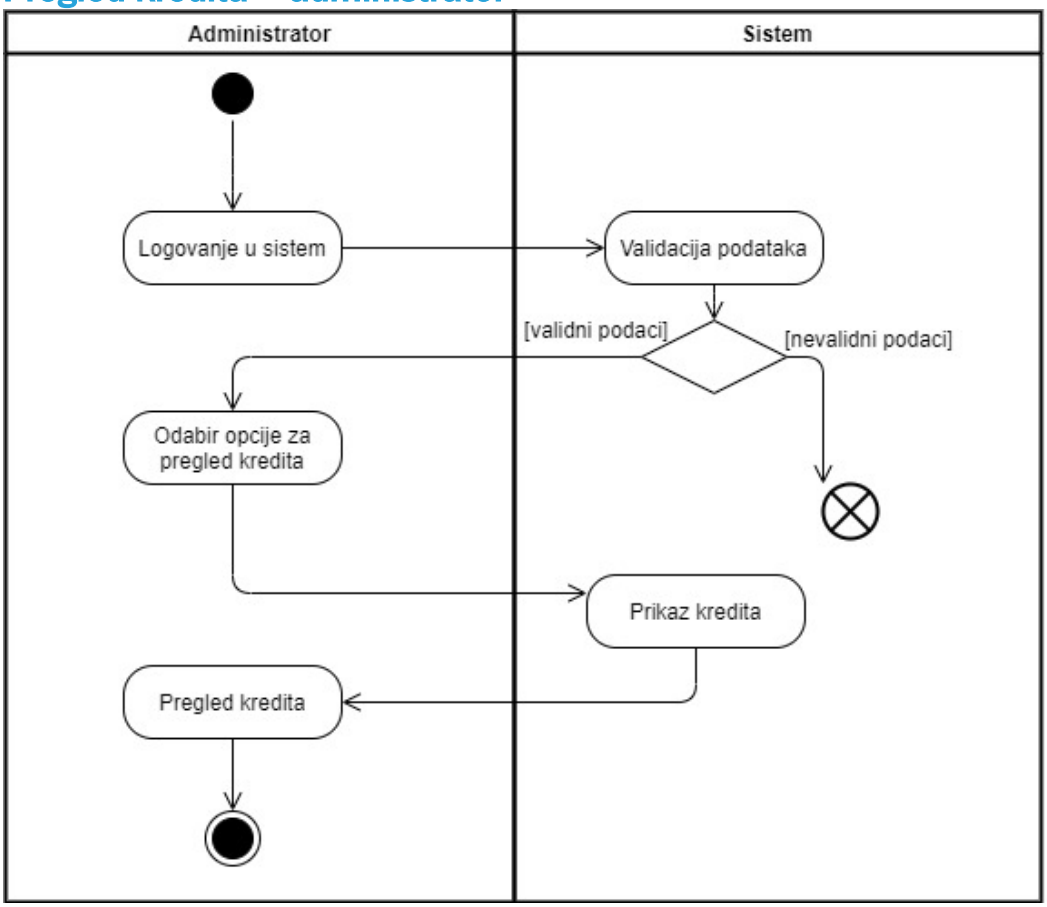
Pregled klijenata i transakcija – administrator



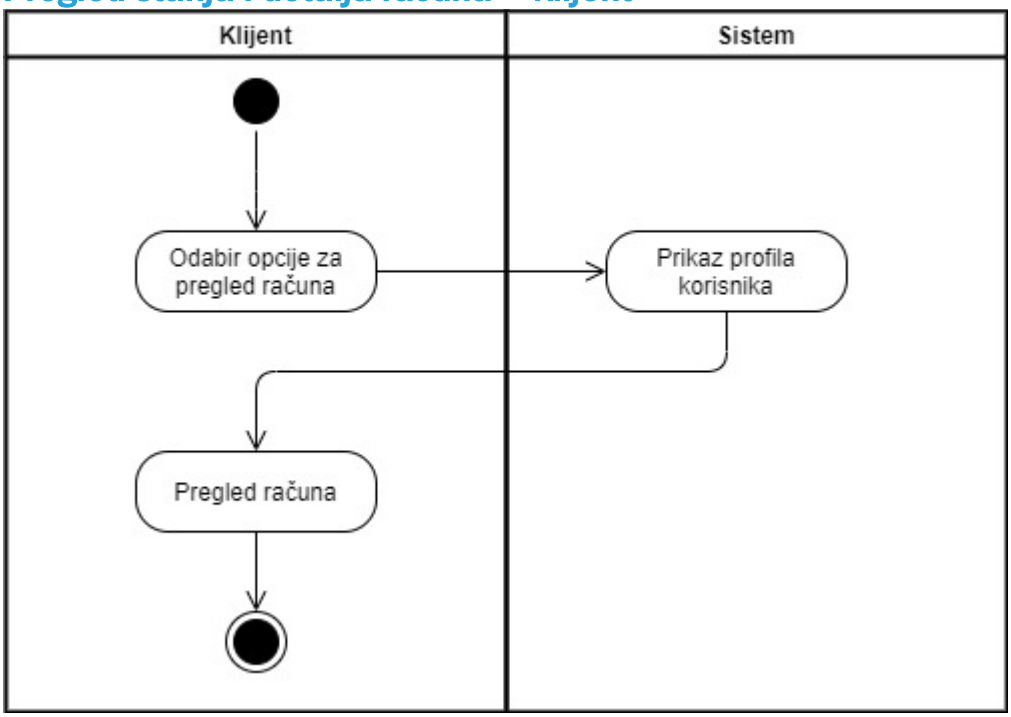
Pregled kontakata i bankomata



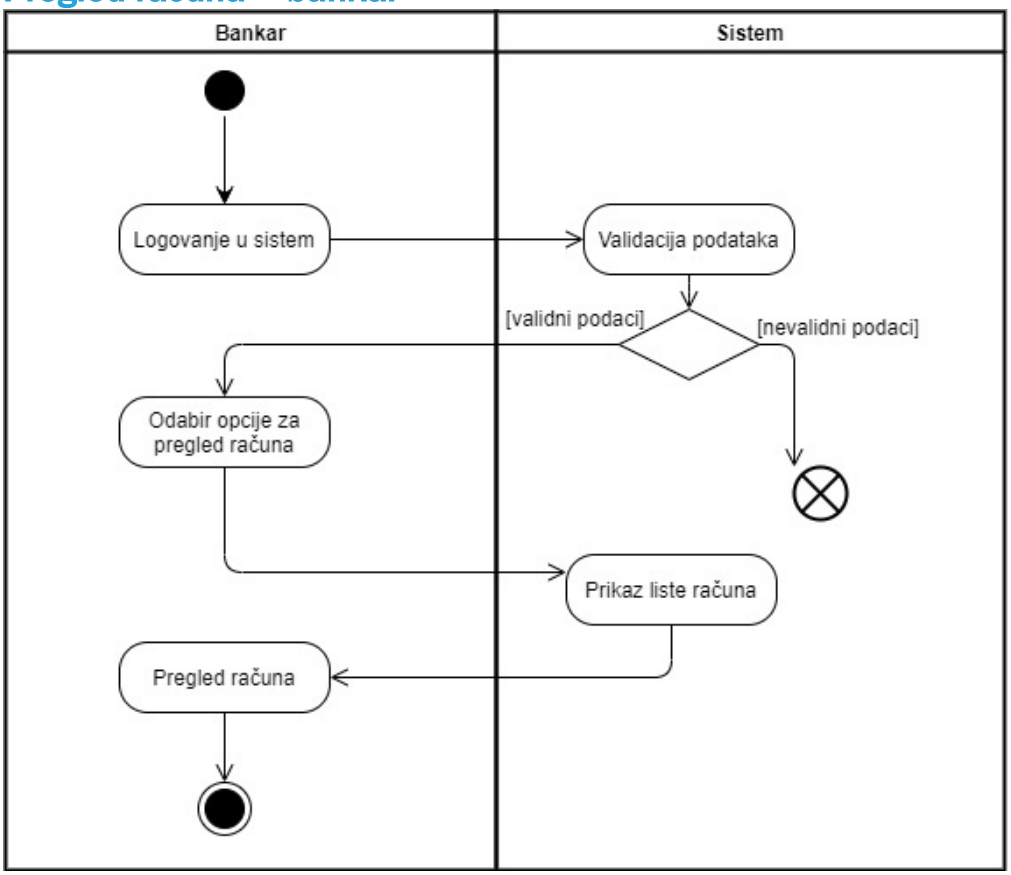
Pregled kredita – administrator



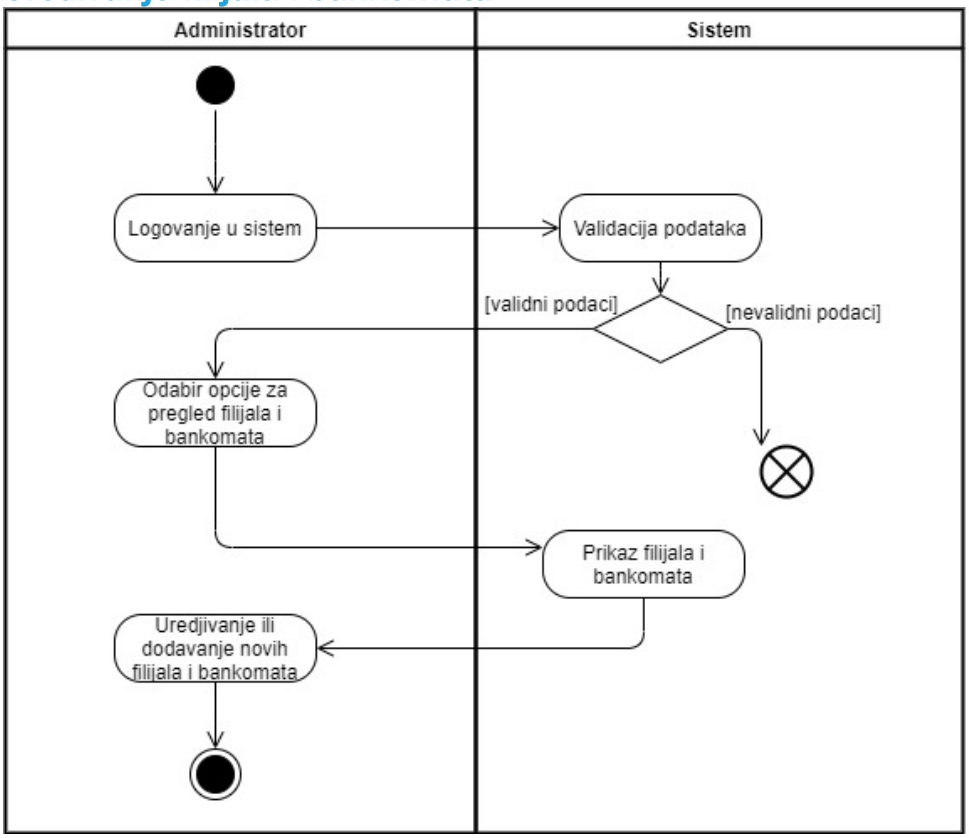
Pregled stanja i detalja računa – klijent



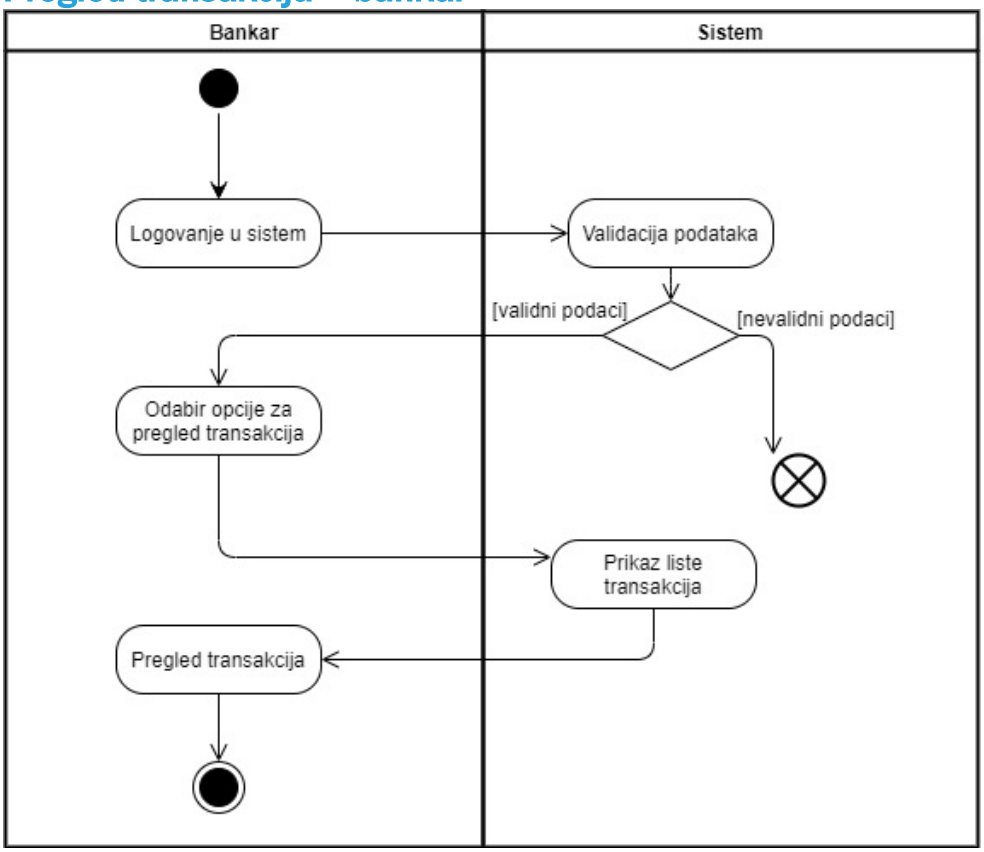
Pregled računa – bankar



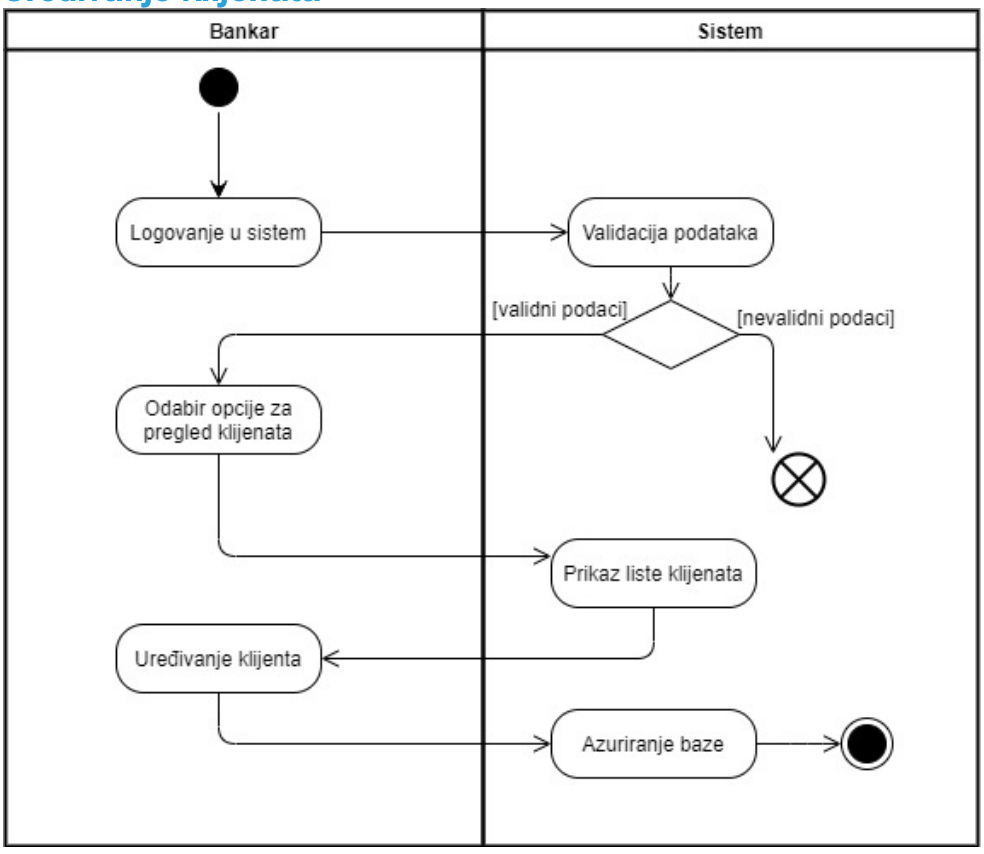
Uređivanje filijala i bankomata



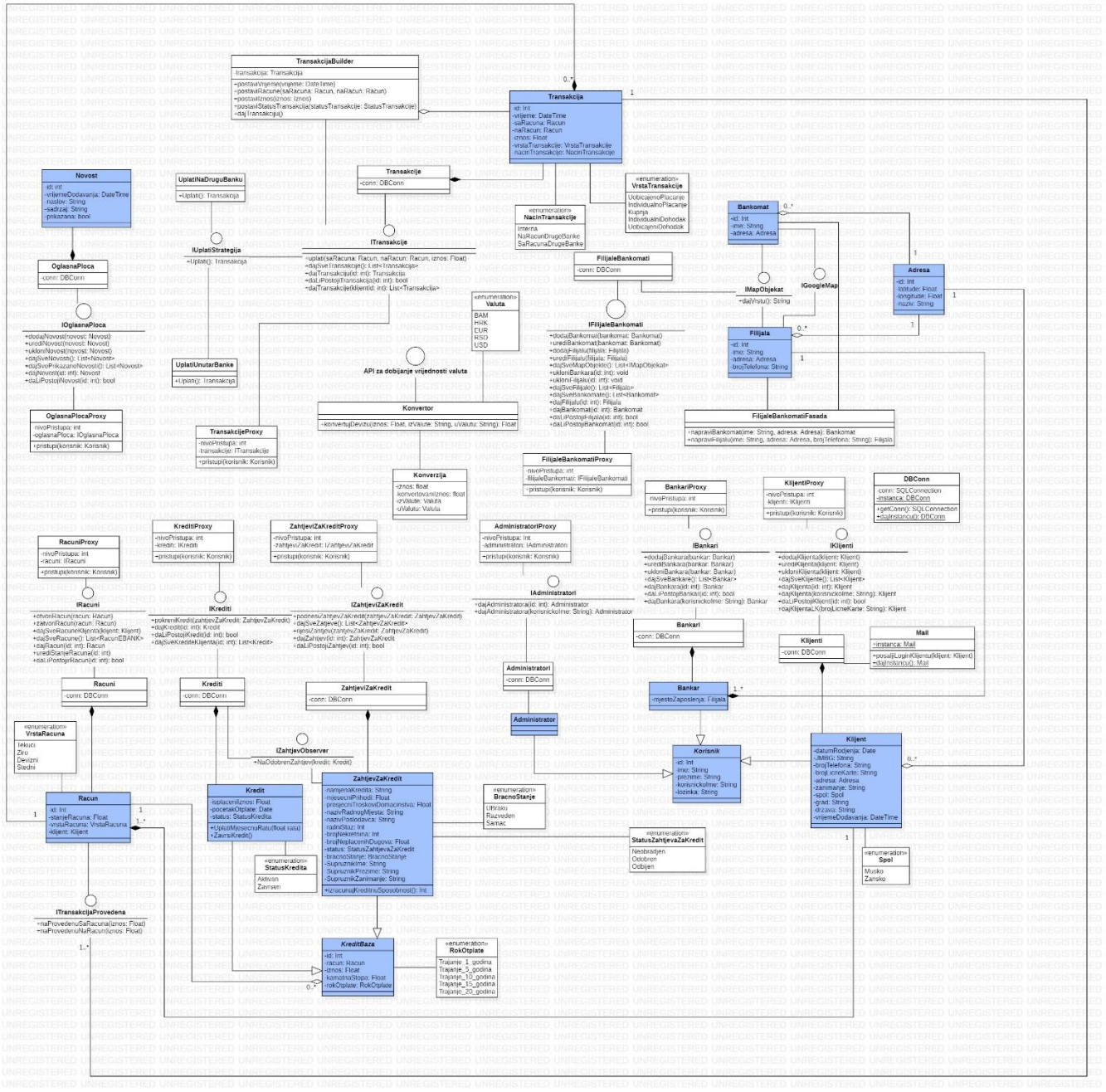
Pregled transakcija – bankar



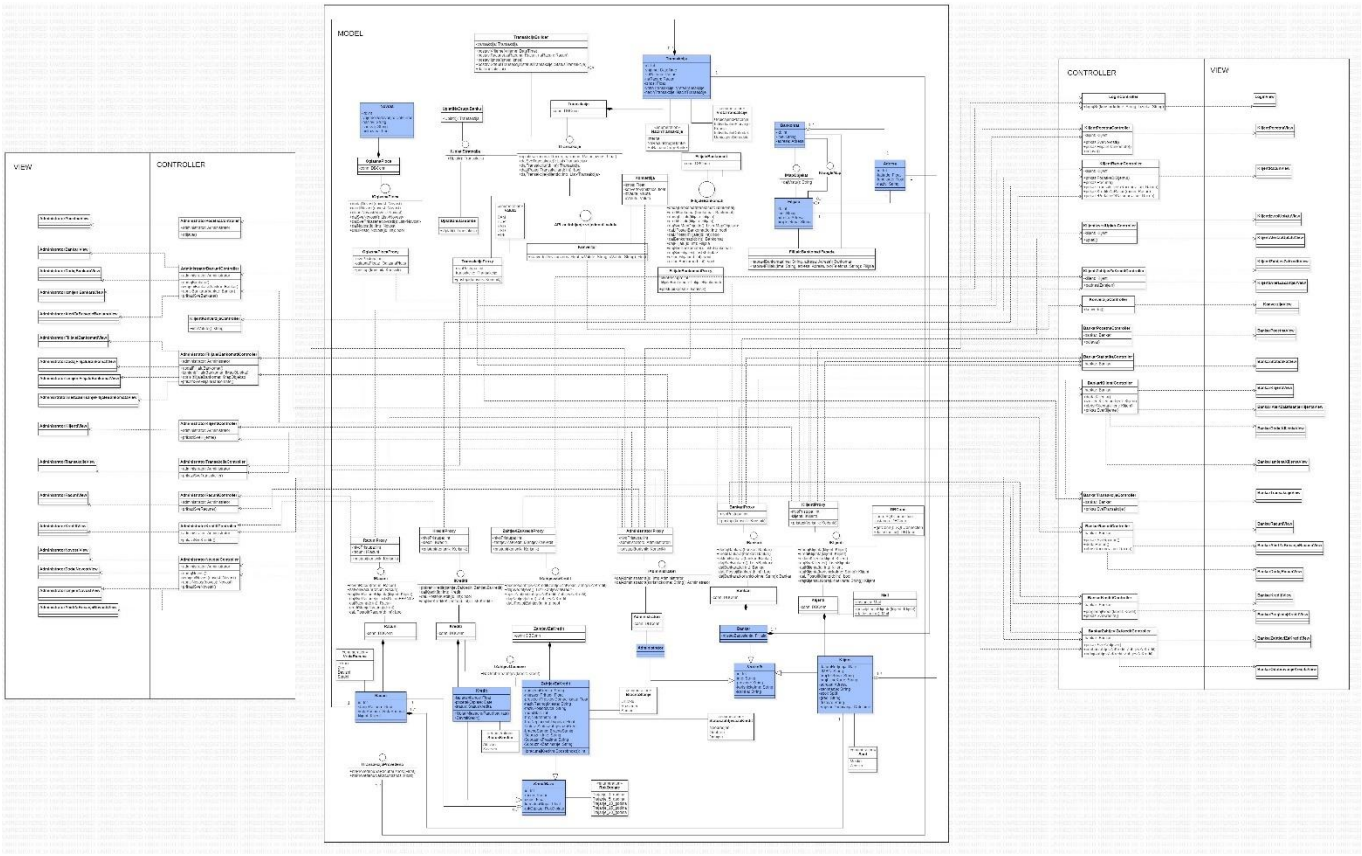
Uređivanje klijenata



Klasni dijagram



Klasni dijagram - MVC



Scenariji

Scenarij 1 - Vršenje uplata:

Naziv	Vršenje uplata
Opis	Klijent putem Web interfejsa vrši uplatu na drugi račun
Vezani zahtjevi	/
Preduvjeti	Klijent prijavljen na sistem, ima dovoljan iznos na računu
Posljedice – uspješan završetak	Izvršena uplata, umanjen iznos na klijentovom računu
Posljedice – neuspješan završetak	Prikaz poruke s greškom
Primarni akteri	Klijent
Ostali akteri	/
Proširenja/Alternative	/
Glavni tok	Klijent u aplikaciji bira opciju vršenja uplate, nakon čega unosi potrebne podatke a sistem provjeri transakciju i izvrši uplatu akoje ona odobrena

Tok događaja:

Klijent	Sistem banke
1. Pristupanje interfejsu za vršenje uplata	2. Prikaz polja za unos
3. Unos iznosa i podataka primaoca	4. Odobravanje uplate
	5. Uspješno uplaćen iznos

Alternativni tok događaja:

Klijent	Sistem banke
	4.1. Odbijanje uplate
	4.2. Prikaz poruke s greškom

Scenarij 2 - Pregled transakcija:

Naziv	Pregled transakcija
Opis	Klijent ima uvid u sve svoje izvršene transakcije
Vezani zahtjevi	/
Preduvjeti	Prijavljen na sistem
Posljedice – uspješan završetak	Prikaz izvršenih transakcija klijenta
Posljedice – neuspješan završetak	Prikaz poruke s greškom
Primarni akteri	Klijent
Ostali akteri	/
Proširenja/Alternative	/
Glavni tok	Klijent u aplikaciji bira opciju pregleda svojih transakcija

Tok događaja:

Klijent	Sistem banke
1. Slanje zahtjeva za prikaz izvršenih transakcija	2. Prikaz izvšenih transakcija

Alternativni tok događaja:

Klijent	Sistem banke
	2. Prikaz poruke s greškom

Scenarij 3 - Uklanjanje uposlenika:

Naziv	Uklanjanje uposlenika
Opis	Administrator nakon prestanka radnog odnosa bankara uklanja iz sistema
Vezani zahtjevi	/
Preduvjeti	Prijavljen na sistem
Posljedice – uspješan završetak	Bankar uklonjen iz sistema
Posljedice – neuspješan završetak	Prikaz poruke s greškom
Primarni akteri	Administrator
Ostali akteri	/
Proširenja/Alternative	/
Glavni tok	Administrator u aplikaciji bira opciju uklanjanja uposlenika i iz prikazane liste uposlenika bira željenog, nakon čega on biva uklonjen

Tok događaja:

Administrator	Sistem banke
1. Pristupanje interfejsu za uklanjanje uposlenika	2. Prikaz liste uposlenika
3. Biranje željenog uposlenika	4. Uspješno uklonjen uposlenik

Alternativni tok događaja:

Administrator	Sistem banke
	4. Prikaz poruke s greškom

Scenarij 4 - Dodavanje nove filijale:

Naziv	Dodavanje nove filijale
Opis	Administrator nakon otvaranja nove filijale unosi istu u sistem
Vezani zahtjevi	/
Preduvjeti	Prijavljen na sistem
Posljedice – uspješan završetak	Filijala dodana u sistem
Posljedice – neuspješan završetak	Prikaz poruke s greškom
Primarni akteri	Administrator
Ostali akteri	/
Proširenja/Alternative	/
Glavni tok	Administrator u aplikaciji bira opciju dodavanja filijale i unosi potrebne podatke za dodavanje

Tok događaja:

Korisnik	Sistem banke
1. Pristupanje interfejsu za uklanjanje uposlenika	2. Prikaz polja za unos potrebnih podataka
3. Unos podataka	4. Filijala uspješno dodana u sistem

Alternativni tok događaja:

Administrator	Sistem banke
	4. Prikaz poruke s greškom

Scenarij 5 - Konverzija valuta:

Naziv	Konverzija valuta
Opis	Klijent vrši zahtjev za konverziju valute
Vezani zahtjevi	/
Preduvjeti	Prijavljen na sistem
Posljedice – uspješan završetak	Prikazana vrijednost iznosa u traženoj valuti
Posljedice – neuspješan završetak	Prikaz poruke s greškom
Primarni akteri	Klijent
Ostali akteri	/
Proširenja/Alternative	/
Glavni tok	Klijent u aplikaciji bira opciju konverzije valute i unosi iznos, valutu iz koje se konvertuje kao i valutu u koju se konvertuje

Tok događaja:

Korisnik	Sistem banke
1. Pristupanje interfejsu za konverziju valuta	2. Prikaz polja za unos potrebnih podataka
3. Unos podataka	4. Dobavljanje vrijednosti deviza sa servisa
	5. Konverzija izvršena
	6. Prikaz iznosa u traženoj valuti

Alternativni tok događaja:

Korisnik	Sistem banke
	5. Prikaz poruke s greškom

Analiza sistema

Osnovne Klase (Modeli)

1. Korisnik – *apstraktna klasa koja opisuje korisnika koji se može logovati na sistem*
    - Atributi:
        -Id (int)
        -Ime (string)
        -Prezime (string)
        -KorisnickoIme (string)
        -Lozinka (string)
    - Metode
        -getteri i setteri

2. Adresa – *klasa koja opisuje mjesto na mapi*
    - Atributi:
        -Id (int)
        -Latitude (float)
        -Longitude (float)
        -Naziv (String)
    - Metode
        -konstruktor
        -getteri i setteri

3. Klijent extends Korisnik – *klasa koja opisuje klijenta banke*
    - Atributi:
        -DatumRodjenja (DateTime)
        -Spol (Spol)
        -JMBG (string)
        -BrojTelefona (string)
        -BrojLicneKarte (string)
        -Adresa (Adresa)
        -Zanimanje (string)
        -Grad (string)
        -Drzava (string)
        -VrijemeDodavanja (DateTime)
    - Metode
        -konstruktor
        -getteri i setteri

4. Racun – *opisuje bankovni račun*
    - Atributi:
        -Id (Int)
        -StanjeRacuna (Float)
        -VrstaRacuna (VrstaRacuna)
        -Klijent (Klijent)
    - Metode
        -konstruktor
        -getteri i setteri

5. Transakcija
    - Atributi:
        -Id (int)
        -Vrijeme (DateTime)
        -SaRacuna (Racun)
        -NaRacun (Racun)
        -Iznos (Float)
        -VrstaTransakcije (VrstaTransakcije)
        -NacinTransakcije(NacinTransakcije)
    - Metode
        -konstruktor
        -getteri i setteri

6. KreditBaza – *apstraktna klasa koja opisuje zahtjev za kredit ili kredit koji je u toku ili završen*
    - Atributi:
        -Id (int)
        -Racun (Racun)
        -Iznos (float)
        -KamatnaStopa (float)
        -RokOtplate (RokOtplate)
    - Metode
        -getteri i setteri

7. ZahtjevZaKredit extends KreditBaza
    - Atributi:
        -NamjenaKredita (string)
        -MjesecniPrihodi (float)
        -ProsjecniTroskoviDomacinstva (float)
        -NazivRadnogMjesta (string)
        -NazivPoslodavca (string)
        -RadniStaz (int)
        -BrojNekretnina (int)
        -BracnoStanje (BracnoStanje)
        -SupruznikIme (string)

-SupruznikPrezime (string)
-SupruznikZanimanje (string)
-ImaNeplacenihDugova (bool)
-BrojNeplacenihDugova (float)
-StatusZahteva (StatusZahtjevaZaKredit)

- Metode
  -konstruktor
  -getteri i setter
  -DajKreditnuSposobnost(): char

8. Kredit extends KreditBaza
   - Atributi:
     -Isplacenilznos (float)
     -PocetakOtplate (DateTime)
     - StatusKredita (StatusKredita)
   - Metode
     -konstruktor
     -getteri i setteri
     -UplatiMjesecnuRatu(): void
     -ZavrsiKredit(): void

9. Bankar extends Korisnik
   - Atributi:
     - mjestoZaposlenja (Filijala)
   - Metode
     -konstruktor
     -getteri i setteri

10. Administrator extends Korisnik
    - Metode
      -konstruktor
      -getteri i setteri

11. Filijala extends IMapObjekat
    - Atributi:
      - Id (int)
      - Ime (string)
      - Adresa (Adresa)
      - BrojTelefona (string)
    - Metode
      -konstruktori
      -getteri i setteri
      -string DajVrstu()

12. Bankomat extends IMapObjekat
    - Atributi:
        - Id (int)
        - Adresa (Adresa)
        -Ime (string)Metode
        -konstruktor
        -getteri i setteri
    - Metode
        -konstruktori
        -getteri i setteri
        -string DajVrstu()

13. Novost – *klasa koja predstavlja jednu vijest na oglasnoj ploči*
    - Atributi:
        - Id (int)
        - VrijemeDodavanja (DateTime)
        - Naslov (string)
        - Sadrzaj (string)
        - Prikazana (bool)
    - Metode
        -konstruktor
        -getteri i setteri
14. Konverzija
    - Atributi:
        - Iznos (float)
        - KonvertovaniIznos (float)
        - IzValute (Valute)
        - UValutu (Valuta)
    - Metode
        -konstruktor
        -getteri i setteri

15. StatusKredita (enum)
    -Aktivan
    -Zavrsen

16. VrstaTransakcije (enum)
    -UobicajenoPlacanje
    -IndividualnoPlacanje
    -Kupnja
    -IndividualniDohodak
    -UobicajeniDohodak

17. VrstaRacuna (enum)
    -Tekuci
    -Ziro
    -Devizni
    -Stedni
17. BracnoStanje (enum)
    -Ubraku
    -Razveden
    -Samac

18. NacinTransakcije (enum)
    -Interna
    -NaRacunDrugeBanke
    -SaRacunaDrugeBane

19. RokOtplate (enum)
    -Trajanje_1_godina
    -Trajanje_5_godina
    -Trajanje_10_godina
    -Trajanje_15_godina
    -Trajanje_20_godina

20. Spol (enum)
    -Muško
    -Žensko

21. StatusZahtjevaZaKredit (enum)
    -Neobradjen
    -Odobren
    -Odbijen

22. Valuta (enum)
    -BAM
    -EUR
    -RSD
    -USD
    -HRK

Interface-i

1. IOglasnaPloca
    - Metode
        -DodajNovost(novost: Novost): void
        -UrediNovost(novost: Novost): void
        -UkloniNovost(novost: Novost): void
        -DajSveNovosti() : List<Novost>
        -DajSvePrikazaneNovosti() : List<Novost>
        -DajNovost(id: int) : Novost
        -DaLiPostojiNovost(id: int) : bool

2. IBankari
    - Metode
        -DodajBankara(bankar: Bankar)
        -UrediBankara(bankar: Bankar)
        -UkloniBankara(id: int): void
        -DajSveBankare(): List<Bankar>
        -DajBankara(id: int): Bankar
        -DaLiPostojiBankar(id: int): bool
        -DajBankara(korisnickoIme: string): Bankar

3. IAdministratori
    - Metode
        -DajAdministratora(korisnickoIme: string): Administrator
        - DajAdministratora (id: int)

4. IFilijaleBankomati
    - Metode
        -DodajBankomat(bankomat: bankomat)
        -UrediBankomat(bankomat: bankomat)
        -DodajFilijalu(filijala: filijala): void
        -UrediFilijalu(filijala: filijala): void
        -UkloniBankomat(id: int): void
        -UkloniFilijalu(id: int): void
        -DajSveMapObjekte(): List<ImapObjekat>
        -DajSveFilijale(): List<Filijala>
        -DajSveBankomate(): List<Bankomat>
        -DajFilijalu(id: int): Filijala
        -DajBankomat(id: int): Bankomat
        -DaLiPostojiFilijala(id: int): bool
        -DaLiPostojiBankomat(id: int): bool

5. IKrediti
    - Metode
        -DajKredit(id: int): Kredit
        -DaLiPostojiKredit(id: int): bool
        -DajSveKredite(): List<Kredit>
        -DajSveKrediteKlijenta(id: int): List<Kredit>

6. IRacuni
    - Metode
        -OtvoriRacun(racun: Racun): void
        -ZatvoriRacun(id: int): void
        -DajRacun(id: int): Racun
        -DajSveRacune(): List<Racun>
        -DaLiPostojiRacun(id: int): bool
        -DajSveRacuneKlijenta(id: int): List<Racun>
        -UrediStanjeRacuna(racun: Racun): void
        -DajRacune(id: int): List<Racun>

7. IKlijenti
    - Metode
        -DodajKlijenta(klijent: Klijent)
        -UrediKlijenta(klijent: Klijent)
        -UkloniKlijenta(klijent: Klijent)
        -DajSveKlijente(): List<Klijent>
        -DajKlijenta(id: int): Klijent
        -DajKlijentaLK(brojLicneKarte: string): Klijent
        -DaLiPostojiKlijent(id: int): bool
        -DajKlijenta(korisnickoIme: string): Klijent

8. ITransakcije
    - Metode
        -Uplati(transakcija: Transakcija)
        -DajSveTransakcije(): List<Transakcija>
        -DajTransakciju(id: int): Klijent
        -DajTransakcije(id: int): List<Transakcija>
        -DaLiPostojiTransakcija(id: int): bool

9. IZahtjeviZaKredit
    - Metode
        -PodnesiZahtjevZaKredit(zahtjevZaKredit: ZahtjevzaKredit): void
        -RijesiZahtjev(id: int, bool: prihvacen): void
        -DajSveZahtjeve(): List<ZahtjevZaKredit>
        -DajZahtjev(id: int): ZahtjevZaKredit
        -DaLiPostojizahtejv(id: int): bool

Repository klase

1. OglasnaPloca implements IOglasnaPloca
2. Bankari implements IBankari
3. Administratori implements IAdministratori
4. FilijaleBankomati implements IFilijaleBankomati
5. Krediti implements Krediti
6. Racuni implements IRacuni
7. Klijenti implements IKlijenti
8. Transakcije implements ITransakcije
9. ZahtjeviZaKredit implements IZahtjeviZaKredit

Baza podataka

Biti će neophodna baza podataka koja će čuvati sve tabele slične klasama iznad.

API - API će se koristiti za dobijanje informacija o trenutnom stanju deviza pri konvertovanju istih, te za mapu.

WEB SERVIS

API (Application Programming Interface) je skup specifikacija kojima se služe programeri kako bi komunicirali i interaktirali sa ostalim aplikacijama. U našem slučaju, bit će nam potrebne vrijednosti valuta kako bismo omogućili korisnicima da konvertuju valute. Da bismo to postigli, treba nam API web servisa fixer.io, koji nam to može ponuditi. Pošto sad za sad ne mislimo prikazivati vrijednosti real-time valuta, jedini endpoint koji ćemo koristiti je Convert endpoint. U nastavku je oficijelna dokumentacija tog endpointa u kojem su objašnjeni parametri i povratna vrijednost.

Convert Endpoint

The Fixer API comes with a separate currency conversion endpoint, which can be used to convert any amount from one currency to another. In order to convert currencies, please use the API's convert endpoint, append the from and to parameters and set them to your preferred base and target currency codes.

It is also possible to convert currencies using historical exchange rate data. To do this, please also use the API's date parameter and set it to your preferred date. (format YYYY-MM-DD)

API Request:

```
https://data.fixer.io/api/convert

    ? access_key = API_KEY

    & from = GBP

    & to = JPY

    & amount = 25
```

Request Parameters:

Parameter	Description
access_key	[required] Your API Key.
from	[required] The three-letter currency code of the currency you would like to convert from.
to	[required] The three-letter currency code of the currency you would like to convert to.

Finalna dokumentacija modela

Parameter	Description
`amount`	[required] The amount to be converted.
`date`	[optional] Specify a date (format `YYYY-MM-DD`) to use historical rates for this conversion.

API Response:


```
{
    "success": true,
    "query": {
        "from": "GBP",
        "to": "JPY",
        "amount": 25
    },
    "info": {
        "timestamp": 1519328414,
        "rate": 148.972231
    },
    "historical": ""
    "date": "2018-02-22"
    "result": 3724.305775
}
```


Response Objects:

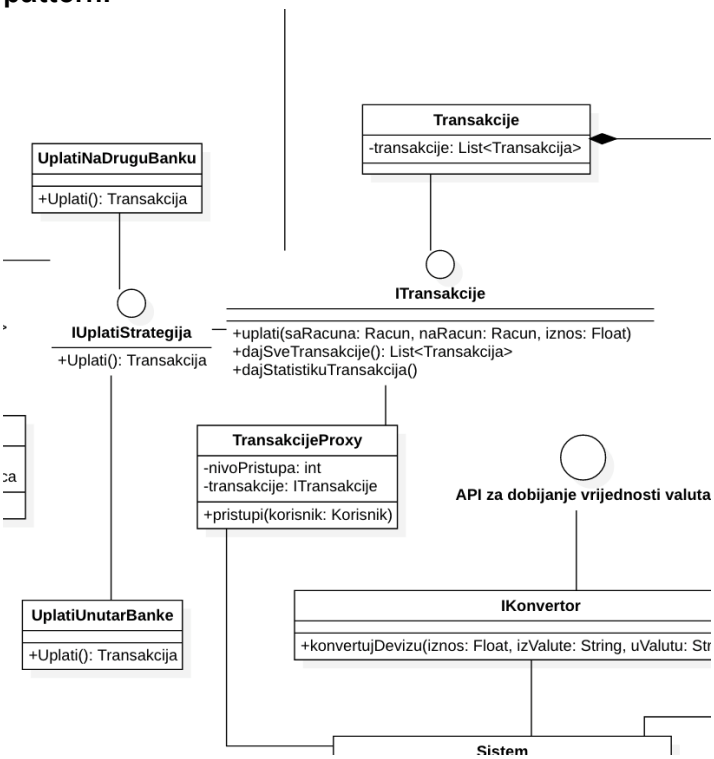
Response Object	Description
`success`	Returns **true** or **false** depending on whether or not your API request has succeeded.

Response Object	Description
`query` > `from`	Returns the three-letter currency code of the currency converted from.
`query` > `to`	Returns the three-letter currency code of the currency converted to.
`query` > `to`	Returns the amount that is converted.
`info` > `timestamp`	Returns the exact date and time (UNIX time stamp) the given exchange rare was collected.
`info` > `rate`	Returns the exchange rate used for your conversion.
`historical`	Returns **true** if historical rates are used for this conversion.
`date`	Returns the date (format `YYYY-MM-DD`) the given exchange rate data was collected.
`result`	Returns your conversion result.

Paterni ponašanja

Strategy pattern

Strategy patern izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Pogodan je kada postoje različiti primjenjivi algoritmi (strategije) za neki problem. Ovaj patern smo primijenili kod metode uplati u interfejsu ITransakcije tako što smo dodali interfejs IUplatiStrategija koji implementiraju klase UplatiNaDruguBanku i UplatiUnutarBanke. Ovo smo primijenili jer metoda uplati se treba implementirati različito u zavisnosti da li je klijent odabrao uplatu na račun druge banke ili uplatu na račun EBANK.
Ovakvu funkcionalnost smo ranije bili zanemarili u našem dijagramu klasa, iako smo je imali na dijagramima aktivnosti i dijagramu slučajeva upotrebe. Čitajući o strategy paternu smo shvatili da nam dio funkcionalnosti nedostaje, te smo iz klase Račun naslijedili klasu RačunEBANK koja označava račun unutar naše banke. Zatim smo ovaj patern primijenili da bi metodu uplati izvršavali drugačije u zavisnosti od načina uplate.
Na sljedećoj slici se nalazi prikaz sa klasnog dijagrama gdje smo primijenili Strategy pattern:



State pattern

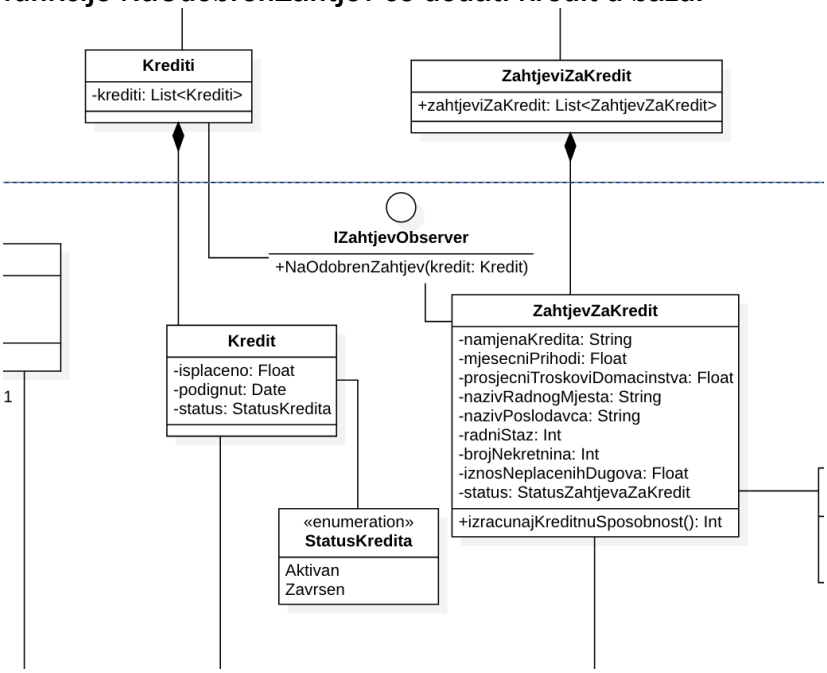
State Pattern je dinamička verzija Strategy paterna. Objekat mijenja način ponašanja na osnovu trenutnog stanja. Ovaj patern nismo iskoristili jer nemamo algoritam koji bi se izvrsavao na različite načine u odnosu na neko stanje. U našem projektu npr. Imamo 3 stanja kredita: u zahtjevu, aktivan i završen. Kada bi imali neku metodu koja se izvršava drugačije u zavisnosti od stanja kredita, mogli bi iskoristiti ovaj patern. Ili, kada bi imali atribut hitno u transakciji, mogli bismo vršiti uplate različito u zavisnosti od toga da li je uplata hitna ili ne.

TemplateMethod patern

Omogućava izdvajanje određenih koraka algoritma u odvojene podklase. Struktura algoritma se ne mijenja - mali dijelovi operacija se izdvajaju i ti se dijelovi mogu implementirati različito. U našem slučaju, kada bismo imali funkcionalnost da sortiramo listu transakcija po nekom kriteriju, ovaj patern bismo mogli iskoristiti za različite kriterije. Tako bismo za sortiranje po vremenu mogli koristiti jednu klasu, za sortiranje po iznosu drugu, itd.

Observer patern

Uloga Observer paterna je da uspostavi relaciju između objekata tako kada jedan objekat promijeni stanje drugi zainteresirani objekti se obavještavaju. U našem projektu, observer patern smo iskoristili kada neki od zahtjeva za kredit biva odobren. Klasa Krediti će implementirati interfejs IZahtjevObserver, na poziv funkcije NaOdobrenZahtjev će dodati kredit u bazu.



Također ovaj patern smo iskoristili kada stanje transakcije pređe iz UToku u Provedena. Pomoću njega će se sa jednog računa skinuti iznos transakcije, a na drugi će se isti iznos položiti.

Iterator patern

Iterator patern omogućava sekvencijalni pristup elementima kolekcije bez poznavanja kako je kolekcija strukturirana. Implementirat ćemo ovaj patern tako što ćemo interfejs IEnumerable implementirati u naše interfejse IOglasaPloča, ITransakcije, IFilijaleBankomati, IRacuni, iKrediti, IZahtjeviZaKredit, IBankari i IKlijenti. Pošto njega implementiraju i proxy i DAO klase, u proxy-u će getEnumerator() samo proslijediti vrijednost one iz DAO, a ona iz DAO će vratiti iterator liste stavki koju dobije iz baze.

Strukturalni paterni

Adapter patern

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 obezbjeđuje 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. Ovaj patern nam nije bio potreban u našem dijagramu klasa, pa ga nismo ni iskoristili. Hipotetički bi mogao biti koristan kada bismo pravili funkcionalnost koja bi dozvolila da se iz nekog drugog bankarskog sistema prebaci račun u naš.

Facade patern

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 Dodali smo klasu FilijaleIBankomatiFasada koja ima dvije metode napraviFilijalu() i napraviBankomat(). Na taj način, korisnik ne mora dobro poznavati sistem kako bi ga mogao koristiti, ali i dalje će moći koristiti klase Bankomat i Filijala.

Decorator patern

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. Mi ga nismo iskoristili, jer nije bilo potrebe.

Bridge patern

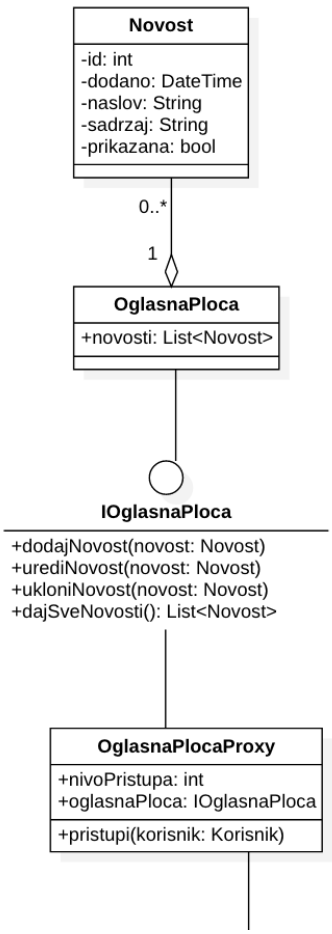
Bridge patern služi kako bi se apstrakcija nekog objekta odvojila od njegove implementacije. Ovaj patern veoma je važan jer omogućava ispunjavanje Open-Closed SOLID principa, odnosno uz poštivanje ovog paterna omogućava se nadogradnja modela klasa u budućnosti te osigurava da se neće morati vršiti određene promjene u postojećim klasama. Ni ovaj patern nismo nigdje iskoristili, jer nemamo dvije klase koje koriste istu metodu na drugaciji nacin. Hipotetički, ovaj patern bismo mogli iskoristiti kada bismo imali klase VISA, MasterCard i Maestro, u svakoj od kojih se nalazi metoda dajRaspolozivaSredstva(). Tu metodu bismo mogli iskoristiti da primijenimo Bridge patern.

Composite patern

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. Njega smo mogli iskoristiti u slučaju bankomata i filijala. Kada bismo imali zajedničke metode u te dvije klase, ovaj patern bi bio pogodniji od Flyweight paterna.

Proxy patern

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



Ovaj patern smo iskoristili u više slučajeva, a ovdje ćemo navesti par njih:

- Klasa OglasnaPloca ima četiri metode: dodajNovost, urediNovost, ukloniNovost, dajSveNovosti. Ovim metodama ne bi trebali pristupati svi korisnici. Na osnovu nivoa pristupa možemo osigurati da administrator može dodavati, uređivati, ukloniti i vidjeti sve novosti, dok klijent može samo vidjeti novosti, a ne i koristiti ostale metode. Zato smo ovdje iskoristili Proxy patern tako što smo dodali klasu OglasnaPlocaProxy.
- Korisnik može pozvati metodu uplati nad klasom Transakcije preko klase TransakcijeProxy i tako pokrenuti transakciju. To može izvršiti samo korisnik koji je klijent. Klasom TransakcijeProxy smo obezbijedili da samo korisnik može vršiti plaćanje.
- Bankar može pozvati metodu dajSveKredite() i pokreniKredit(), Administrator može pozvati metodu dajSveKredite(), dok Klijent ne može pozvati niti jednu od ovih metoda. Koristeći Proxy patern onemogućili smo nekim korisnicima da koriste neke metode, dok smo pojedinim omogućili. Time je uspostavljen neki nivo pristupa ovim metodama.

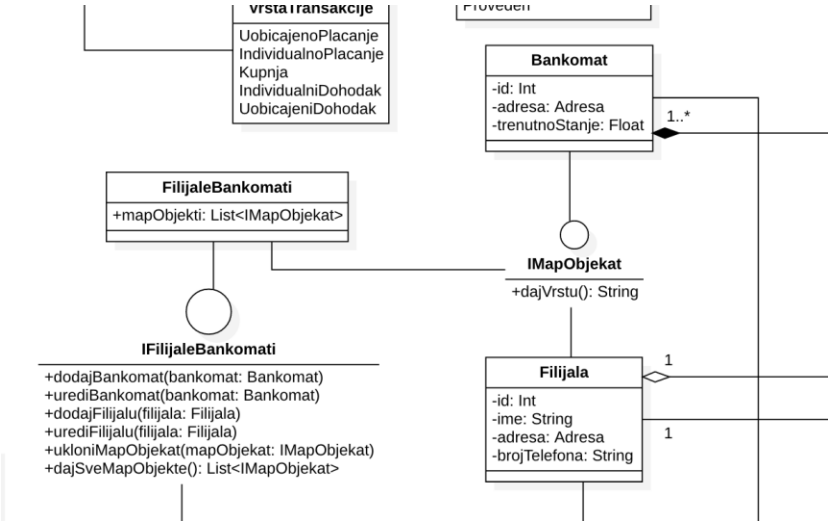
Iskoristili smo Proxy patern u jako puno slucajeva koji su slicni, tako da smatramo da je njihovo navodjenje bespotrebno. Svakako, bit ce prikazano na dijagramu klasa. Jedna od primjena ovog paterna u našem dijagramu klasa prikazana je na slici.

Flyweight patern

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 instanca objekta (tzv. bezlično stanje). Korištenje ovog paterna veoma je korisno u slučajevima kada je potrebno vršiti uštedu memorije.

Ovaj patern smo uspjeli primijeniti u našem projektu. Klasa FilijaleIBankomati (klasa koja je generalizacija dvije klase: Filijala i Bankomat) u sebi je imala dvije liste: listu bankomata i listu filijala, a takodjer je imala i metode dajBankomate(), dajFilijale(), kao i ukloniBankomat(), ukloniFilijalu(). Primijenili smo Flyweight patern tako sto smo dodali interfejs IMapObjekat koji je povezan sa klasama Filijala i Bankomat, kao i sa klasom FilijaleIBankomati. Taj interfejs ima metodu dajVrstu() umjesto dvije metode za filijale i bankomate. Sada, unutar interfejsa IFilijaleIBankomati, sada imamo metode dajSveMapObjekte() i ukloniMapObjekat().

Na sljedecoj slici vidimo primjenu ovog paterna u našem klasnom dijagramu:

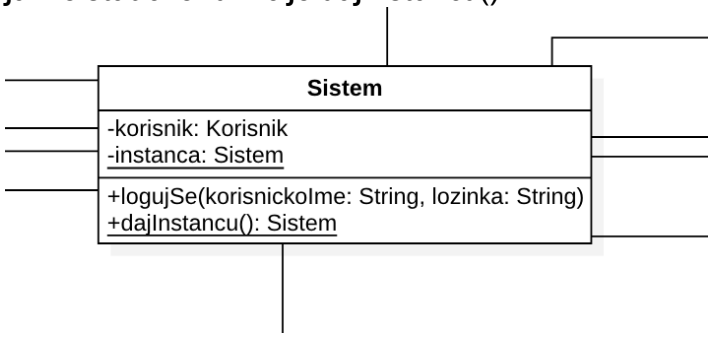


Kreacijski paterni

Singleton patern

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.

Ovaj patern smo iskoristili u našoj glavnoj klasi Sistem. U njoj ima privatna statička varijabla instance koja označava jedinu instancu klase. Nju možemo dobiti pomoću javne statičke funkcije dajInstancu().



Prototype patern

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. Mi ga nismo iskoristili, jer nije bilo potrebe.

Factory Method patern

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 naslijeđenih 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. Mi ga nismo iskoristili, jer nije bilo potrebe.

Abstract Factory patern

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 nasljeđivanja, ukida potreba za postojanjem if-else uslova jer određeni tip fabrike sadrži određene tipove objekata i zna se tačno koju podklasu će instancirati. Mi ga nismo iskoristili, jer nije bilo potrebe.

Builder patern

Builder patern služi za apstrakciju procesa konstrukcije objekta, kako bi se kao rezultat mogle 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.

Ovaj patern smo iskoristili kod Transakcije. Napravili smo klasu TransakcijaBuilder koja ima sve potrebne metode da napravi novu transakciju od datih informacija, i metodu dajTransakciju() koja vraća Transakciju napravljenu sa tim informacijama.

Ovo je prikazano na sljedećoj slici: