

Univerzitet u Beogradu
Fakultet organizacionih nauka
Laboratorija za softversko inženjerstvo

Seminarski rad iz predmeta

Softverski proces

Tema: Softverski sistem za vođenje evidencije o
postignutim rezultatima u Sudoku igrici

Profesor:

dr Siniša Vlajić

Student:

Danica Zdravković 3709/22

Beograd, 2023.

Sadržaj

1	Korisnički zahtevi.....	4
1.1	Verbalni opis.....	4
1.2	Slučajevi korišćenja	5
1.2.1	SK 1: Slučaj korišćenja – Unos novog igrača	6
1.2.2	SK 2: Slučaj korišćenja – Izmena podataka o igraču	7
1.2.3	SK 3: Slučaj korišćenja – Brisanje igrača.....	8
1.2.4	SK 4: Slučaj korišćenja – Prijava igrača.....	9
1.2.5	SK 5: Slučaj korišćenja – Ažuriranje poena	10
2	Analiza.....	11
2.1	Ponašanje softverskog sistema – Sistemski dijagram sekvenci	11
2.1.1	DS 1: Dijagram sekvenci slučaja korišćenja – Unos novog igrača.....	11
2.1.2	DS 2: Dijagram sekvenci slučaja korišćenja – Izmena podataka o igraču .	12
2.1.3	DS 3: Dijagram sekvenci slučaja korišćenja – Brisanje igrača	14
2.1.4	DS 4: Dijagram sekvenci slučaja korišćenja – Prijava igrača	16
2.1.5	DS 5: Dijagram sekvenci slučaja korišćenja – Ažuriranje poena	17
2.2	Ponašanje softverskog sistema – Definisanje ugovora o sistemskim operacijama	19
2.3	Struktura softverskog sistema – Konceptualni (domenski) model	20
2.4	Struktura softverskog sistema – Relacioni model.....	20
3	Projektovanje.....	22
3.1	Arhitektura softverskog sistema	22
3.2	Projektovanje korisničkog interfejsa	23
3.3	Projektovanje ekranskih formi	23
3.3.1	SK1: Slučaj korišćenja – Unos novog igrača	23
3.3.2	SK 2: Slučaj korišćenja – Izmena podataka o igraču	26
3.3.3	SK 3: Slučaj korišćenja – Brisanje igrača.....	28

3.3.4	SK 4: Slučaj korišćenja – Prijava igrača.....	30
3.3.5	SK 5: Slučaj korišćenja – Ažuriranje poena	32
3.4	Komunikacija sa klijentima	34
3.5	Kontroler aplikacione logike.....	36
3.6	Projektovanje strukture softverskog sistema	38
3.7	Projektovanje sistemskih operacija.....	44
3.8	Broker baze podataka	47
3.9	Projektovanje skladišta podataka	49
3.10	Principi, metode i strategije projektovanja softvera	50
3.10.1	Principi projektovanja softvera	50
3.10.2	Strategije projektovanja softvera	57
3.10.3	Metode projektovanja softvera.....	60
3.11	Primena paterna u projektovanju	64
4	Faza implementacije.....	68
5	Faza testiranja.....	72
6	Zaključak	72
7	Literatura	73

1 Korisnički zahtevi

1.1 Verbalni opis

Potrebno je napraviti informacijski sistem za igranje igrice Sudoku. Sudoku je logička igra sa brojevima.

Sistem treba da omogući prijavu igrača, registraciju igrača, vođenje evidencije o poenima igrača.

Evidencija o igračima podrazumeva: unos novog igrača u informacijski sistem, izmenu podataka o igračima, brisanje igrača kao i ažuriranje njegovih poena.

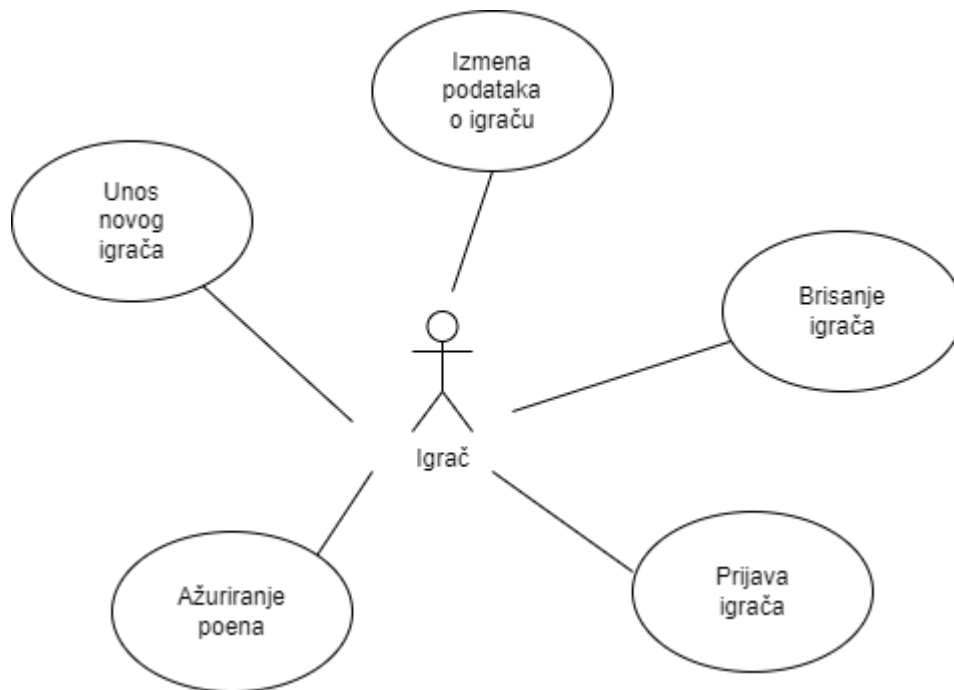
Svaki put kada igrač igra igru, nakon uspešnog popunjavanja nedostajućih polja dodeljuje mu se 1 poen za osnovni i 3 poena za napredni nivo.

Igrač može da pročita pravila igre, autora igre, kao i da pregleda svoje prikupljene poene.

1.2 Slučajevi korišćenja

U konkretnom slučaju identifikovani su sledeći slučajevi korišćenja koji su prikazani i na slici:

- 1) Unos novog igrača
- 2) Izmena podataka o igraču
- 3) Brisanje igrača
- 4) Prijava igrača
- 5) Ažuriranje poena



Slika 1-Dijagram slučajeva korišćenja

1.2.1 SK 1: Slučaj korišćenja – Unos novog igrača

Naziv SK

Unos novog igrača

Aktori SK

Igrač

Učesnici SK

Igrač i sistem (program)

Preduslov: **Sistem** je uključen i prikazana je forma za prijavu **igrača**. Klikom na dugme registruj se otvara se forma za registrovanje novog igrača. **Sistem** prikazuje formu za registraciju igrača.

Osnovni scenario SK

1. **Igrač** unosi svoje podatke potrebne za unos novog igrača u sistem. (APUSO)
2. **Igrač** kotroliše da li je korektno uneo podatke o sebi. (ANSO)
3. **Igrač** poziva **sistem** da zapamti podatke o njemu. (APSO)
4. **Sistem** pamti podatke o novom igraču. (SO)
5. **Sistem** prikazuje **igraču** poruku: "**Sistem** je zapamtio igrača". (IA)

Alternativna scenarija

- 5.1. Ukoliko **sistem** ne može da zapamti podatke o novom igraču prikazuje **igraču** poruku "**Sistem** ne može da zapamti novog igrača". (IA)

1.2.2 SK 2: Slučaj korišćenja – Izmena podataka o igraču

Naziv SK

Izmena podataka o igraču

Aktori SK

Igrač

Učesnici SK

Igrač i sistem (program)

Preduslov Sistem je uključen i **igrač** je ulogovan sa svojom šifrom. Izabrana je opcija "izmeni podatke" sa početne forme. **Sistem** prikazuje formu za rad sa igračem. Prikazani su trenutni podaci o igraču.

Osnovni scenario SK

1. **Igrač** poziva **sistem** da prikaže podatke o ulogovanom **igraču**. (APSO)
2. **Sistem** traži podatke o ulogovanom **igraču**. (SO)
3. **Sistem** prikazuje **igraču** njegove podatke i poruku: "Sistem je pronašao podatke o igraču". (IA)
4. **Igrač** unos (menja) svoje podatke. (APUSO)
5. **Igrač** kotroliše da li je korektno uneo svoje podatke. (ANSO)
6. **Igrač** poziva **sistem** da zapamti izmene. (APSO)
7. **Sistem** pamti podatke o **igraču**. (SO)
8. **Sistem** prikazuje **igraču** poruku: "**Sistem** je zapamtio promene" (IA)

Alternativna scenarija:

- 3.1. Ukoliko **sistem** ne može da nađe podatke o **igraču** prikazuje poruku: "**Sistem** ne može da nađe podatke o **igraču**". Prekida se izvršenje scenarija. (IA)
- 8.1. Ukoliko **sistem** ne može da zapamti promene nad podacima prikazuje **igraču** poruku: "**Sistem** ne može da zapamti promene nad podacima". (IA)

1.2.3 SK 3: Slučaj korišćenja – Brisanje igrača

Naziv SK

Brisanje igrača

Aktori SK

Igrač

Učesnici SK

Igrač i sistem (program)

Preduslov: **Sistem** je uključen i **igrač** je ulogovan sa svojom šifrom. Izabrana je opcija "brisanje naloga" sa početne forme. **Sistem** prikazuje formu za brisanje naloga.

Osnovni scenario SK

1. **Igrač** poziva **sistem** da obriše njegov nalog. (APSO)
2. **Sistem** briše igrača. (SO)
3. **Sistem** prikazuje **igraču** poruku: "**Sistem** je uspešno obrisao igrača." (IA)

Alternativna scenarija

- 3.1. Ukoliko **sistem** ne može da obriše igrača prikazuje **igraču** poruku: "**Sistem** ne može da obriše igrača". (IA)

1.2.4 SK 4: Slučaj korišćenja – Prijava igrača

Naziv SK

Prijava igrača

Aktori SK

Igrač

Učesnici SK

Igrač i sistem (program)

Preduslov: Sistem je uključen i prikazuje formu za prijavu igrača.

Osnovni scenario SK

1. **Igrač** unosi podatke za prijavu **igrača**. (APUSO)
2. **Igrač** kotroliše da li je korektno uneo podatke za prijavu. (ANSO)
3. **Igrač** poziva **sistem** da pronađe **igrača** sa zadatim podacima. (APSO)
4. **Sistem** pretražuje **igrača**. (SO)
5. **Sistem** prikazuje igraču poruku :"**Igrač** je uspešno ulogovan u sistem" i omogućava pristup sistemu. (IA)

Alternativna scenarija

- 5.1. Ukoliko **sistem** ne može da nađe **igrača** prikazuje poruku: "Sistem ne može da nađe **igrača** na osnovu unetih vrednosti za prijavljivanje". (IA)

1.2.5 SK 5: Slučaj korišćenja – Ažuriranje poena

Naziv SK

Ažuriranje poena

Aktori SK

Igrač

Učesnici SK

Igrač i sistem (program)

Preduslov Sistem je uključen i **igrač** je ulogovan sa svojom šifrom. **Sistem** prikazuje formu za igranje igre.

Osnovni scenario SK

1. **Igrač** poziva **sistem** da ažurira poene. (APSO)
2. **Sistem** pamti podatke o ažuriranim poenima **igrača**. (SO)
3. **Sistem** prikazuje **igraču** poruku: “**Sistem** je zapamtio promene poena.” (IA)

Alternativna scenarija:

- 3.1. Ukoliko **sistem** ne može da zapamti promene nad podacima prikazuje **igraču** poruku: “**Sistem** ne može da zapamti promene nad podacima”. (IA)

2 Analiza

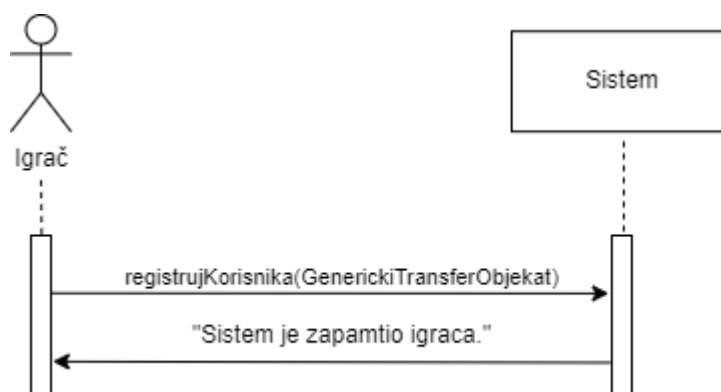
Faza analize opisuje logičku strukturu i ponašanje softverskog sistema. Ponašanje softverskog sistema opisano je preko sistemskih operacija, a struktura preko konceptualnog i relacionog modela.

2.1 Ponašanje softverskog sistema – Sistemski dijagram sekvenci

Ponašanje softverskog sistema se opisuje preko UML sekvencnih dijagrama, odnosno preko dijagrama saradnje.

2.1.1 DS 1: Dijagram sekvenci slučaja korišćenja – Unos novog igrača

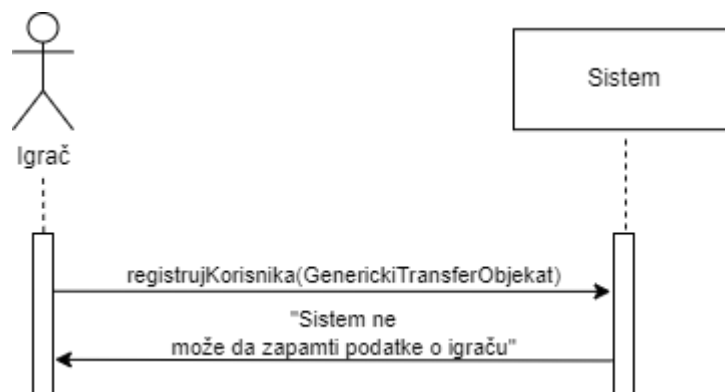
1. **Igrač** poziva **sistem** da zapamti podatke o njemu. (APSO)
2. **Sistem** prikazuje **igraču** poruku: “**Sistem** je zapamtio igrača”. (IA)



Slika 2-Dijagram sekvenci za unos novog igrača

Alternativna scenarija

- 2.1 Ukoliko **sistem** ne može da zapamti igrača prikazuje **igraču** poruku: “**Sistem** ne može da zapamti podatke o igraču”. (IA)



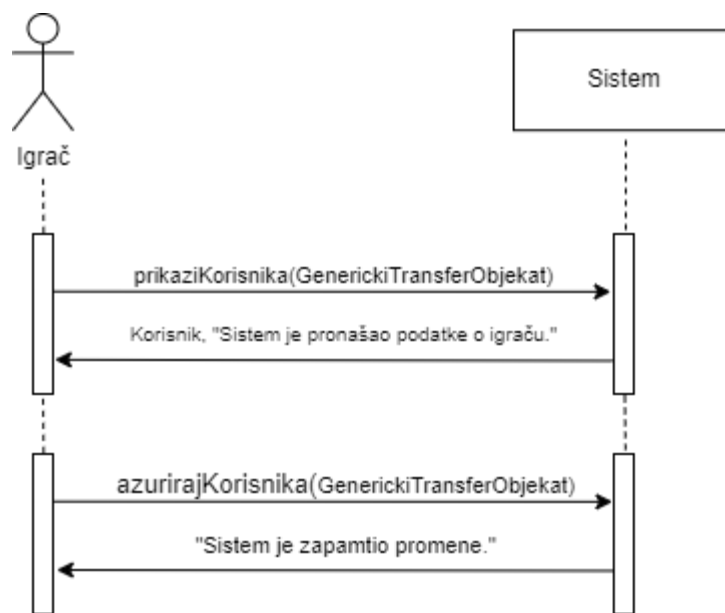
Slika 3-Dijagram sekvenci za unos novog igrača alternativni scenario

Sa navedenih sekvencnih dijagrama uočava se 1 sistemska operacija koju treba projektovati:

1. *signal* **registrujKorisnika**(*Korisnik*)

2.1.2 DS 2: Dijagram sekvenci slučaja korišćenja – Izmena podataka o igraču

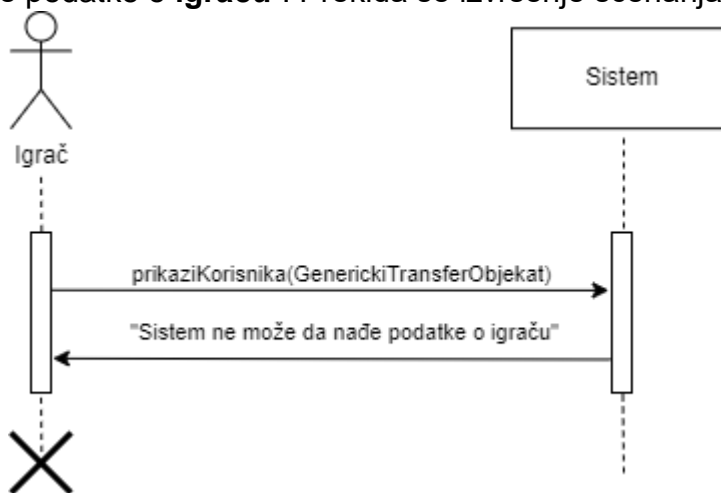
1. **Igrač** poziva **sistem** da prikaže podatke o ulagovanom igraču. (APSO)
2. **Sistem** prikazuje **igraču** njegove podatke i poruku: "**Sistem** je pronašao podatke o igraču". (IA)
3. **Igrač** poziva **sistem** da zapamti izmene. (APSO)
4. **Sistem** prikazuje **igraču** poruku: "**Sistem** je zapamtio promene". (IA)



Slika 4-Dijagram sekvenci za ažuriranje podataka o igraču

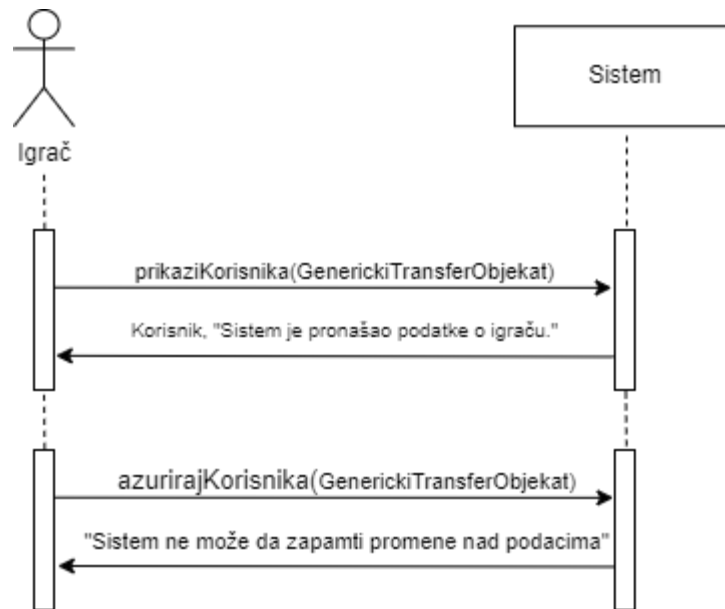
Alternativna scenarija

2.1. Ukoliko **sistem** ne može da nađe podatke o **igraču** prikazuje poruku: "**Sistem** ne može da nađe podatke o **igraču**". Prekida se izvršenje scenarija. (IA)



Slika 5-Dijagram sekvenci za ažuriranje podataka o igraču alternativni scenario 1

4.1. Ukoliko **sistem** ne može da zapamti promene nad podacima prikazuje **igraču** poruku: "**Sistem** ne može da zapamti promene nad podacima". (IA)



Slika 6-Dijagram sekvenci za ažuriranje podataka o igraču alternativni scenario 2

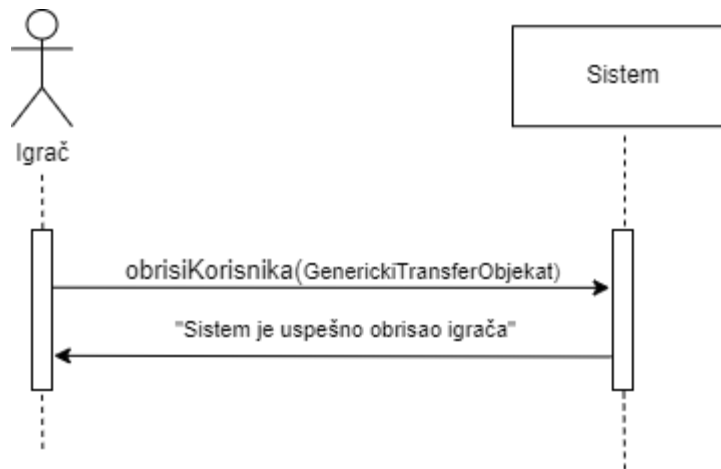
Sa navedenih sekvencnih dijagrama uočavaju se 2 systemske operacije koje treba projektovati:

1. *signal* **prikaziKorisnika**(*Korisnik*)
2. *signal* **azurirajKorisnika**(*Korisnik*)

2.1.3 DS 3: Dijagram sekvenci slučaja korišćenja – Brisanje igrača

Osnovni scenario SK

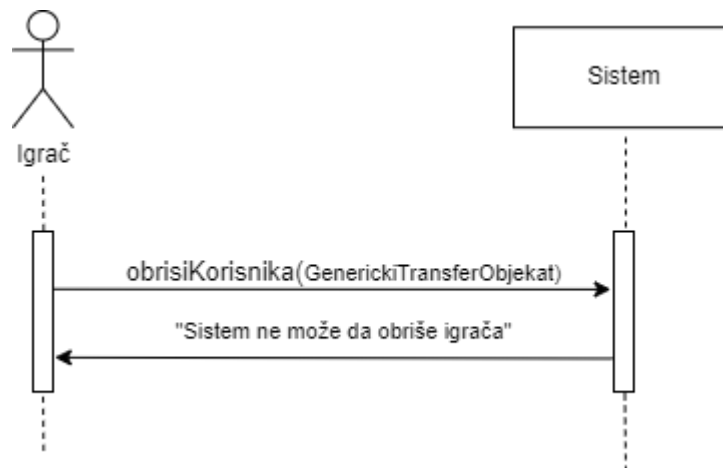
1. **Igrač** poziva **sistem** da obriše njegov nalog. (APSO)
2. **Sistem** prikazuje **igraču** poruku: "**Sistem** je uspešno obrisao igrača." (IA)



Slika 7-Dijagram sekvenci za brisanje igrača

Alternativna scenarija

- 2.1. Ukoliko **sistem** ne može da obriše igrača prikazuje **igraču** poruku: "**Sistem ne može da obriše igrača**". (IA)



Slika 8-Dijagram sekvenci za brisanje igrača alternativni scenario

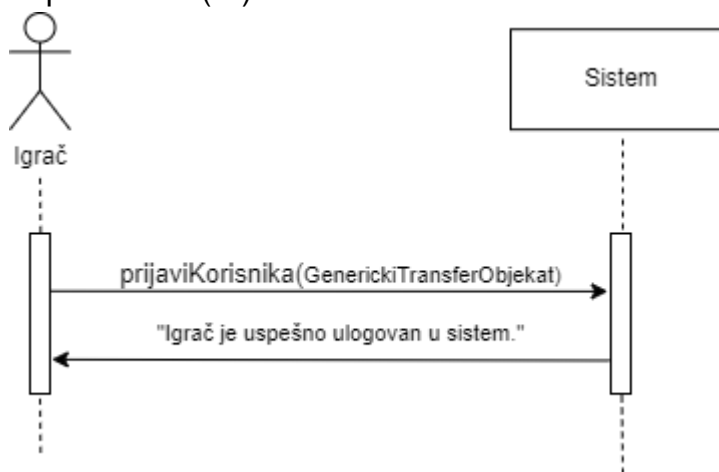
Sa navedenih sekvencnih dijagrama uočava se 1 sistemska operacija koju treba projektovati:

1. *signal* **obrisiKorisnika(Korisnik)**

2.1.4 DS 4: Dijagram sekvenci slučaja korišćenja – Prijava igrača

Osnovni scenario SK

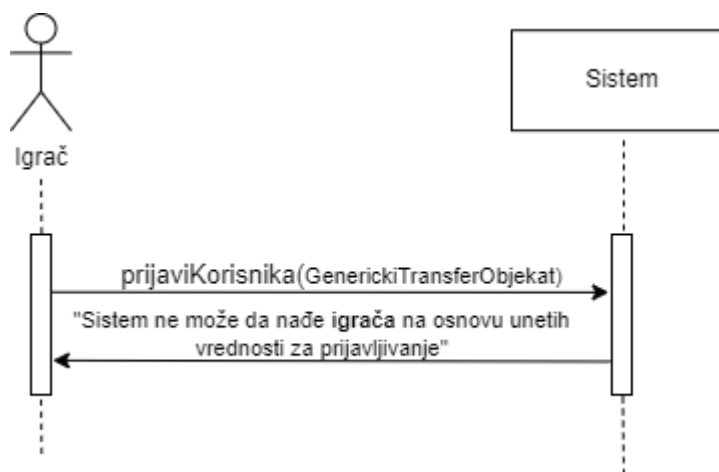
1. **Igrač** poziva **sistem** da pronade **igrača** sa zadatim podacima. (APSO)
2. **Sistem** prikazuje igraču poruku :"**Igrač** je uspešno ulogovan u sistem" i omogućava pristup sistemu. (IA)



Slika 9-Dijagram sekvenci za prijavu igrača

Alternativna scenarija

- 2.1. Ukoliko **sistem** ne može da nađe **igrača** prikazuje poruku: "Sistem ne može da nađe **igrača** na osnovu unetih vrednosti za prijavljivanje". (IA)



Slika 10-Dijagram sekvenci za prijavu igrača alternativni scenario

Sa navedenih sekvencnih dijagrama uočavaju se 1 sistemska operacija koju treba projektovati:

1. *signal prijaviKorisnika*(Korisnik)

2.1.5 DS 5: Dijagram sekvenci slučaja korišćenja – Ažuriranje poena

Naziv SK

Ažuriranje poena

Aktori SK

Igrač

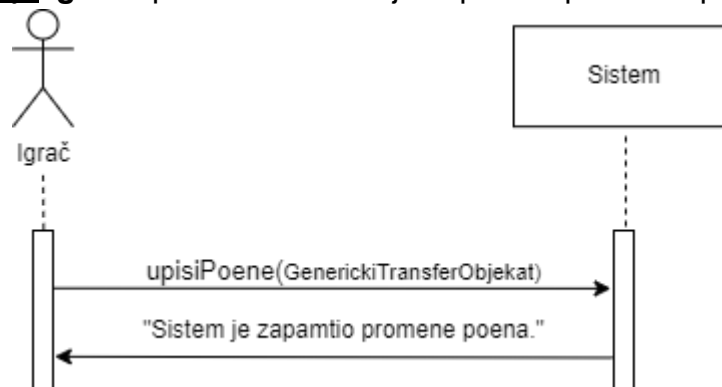
Učesnici SK

Igrač i sistem (program)

Preduslov Sistem je uključen i **igrač** je ulogovan sa svojom šifrom. **Sistem** prikazuje formu za igranje igre.

Osnovni scenario SK

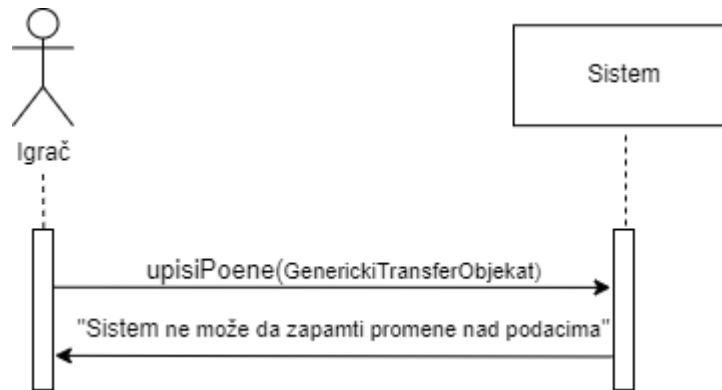
1. **Igrač** poziva **sistem** da ažurira poene. (APSO)
2. **Sistem** pamti podatke o ažuriranim poenima **igrača**. (SO)
3. **Sistem** prikazuje **igraču** poruku: “**Sistem** je zapamtio promene poena.” (IA)



Slika 11-Dijagram sekvenci za ažuriranje poena igrača

Alternativna scenarija:

- 3.1. Ukoliko **sistem** ne može da zapamti promene nad podacima prikazuje **igraču** poruku: “**Sistem** ne može da zapamti promene nad podacima”. (IA)



Slika 12-Dijagram sekvenci za ažuriranje poena igrača alternativni scenario

Sa navedenih sekvencnih dijagrama uočava se 1 sistemska operacija koju treba projektovati:

1. *signal* **upisiPoene**(*Korisnik*)

Kao rezultat analize scenarija dobijeno je ukupno 8 sistemskih operacija koje treba projektovati:

1. *signal* **registrujKorisnika**(*Korisnik*)
2. *signal* **azurirajKorisnika**(*Korisnik*)
3. *signal* **obrisiKorisnika**(*Korisnik*)
4. *signal* **prijaviKorisnika**(*Korisnik*)
5. *signal* **azurirajPoene**(*Korisnik*)

2.2 Ponašanje softverskog sistema – Definisanje ugovora o sistemskim operacijama

Ugovor UG1: RegistrujKorisnika

Operacija: registrujKorisnika(*Korisnik*):signal;

Veza sa SK: SK1

Preduslovi: Vrednosna ograničenja nad objektom Korisnik moraju biti zadovoljena.

Strukturno ograničenje nad objektom Korisnik mora biti zadovoljeno.

Postuslovi: Podaci o korisniku su zapamćeni.

Ugovor UG2: AzurirajKorisnika

Operacija: azurirajKorisnika(*Korisnik*):signal;

Veza sa SK: SK2

Preduslovi: Vrednosna ograničenja nad objektom Korisnik moraju biti zadovoljena.

Strukturno ograničenje nad objektom Korisnik mora biti zadovoljeno.

Postuslovi: Podaci o korisniku su zapamćeni.

Ugovor UG3: ObrisiKorisnika

Operacija: obrisiKorisnika(*Korisnik*):signal;

Veza sa SK: SK3

Preduslovi: -

Postuslovi: Korisnik je obrisao. Strukturno ograničenje nad objektom Korisnik mora biti zadovoljeno.

Ugovor UG4: PrijavaKorisnika

Operacija: prijavaKorisnika(*Korisnik*):signal;

Veza sa SK: SK4

Preduslovi: -

Postuslovi: -

Ugovor UG5: AzurirajPoene

Operacija: upisiPoene(Korisnik):signal;

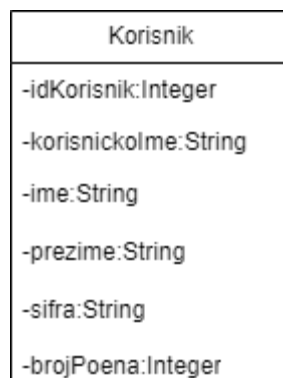
Veza sa SK: SK5

Preduslovi: -

Postuslovi: -

2.3 Struktura softverskog sistema – Konceptualni (domenski) model

Konceptualnim modelom je opisana struktura sistema. Konceptualni model se sastoji od konceptualnih klasa i asocijacija između konceptualnih klasa.



Slika 13-Konceptualni model

2.4 Struktura softverskog sistema – Relacioni model

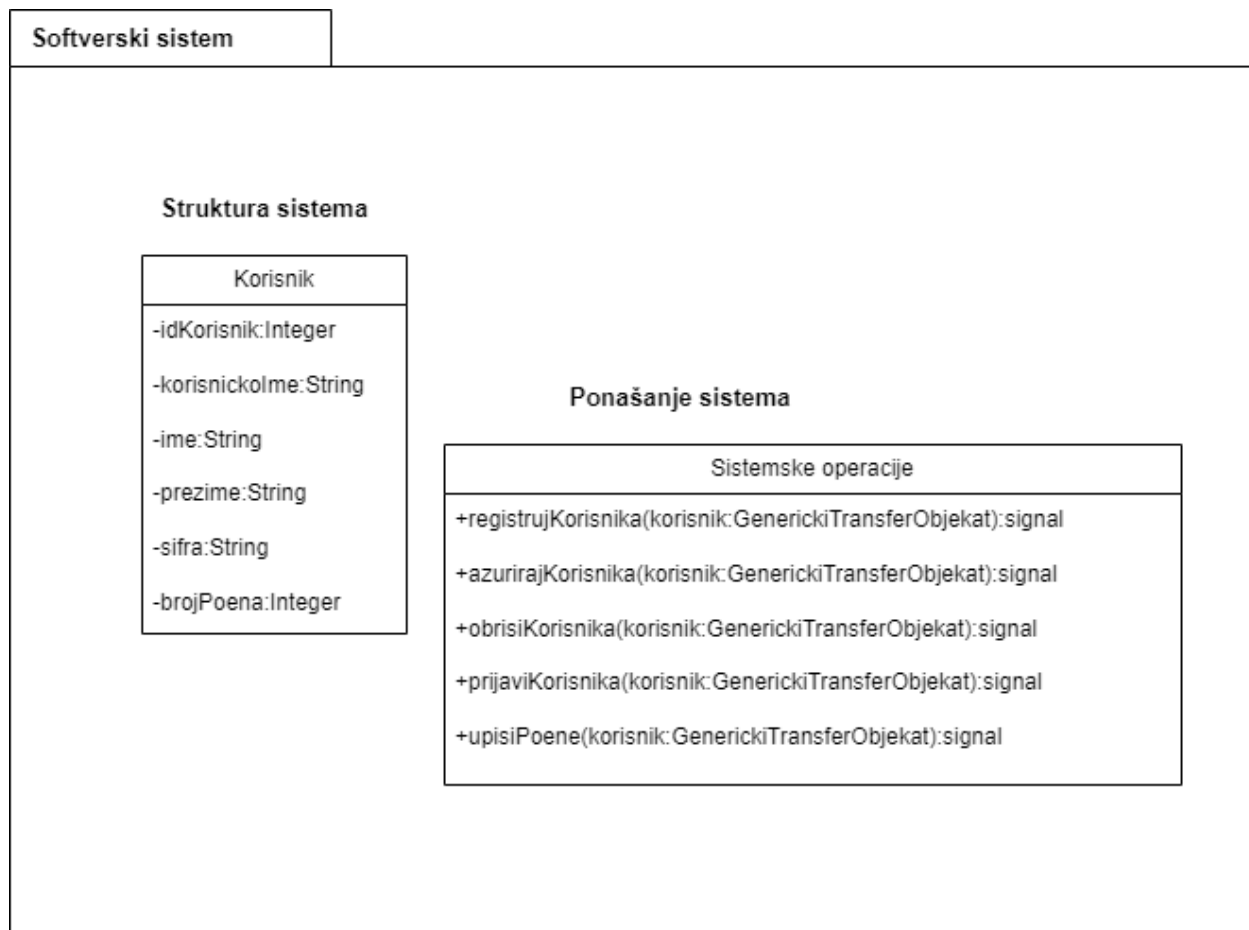
Korisnik(idKorisnik, korisnickolme, ime, prezime, sifra, brojPoena)

Tabela Korisnik		Prosto vrednosno ograničenje		Složeno vrednosno ograničenje		Strukturno ograničenje
Atributi	Ime	Tip atributa	Vrednost atributa	Međuzav. atributa jedne tabele	Međuzav. atributa više tabela	INSERT / UPDATE / DELETE /
	idKorisnik	Integer	not null and >0			
	korisnickolme	String	not null			
	ime	String	not null			
	prezime	String	not null			

	sifra	String	not null			
	brojPoena	Integer	>0			

Tabela 1-Tabela korisnik

Kao rezultat analize scenarija SK i pravljenja konceptualnog modela dobija se logička struktura i ponašanje softverskog sistema:



Slika 14-Logička stuktura i ponašanje softverskog sistema

3 Projektovanje

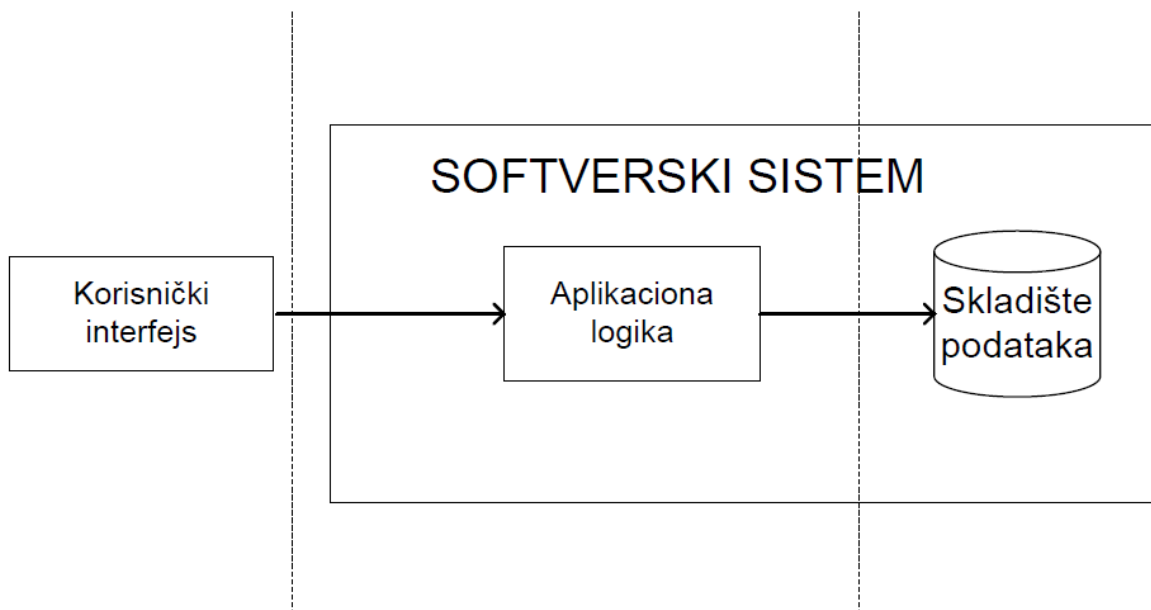
Faza projektovanja opisuje fizičku strukturu i ponašanje softverskog sistema (arhitekturu softverskog sistema).

3.1 Arhitektura softverskog sistema

Arhitektura sistema je tronivojska i sastoji se od sledećih nivoa:

- korisnički interfejs
- aplikaciona logika
- skladište podataka

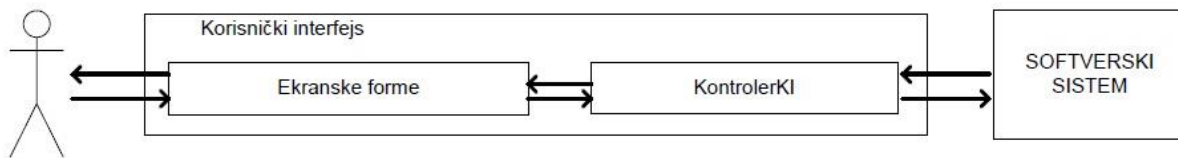
Nivo korisničkog interfejsa je na strani klijenta, dok su aplikaciona logika i skladište podataka na strani servera.



Slika 15-Tronivojska arhitektura

3.2 Projektovanje korisničkog interfejsa

Projektovanje korisničkog interfejsa predstavlja realizaciju ulaza i/ili izlaza softverskog sistema.



Slika 16-Stuktura korisničkog interfejsa

Ekranska forma ima ulogu da prihvati podatke koje korisnik unosi, prihvata događaje koje pravi korisnik, poziva kontrolera korisničkog interfejsa kako bi mu prosledila podatke i prikazuje podatke dobijene od kontrolera korisničkog interfejsa.

3.3 Projektovanje ekranskih formi

Korisnički interfejs defenisan je preko skupa ekranskih formi. Scenarija korišćenja ekranskih formi su direktno povezana sa scenarijima slučajeva korišćenja.

3.3.1 SK1: Slučaj korišćenja – Unos novog igrača

Naziv SK

Unos novog igrača

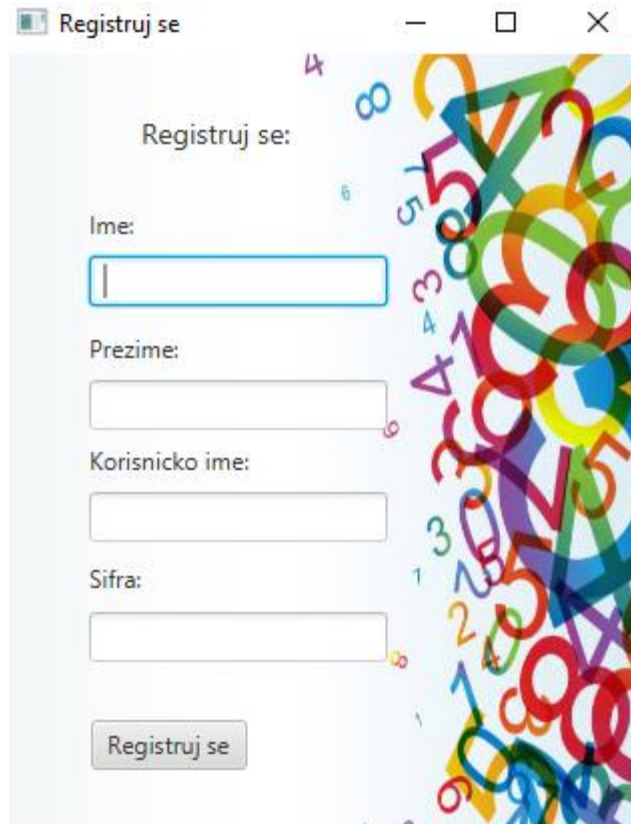
Aktori SK

Igrač

Učesnici SK

Igrač i sistem (program)

Preduslov: **Sistem** je uključen i prikazana je forma za prijavu **igrača**. Klikom na dugme registruj se otvara se forma za registrovanje novog igrača. **Sistem** prikazuje formu za registraciju igrača.

A screenshot of a web application window titled "Registruj se". The window contains a registration form with the following elements: a title "Registruj se:", a label "Ime:" followed by a text input field, a label "Prezime:" followed by a text input field, a label "Korisnicko ime:" followed by a text input field, a label "Sifra:" followed by a text input field, and a "Registruj se" button at the bottom. The right side of the window features a decorative background with colorful, overlapping numbers and symbols.

Slika 17- Forma registracija

Osnovni scenario SK

1. **Igrač** unosi svoje podatke potrebne za unos novog igrača u sistem. (APUSO)
2. **Igrač** kotroliše da li je korektno uneo podatke o sebi. (ANSO)

Registruj se

Registruj se:

Ime:
Pera

Prezime:
Peric

Korisnicko ime:
pera

Sifra:
....

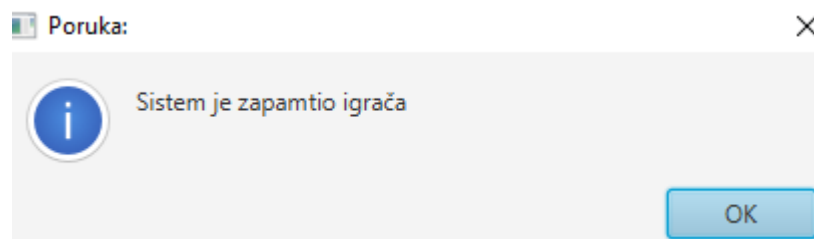
Registruj se

Slika 18-Popunjena forma registracija

3. **Igrač poziva sistem** da zapamti podatke o njemu. (APSO)

Opis akcije: Igrač pritiskom na dugme "Registruj se" poziva sistemsku operaciju **registrujKorisnika**(Korisnika) koja proverava da li korisnik koji se registruje postoji u bazi, ako ne postoji dodaje novog korisnika.

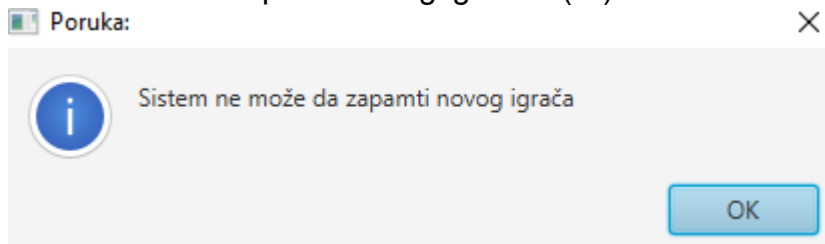
4. **Sistem pamti** podatke o novom igraču. (SO)
5. **Sistem prikazuje igraču** poruku: "**Sistem** je zapamtio igrača". (IA)



Slika 19-Poruka da je igrač upamćen

Alternativna scenarija

5.1. Ukoliko **sistem** ne može da zapamti podatke o novom igraču prikazuje **igraču** poruku “**Sistem** ne može da zapamti novog igrača”. (IA)



Slika 20-Poruka da igrač nije upamćen

3.3.2 SK 2: Slučaj korišćenja – Izmena podataka o igraču

Naziv SK

Izmena podataka o igraču

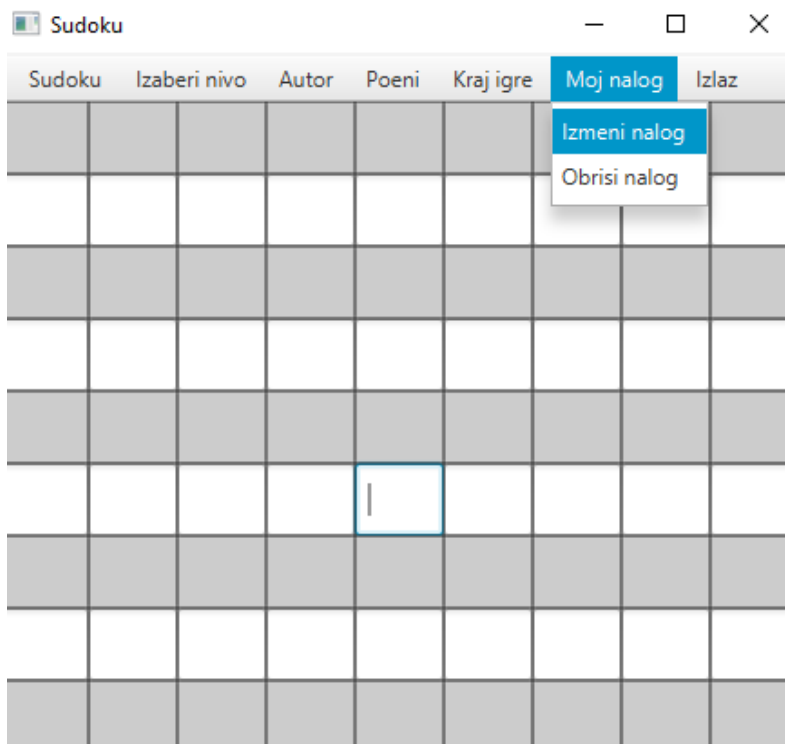
Aktori SK

Igrač

Učesnici SK

Igrač i sistem (program)

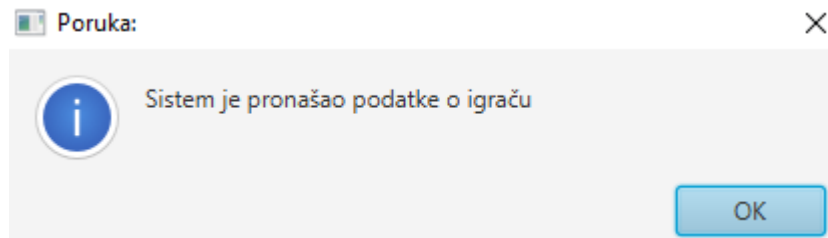
Preduslov Sistem je uključen i **igrač** je ulogovan sa svojom šifrom. Izabrana je opcija “izmeni podatke” sa početne forme. **Sistem** prikazuje formu za rad sa igračem. Prikazani su trenutni podaci o igraču.



Slika 21-Glavna forma

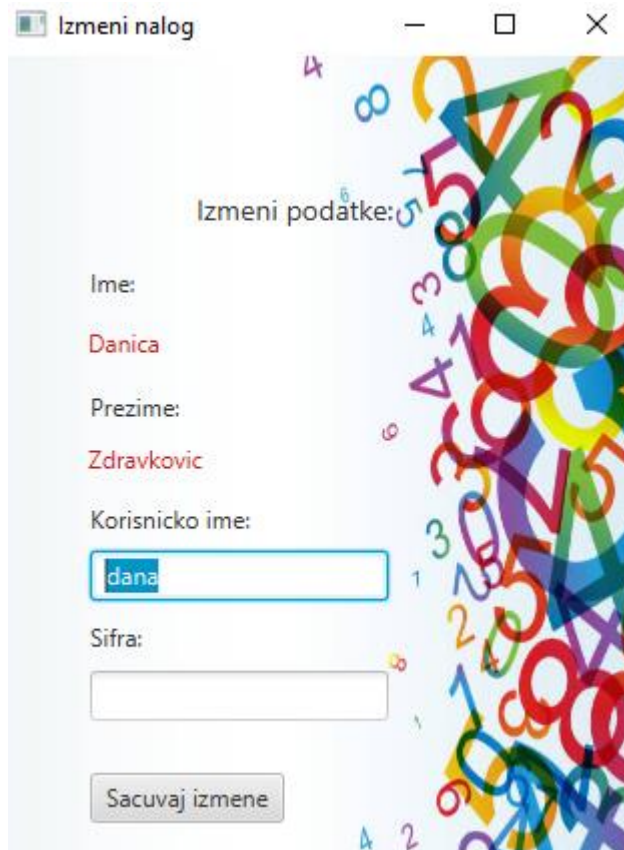
Osnovni scenario SK

1. **Igrač** poziva **sistem** da prikaže podatke o ulogovanom **igraču**. (APSO)
2. **Sistem** traži podatke o ulogovanom **igraču**. (SO)
3. **Sistem** prikazuje **igraču** njegove podatke i poruku: "Sistem je pronašao podatke o igraču". (IA)



Slika 22-Poruka da je igrač pronađen

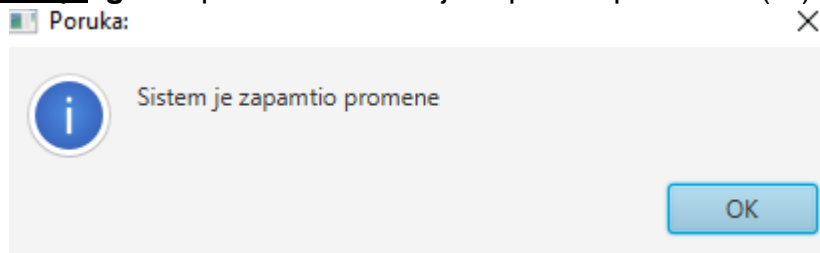
4. **Igrač** unosi (menja) svoje podatke. (APUSO)
5. **Igrač** kotroliše da li je korektno uneo svoje podatke. (ANSO)



Slika 23-Forma za izmenu podataka igrača

Opis akcije: Igrač pritiskom na dugme "Sacuvaj izmene" poziva sistemsku operaciju **azurirajKorisnika**(Korisnika) koja ažurira podatke o korisniku.

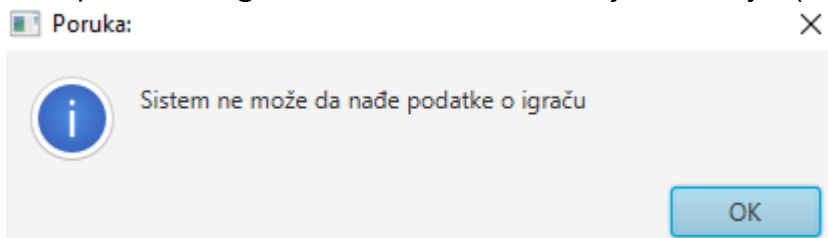
6. **Igrač** poziva **sistem** da zapamti izmene. (APSO)
7. **Sistem** pamti podatke o **igraču**. (SO)
8. **Sistem** prikazuje **igraču** poruku: “**Sistem** je zapamtio promene” (IA)



Slika 24-Poruka da je sistem zapamtio promene

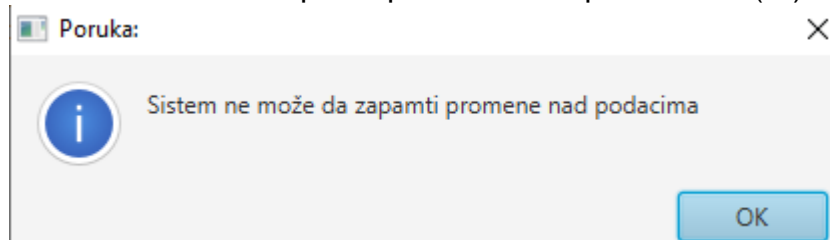
Alternativna scenarija:

- 3.1. Ukoliko **sistem** ne može da nađe podatke o **igraču** prikazuje poruku: “**Sistem** ne može da nađe podatke o **igraču**”. Prekida se izvršenje scenarija. (IA)



Slika 25-Poruka da sistem ne može da nađe podatke o igraču

- 8.1. Ukoliko **sistem** ne može da zapamti promene nad podacima prikazuje **igraču** poruku: “**Sistem** ne može da zapamti promene nad podacima”. (IA)



Slika 26-Poruka da sistem ne može da zapamti izmene

3.3.3 SK 3: Slučaj korišćenja – Brisanje igrača

Naziv SK

Izmena podataka o igraču

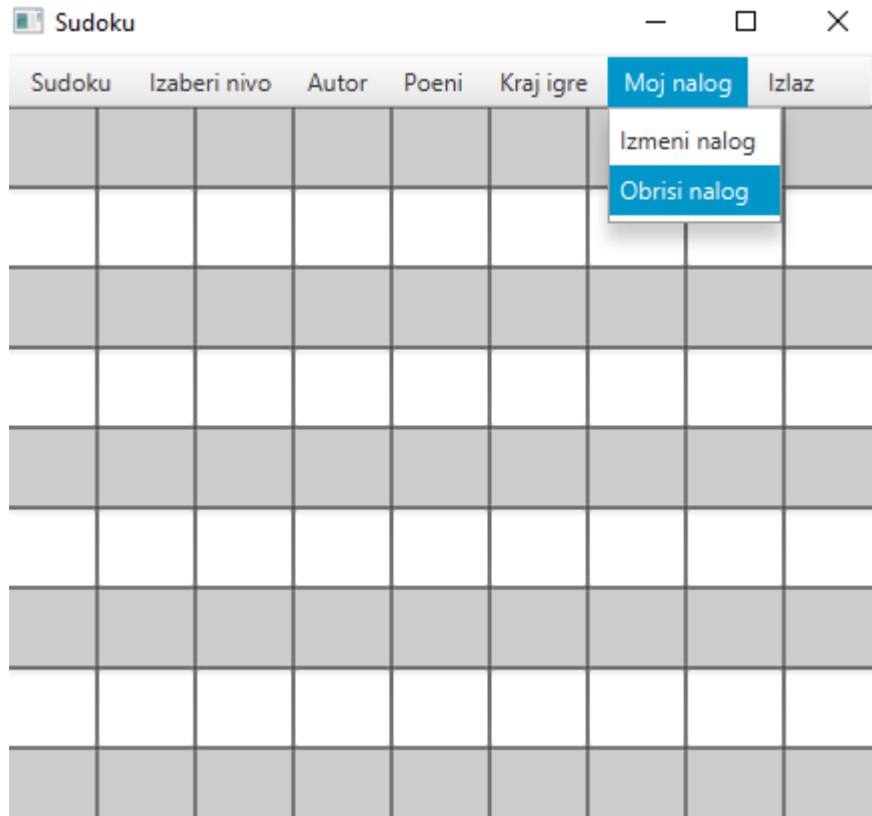
Aktori SK

Igrač

Učesnici SK

Igrač i sistem (program)

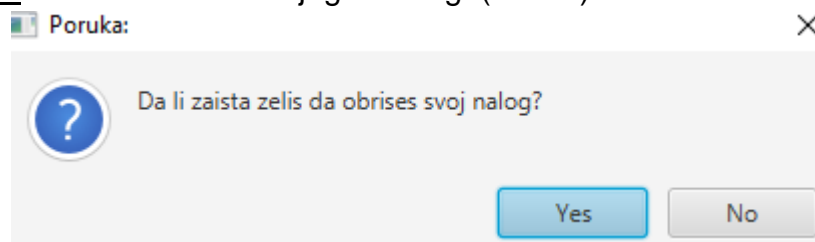
Preduslov: **Sistem** je uključen i **igrač** je ulogovan sa svojom šifrom. Izabrana je opcija “brisanje naloga” sa početne forme. **Sistem** prikazuje formu za brisanje naloga.



Slika 27-Glavna forma

Osnovni scenario SK

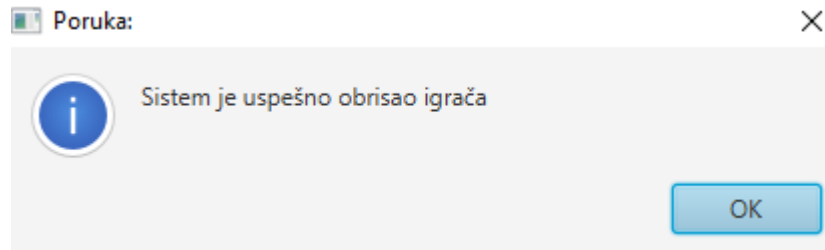
1. **Igrač** poziva **sistem** da obriše njegov nalog. (APSO)



Slika 28-Poruka potvrde brisanja naloga

Opis akcije: Igrač pritiskom na dugme “Yes” poziva sistemsku operaciju **obrisiKorisnika**(Korisnika) koja briše korisnika iz baze podataka.

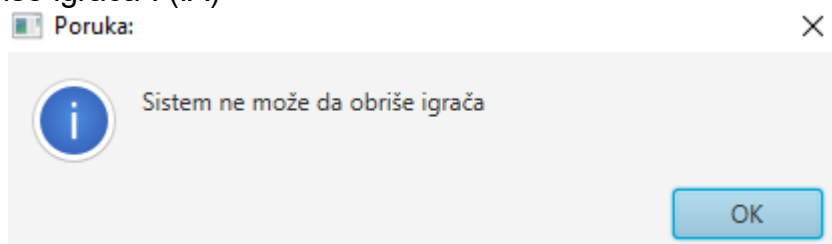
2. **Sistem** briše igrača. (SO)
3. **Sistem** prikazuje **igraču** poruku: “**Sistem** je uspešno obrisao igrača.” (IA)



Slika 29-Poruka da je igrač izbrisan

Alternativna scenarija

3.1. Ukoliko **sistem** ne može da obriše igrača prikazuje **igraču** poruku: “**Sistem** ne može da obriše igrača”. (IA)



Slika 30-Poruka da igrač nije izbrisan

3.3.4 SK 4: Slučaj korišćenja – Prijava igrača

Naziv SK

Prijava igrača

Aktori SK

Igrač

Učesnici SK

Igrač i sistem (program)

Preduslov: Sistem je uključen i prikazuje formu za prijavu igrača.

Slika 31-Forma prijava

Osnovni scenario SK

1. **Igrač** unosi podatke za prijavu **igrača**. (APUSO)
2. **Igrač** kotroliše da li je korektno uneo podatke za prijavu. (ANSO)

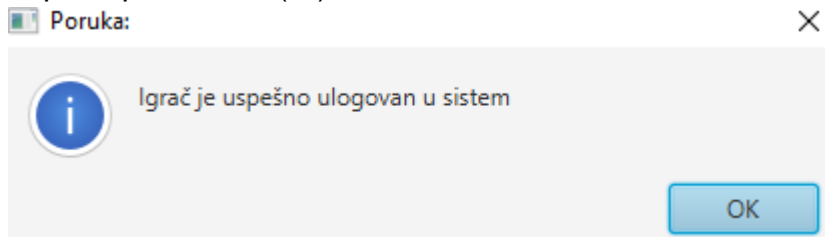
Slika 32-Popunjena forma prijava

3. **Igrač** poziva **sistem** da pronade **igrača** sa zadatim podacima. (APSO)

4. **Sistem** pretražuje igrača. (SO)

*Opis akcije: Igrač pritiskom na dugme "Prijavi se" poziva sistemsku operaciju **prijaviKorisnika**(Korisnika) koja proverava da li korisnik koji se prijavljuje postoji u bazi.*

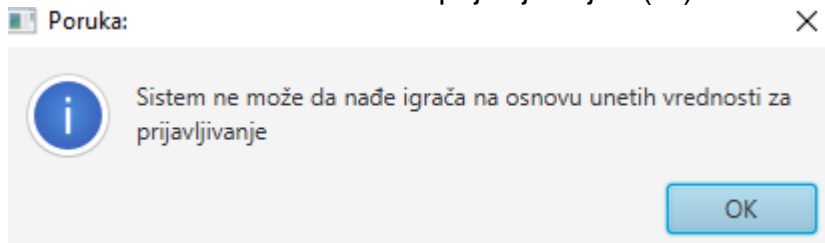
5. **Sistem** prikazuje igraču poruku :"**Igrač** je uspešno ulogovan u sistem" i omogućava pristup sistemu. (IA)



Slika 33-Poruka o uspešnom prijavljivanju

Alternativna scenarija

5.1. Ukoliko **sistem** ne može da nađe **igrača** prikazuje poruku: "Sistem ne može da nađe **igrača** na osnovu unetih vrednosti za prijavljivanje". (IA)



Slika 34-Poruka o neuspešnom prijavljivanju

3.3.5 SK 5: Slučaj korišćenja – Ažuriranje poena

Naziv SK

Ažuriranje poena

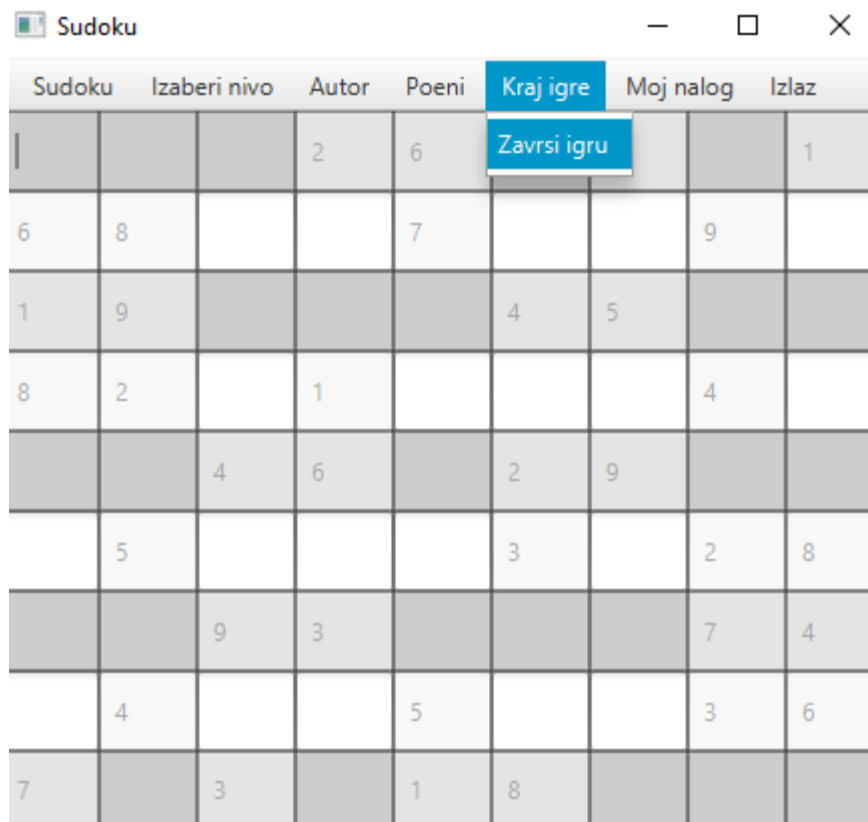
Aktori SK

Igrač

Učesnici SK

Igrač i sistem (program)

Preduslov Sistem je uključen i **igrač** je ulogovan sa svojom šifrom. **Sistem** prikazuje formu za igranje igre.



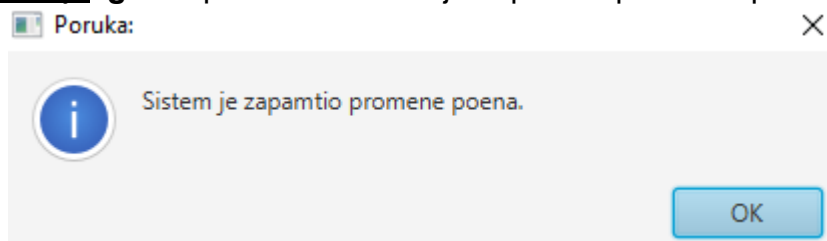
Slika 35-Glavna forma sa postavkom igre

Osnovni scenario SK

1. **Igrač** poziva **sistem** da ažurira poene. (APSO)
2. **Sistem** pamti podatke o ažuriranim poenima **igrača**. (SO)

Opis akcije: Igrač pritiskom na podopciju "Kraj igre" poziva sistemsku operaciju **upisiPoene**(Korisnika) koja ažurira poene igrača u bazi.

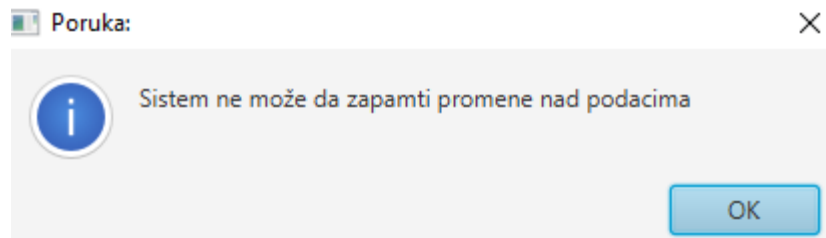
3. **Sistem** prikazuje **igraču** poruku: "**Sistem** je zapamtio promene poena." (IA)



Slika 36-Poruka o ažuriranju poena

Alternativna scenarija:

- 3.1. Ukoliko **sistem** ne može da zapamti promene nad podacima prikazuje **igraču** poruku: "**Sistem** ne može da zapamti promene nad podacima". (IA)



Slika 37-Poruka da poeni ne mogu da se ažuriraju

3.4 Komunikacija sa klijentima

Deo za komunikaciju podiže serverski soket koji će da osluškuje mrežu. Kada klijentski soket uspostavi konekciju sa serverskim soketom, tada server generiše nit koja će uspostaviti dvosmernu vezu sa klijentom.

Slanje i primanje podataka od klijenta se obavlja razmenom objekata klase `GenerickiTransferObjekat` i ostvaruje se preko soketa.

Klijent šalje zahtev za izvršenje neke od sistemskih operacija do odgovarajuće niti koja je povezana sa tim klijentom. Ta nit prihvata zahtev i prosleđuje ga do kontrolera aplikacione logike. Nakon izvršenja sistemske operacije rezultat se, preko kontrolera aplikacione logike, vraća do niti klijenta koja taj rezultat šalje nazad do klijenta.

```
package TransferObjekat;

import DomenskiObjekat.GeneralDObject;
import java.io.Serializable;

public class GenerickiTransferObjekat implements Serializable {

    private static final long serialVersionUID = 6529685098267757690L;

    public GeneralDObject gdo;

    public String poruka;

    public boolean signal;

    public String nazivOperacije;

    public int currentRecord = -1;

    public void setDK(GeneralDObject gdo) {
```

```

        this.gdo = gdo;
    }

    public String getPoruka() {
        return poruka;
    }

    public boolean getSignal() {
        return signal;
    }

    public GeneralDObject getDK() {
        return gdo;
    }

    public void setNazivOperacije(String nazivOperacije) {
        this.nazivOperacije = nazivOperacije;
    }

    public void setSignal(boolean signal) {
        this.signal = signal;
    }

    public void setGdo(GeneralDObject gdo) {
        this.gdo = gdo;
    }

    public void setPoruka(String poruka) {
        this.poruka = poruka;
    }

```

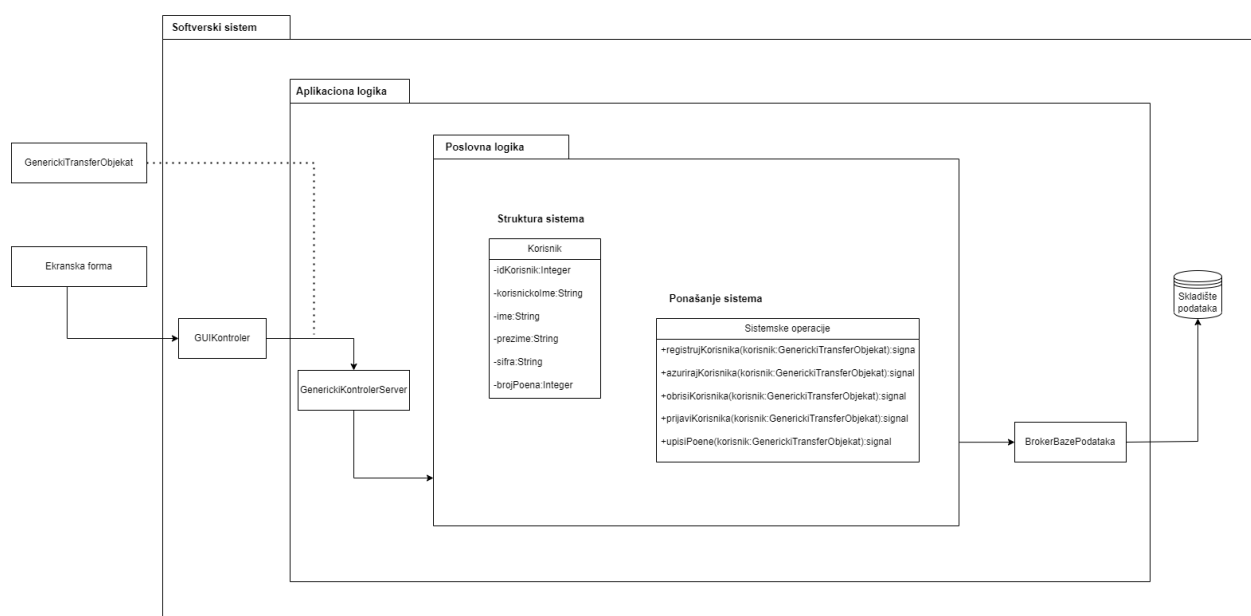
```

}
}

```

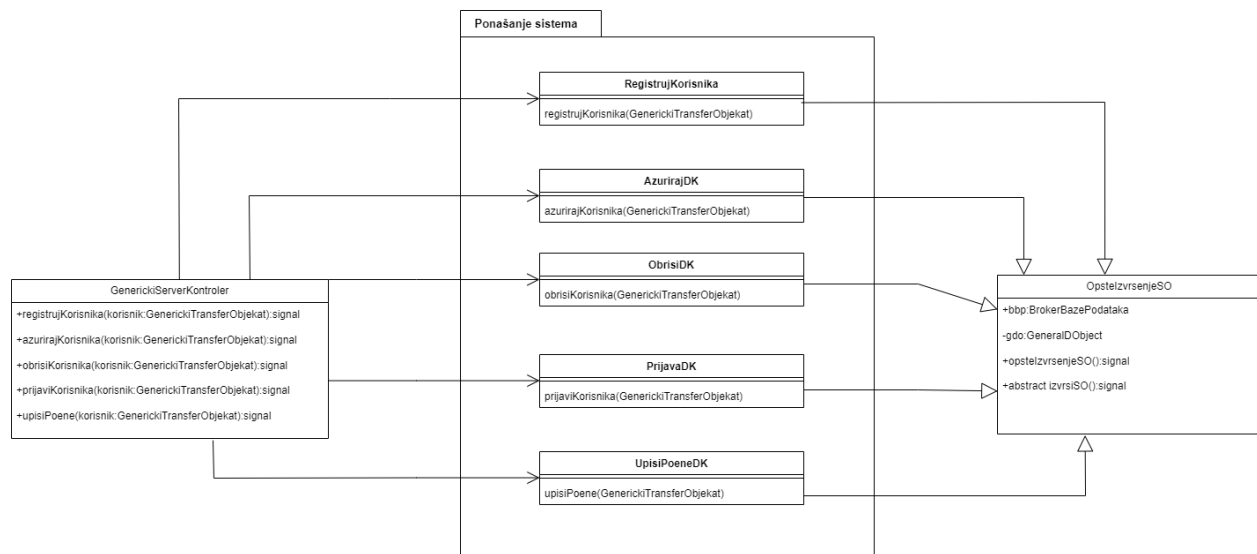
3.5 Kontroler aplikacione logike

GenerickiKontrolerServer (kontroler aplikacione logike) prihvata zahtev za izvršenje sistemske operacije od GUIKontrolera(kontrolera sa klijentske strane) i dalje ga preusmerava do klasa koje su odgovorne za izvršenje sistemskih operacija. Nakon izvršenja sistemske operacije kontroler aplikacione logike(GenerickiKontrolerServer) prihvata rezultat i prosleđuje ga pozivaocu (GUIKontroler).



Slika 38-Arhitetktura softverskog sistema nakon projektovanja aplikacione logike

Za svaku sistemsku operaciju prave se softverske klase koje treba da realizuju sistemsku operaciju.



Slika 39-Klase koje su odgovorne za SO nasleđuju OpstuSO klasu

```
package SO;
```

```
import BrokerBazePodataka.BrokerBazePodataka;
import BrokerBazePodataka.BrokerBazePodatakImpl;
import DomenskiObjekat.GeneralIDObject;
```

```
public abstract class OpstelzvršenjeSO {
```

```
    public BrokerBazePodataka bbp = new BrokerBazePodatakImpl();
```

```
    int recordsNumber;
```

```
    int currentRecord = -1;
```

```
    GeneralIDObject gdo;
```

```
    public boolean opstelzvršenjeSO() {
```

```
        bbp.makeConnection();
```

```
        boolean signal = izvrsiSO();
```

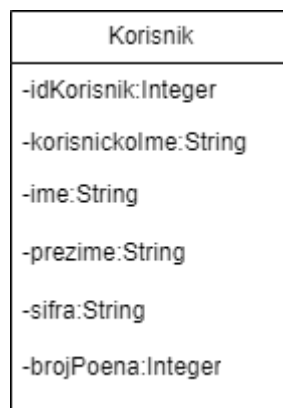
```
        if (signal == true) {
```

```

        bbp.commitTransation();
    } else {
        bbp.rollbackTransation();
    }
    bbp.closeConnection();
    return signal;
}
abstract public boolean izvrsiSO();
}

```

3.6 Projektovanje strukture softverskog sistema



Slika 40-Softverska klasa

Klasa Korisnik ima privatna polja atributa, gettere i settere za te attribute, besparametarski konstruktor. Kao primer dajem klasu “Korisnik”:

```

package DomenskiObjekat;
import java.io.Serializable;
import java.sql.ResultSet;
import java.sql.SQLException;
public class Korisnik extends GeneralIDObject implements Serializable {

```

```
public Long idKorisnik;  
String korisnickolme;  
String sifra;  
String ime;  
String prezime;  
int brojPoena;
```

```
public Korisnik() {  
    this.idKorisnik = 0L;  
    this.korisnickolme = "";  
    this.sifra = "";  
    this.ime = "";  
    this.prezime = "";  
    brojPoena=0;  
}
```

```
public Korisnik(Long idKorisnik, String korisnickolme, String sifra, String ime, String  
prezime,int brojPoena) {  
    this.idKorisnik = idKorisnik;  
    this.korisnickolme = korisnickolme;  
    this.sifra = sifra;  
    this.ime = ime;  
    this.prezime = prezime;  
    this.brojPoena=brojPoena;  
  
}
```

```
// alternativni primarni ključ
public Korisnik(String korisnickolme) {
    this.korisnickolme = korisnickolme;
}
```

```
// primarni ključ
public Korisnik(Long idKorisnik) {
    this.idKorisnik = idKorisnik;
}
```

```
public Long vratiIDKorisnik() {
    return this.idKorisnik;
}
```

```
public String getKorisnickolme() {
    return this.korisnickolme;
}
```

```
public String getSifra() {
    return this.sifra;
}
```

```
public String getIme() {
    return this.ime;
}
```

```
public String getPrezime() {
    return this.prezime;
}
```



```

    }

    public void setIDKorisnika(Long idKorisnik) {
        this.idKorisnik = idKorisnik;
    }

    public void setKorisnickolme(String korisnickolme) {
        this.korisnickolme = korisnickolme;
    }

    public void setSifra(String sifra) {
        this.sifra = sifra;
    }

    public void setIme(String ime) {
        this.ime = ime;
    }

    public void setPrezime(String prezime) {
        this.prezime = prezime;
    }

    @Override
    public String getAtrValue() {
        return idKorisnik + ", " + korisnickolme + ", " + sifra + ", " + ime + ", " + prezime +
        ", " + brojPoena;
    }

    @Override
    public String getAtrValue2() {

```

```

        return ""+ korisnickolme + ", " + sifra + ", " + ime + ", " + prezime + ", "
+brojPoena;
    }

```

@Override

```

public String setAtrValue() {
    return "idKorisnik=" + idKorisnik + ", " + "korisnickolme=" + korisnickolme + ", " +
"sifra=" + sifra + ", ime=" + ime + ", prezime=" + prezime + ", "
+"brojPoena="+brojPoena;
}

```

@Override

```

public String getClassName() {
    return "Korisnik";
}

```

@Override

```

public String getWhereCondition() {
    return "idKorisnik=" + idKorisnik;
}

```

@Override

```

public String getNameByColumn(int column) {
    String names[] = {"idKorisnik", "korisnickolme", "sifra", "ime",
"prezime","brojPoena"};
    return names[column];
}

```

@Override

```

    public DomenskiObjekat.GeneralIDObject getNewRecord(ResultSet rs) throws
SQLException {

        return new DomenskiObjekat.Korisnik(rs.getLong("idKorisnik"),
rs.getString("korisnickolme"), rs.getString("sifra"), rs.getString("ime"),
rs.getString("prezime"), rs.getInt("brojPoena"));

    }

    @Override

    public String columnsForInsert() {

        return " (korisnickolme, sifra, ime, prezime, brojPoena) ";

    }

    @Override

    public String toString() {

        return "Korisnik{" + "idKorisnik=" + idKorisnik + ", korisnickolme=" + korisnickolme
+ ", sifra=" + sifra + ", ime=" + ime + ", prezime=" + prezime + ", brojPoena=" +
brojPoena + '}';

    }

    public Long vratildKorisnik() {

        return idKorisnik;

    }

    public void setldKorisnik(Long idKorisnik) {

        this.idKorisnik = idKorisnik;

    }

    public int getBrojPoena() {

        return brojPoena;

    }

    public void setBrojPoena(int brojPoena) {

        this.brojPoena = brojPoena;

    }

}

```

3.7 Projektovanje sistemskih operacija

Za svaku sistemsku operaciju treba napraviti konceptualna rešenja koja su direktno povezana sa logikom problema. Za svaki od ugovora projektuje se konceptualno rešenje.

Ugovor UG1: RegistrujKorisnika

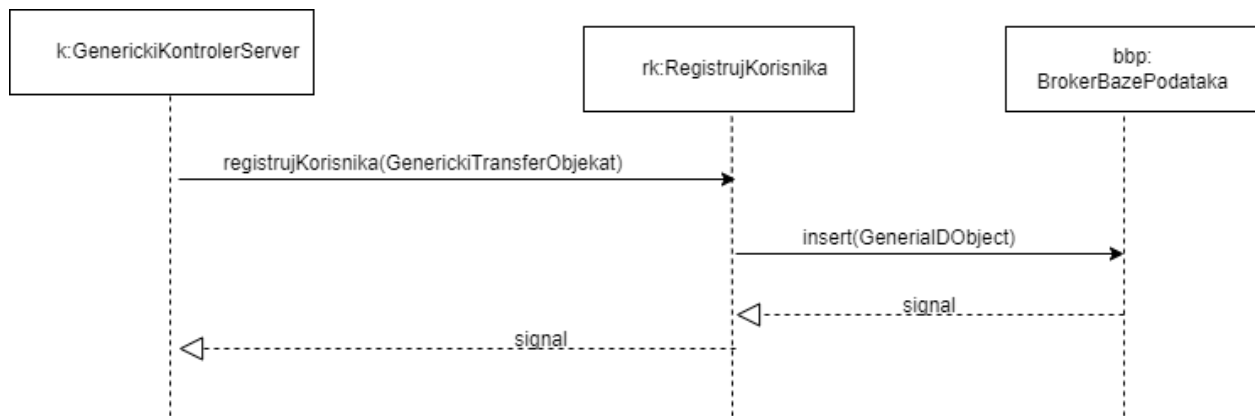
Operacija: registrujKorisnika(*Korisnik*):signal;

Veza sa SK: SK1

Preduslovi: Vrednosna ograničenja nad objektom Korisnik moraju biti zadovoljena.

Strukturno ograničenje nad objektom Korisnik mora biti zadovoljeno.

Postuslovi: Podaci o korisniku su zapamćeni.



Slika 41-Dijagram sekvenci za ugovor RegistrujKorisnika

Ugovor UG2: AzurirajKorisnika

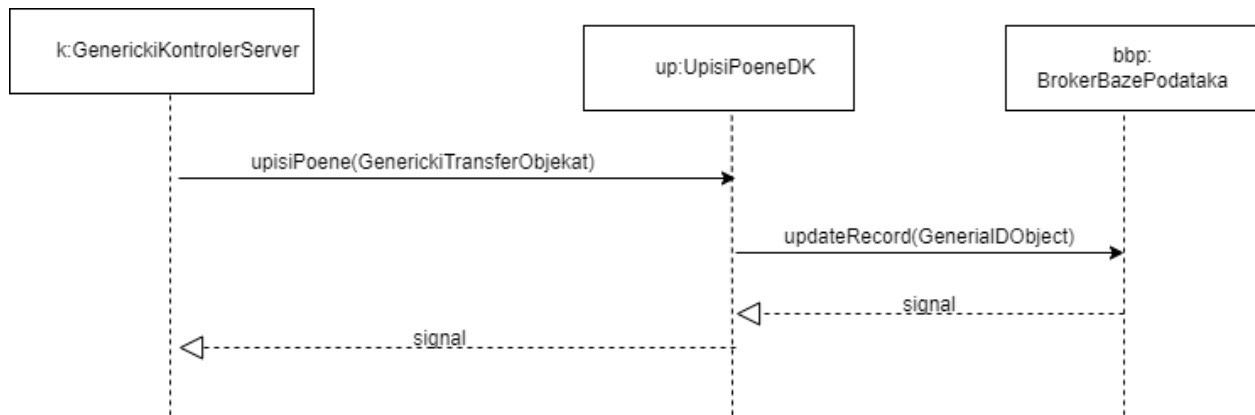
Operacija: azurirajKorisnika(*Korisnik*):signal;

Veza sa SK: SK2

Preduslovi: Vrednosna ograničenja nad objektom Korisnik moraju biti zadovoljena.

Strukturno ograničenje nad objektom Korisnik mora biti zadovoljeno.

Postuslovi: Podaci o korisniku su zapamćeni.



Slika 42-Dijagram sekvenci za ugovor ažuriraj korisnika

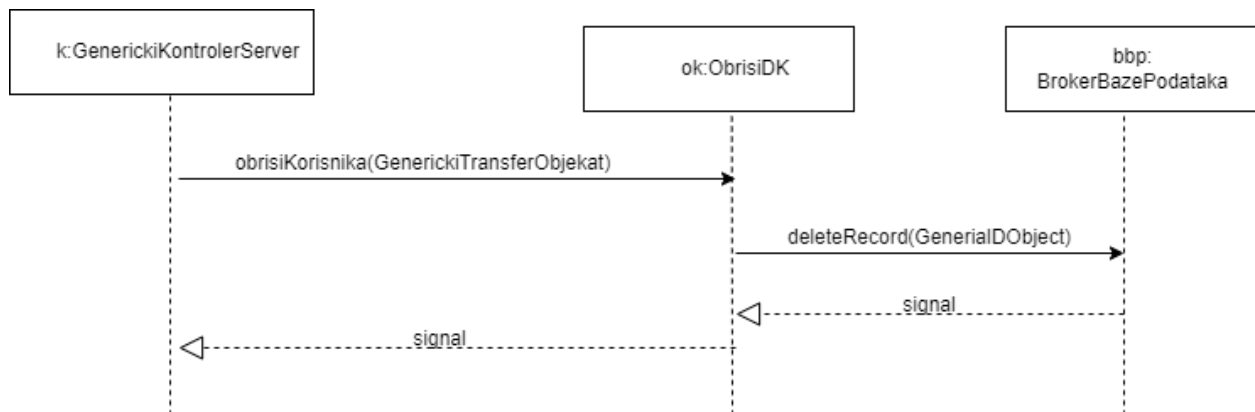
Ugovor UG3: ObrisiKorisnika

Operacija: obrisiKorisnika(*Korisnik*):signal;

Veza sa SK: SK3

Preduslovi: -

Postuslovi: Korisnik je obrisan. Strukturno ograničenje nad objektom Korisnik mora biti zadovoljeno.



Slika 43-Dijagram sekvenci za ugovor obriši korisnika

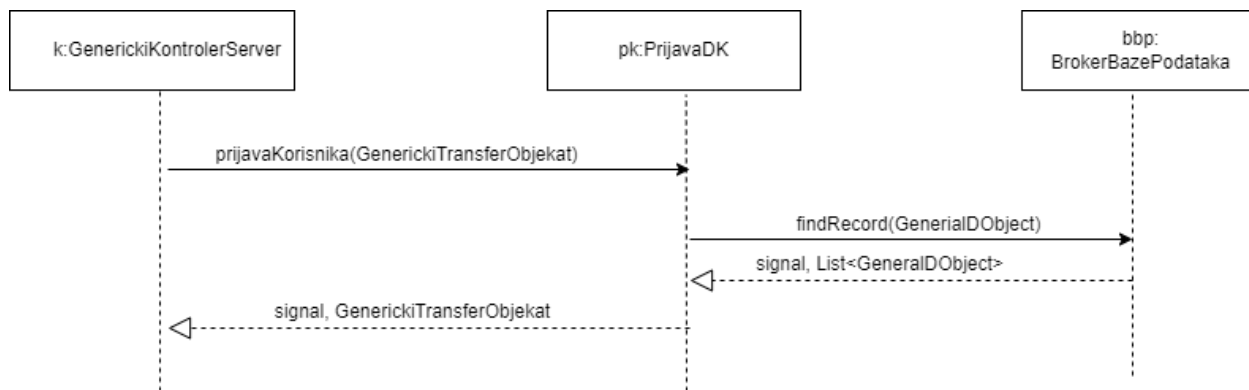
Ugovor UG4: PrijavaKorisnika

Operacija: prijavaKorisnika(*Korisnik*):signal;

Veza sa SK: SK4

Preduslovi: -

Postuslovi: -



Slika 44-Dijagram sekvenci za ugovor prijavi korisnika

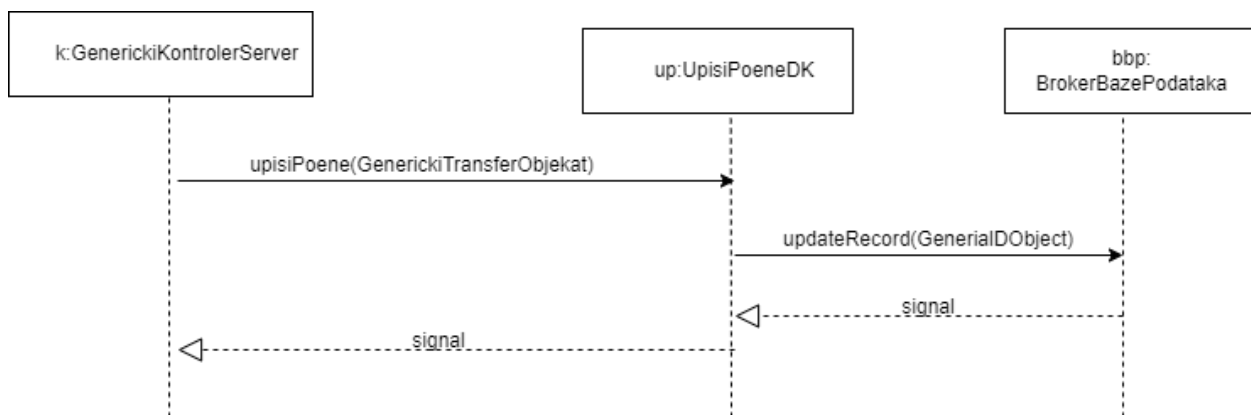
Ugovor UG5: Ažuriranje poena

Operacija: **upisiPoene(Korisnik):signal;**

Veza sa SK: SK5

Preduslovi: -

Postuslovi: -



Slika 45-Dijagram sekvenci za ugovor ažuriranje poena

3.8 Broker baze podataka

Klasa `BrokerBazePodataka` se projektuje kako bi se obezbedio perzistentni servis objektima domenskih klasa koji se čuvaju u bazi podataka. Tako klasa `BrokerBazePodatakaImpl` predstavlja perzistentni okvir koji posreduje u svim operacijama nad bazom podataka i realizuje sledeće metode:

1. `public boolean makeConnection()`
2. `public boolean insertRecord(GeneralDBObject odo)`
3. `public PreparedStatement insert(GeneralDBObject gdo) throws SQLException`
4. `public boolean deleteRecord(GeneralDBObject odo)`
5. `public boolean deleteRecords(GeneralDBObject odo, String where)`
6. `public boolean updateRecord(GeneralDBObject odo, GeneralDBObject odoold)`
7. `public boolean updateRecord(GeneralDBObject odo)`
8. `public boolean executeUpdate(String upit)`
9. `public GeneralDBObject findRecord(GeneralDBObject odo)`
10. `public boolean commitTransation()`
11. `public boolean rollbackTransation()`
12. `public void closeConnection()`
13. `public void close(Connection conn, Statement st, ResultSet rs)`
14. `public int getRecordsNumber(GeneralDBObject odo)`
15. `public GeneralDBObject getRecord(GeneralDBObject odo, int index)`

Sve metode "BrokerBazePodataka" klase su projektovane kao generičke, što znači da mogu da prihvate različite domenske objekte preko parametara kako ne bi u `BrokerBazePodataka` -u implementirali pojedinačne metode za svaku domensku klasu i bespotrebno umnožavali kod. Ovo je ostvareno definisanjem interfejsa "GeneralDBObject" koga implementiraju (i sve njegove metode) sve domenske klase:

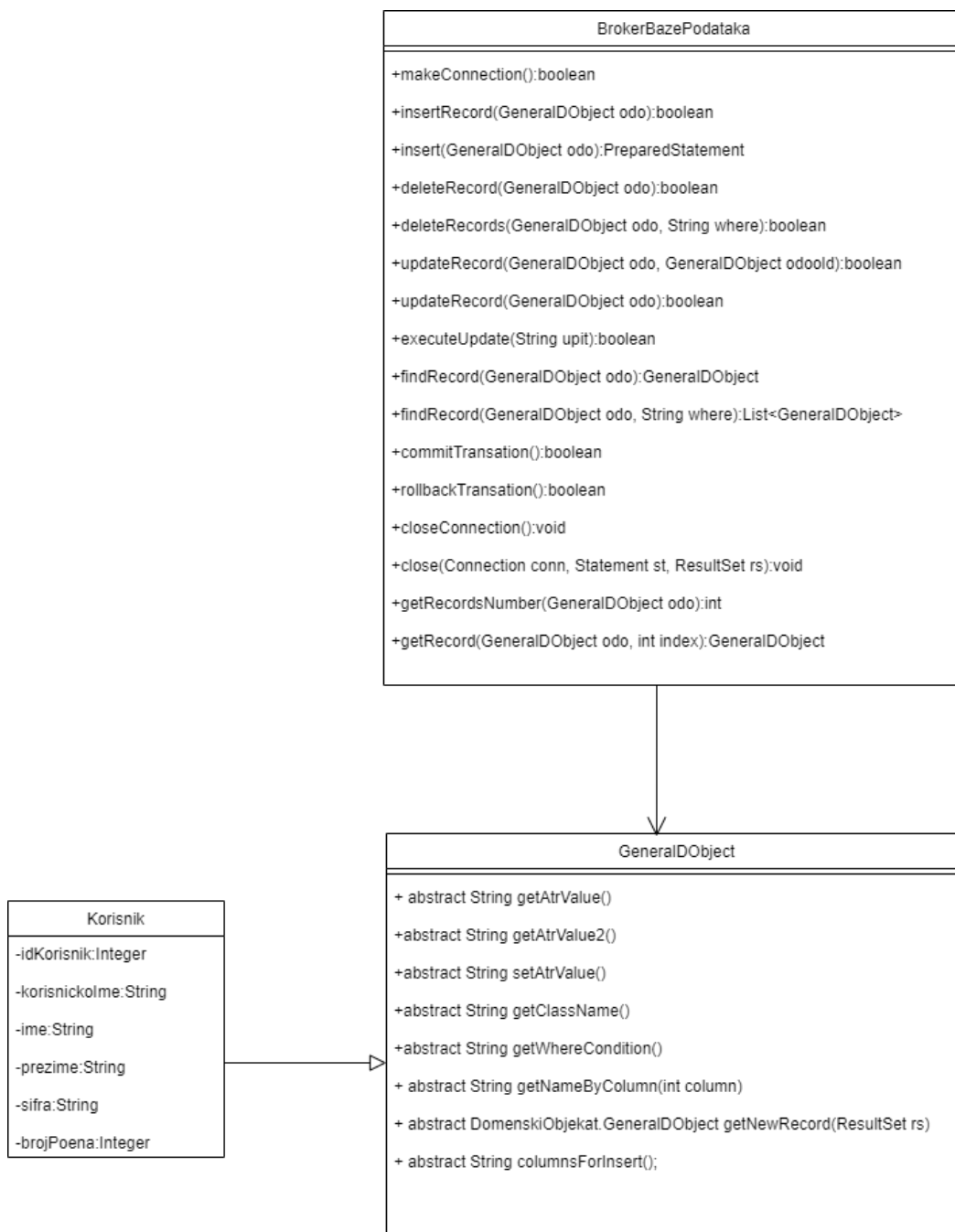
```
public abstract class GeneralDBObject implements Serializable {  
  
    abstract public String getAtrValue();  
  
    abstract public String getAtrValue2();  
  
    abstract public String setAtrValue();  
  
    abstract public String getClassName();  
  
    abstract public String getWhereCondition();  
  
    abstract public String getNameByColumn(int column);  
}
```

```

    abstract public DomenskiObjekat.GeneralDOObject getNewRecord(ResultSet rs)
    throws SQLException;

    public abstract String columnsForInsert();
}

```



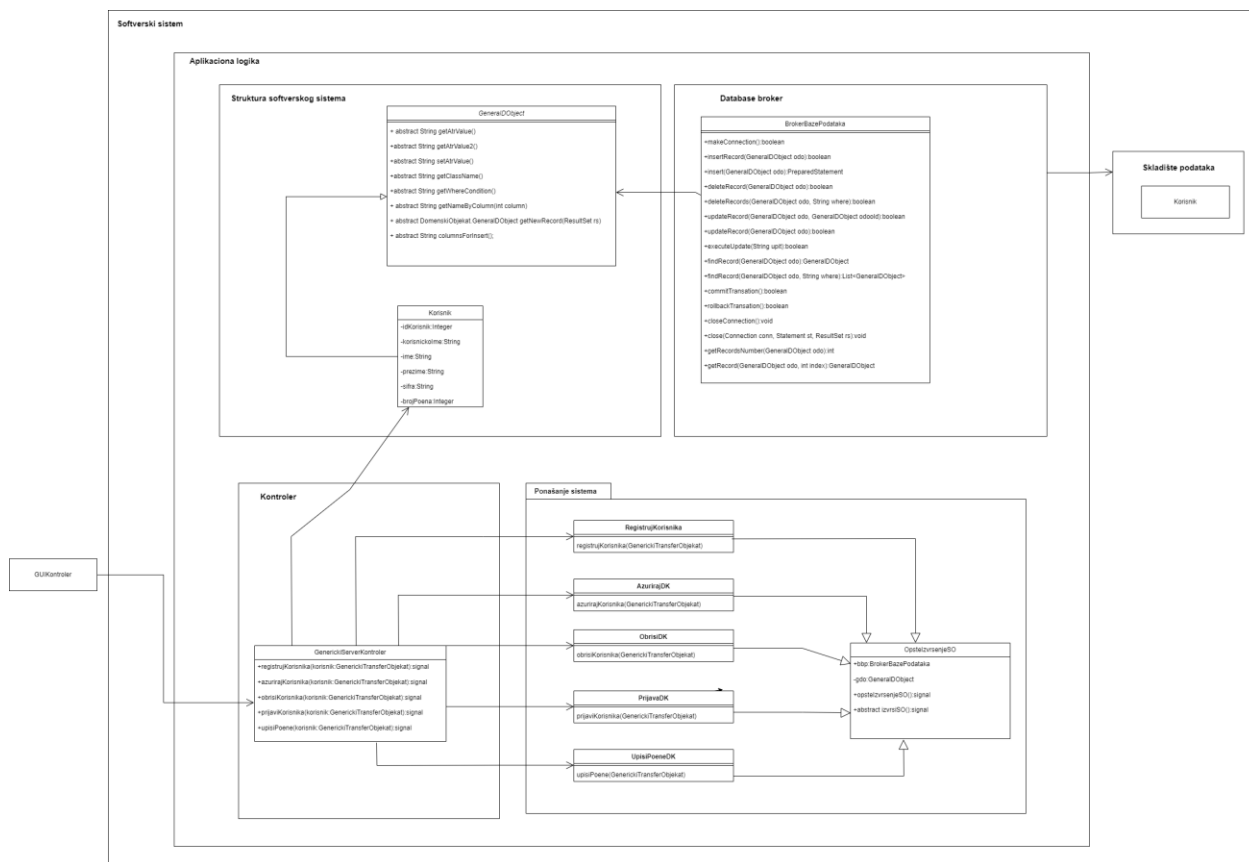
Slika 46-BrokerBazePodataka klasa se povezuje sa klasom GeneralDOObject

3.9 Projektovanje skladišta podataka

Na osnovu softverskih klasa strukture projektovane su tabele (skladišta podataka) relacionog sistema za upravljanje bazom podataka. U ovom radu korišćen je "MySQL":

Tabela: Korisnik		
Ime polja	Tip	Veličina
idKorisnik	int	11
korisnickolme	varchar	30
ime	varchar	30
prezime	varchar	30
sifra	varchar	30
brojPoena	int	11

Nakon projektovanja tabela iz baze podataka dobijamo potpunu sliku arhitekture softverskog sistema koji razvijamo.



Slika 47-ArHITEKTURA softverskog sistema

3.10 Principi, metode i strategije projektovanja softvera

Razvoj igre Sudoku opisan je uprošćenom Larmanovom metodom kroz nekoliko faza: prikupljanje zahteva, analiza, projektovanje, implementacija i testiranje.

3.10.1 Principi projektovanja softvera

Apstrakcija

Apstrakcija je svesno izdvajanje opštih informacija i zanemarivanje detalja, kako bi se naglasila suština. Sama Larmanova metoda se zasniva na principu apstrkcije, od početne faze prikupljana zahteva u kojoj je najizraženija apstrakcija i najviše se zanemaruju detalji softvera, do faze projektovanja gde su detalji mnogo izraženiji.

U kontekstu softverskog projektovanja postoje dva ključna mehanizma apstrakcije:

- Parametrizacija
- Specifikacija

Apstrakcija specifikacijom vodi do 3 glavne vrste apstrakcija:

- Proceduralna apstrakcija

- Apstrakcija podataka
- Apstrakcija kontrolom

Parametrizacija

Parametrizacija je apstrakcija koja **izdvaja** iz nekog skupa elemenata njihova opšta svojstva koja su predstavljena preko parametara.

Razlikujemo pet slučajeva parametrizacije:

- Parametrizacija skupa elemenata prostog tipa
- Parametrizacija skupa elemenata složenog tipa
- Parametrizacija skupa operacija
- Parametrizacija skupa procedura
- Parametrizacija skupa naredbi

Parametrizacija skupa elemenata prostog tipa

Ako posmatramo skup celih brojeva 1, 2, 3... oni se mogu predstaviti npr. preko parametra x koji je u tom slučaju opšti predstavnik svih celih brojeva. Programerski bismo to zapisali kao: **int x;**

Parametrizacija skupa elemenata složenog tipa

Ukoliko npr. Imamo skup igrača (1, Danica, Zdravkovic, dana, 22), (2, Nevena, Nenic, nena, 23), tada se parametrizacijom dobijaju njihova opšta svojstva: korisnikID, ime, prezime, korisnickolme, brojPoena.

Parametrizacija se radi za svaki skup vrednosti atributa. Skup imena igrača (Danica, Nevena) se parametrizuje preko svojstva ime, skup prezimena igrača (Zdravkovic, Nenic) se parametrizuje preko svojstva prezime, skup ID igrača (1, Nenic) se parametrizuje preko svojstva korisnikID, skup korisničkih imena igrača (dana, nena) se parametrizuje preko svojstva korisnickolme, skup poena igrača (22, 23) se parametrizuje preko svojstva brojPoena igrača.

Parametrizacijom dolazimo do opštih svojstava elemenata skupa. Navođenje opštih svojstava elemenata skupa predstavlja specifikaciju skupa.

Apstrakcija podataka je definisana imenom i specifikacijom skupa: Korisnik (idKorisnika, ime, prezime, korisnickolme, brojPoena). Parametrizacija prethodi specifikaciji, jer specifikacija koristi rezultate parametrizacije.

Parametrizacija skupa operacija

Ukoliko imamo skup nekih operacija ((1+2), (2+3), (3+4)) tada se parametrizacijom dobija opšta operacija $x+y$ (x, y su operandi, dok $+$ ukazuje na to šta operacija radi).

Kao rezultat parametrizacije dobija se x+y, opšta operacija za sabiranje dva broja.

Parametrizacija skupa procedura

Ukoliko posmatramo skup procedura (pr1, pr2, pr3):

1. List vratiListuObjekata (ResultSet rs){metoda koja vraća listu Korisnika iz resultset-a}
2. List vratiListuObjekata (ResultSet rs){metoda koja vraća listu Igra iz resultset-a}

Tada se parametrizacijom dobija opšta procedura: List vratiListuObjekata(ResultSet rs). Elementi neke procedure su: povratna vrednost, naziv procedure, argumenti procedure i telo procedure.

Parametrizacijom dolazimo do opštih svojstava procedura, njihovim navođenjem dolazimo do potpisa procedura. Potpis procedure je jedan od glavnih delova specifikacije procedure (proceduralna apstrakcija).

Parametrizacija skupa naredbi

Ukoliko posmatramo skup naredbi:

```
System.out.println("Igrač:"+igrači[0].getIme()+"sa prezimenom:"+ igrači[0].getPrezime())
```

```
System.out.println("Igrač:"+igrači[1].getIme()+"sa prezimenom:"+ igrači[1].getPrezime())
```

```
System.out.println("Igrač:"+igrači[2].getIme()+"sa prezimenom:"+ igrači[2].getPrezime())
```

Specifikacija

Koje prikazuju elemente liste igrači, tada se parametrizacijom dobija opšta naredba:

```
System.out.println ("Igrač:"+igrači[i].getIme()+"sa prezimenom" + igrači[i].getPrezime())
```

Parametrizacijom dolazimo do opštih svojstava naredbi. Navođenjem opštih svojstava naredbi dolazi se do specifikacije naredbe koja postaje opšta naredba. Navedena specifikacija je u stvari apstrakcija naredbi, jedan od oblika apstrakcije kontrolom.

Specifikacija

Specifikacija je apstrakcija koja izdvaja iz nekog skupa elemenata njihova opšta svojstva koja mogu biti predstavljena preko procedure, podatka ili kontrole.

Proceduralna apstrakcija

Proceduralnom apstrakcijom se izdvaja iz nekog skupa procedura ono što su njihova opšta svojstva:

- Tip onoga što vraća procedura
- Ime procedure
- Argumenti procedure

Proceduralnom apstrakcijom se dobija potpis procedure. Pored potpisa, proceduralna apstrakcija obuhvata i dopunske sekcije koje detaljno opisuju šta radi procedura i koji su uslovi izvršenja procedure. Opšti slučaj proceduralne apstrakcije prema uprošćenoj Larmanovoj metodi, ima sledeće elemente: Potpis procedure, Veza sa SK, Preduslovi i postuslovi.

Kao primer opšteg slučaja proceduralne apstrakcije možemo navesti ugovor o sistenskoj operaciji: registrujKorisnika

Potpis procedure: registrujKorisnika(*Korisnik*):signal;

Veza sa SK: SK1

Preduslovi: Vrednosna ograničenja nad objektom Korisnik moraju biti zadovoljena.

Strukturno ograničenje nad objektom Korisnik mora biti zadovoljeno.

Postuslovi: Podaci o korisniku su zapamćeni.

Apstrakcija podataka

Apstrakcijom podataka se izdvaja iz nekog skupa podataka ono što su njihova opšta svojstva. Na prethodnom primeru pokazano je da ukoliko posmatramo skup korisnika (i1, i2) iz ovog studijskog primera: (1, Danica, Zdravkovic, dana, 22), (2, Nevena, Nenic, nena, 23), tada se parametrizacijom dobijaju njihova opšta svojstva: korisnikID, ime, prezime, korisnickolme, brojPoena. Navođenje opštih svojstava elemenata skupa (korisnikID, ime, prezime, korisnickolme, brojPoena) predstavlja specifikaciju skupa. Apstrakcija podataka je definisana imenom (Korisnik) skupa i specifikacijom skupa. Ukoliko elementi, kao u ovom slučaju, imaju i strukturu i ponašanje tada se nad njima radi i proceduralna apstrakcija i apstrakcija podataka. Kao rezultat navedenih apstrakcija dobija se klasa, koja se sastoji od atributa i procedura (metoda), koji su učaureni u klasi.

Kod uprošćene Larmanove metode, struktura je apstrakcija podataka, dok su sistemske operacije proceduralna apstrakcija.

Apstrakcija kontrolom

Postoje dva oblika apstrakcije kontrole:

- Apstrakcija **naredbi**, kojom se izdvaja iz nekog skupa naredbi ono što su njihova opšta svojstva i predstavljaju se preko kontrolne strukture i opšte naredbe.
- Apstrakcija **struktura podataka**, kojoj se izdvaja iz nekog skupa struktura podataka ono što su njihova opšta svojstva i predstavljaju se preko iteratora, koji u opštem smislu kontroliše kretanje kroz strukturu podataka.

Spojenost – Kohezija

U razvoju obejktno-orijentisanog softvera potrebno je postići dva cilja:

- Treba izgraditi klase koje imaju visoku koheziju (high cohesion)
- Klase treba da budu slabo povezane (weak coupling)

Spojenost se odnosi na veze između klasa, dok se kohezija odnosi na povezanost metoda unutar klase.

Postizanje visoke kohezije unutar klase prikazano je u klasi OpstelzvršenjeSO.

```
public abstract class OpstelzvršenjeSO {
    public BrokerBazePodataka bbp = new BrokerBazePodatakaImpl();
    public boolean opstelzvršenjeSO() {
        bbp.makeConnection();
        boolean signal = izvrsiSO();
        if (signal == true) {
            bbp.commitTransation();
        } else {
            bbp.rollbackTransation();
        }
        bbp.closeConnection();
        return signal;
    }

    abstract public boolean izvrsiSO();
}
```

Sve metode unutar ove klase povezane su ka glavnoj metodi opstelzvršenjeSO, tako da doprinose zajedničkom cilju-komunikacija sa bazom, a da ne menjaju konzistentnost njenog stanja.

Spojenost

Spojenost (kuplovanje) znači da su klase u međusobnoj zavisnosti, odnosno da klasa ne može da obavi neku operaciju ako ne postoji neka druga klasa.

Klase koje nasleđuju klasu OpstelzvršenjeSO odgovaraju principu slabe povezanosti među klasama, što je jedan od dobrih principa projektovanja softvera. Klasa PrijavaDK osim što nasleđuje OpstelzvršenjeSO, te implementira njenu apstaktnu metodu

izvrsiSO, ni na koji način nije zavisna niti povezana sa ovom klasom, njene metode se odvijaju nezavisno od klase OpstelzvršenjeSO.

```
public class PrijavaDK extends OpstelzvršenjeSO {
    GenerickiTransferObjekat gto;

    public void prijavaKorisnika(GenerickiTransferObjekat gto) {
        this.gto = gto;
        opstelzvršenjeSO();
    }

    @Override
    public boolean izvrsiSO() {

        Korisnik neuloKorisnik = (Korisnik) gto.getDK();
        String whereUslov = " WHERE korisnickolme like '" +
        neuloKorisnik.getKorisnickolme() + "' AND sifra like '" + neuloKorisnik.getSifra() + "' ";
        List<GeneralDObject> korisnici = bbp.findRecord(gto.getDK(), whereUslov);
        System.out.println("Lista korisnici"+korisnici);
        if (korisnici.isEmpty()) {
            gto.setSignal(false);
            return false;

        }
        gto.setDK(korisnici.get(0)); //prvi pronadjen
        gto.setSignal(true);
        gto.setPoruka("Korisnik je pronadjen.");

        return gto.getSignal();
    }
}
```

}

Dekompozicija i modularizacija

Dekompozicija (rašćlanjivanje), u najopštijem smislu, je proces koji početni problem deli u skup podproblema, koji se nezavisno rešavaju i time omogućavaju lakše rešavanje početnog problema. Ukoliko dekompoziciju posmatramo u kontekstu razvoja softverskog sistema može se reći da se dekompozicijom softverski sistem deli u više modula, odnosno da je modularizacija rezultat procesa dekompozicije.

Dekompozicija se javlja u nekoliko različitih aspekata razvoja softvera:

- *Dekompozicija kod prikupljanja zahteva*

Korisnički zahtev se dekomponuje na skup zahteva koji se opisuju preko slučajeva korišćenja (Unos novog igrača, Izmena podataka o igraču, Brisanje igrača, Prijava igrača, Ažuriranje poena)

- *Dekompozicija kod projektovanja softvera*

U fazi projektovanja pravi se arhitektura softverskog sistema koja se obično dekomponuje na tri modula (komponente): korisnički interfejs, aplikaciona logika i skladište podataka. Nakon toga se ti moduli dalje dekomponuju a svaki od modula koji je dobijen dekompozicijom može se nezavisno projektovati i implementirati u odnosu na druge module. Pri dekomponovanju softverskog sistema u module neophodno je zadovoljiti sledeće principe: jaka kohezija, slaba povezanost, čuvanje internih informacija unutar modula.

- *Dekompozicija funkcija (metoda)*

Funkcija (metoda) neke klase ukoliko je složena može se dekomponovati u više podfunkcija, koje se zatim nezavisno rešavaju da bi se na kraju integrisale u jednu celinu i na taj način realizovala početna funkcija. Primer je metoda `OpstelzvršenjeSO()` koja unutar sebe poziva metode `makeConnection()`, `izvrsiSO()`, `closeConnection()`, `commitTransaction()`, `rollbackTransaction()`. Na ovaj način izbegavamo kompleksnost koda na jednom mestu.

Učaurjenje (encapsulation) / Sakrivanje informacija (Information hiding)

Učaurjenje je proces u kome se razdvajaju osobine modula (klase) koje su javne za druge module od osobina koje su sakrivene od drugih modula sistema. Sakrivanje informacija nastaje kao rezultat procesa učaurjenja. Na primer, atribut `nivou` u klasi `FXMLDocumentController` je privatni atribut, jer nije od značaja za ostale klase i učaurjen je, dok su atributi koji predstavljaju polja sa grafičke komponente javne jer im se pristupa iz kontrolera `GUIKontrolerMeni`.


```

public class FXMLDocumentController {
    String nivo = "";
    @FXML
    public TextField p00;
    @FXML
    public TextField p01;
    @FXML
    public TextField p02;
    .....
}

```

Odvajanje interfejsa i implementacije (Separation of interface and implementation)

Interfejs se odvađa od implementacije i izlaže se korisniku, koji ne zna kako su operacije interfejsa implementirane. U navedenom primeru korisnik je BrokerBazePodataka, dok je GeneralIDObject interfejs koji izlaže operacije korisniku. BrokerBazePodataka ne zna kako su operacije domenskih klasa (getAtrValue(),getWhereCondition()...) implementirane. Sve domenske klase realizuju interfejs OpstiDomenskiObjekat.

@Override

```

    public String getAtrValue() {
        return idKorisnik + ", " + korisnickolme + ", " + sifra + ", " + ime + ", " + prezime +
        ", " + brojPoena;
    }

```

@Override

```

    public String getWhereCondition() {
        return "idKorisnik=" + idKorisnik;
    }

```

3.10.2 Strategije projektovanja softvera

Neke od poznatih strategija projektovanja softvera su:

- Pristup podeli i vladaj
- Pristup s vrha na dole

- Pristup odozgo na gore
- Iterativno – inkrementalni pristup

Podeli i vladaj

Podeli i vladaj strategija zasnovana je na principu dekompozicije koja početni problem deli u skup podproblema koji se nezavisno rešavaju, u cilju lakšeg rešavanja početnog problema. Ova strategija se najčešće primenjuje u sledećim fazama razvoja softvera:

- Prikupljanje zahteva – Korisnički zahtevi se opisuju preko skupa nezavisnih slučajeva korišćenja
- Analiza – Struktura se opisuje preko skupa koncepata koji čine konceptualni model. Ponašanje se opisuje preko skupa nezavisnih sistemskih operacija.
- Projektovanje – Arhitektura sistema se deli u tri nivoa: korisnički interfejs, aplikaciona logika i skladište podataka. Korisnički interfejs se deli na dva dela: ekranske forme i kontroler KI. Aplikaciona logika se deli u tri dela: kontroler aplikacione logike, poslovna logika i broker baze podataka.

Pristup s vrha na dole (top down) i odozgo na gore (bottom-up)

Strategija s vrha na dole je zasnovana na principu dekompozicije funkcija koja početnu funkciju dekomponuje u više podfunkcija koje se rešavaju nezavisno, kako bi se njihovim integrisanjem u celinu na kraju realizovala početna funkcija.

Strategija odozgo na gore je zasnovana na principu generalizacije, koji u nekoj funkciji koja je složena uočava jednu ili više logičkih celina koje proglašava za podfunkcije. Na taj način je jedna složena funkcija dekomponovana u više nezavisnih funkcija, a kompozicija tih funkcija obezbeđuje istu funkcionalnost kao i složena funkcija. Funkciju `opstelzvršenjeSO()` smo podelili na više logičkih podfunkcija `makeConnection()`, `commitTransation()`, `rollbackTransation()`, `closeConnection()`. Svaka podfunkcija je zasebno implementirana.

```
public boolean opstelzvršenjeSO() {
    bbp.makeConnection();
    boolean signal = izvršiSO();
    if (signal == true) {
        bbp.commitTransation();
    } else {
        bbp.rollbackTransation();
    }
    bbp.closeConnection();
}
```

```

        return signal;
    }

@Override
    public boolean makeConnection() {
        String Url;
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Url = "jdbc:mysql://127.0.0.1:3306/sudoku_igra";
            conn = DriverManager.getConnection(Url, "root", "");
            conn.setAutoCommit(false);
        } catch (SQLException | ClassNotFoundException ex) {
            Logger.getLogger(BrokerBazePodatakaImpl.class.getName()).log(Level.SEVERE, null,
ex);
            return false;
        }
        return true;
    }

@Override
    public boolean commitTransation() {
        try {
            conn.commit();
        } catch (SQLException esql) {
            return false;
        }
        return true;
    }

@Override
    public boolean rollbackTransation() {

```

```

try {
    conn.rollback();
} catch (SQLException esql) {
    return false;
}

return true;
}

```

```

@Override
public void closeConnection() {
    close(conn, null, null);
}

```

Iterativno – inkrementalni pristup (iterative and incremental approach)

Sistem se deli u više mini projekata koji se nezavisno razvijaju i prolaze kroz sve faze razvoja softvera. Mini projekat može biti vezan za slučaj korišćenja ili za složenu sistemsku operaciju. Svaki mini projekat prolazi kroz više iteracija, čiji rezultat, interno izdanje (release) ili build predstavlja inkrement za sistem. Na kraju se implementirani mini projekti integrišu u softverski sistem. Primer: U primeru razvoja softverskog sistema imali smo nekoliko nezavisnih slučajeva korišćenja iz kojih smo dobili nekoliko nezavisnih sistemskih operacija. Razvoj pojedinačne sistemske operacije nije zavisio od razvoja bilo koje druge sistemske operacije, odnosno neka sistemska operacija može biti implementirana i integrisana u sistem dok druga sistemska operacija može tek biti identifikovana u fazi analize.

3.10.3 Metode projektovanja softvera

Neke od najvažnijih metoda projektovanja softvera su:

1. Funkcionalno orijentisano projektovanje
2. Objektno orijentisano projektovanje
3. Projektovanje zasnovano na strukturi podataka
4. Projektovanje zasnovano na komponentama

Za razvoj ovog softvera korišćena je objektno orijentisana metoda.

Funkcionalno orijentisano projektovanje je zasnovano na funkcijama. Problem se posmatra iz perspektive njegovog ponašanja odnosno funkcionalnosti. Prvo se uočavaju funkcije sistema, zatim se određuju strukture podataka nad kojima se izvršavaju te funkcije.

Objektno orijentisano projektovanje je zasnovano na objektima (klasama). Objekti mogu da reprezentuju i strukturu (atribute) i ponašanje (metode) softverskog sistema. Kod objektno orijentisanog projektovanja paralelno se razvijaju i struktura i ponašanje.

Projektovanje zasnovano na strukturi podataka problem posmatra iz perspektive strukture. Prvo se uočava struktura sistema, zatim se određuju funkcije koje se izvršavaju nad tom strukturom.

Projektovanje zasnovano na komponentama problem posmatra iz perspektive postojećih komponenti koje se mogu (ponovo) koristiti u rešavanju problema. Prvo se uočavaju delovi problema koji se mogu realizovati preko postojećih komponenti a nakon toga se implementiraju delovi za koje nije postojalo rešenje.

Objektno orijentisano projektovanje (Object – oriented design)

U ovom primeru kao rezultat analize se dobija logička struktura i ponašanje softverskog sistema. Struktura je opisana preko konceptualnog modela, dok je ponašanje opisano preko dijagrama sekvenci. Konceptualni model se sastoji od skupa međusobno povezanih koncepata (klasa, odnosno njihovih pojavljivanja objekata), dok sekvenci dijagram opisuje interakciju između objekata (aktora i softverskog sistema). To znači da su u osnovi logičke strukture i ponašanja softverskog sistema objekti. Zato se ovakva analiza zove objektno orijentisana analiza. Kasnije se u fazi projektovanja i implementacije određuju klase koje će da realizuju strukturu (domenske klase) i ponašanje (klase koje realizuju sistemske operacije). To znači da su u osnovi fizičke strukture i ponašanja softverskog sistema takođe objekti, odnosno klase. Zato se ovakvo projektovanje naziva objektno – orijentisano projektovanje.

Principi objektno orijentisanog projektovanja klasa (Principles of Object Oriented Class Design)

Postoje sledeći principi kod objektno – orijentisanog projektovanja klasa:

1. Princip Otvoreno – Zatvoreno
2. Princip zamene Barbare Liskov
3. Princip inverzije zavisnosti
4. Princip umetanja zavisnosti
5. Princip izdvajanja interfejsa

Princip Otvoreno – Zatvoreno (Open – Closed principle)

Princip Otvoreno – Zatvoreno: Modul treba da bude otvoren za proširenje ali i zatvoren za modifikaciju.

U ovom primeru klasa OpstelzvršenjeSO je zatvorena za promenu njenog ponašanja ali je u isto vreme i otvorena za promenu njenog ponašanja u klasama koje su iz nje izvedene kao npr. Klasa RegistrujKorisnika:

```
public abstract class OpstelzvršenjeSO {

    public BrokerBazePodataka bbp = new BrokerBazePodatakaImpl();

    int recordsNumber;

    int currentRecord = -1;

    GeneralIDObject gdo;

    public boolean opstelzvršenjeSO() {
        bbp.makeConnection();
        boolean signal = izvrsiSO();
        if (signal == true) {
            bbp.commitTransation();
        } else {
            bbp.rollbackTransation();
        }
        bbp.closeConnection();
        return signal;
    }

    abstract public boolean izvrsiSO();
}
```

```
public class RegistrujKorisnika extends OpstelzvršenjeSO {
```

GenerickiTransferObjekat gto;

```
public void registrujKorisnika(GenerickiTransferObjekat gto) {  
    this.gto = gto;  
    opstelzvršenjeSO();  
}
```

@Override

```
public boolean izvršiSO() {  
    Korisnik k = gto.gdo;  
    boolean s = false;  
  
    String where = " WHERE korisnickolme LIKE '" + k.getKorisnickolme() + "' ";  
    List<GeneralDObject> lista = bbp.findRecord(gto.gdo, where);  
  
    if (lista.isEmpty()) {  
        s = bbp.insertRecord(gto.gdo);  
        gto.setSignal(s);  
    }  
  
    return s;  
}  
}
```

Princip zamene Barbare Liskov (The Liskov Substitution Principle)

Princip zamene Barbare Liskov: Podklase treba da budu zamenljive. Klasa Korisnik je zamenljiva nadklasom GeneralDObject:

Princip inverzije zavisnosti (The dependency Inversion Principle)

Princip inverzije zavisnosti: Zavisi od apstrakcije a ne od konkretizacije.

Moduli najvišeg nivoa ne treba da zavise od modula nižeg nivoa. Apstrakcije ne treba da zavise od detalja, a detalji treba da zavise od apstrakcije. BrokerBazePodataka izbegava direktnu povezanost sa domenskim klasama, već se između njih umeće apstrakcija GeneralDObject.

Princip umetanja zavisnosti (The dependency injection principle)

Princip umetanja zavisnosti: Zavisnosti između dve komponente programa se uspostavljaju u vreme izvršenja programa preko neke treće komponente. U ovom slučaju, zavisnost BrokeraBazePodataka i domenskih klasa se u vreme izvršavanja programa ostvaruje preko klase GeneralDObject.

Princip odvajanja interfejsa

Princip odvajanja interfejsa: Bolje je da postoji više specifičnih interfejsa, nego jedan generalni interfejs opšte namene. Interfejs se odvaja od implementacije i izlaže se korisniku koji može da poziva njegove operacije, bez da zna kako su operacije implementirane. BrokerBazePodatakImpl je korisnik dok je GeneralDObject interfejs koji se izlaže korisniku. BrokerBazePodatakImpl ne zna kako su operacije klase GeneralDObject implementirane.

3.11 Primena paterna u projektovanju

Za projektovanje realnog softverskog sistema, kao i bilo kog drugog velikog sistema, veoma je bitno njegovo održavanje. U praksi, veliki deo softverskog koda može se više puta koristiti. Moguće je da deo koda jednog projekta upotrebimo za neki novi projekat kako bismo skratili vreme njegove realizacije. Svaki problem koji se javi tokom razvoja softvera ne treba rešavati od početka. Potrebno je koristiti rešenja koja su uspešno korišćena ranije. Iz tog razloga se u mnogim objektno orijentisanim sistemima nailazi na iste uzore klasa i objekata koji komuniciraju. Ti uzori rešavaju specifične probleme projektovanja i omogućavaju fleksibilnost i ponovnu upotrebljivost objektno orijentisanog dizajna. Oni projektantima pomažu da ponovo upotrebe uspešne projekte tako što će novi projekat zasnovati na prethodnom iskustvu. Ovi uzori nazivaju se još i paterni. Paterni imaju cilj da nam pomognu u održavanju i nadogradnji softverskog sistema.

Makro arhitektura

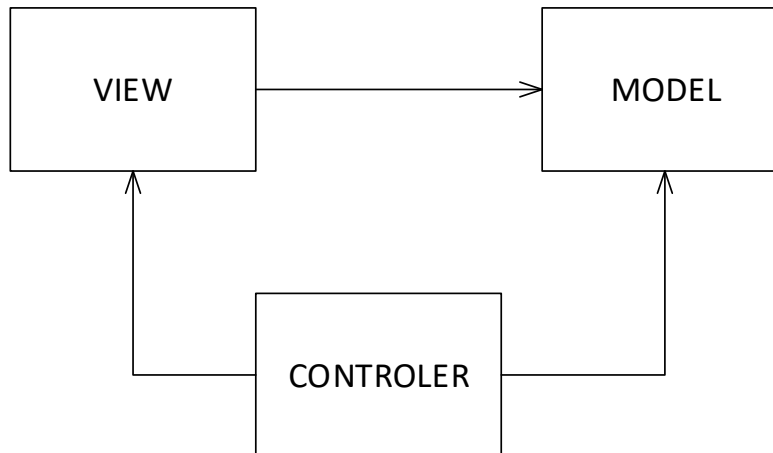
Makroarhitektura opisuje strukturu i organizaciju softverskog sistema na najvišem nivou. Makro paterni su EFC i MVC paterni.

MVC (Model-View-Controller) patern

Patern deli softverski sistem u 3 dela:

1. **View** - obezbeđuje korisniku interfejs pomoću koga će korisnik da unosi podatke i poziva odgovarajuće operacije koje treba da se izvrše nad modelom. View prikazuje korisniku stanje modela. To je na primer forma za prijavu korisnika.

2. **Controller** - osluškuje i prihvata zahtev od klijenta za izvršenje operacije. Nakon toga poziva operaciju koja je definisana u modelu. Ukoliko model promeni stanje, controller obaveštava view da je promenjeno stanje. U našem slučaju to je GUIControllerMeni.
3. **Model** – predstavlja stanje sistema. Stanje mogu menjati neke od operacija modela. Model ne mora da zna ko je pogled, a ko kontroler.



Slika 48-MVC patern

Opšte o paternima

Paterni tj. uzori su gotova rešenja koja se primenjuju u projektovanju softvera. „Svaki uzorak opisuje problem koji se stalno ponavlja u našem okruženju i zatim opisuje suštinu rešenja problema tako da se to rešenje može upotrebiti milion puta, a da se dva puta ne ponovi na isti način.“(Cristopher Alexander)

Svaki patern sastoji se od tri dela: **problema** koji se rešava, **konteksta** u smislu određenih ograničenja i **rešenja** problema. Kada je kontekst odgovarajući, ista pravila se mogu primeniti da bi se problem rešio u tom konkretnom slučaju. Paterni su razvijeni kako bi podržali izgradnju velikih, složenih sistema, da bi se dokumentovao svaki potencijalni problem i njegovo rešenje i da bi ovakvo znanje programeri u budućnosti koristili i primenjivali.

Paterni su definisani i kao opisi komunikacije objekata i klasa koji su prilagođene za rešavanje opštih problema u datom kontekstu. Imena paterna, apstrakcije i identifikacija ključnih aspekata zajedničke strukture čine ih korisnim za kreiranje objektno orijentisanog dizajna za ponovno korišćenje. Paterni identifikuju klase i njihove instance koje učestvuju u strukturi, njihove uloge i veze i raspodelu odgovornosti, odnosno daju apstraktan opis problema projektovanja i kako se on rešava opštim uređenjem elemenata (klasa i objekata). Svaki patern koncentrisan je na određeni problem. Opisuje kada se može primeniti, da li se ili ne može primeniti uzimajući u obzir ograničenja ili posledice njegove primene.

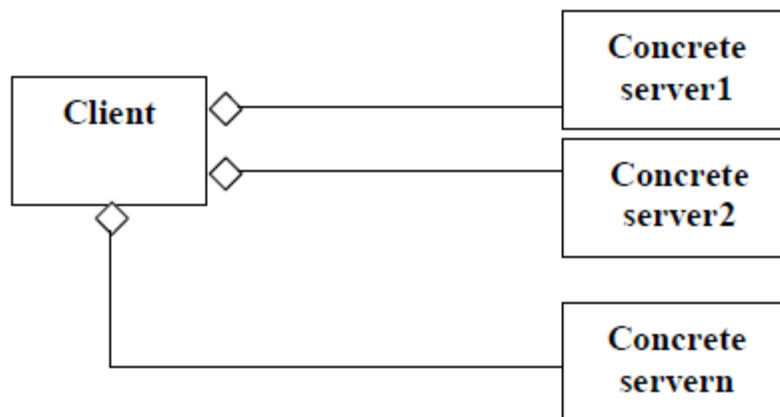
Postoji kategorizacija paterna u tri grupe: paterni za kreiranje objekata, strukturni paterni i paterni ponašanja.

Korišćenje paterna

U ovom radu se može prepoznati primena nekih paterna. U daljem tekstu je opisan svaki pojedinačno.

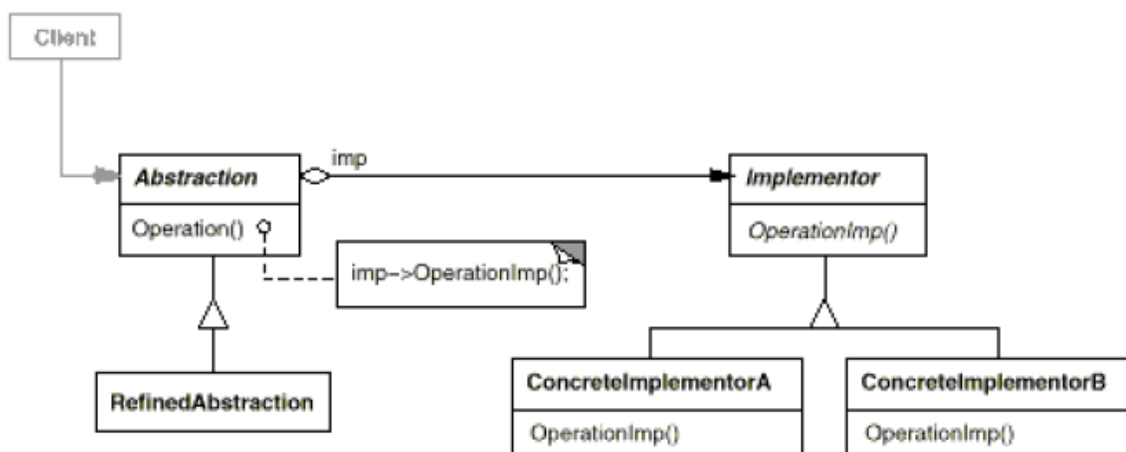
Bridge patern

Ovim paternom se podrazumeva da se za svakog klijenta kreira konkretan server, što predstavlja problem u održavanju fleksibilnosti programa.



Slika 49-Bridge patern ilustracija

Ovaj patern odvaja apstrakciju od njene implementacije tako da se one mogu menjati nezavisno. Struktura samog paterna je slična strukturi rada koji je opisan u ovoj dokumentaciji.

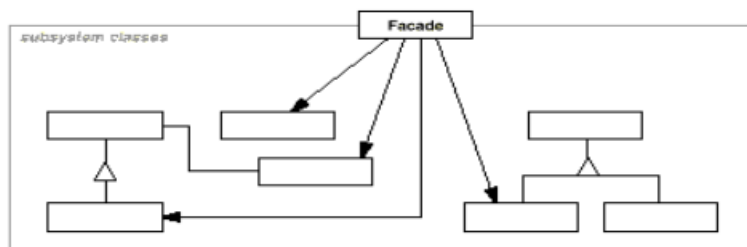


Slika 50-Struktura Bridge paterna

Apstraktnoj klasi Abstraction odgovara klasa BrokerBazePodataka, koju implementira klasa BrokerBazePodatakImpl. BrokerBazePodataka ima referencu na Apstraktnu klasu GeneralIDObject, tj. Abstraction ima reference na Implementor. Kako je Implementor u našem slučaju GenrealIDObject, klase ConcretImplementorA je naša klasa Korisnik. Metoda Operation() klase Abstraction je bilo koja metoda iz BrokeraBazePodataka(insertRecord(GeneralIDObject), findRecord(GeneralIDObject), updateRecord(GeneralIDObject), deleteRecord(GeneralIDObject)) koju će na specifičan način da implementira klasa BrokerBazePodatakImpl, a izvršiće je na klasom GeneralIDObject koja je apstraktna i koju nasleđuje klasa Korisnik, ali u ovom slučaju i nad bilo kojom drugom klasom koja nasleđuje GeneralIDObject će moći da se izvrše metode iz BrokerBazePodatakImpl klase.

Facade petern

Ovim paternom se obezbeđuje jedinstven interfejs za skup interfejsa nekog podsistema. Na taj način se definiše interfejs visokog nivoa.

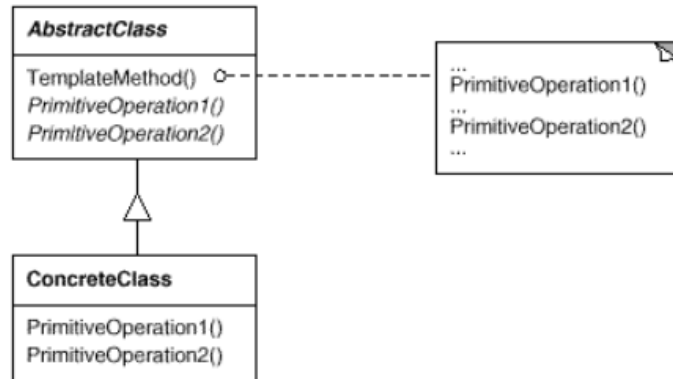


Slika 51-Struktura Facade paterna

Facade patern omogućava lakše korišćenje sistema od strane korisnika, tako što korisnik pozivom operacija ima jednostavan pristup bez obzira na to kako su operacije implementirane. Fasada prihvata zahtev od korisnika i prosleđuje ga dalje do komponenti zaduženih za njegovo izvršenje. Svaki od kontrolera GUIKontrolerRegistracija, GUIKontrolerMeni, GUIKontrolerLogin je zadužen da prima zahteve od korisnika i usmerava ih ka odgovarajućoj klasi zaduženoj za njihovo obavljanje. Neke od operacija se izvršavaju na klijentskoj strani, neke je potrebno realizovati povezivanjem na server i sl.

Template method patern

Definiše skelet algoritma u operaciji, prepuštajući izvršenje operacija podklasama. Omogućava podklasama redefinisane nekih koraka algoritma bez da se menja struktura algoritma.

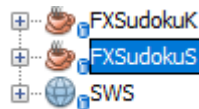


Slika 52-Struktura Template method paterna

U ovom slučaju klasa OpstelzvršenjeSO (AbstractClass sa slike) sadrži metodu opstelzvršenjeS() (metoda TemplateMethod() klase AbstractClass) koja u svom algoritmu sadrži abstraktnu metodu izvršiSO() (PrimitiveOperation1()) koja se implementira u svakoj od konkretnih klasa RegistrujKorisnika, PrijavaDK, AzurirajDK(ConcreteClass) koje nasleđuju OpstelzvršenjeSO.

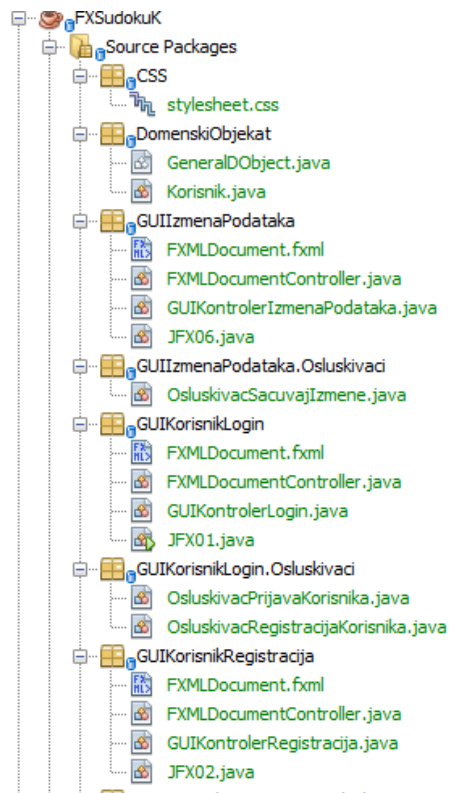
4 Faza implementacije

Softverski sistem je razvijen u programskom jeziku "Java". Sistem je projektovan kao klijent-server. Kao sistem za upravljanje bazom podataka korišćen je "MySQL", dok je razvojno okruženje "NetBeans IDE 8.2". Na osnovu arhitekture softverskog sistema dobijene su tri projekta:

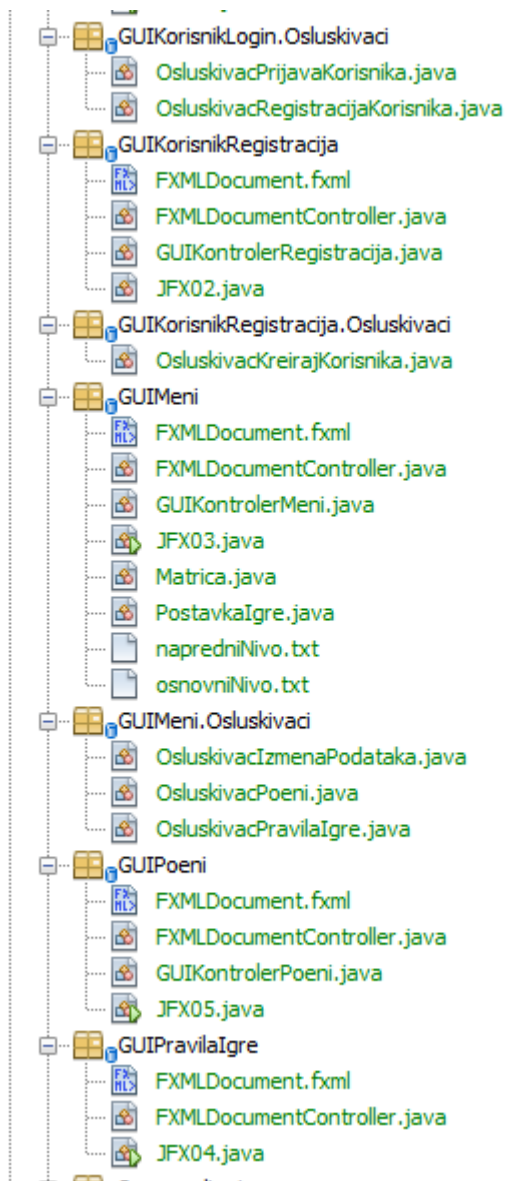


Slika 53-Projekti softverskog sistema

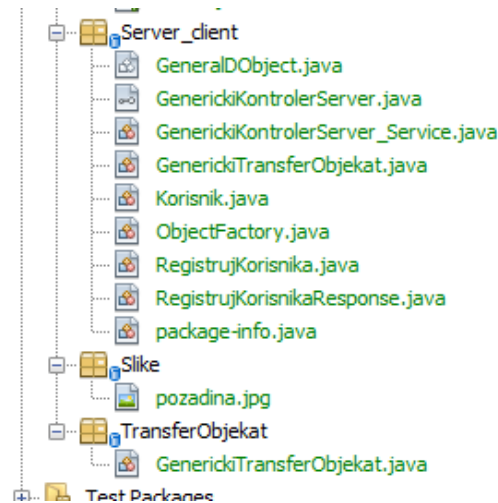
Struktura klijentskog projekta FXSudokuK je data narednim slikama.



Slika 54-Struktura klijentskog programa



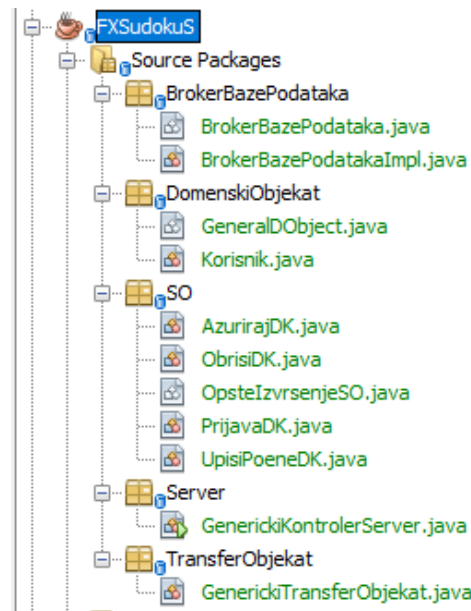
Slika 55-Struktura klijentskog programa



Slika 56-Struktura klijentskog programa

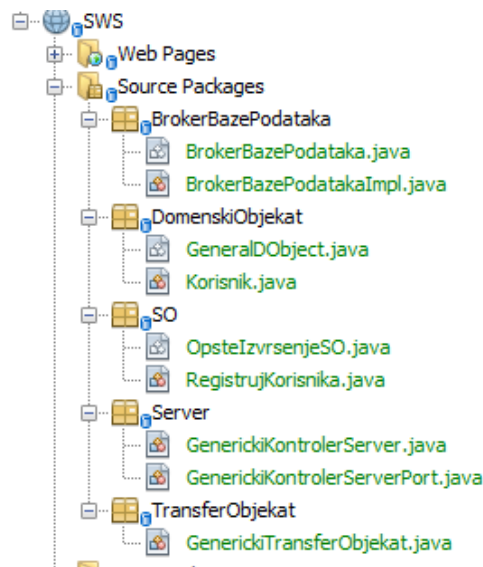
Klijentski deo programa zadužen je za praćenje događaja koji se dešavaju nad grafičkim komponentama opisanim u FXMLDocument.fxml fajlovima, prosleđivanjem događaja do GUIKontrolera i određivanjem operacije koja će se pozvati. GUIKontroler određuje da li je neophodno da se operacija poziva na serverskoj strani ili je dovoljan poziv samo sa klijentske strane.

Na serverskoj strani implementacija je dvostruka, preko soketa i preko servisa. Implementacija preko soketa izgleda ovako:



Slika 57-Struktura serverskog soket programa

Dok struktura serverskog programa preko servisa izgleda ovako:



Slika 58-Struktura serverskog servis programa

Serverski programi su zaduženi za povezivanje sa bazom podataka i primanje i obradu zahteva od klijentskog kontrolera. Komunikacija između klijenta i servera se obavlja preko `GenerickiTransferObjekat` klase kojom se objekti šalju u oba smera.

5 Faza testiranja

Automatski testovi nisu pokretani, jedini vid testiranja zasniva se na proveru podataka koji se unose kao ulazni argument u sistem. Program je testiran kada radi sa odgovarajućim, ali i neodgovarajućim podacima.

6 Zaključak

Ovaj rad prikazuje razvoj softverskog sistema za igru Sudoku. Korišćena metoda razvoja softverskog sistema je uprošćena Larmanova metoda. Prikazane su faze životnog ciklusa:

- Faza prikupljanja zahteva koja se sastoji od slučajeva korišćenja
- Faza analize je opisana pomoću sistemskih dijagrama sekvenci i ugovora (ponašanje softverskog sistema), konceptualnog i relacionog modela (struktura softverskog sistema).
- Faza projektovanja predstavljena pomoću dijagrama klasa i dijagrama sekvenci, a sadrži i projektovanje korisničkog interfejsa.
- Faza implementacije se zasniva na rezultatima faze projektovanja, a izvršena je korišćenjem NetBeans razvojnog okruženja
- Faza testiranja nije razrađena u potpunosti jer nisu pisani automatski testovi.

7 Literatura

„Softverski proces skripta”-radni materijal, ver. 1.2, dr Siniša Vlajić, Beograd 2020.