

Универзитет у Београду
Факултет организационих наука



Семинарски рад

Назив предмета: Софтверски процес, Катедра за софтверско инжењерство

тема:

Развој софтверског систем ИГРА МЕМОРИЈЕ уз помоћ
ЈаваФХ развојног окружења

Ментор: Проф.др. Синиша С. Влајић

Студент: Жељана Грујић 2022/3703

Београд, Март 2023.

Садржај

1. Фаза прикупљања корисничких захтева.....	5
1.1 Вербални опис.....	5
1.2 Случајеви коришћења.....	6
СК1: Случај коришћења- Регистрација корисника	7
СК2: Случај коришћења- Пријављивање корисника.....	8
СК3: Случај коришћења- Преглед правила игре	9
СК4: Случај коришћења- Информације о аутору програма	10
СК5: Случај коришћења- Покретање игре	11
СК6: Случај коришћења- Преглед активности и профила	12
СК7: Случај коришћења- Излаз из програма	12
2. Фаза анализе	13
2.1 Понашање софтверског система.....	13
ДС1: Дијаграм секвенци случаја коришћења – Регистрација корисника.....	14
ДС2: Дијаграм секвенци случаја коришћења – Пријављивање корисника.....	15
ДС3: Дијаграм секвенци случаја коришћења –Преглед правила игре	17
ДС4: Дијаграм секвенци случаја коришћења –Информације о аутору програма	18
ДС5: Дијаграм секвенци случаја коришћења –Покретање игре	19
ДС6: Дијаграм секвенци случаја коришћења –Преглед активности и профила	20
ДС7: Дијаграм секвенци случаја коришћења –Излаз из програма	21
Резултирајуће системске операције	22
2.2 Дефинисање уговора о системским операцијама	23
Уговор UG1: kreirajDK	23
Уговор UG2: prijaviDK	23
Уговор UG3: prikaziPravilaIgre	23
Уговор UG4: prikaziAutoraIgre.....	23
Уговор UG5: vratiIDKorisnika	23
Уговор UG6: dodajPartiju.....	23
Уговор UG7: vratiUlogovanogKorisnika	24
Уговор UG8: pronadjiPartije	24
Уговор UG9: prikaziMojProfil	24
Уговор UG10: izlazIzPrograma	24
Структура софтверског система - Концептуални (доменски) модел.....	25
Структура софтверског система- Релациони модел	26
3. Фаза пројектовања	28
3.1 Пројектовање корисничког интерфејса	29
3.1.1 Пројектовање екранских форми	30

СК1: Случај коришћења- Регистрација корисника	30
СК2: Случај коришћења- Пријављивање корисника.....	33
СК3: Случај коришћења- Преглед правила игре	36
СК4: Случај коришћења- Информације о аутору програма	38
СК5: Случај коришћења- Покретање игре	40
СК6: Случај коришћења- Преглед активности и профила	43
СК7: Случај коришћења- Излаз из програма	45
3.1.2 Пројектовање контролера корисничког интерфејса.....	46
3.2 Пројектовање апликационе логике	46
3.2.1 Конторлер апликационе логике.....	46
3.2.2 Пословна логика.....	47
Уговор UG1: kreirajDK	48
Уговор UG2: prijaviDK	48
Уговор UG3: prikaziPravilaIgre	49
Уговор UG4: prikaziAutoraIgre.....	49
Уговор UG5: vratiIDKorisnika	50
Уговор UG6: dodajPartiju.....	50
Уговор UG7: vratiUlogovanogKorisnika	51
Уговор UG8: pronadjiPartije	51
Уговор UG9: prikaziMojProfil	52
Уговор UG10: izlazIzPrograma	52
3.2.3 Пројектовање складишта података	59
4. Фаза имплементације.....	61
5. Тестирање	62
6. Принципи методе и стратегије пројектовања софтвера.....	62
6.1 Принципи (технике) пројектовања софтвера	62
Апстракција	62
Параметризација.....	62
Спецификација	63
Спојеност (coupling) и Кохезија (cohesion)	64
Декомпозиција и модуларизација	66
Енкапсулација и сакривање информација	67
Одвајање интерфејса и имплементације.....	68
6.2 Стратегије пројектовања софтвера.....	68
Подели и владај (Divide and conquer).....	68
Приступ са врха на доле (Top down) и Приступ одоздо на горе (Bottom up)	69
Итеративно инкрементални приступ	69

6.3 Методе пројектовања софтвера	70
Принципи објекто-оријентисаног пројектовања	70
7. Примена патерна у пројектовању софтвера	71
МСV (Model – View – Controller)	72
Опште о патернима	72
Патерни као решење проблема пројектовања	73
8. Закључак	75
Коришћена литература	75

1.Фаза прикупљања корисничких захтева

1.1 Вербални опис

Тема предмета Софтверски процес јесте развој софтверског система који је логички заснован на матрици, а уз помоћ принципа, метода и стратегија пројектовања софтвера. Као одговор на постављени захтев развијен је софтверски систем под називом „ИГРА МЕМОРИЈЕ“.

Систем новим корисницима треба да омогући регистрацију путем уношења свог имена, презимена, корисничког имена и лозинке, а затим пријаву на систем са претходно унетим креденцијалима, док постојећим корисницима систем нуди пријаву путем корисничког имена и лозинке.

Након успешне регистрације и/или пријаве систем корисницима омогућава да се упознају са правилима игре, одаберу тему и тежину нивоа игре коју желе да играју, погледају информације о свом корисничком налогу и својим активностима (колико пута су одиграли неку игру и датум када су последњи пут одиграли конкретну игру), излазак из система.

1.2 Случајеви коришћења

Идентификовани случајеви коришћења:

1. [Регистрација корисника](#)
2. [Пријављивање корисника](#)
3. [Преглед правила игре](#)
4. [Информације о аутору програма](#)
5. [Покретање игре](#)
6. [Преглед активности и профила](#)
7. [Изаз из програма](#)



Слика 1. Случајеви коришћења

СК1: Случај коришћења- Регистрација корисника

Назив СК

Регистровање **корисника**.

Актори СК

Корисник

Учесници СК

Корисник и **систем** (програм)

Предуслов: **Систем** је укључен и приказује форму за регистровање новог **корисника**.

Основни сценарио СК

1. **Корисник** уноси податке за регистрацију. (АПУСО)
2. **Корисник** контролише да ли је коректно унео податке за регистрацију. (АНСО)
3. **Корисник** позива **систем** да креира **корисника** са задатим подацима.(АПСО)
4. **Систем** креира новог **корисника**. (СО)
5. **Систем** приказује поруку „ **Систем** је успешно креирао новог **корисника**“. (ИА)

Алтернативна сценарија

- 5.1. Уколико **систем** не може да креира новог **корисника**, приказује **кориснику** поруку
“**Систем** не може да креира **корисника** на основу унетих података”.(ИА)

СК2: Случај коришћења- Пријављивање корисника

Назив СК

Пријављивање **корисника**.

Актори СК

Корисник

Учесници СК

Корисник и **систем** (програм)

Предуслов: **Систем** је укључен и приказује форму за пријављивање постојећег **корисника**.

Основни сценарио СК

6. **Корисник** уноси податке за пријаву. (АПУСО)
7. **Корисник** контролише да ли је коректно унео податке за пријаву. (АНСО)
8. **Корисник** позива **систем** да пронађе **корисника** са задатим подацима.(АПСО)
9. **Систем** претражује **кориснике**. (СО)
10. **Систем** враћа корисника и **приказује** поруку „Успешна пријава на **систем** “. (ИА)

Алтернативна сценарија

- 5.2. Уколико **систем** није пронашао **корисника**, приказује **кориснику** поруку
“**Систем** не може да пронађе **корисника** на основу унетих података“.(ИА)
- 5.3. Уколико **корисник** није успео да се пријави на **систем**, **систем** шаље **кориснику** поруку
“Неуспешна пријава, молимо Вас проверите унете креденцијале”. (ИА)

СКЗ: Случај коришћења- Преглед правила игре

Назив СК

Преглед правила игре

Актори СК

Корисник

Учесници СК

Корисник и **систем** (програм)

Предуслов: **Систем** је укључен и **корисник** је улогован под својом шифром. **Систем** приказује форму главни мени.

Основни сценарио СК

1. **Корисник** позива **систем** да прикаже форму са правилима игре.(АПСО)
2. **Систем** проналази тражену форму.(СО)
3. **Систем** приказује **кориснику** форму са подацима о правилима игре.(ИА)

Алтернативна сценарија

- 3.1 Уколико **систем** не може да прикаже тражену форму, он приказује **кориснику** поруку:
“**Систем** не може да прикаже тражену форму.”.(ИА)

СК4: Случај коришћења- Информације о аутору програма

Назив СК

Информације о аутору програма

Актери СК

Корисник

Учесници СК

Корисник и **систем** (програм)

Предуслов: **Систем** је укључен и **корисник** је улогован под својом шифром. **Систем** приказује форму главни мени.

Основни сценарио СК

1. **Корисник** позива **систем** да прикаже форму са информацијама о аутору програма.(АПСО)
2. **Систем** проналази тражену форму.(СО)
3. **Систем** приказује **кориснику** форму са информацијама о аутору програма.(ИА)

Алтернативна сценарија

- 3.1 Уколико **систем** не може да прикаже тражену форму, он приказује **кориснику** поруку:
“**Систем** не може да прикаже тражену форму.”.(ИА)

СК5: Случај коришћења- Покретање игре

Назив СК

Покретање игре

Актори СК

Корисник

Учесници СК

Корисник и **систем** (програм)

Предуслов: **Систем** је укључен и **корисник** је улогован под својом шифром. **Систем** приказује форму игре из главног менија. Учитан је улоговани корисник.

Основни сценарио СК

1. **Корисник** бира картицу са талона. (АПУСО)
2. **Корисник** контролише да ли је изабрао одговарајућу картицу како би остварио победу. (АНСО)
3. **Корисник** позива **систем** да запамти податке о одиграној **партији**. (АПСО)
4. **Систем** памти податке о **партији**. (СО)
5. **Систем** приказује **кориснику** поруку о исходу игре. (ИА)

Алтернативна сценарија:

5.1 Уколико **систем** не може да запамти податке о одиграној **партији**, он приказује **кориснику** поруку: “Систем не може да запамти податке о **партији**”.

СК6: Случај коришћења- Преглед активности и профила

Назив СК

Преглед активности и профила

Актори СК

Корисник

Учесници СК

Корисник и **систем** (програм)

Предуслов: **Систем** је укључен и **корисник** је улогован под својом шифром. **Систем** приказује форму главни мени. Учитан је улоговани корисник и листа његових одиграних партија.

Основни сценарио СК

1. **Корисник** позива **систем** да прикаже форму са подацима о профилу и активностима одиграних партија.(АПСО)
2. **Систем** проналази тражену форму.(СО)
3. **Систем** приказује **кориснику** форму са информацијама о профилу и одиграним партијама.(ИА)

Алтернативна сценарија

- 3.1 Уколико **систем** не може да прикаже тражену форму, он приказује **кориснику** поруку:
“**Систем** не може да прикаже тражену форму.”.(ИА)

СК7: Случај коришћења- Излаз из програма

Назив СК

Излаз из програма

Актори СК

Корисник

Учесници СК

Корисник и **систем** (програм)

Предуслов: **Систем** је укључен и **корисник** је улогован под својом шифром. **Систем** приказује главни мени.

Основни сценарио СК

1. **Корисник** позива **систем** да изврши опцију излаз из програма.(АПСО)
2. **Систем** проналази тражену опцију.(СО)
3. **Систем** приказује **кориснику** поруку „ До следећег виђења, поздрав.“.(ИА)

2. Фаза анализе

У фази анализе описујемо логичку структуру и понашање софтвера. Заправо, резултат фазе анализе јесте *пословна логика* софтверског система. Понашање софтверског система се описује помоћу *дијаграма секвенци* и *системских операција*, док се структура система описује помоћу *концептуалног* и *релационог* модела.

2.1 Понашање софтверског система

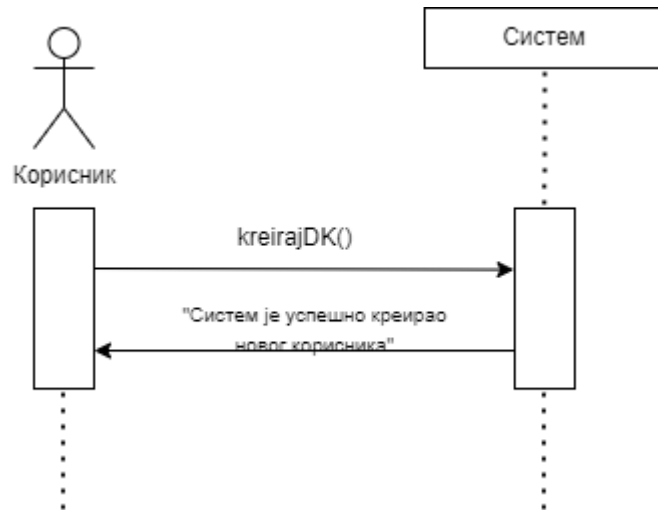
Понашање софтверског система приказујемо путем системских дијаграма секвенци, где ћемо за сваки *случај коришћења*, уочен у *фази прикупљања захтева*, дати дијаграм секвенци. Системски дијаграм секвенци треба да прикаже интеракцију између актора и система, путем активности у одређеном редоследу.

*На дијаграму секвенци, актор не комуницира са системом директно, већ преко посредника (форма). Приликом цртања секвенцих дијаграма, изоставићемо приказ форме, али се њено присуство у комуникацији подразумева.

ДС1: Дијаграм секвенци случаја коришћења – Регистрација корисника

Основни сценарио СК

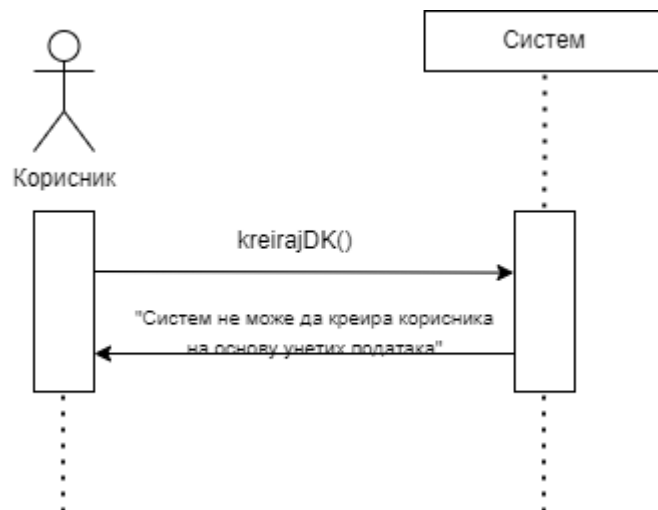
1. **Корисник** позива **систем** да креира **корисника** са задатим подацима.(АПСО)
2. **Систем** приказује поруку „**Систем** је успешно креирао новог **корисника**“. (ИА)



Дијаграм 1: ДС Регистрација корисника (основни сценарио)

Алтернативна сценарија

- 2.1 Уколико **систем** не може да креира новог **корисника**, приказује **кориснику** поруку “**Систем** не може да креира **корисника** на основу унетих података”.(ИА)



Дијаграм 2: ДС Регистрација корисника (алтернативни сценарио 1)

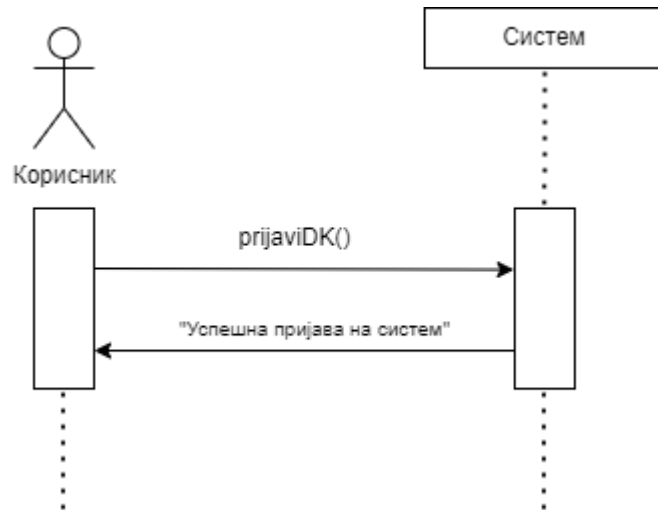
Са наведених секвенцих дијаграма уочава се једна системска операција коју треба пројектовати:

1. *signal* **kreirajDK()**

ДС2: Дијаграм секвенци случаја коришћења – Пријављивање корисника

Основни сценарио СК

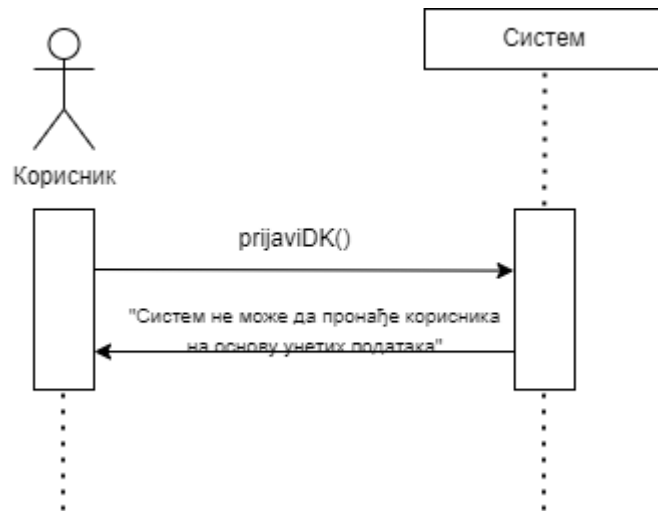
1. **Корисник** позива **систем** да пронађе **корисника** са задатим подацима.(АПСО)
2. **Систем** враћа корисника и **приказује** поруку „Успешна пријава на **систем** “. (ИА)



Дијаграм 3: ДС Пријављивање корисника (основни сценарио)

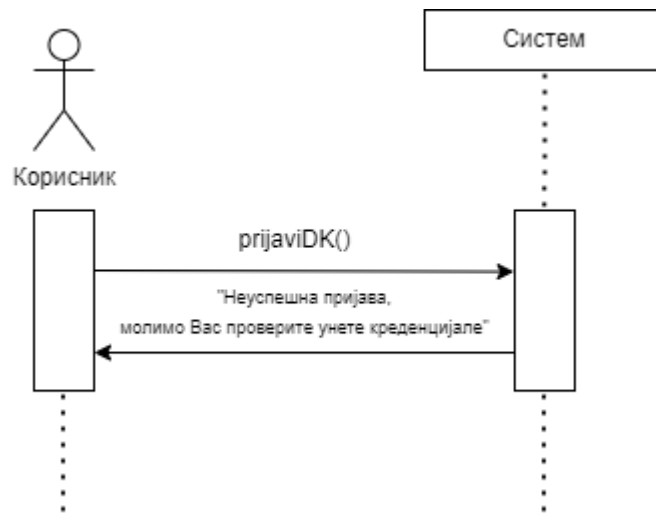
Алтернативна сценарија

- 2.1 Уколико **систем** није пронашао **корисника**, приказује **кориснику** поруку “**Систем** не може да пронађе **корисника** на основу унетих података“.(ИА)



Дијаграм 4: ДС Пријављивање корисника (алтернативни сценарио 1)

2.2 Уколико **корисник** није успео да се пријави на **систем**, **систем** шаље **кориснику** поруку “Неуспешна пријава, молимо Вас проверите унете креденцијале”. (ИА)



Дијаграм 5: ДС Пријављивање корисника (алтернативни сценарио 2)

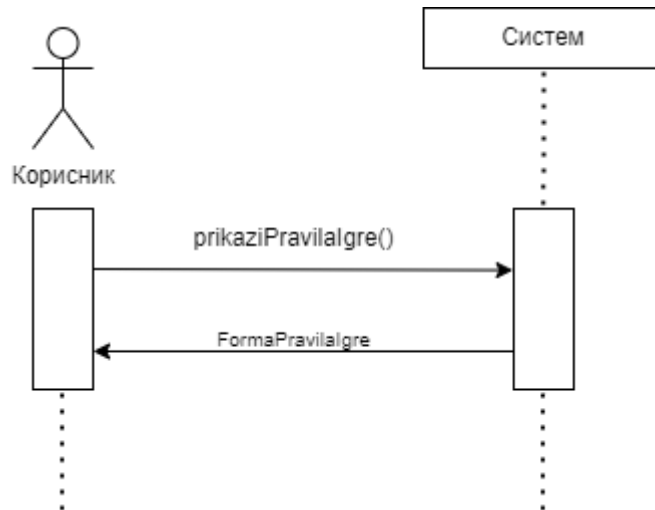
Са наведених секвенцних дијаграма уочава се једна системска операција коју треба пројектовати:

2. *signal* **prijaviDK()**

ДС3: Дијаграм секвенци случаја коришћења –Преглед правила игре

Основни сценарио СК

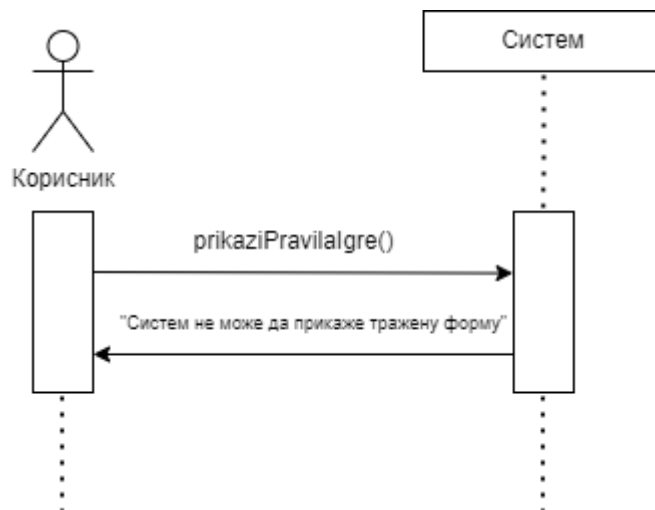
1. **Корисник** позива **систем** да прикаже форму са правилима игре.(АПСО)
2. **Систем** приказује **кориснику** форму са подацима о правилима игре.(ИА)



Дијаграм 6: ДС Преглед правила игре (основни сценарио)

Алтернативна сценарија СК

- 2.1 Уколико **систем** не може да прикаже тражену форму, он приказује **кориснику** поруку: “Систем не може да прикаже тражену форму.”.(ИА)



Дијаграм 7: ДС Преглед правила игре (алтернативни сценарио 1)

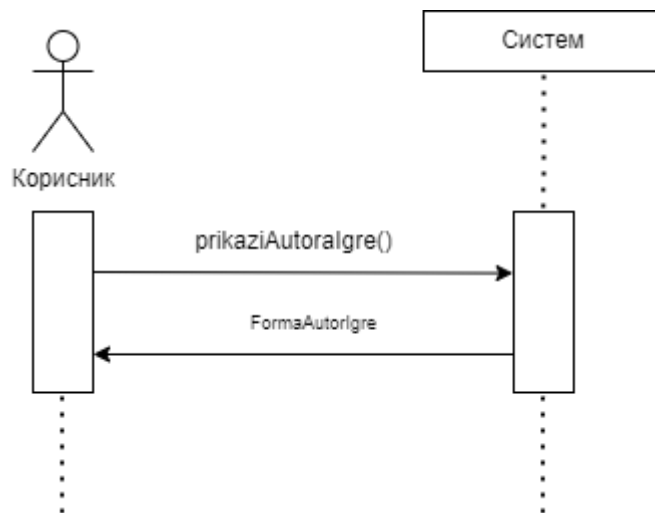
Са наведених секвенцих дијаграма уочава се једна системска операција коју треба пројектовати:

1. *signal* **prikaziPravilaIgre()**

ДС4: Дијаграм секвенци случаја коришћења –Информације о аутору програма

Основни сценарио СК

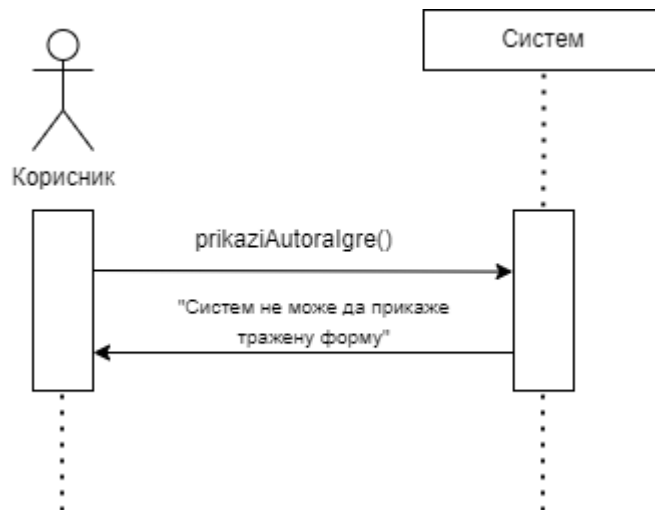
1. **Корисник** позива **систем** да прикаже форму са информацијама о аутору
2. **Систем** приказује **кориснику** форму са информацијама о аутору програма.(ИА)



Дијаграм 8: ДС Информације о аутору програма (основни сценарио)

Алтернативна сценарија СК

- 2.1 Уколико **систем** не може да прикаже тражену форму, он приказује **кориснику** поруку: “Систем не може да прикаже тражену форму.”.(ИА)



Дијаграм 9: ДС Информације о аутору програма (алтернативни сценарио 1)

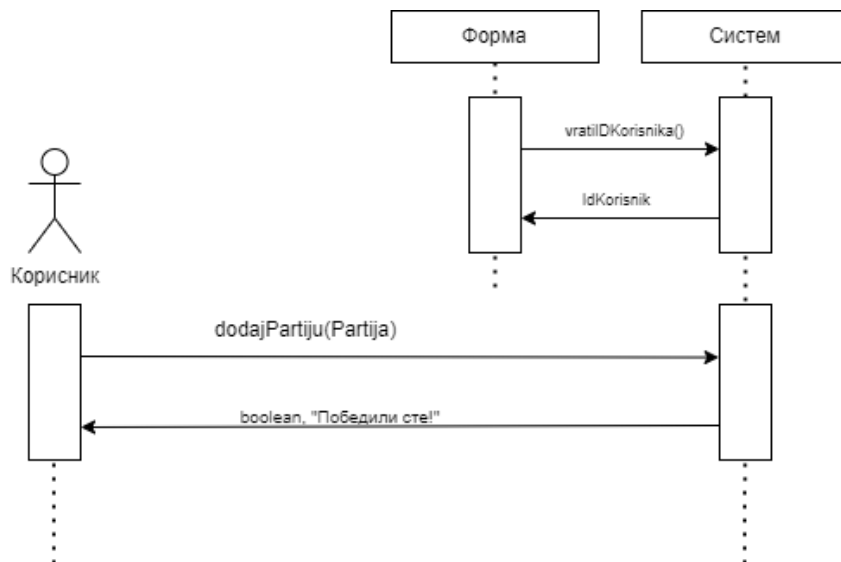
Са наведених секвенцих дијаграма уочава се једна системска операција коју треба пројектовати:

- 1 *signal* `prikaziAutoralgre()`

ДС5: Дијаграм секвенци случаја коришћења –Покретање игре

Основни сценарио СК

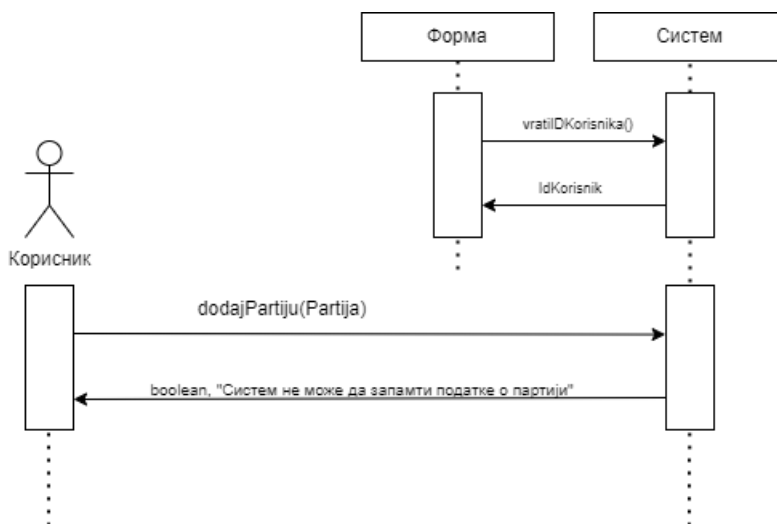
1. **Корисник** позива **систем** да запамти податке о одиграној **партији**. (АПСО)
2. **Систем** приказује **кориснику** поруку о исходу игре. (ИА)



Дијаграм 10: ДС Покретање игре (основни сценарио)

Алтернативна сценарија

- 2.1 Уколико **систем** не може да запамти податке о одиграној **партији**, он приказује **кориснику** поруку: “Систем не може да запамти податке о **партији**”.



Дијаграм 11: ДС Покретање игре (алтернативни сценарио 1)

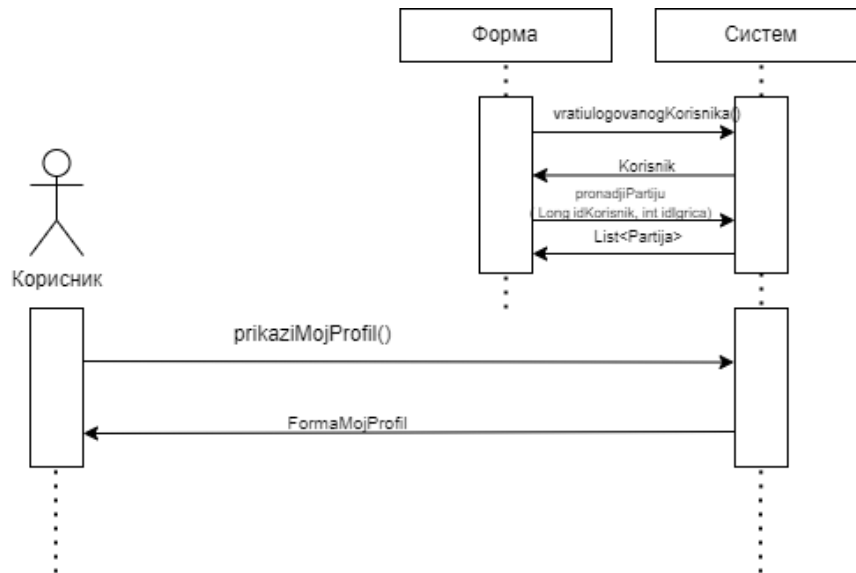
Са наведених секвенцих дијаграма уочавају се две системске операције које треба пројектовати:

1. *signal* `vratiIDKorisnika()`
2. *signal* `dodajPartiju(Partija)`

ДС6: Дијаграм секвенци случаја коришћења –Преглед активности и профила

Основни сценарио СК

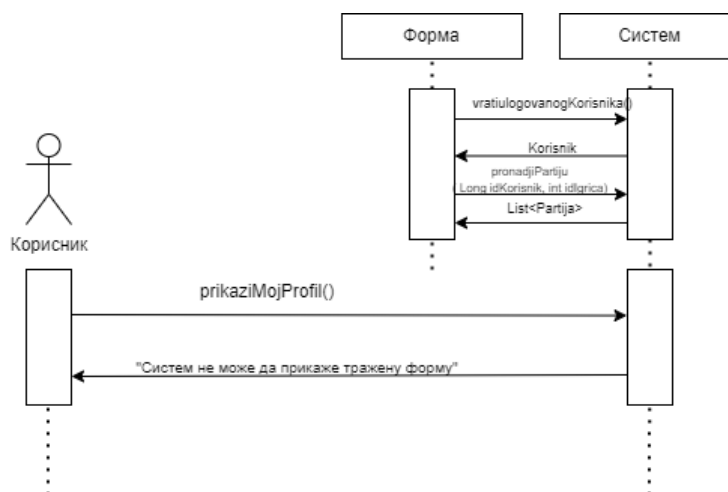
1. **Корисник** позива **систем** да прикаже форму са подацима о профилу и активностима одиграних партија.(АПСО)
2. **Систем** приказује **кориснику** форму са информацијама о профилу и одиграним партијама.(ИА)



Дијаграм 12: ДС Преглед активности и профила (основни сценарио)

Алтернативна сценарија СК

- 2.1 Уколико **систем** не може да прикаже тражену форму, он приказује **кориснику** поруку: “Систем не може да прикаже тражену форму.”.(ИА)



Дијаграм 13: ДС Преглед активности и профила (алтернативни сценарио)

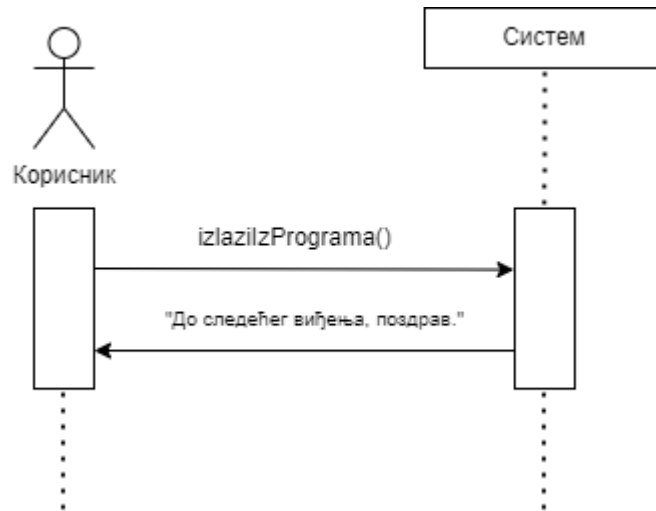
Са наведених секвенцих дијаграма уочавју се три системске операције које треба пројектовати:

- 1 *signal* **vratiUlogovanogKorisnika()**
- 2 *signal* **pronadjiPartije(Long idKorisnik, int idIgrica)**
- 3 *signal* **prikaziMojProfil()**

ДС7: Дијаграм секвенци случаја коришћења –Излаз из програма

Основни сценарио СК

1. **Корисник** позива **систем** да изврши опцију излаз из програма.(АПСО)
2. **Систем** приказује **кориснику** поруку „ До следећег виђења, по здрав.“.(ИА)



Дијаграм 14: ДС Излаз из програма (основни сценарио)

Са наведених секвенчних дијаграма уочава се једна системска операција коју треба пројектовати:

1 *signal* **izlazIzPrograma()**

Резултирајуће системске операције

Као резултат анализе сценарија добијамо укупно 14 системских операција које треба пројектовати:

1. *signal* **kreirajDK()**
2. *signal* **prijaviDK()**
3. *signal* **prikaziPravilaIgre()**
4. *signal* **prikaziAutoraIgre()**
5. *signal* **vratiIDKorisnika()**
6. *signal* **dodajPartiju(Partija)**
7. *signal* **vratiUlogovanogKorisnika()**
8. *signal* **pronadjiPartije(Long idKorisnik, int idIgrica)**
9. *signal* **prikaziMojProfil()**
10. *signal* **izlazIzPrograma()**

2.2 Дефинисање уговора о системским операцијама

Понашање софтверског система се описује преко системских операција, а за сваку системску операцију се прави уговор. Уговор описују понашање системске операције, то јест описује се оно шта та системска опеација треба да одради (али не и како то треба да одради).

Један уговор везује се за једну системску операцију, и састоји се од следећих секција:

- *операција*
- *веза са СК*
- *предуслов*
- *постуслов*

Уговор UG1: kreirajDK

Операција: kreirajDK():signal;

Веза са СК: СК1

Предуслови: *Вредносна и структурна ограничења над објектом Корисник морају бити задовољена.*

Постуслови: *Креиран је нови корисник.*

Уговор UG2: prijavidK

Операција: prijavidK(): signal;

Веза са СК: СК2

Предуслови: /

Постуслови: /

Уговор UG3: prikaziPravilaIgre

Операција: prikaziPravilaIgre(): signal;

Веза са СК: СК3

Предуслови: /

Постуслови: /

Уговор UG4: prikaziAutoraIgre

Операција: prikaziAutoraIgre(): signal;

Веза са СК: СК4

Предуслови: /

Постуслови: /

Уговор UG5: vratiIDKorisnika

Операција: vratiIDKorisnika(): signal;

Веза са СК: СК5

Предуслови: /

Постуслови: /

Уговор UG6: dodajPartiju

Операција: dodajPartiju(Partija):signal;

Веза са СК: СК5

Предуслови: *Вредносна и структурна ограничења над објектом Партија морају бити задовољена.*

Постуслови: *Креирана је нова партија.*

Уговор UG7: vratiUlogovanogKorisnika

Операција: `vratiUlogovanogKorisnika()`: signal;

Веза са СК: СК6

Предуслови: /

Постуслови: /

Уговор UG8: pronadjiPartije

Операција: `pronadjiPartije(Long idKorisnik, int idIgrica)`: signal;

Веза са СК: СК6

Предуслови: /

Постуслови: /

Уговор UG9: prikaziMojProfil

Операција: `prikaziMojProfil()`: signal;

Веза са СК: СК6

Предуслови: /

Постуслови: /

Уговор UG10: izlazIzPrograma

Операција: `izlazIzPrograma()`: signal;

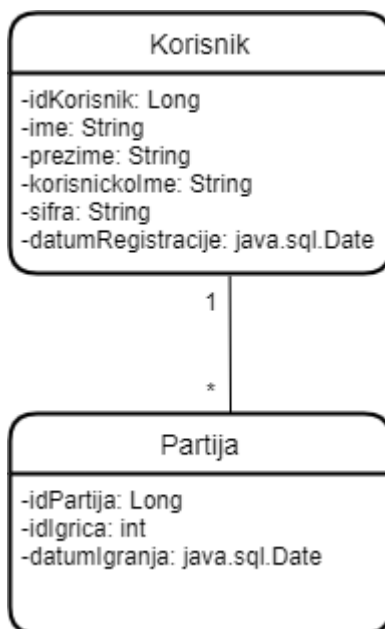
Веза са СК: СК7

Предуслови: /

Постуслови: /

Структура софтверског система - Концептуални (доменски) модел

Помоћу концептуалног модела описујемо структуру система. Концептуални модел садржи концептуалне класе (доменске објекте) и асоцијације између концептуалних класа.



Слика 2 Концептуални модел

Структура софтверског система- Релациони модел

На основу концептуалног модела, прави се релациони модел, а на основу њега се пројектује релациона база података.

У концептуалном моделу се могу идентификовати следеће класе: *Korisnik*, и *Partija*. Свака релација ће бити представљена као једна табела у релационом моделу.

Korisnik(idKorisnik, ime, prezime, korisnickoIme, sifra, datumRegistracije)

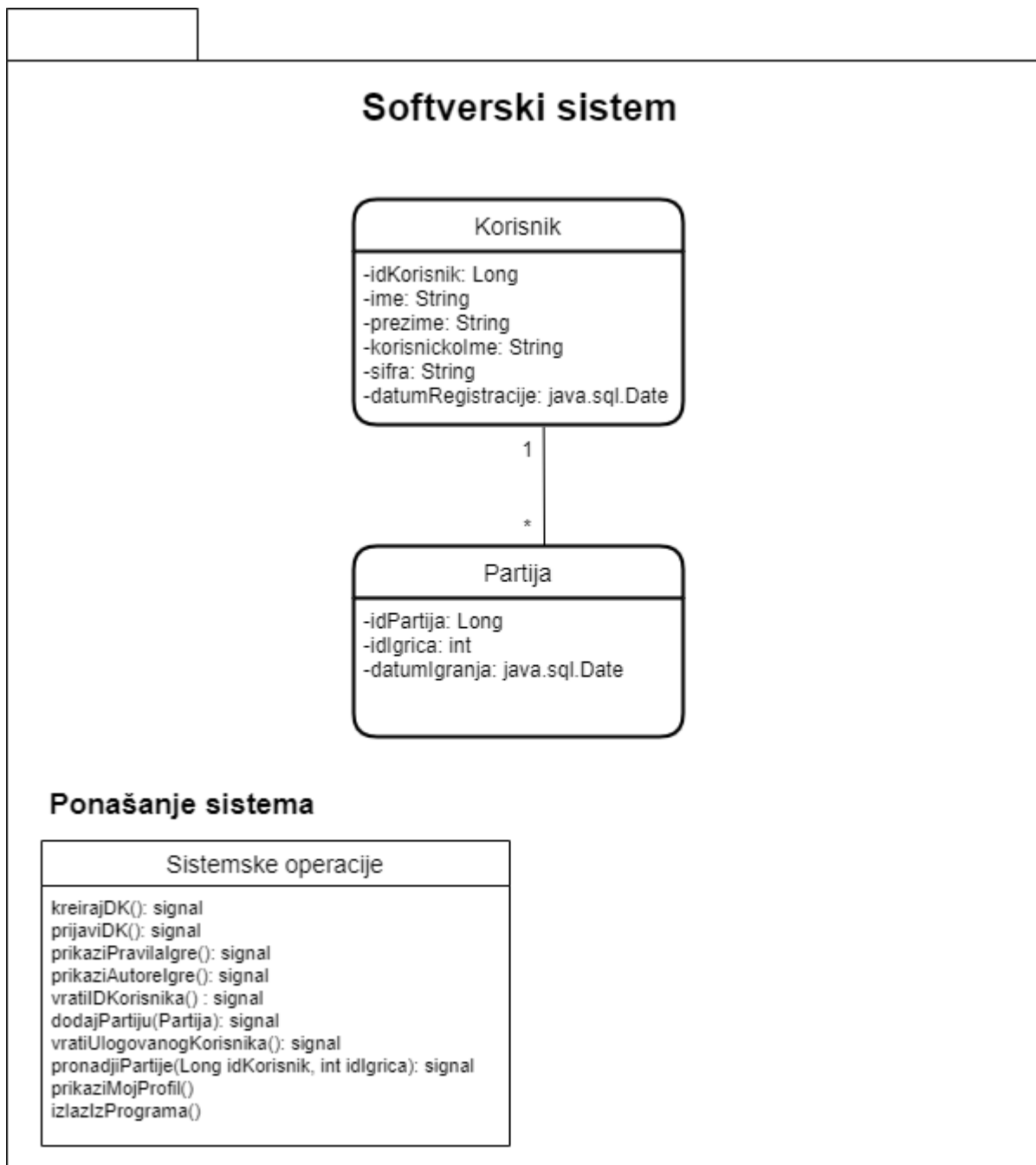
Partija(idPartija, idIgrica, datumIgranja, *idKorisnik*)

Табеле ограничења

Табела <i>Korisnik</i>		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути		Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT / UPDATE CASCADES Partija DELETE RESTRICTED Partija
	idKorisnik	Long	Not null and >0			
	ime	String	Not null			
	prezime	String	Not null			
	korisnickoIme	String	Not null			
	sifra	String	Not null			
	datumRegistracije	Date	Not null			

Табела <i>Partija</i>		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути		Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT RESTRICTED Korisnik UPDATE RESTRICTED Korisnik DELETE/
	idPartija	Long	Not null and >0			
	idIgrica	int	Not null and >0			
	datumIgranja	Date	Not null			
	idKorisnik	Long	Not null and >0			

Као резултат анализе сценарија случајева коришћења и прављења концептуалног модела добија се логичка структура и понашање софтверског система.



Слика 3 Софтверски систем, пословна логика

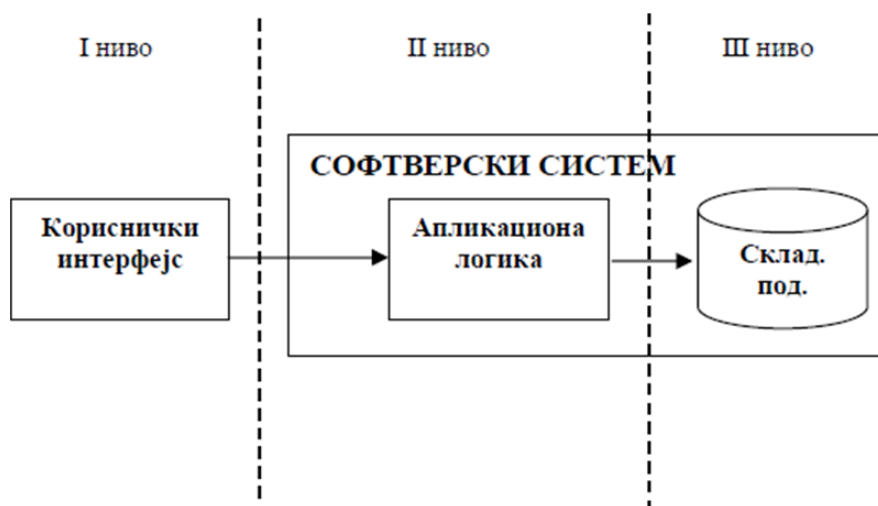
3.Фаза пројектовања

Фаза пројектовања описује физичку структуру и понашање софтверског система. Пројектовање архитектуре софтверског система обухвата пројектовање корисничког интерфејса (пројектовање контролера корисничког интерфејса и екранских форми), апликационе логике (пројектовање контролера апликационе логике и пословне логике) и складишта података (брокер базе података).

Архитектура система је тронивојска и састоји се од следећих нивоа:

- кориснички интерфејс
- апликациона логика
- складиште података

Ниво корисничког интерфејса је на страни клијента, док су апликациона логика и складиште на страни сервера.

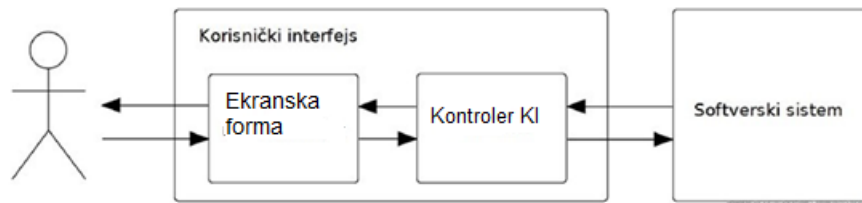


Слика 4 Тронивојска архитектура

3.1 Пројектовање корисничког интерфејса

Кориснички интерфејс представља улазно-излазну реализацију софтверског система. Састоји се од:

1. Екранске форме
2. Контролера корисничког интерфејса



Слика 5 Структура корисничког интерфејса

3.1.1 Пројектовање екранских форми

Кориснички интерфејс је дефинисан преко скупа екранских форми. Сценарио коришћења екранских форми је директно повезан са сценаријима случајева коришћења.

Постоје два аспекта пројектовања екранске форме:

1. Пројектовање сценарија СК који се изводе преко екранске форме
2. Пројектовање метода екранске форме

СК1: Случај коришћења- Регистрација корисника

Назив СК

Регистровање **корисника**.

Актери СК

Корисник

Учесници СК

Корисник и **систем** (програм)

Предуслов: **Систем** је укључен и приказује форму за регистровање новог **корисника**.



The image shows a screenshot of a web application window titled "Registracija". The window has a yellow-to-blue gradient background. At the top, it asks "Želite da igrate igru memorije?" (Do you want to play the memory game?). Below this, it says "Unesite svoje podatke i registrujte se!" (Enter your data and register!). There are four input fields: "Ime" (Name), "Prezime" (Surname), "Korisničko ime:" (Username), and "Šifra:" (Password). At the bottom, there are two buttons: "Registruj se" (Register) and "Prijavi se" (Login). Above the buttons, there are two links: "Nemaš nalog?" (Don't have an account?) and "Imaš nalog?" (Have an account?).

Слика 6 Форма за регистровање корисника на систем

Основни сценарио СК

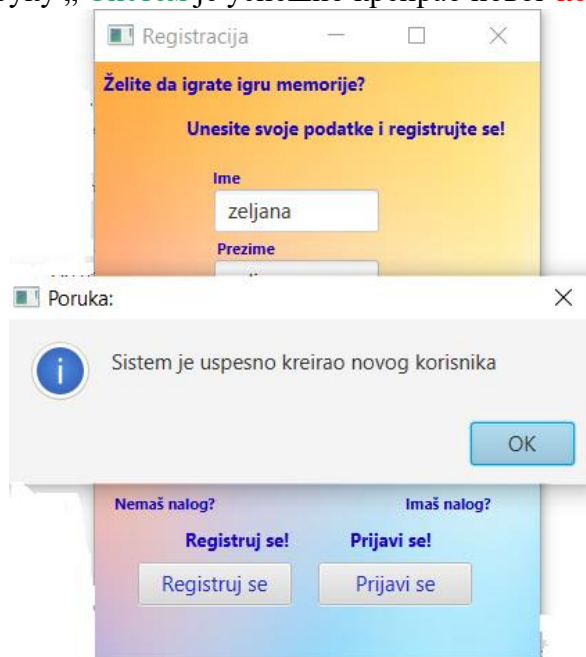
1. **Корисник** уноси податке за регистрацију. (АПУСО)
2. **Корисник** контролише да ли је коректно унео податке за регистрацију. (АНСО)

Слика 7 Форма за регистровање корисника на систем, унети подаци

3. **Корисник** позива **систем** да креира **корисника** са задатим подацима.(АПСО)

Опис акције: Корисник кликом на дугме „Региструј се“ позива системску операцију **kreirajDK()**

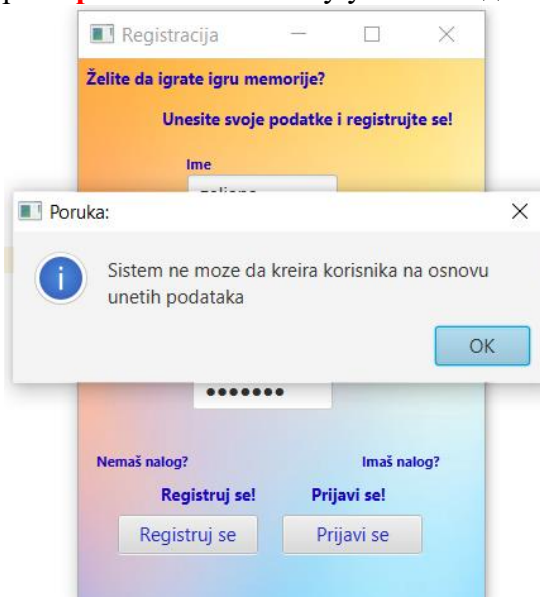
4. **Систем** креира новог **корисника**. (СО)
5. **Систем** приказује поруку „**Систем** је успешно креирао новог **корисника**“. (ИА)



Слика 8 Успешна регистрација на систем

Алтернативна сценарија

- 5.1 Уколико **систем** не може да креира новог **корисника**, приказује **кориснику** поруку “**Систем** не може да креира **корисника** на основу унетих података”.(ИА)



Слика 9 Неуспешна регистрација на систем

СК2: Случај коришћења- Пријављивање корисника

Назив СК

Пријављивање **корисника**.

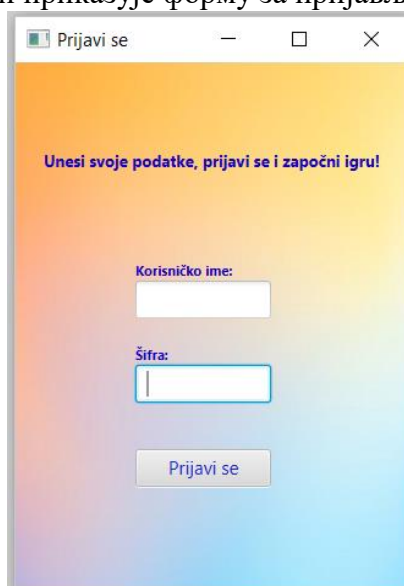
Актори СК

Корисник

Учесници СК

Корисник и **систем** (програм)

Предуслов: **Систем** је укључен и приказује форму за пријављивање постојећег **корисника**.



Слика 10 Форма за пријаву на систем

Основни сценарио СК

1. **Корисник** уноси податке за пријаву. (АПУСО)
2. **Корисник** контролише да ли је коректно унео податке за пријаву. (АНСО)

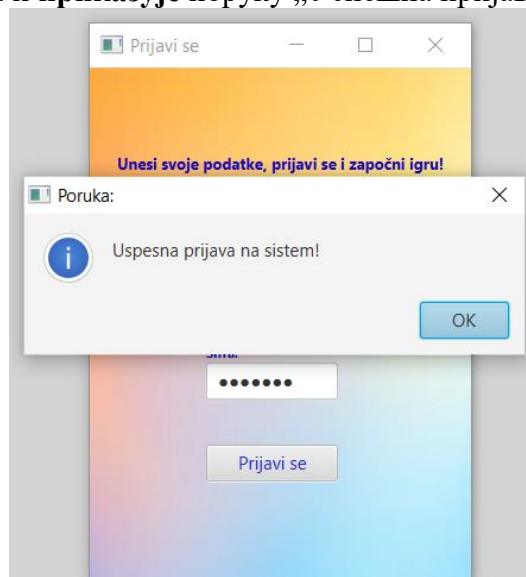


Слика 11 Форма за пријаву на систем, унети подаци

3. **Корисник** позива **систем** да пронађе **корисника** са задатим подацима.(АПСО)

Опис акције: Корисник кликом на дугме „Пријави се“ позива системску операцију **prijaviDK()**

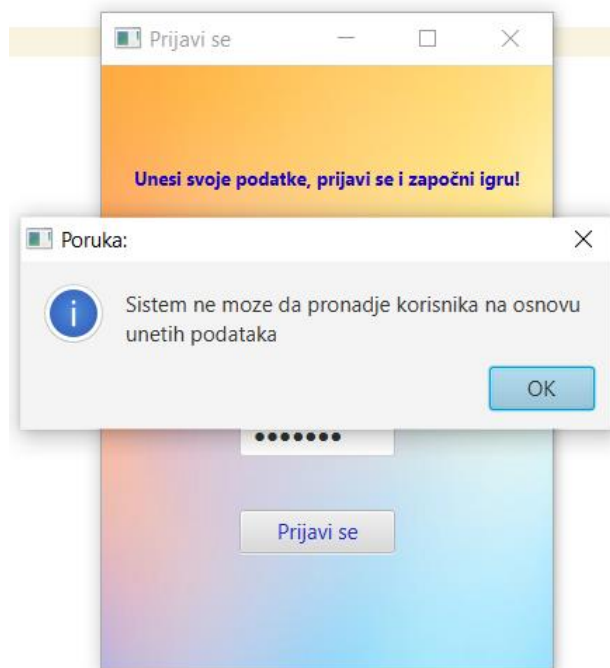
4. **Систем** претражује **кориснике**. (СО)
5. **Систем** враћа корисника и **приказује** поруку „Успешна пријава на **систем** “. (ИА)



Слика 12 Успешна пријава на систем

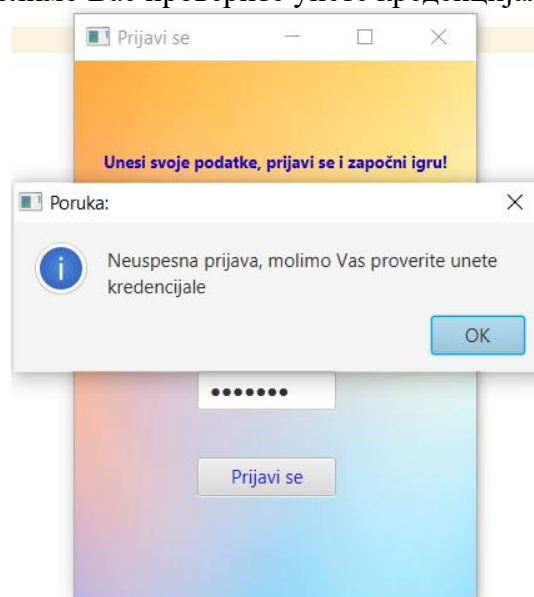
Алтернативна сценарија

- 5.1 Уколико **систем** није пронашао **корисника**, приказује **кориснику** поруку “**Систем** не може да пронађе **корисника** на основу унетих података“.(ИА)



Слика 13 Неуспешна пријава на систем

5.2 Уколико **корисник** није успео да се пријави на **систем**, **систем** шаље **кориснику** поруку “Неуспешна пријава, молимо Вас проверите унете кренцијале”. (ИА)



Слика 14 Неуспешна пријава на систем

СКЗ: Случај коришћења- Преглед правила игре

Назив СК

Преглед правила игре

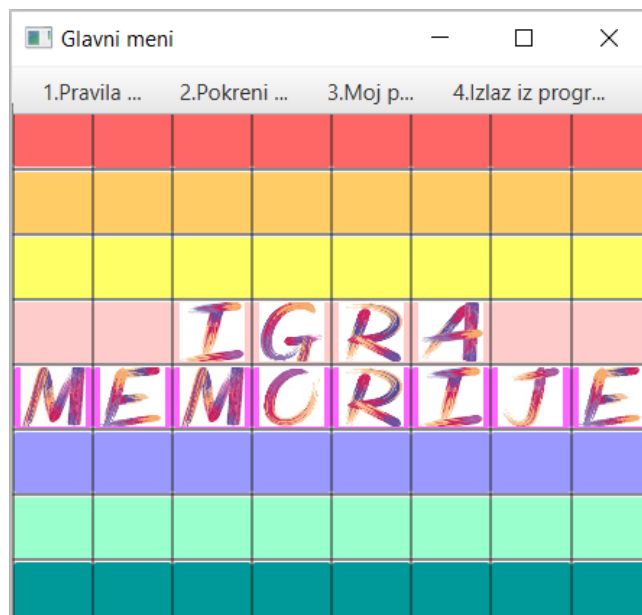
Актори СК

Корисник

Учесници СК

Корисник и **систем** (програм)

Предуслов: **Систем** је укључен и **корисник** је улогован под својом шифром. **Систем** приказује форму главни мени.



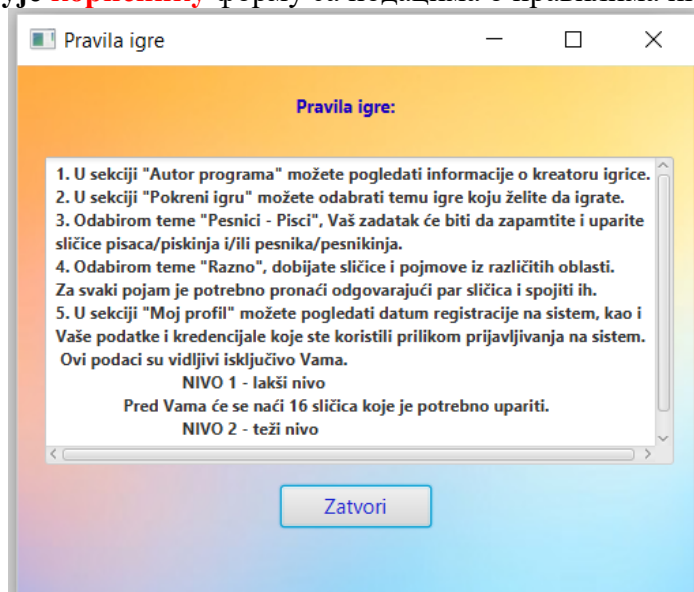
Слика 15 Форма главни мени

Основни сценарио СК

1. **Корисник** позива **систем** да прикаже форму са правилима игре.(АПСО)

Опис акције: Корисник кликом на опцију из падајућег менија „1. Правила игре > Погледај правила игре“ позива системску операцију **prikaziPravilaIgre()**

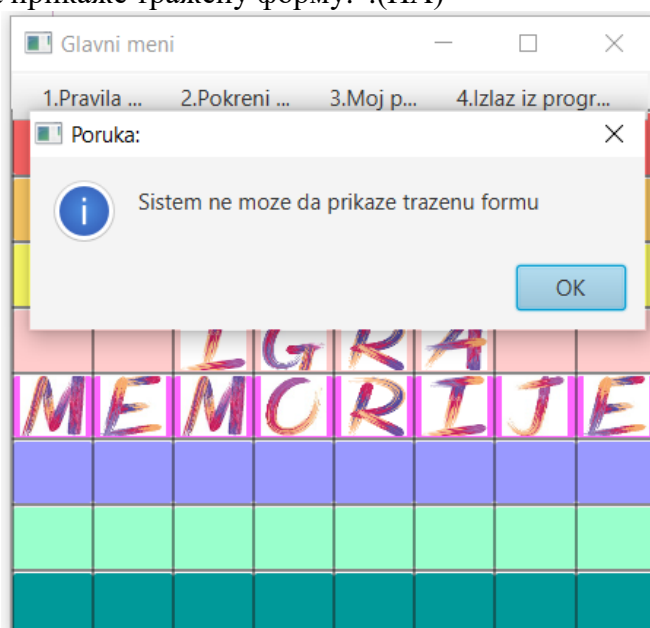
2. **Систем** проналази тражену форму.(СО)
3. **Систем** приказује **кориснику** форму са подацима о правилима игре.(ИА)



Слика 16 Форма правила игре

Алтернативна сценарија

- 3.1 Уколико **систем** не може да прикаже тражену форму, он приказује **кориснику** поруку: “Систем не може да прикаже тражену форму.”.(ИА)



Слика 17 Неуспешни приказ форме правила игре

СК4: Случај коришћења- Информације о аутору програма

Назив СК

Информације о аутору програма

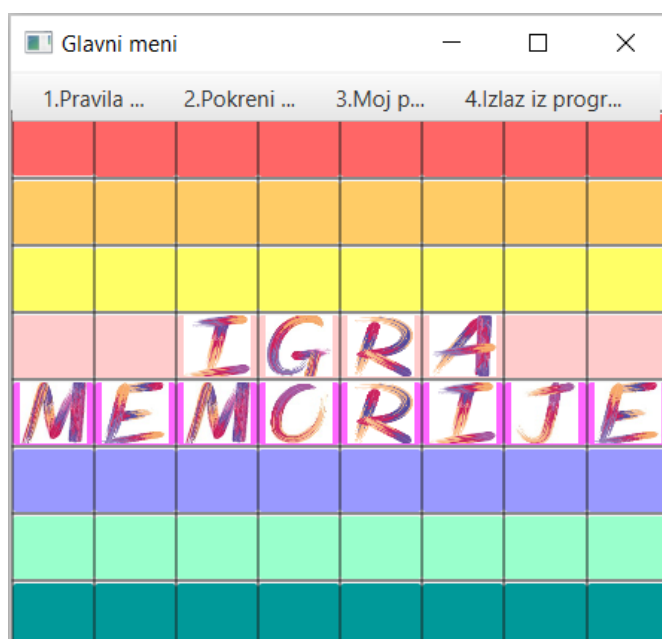
Актори СК

Корисник

Учесници СК

Корисник и **систем** (програм)

Предуслов: **Систем** је укључен и **корисник** је улогован под својом шифром. **Систем** приказује форму главни мени.



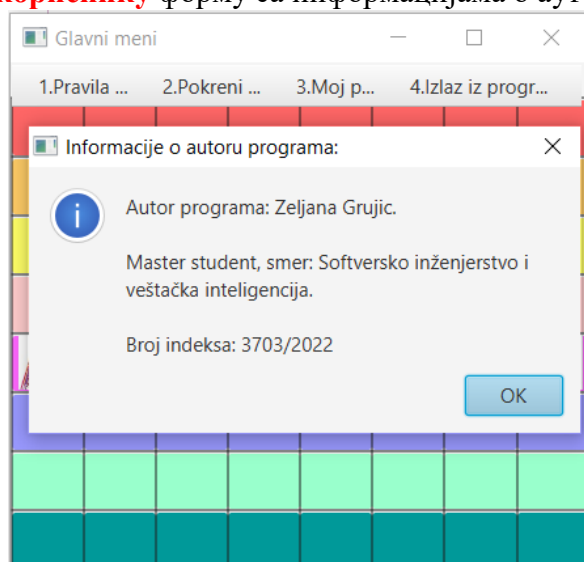
Слика 18 Форма главни мени

Основни сценарио СК

1. **Корисник** **позива** **систем** да прикаже форму са информацијама о аутору програма.(АПСО)

Опис акције: Корисник кликом на опцију из падајућег менија „1. Правила игре > Погледај правила игре“ позива системску операцију **prikaziAutoraIgre()**

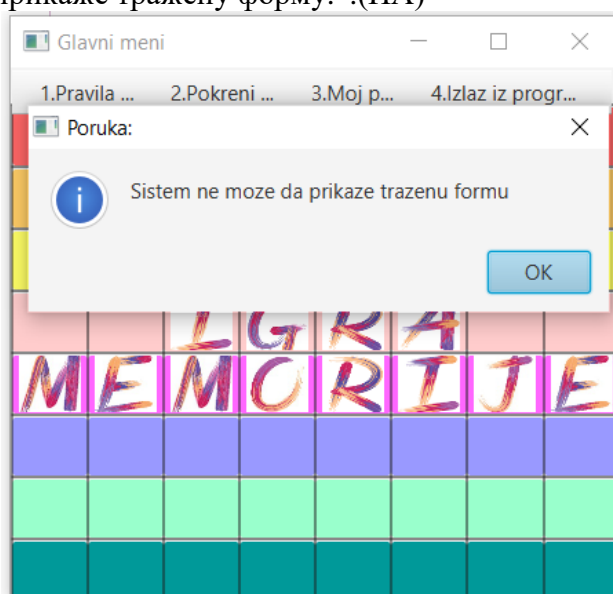
2. **Систем** **проналази** тражену форму.(СО)
3. **Систем** **приказује** **кориснику** форму са информацијама о аутору програма.(ИА)



Слика 19 Информације о аутору програма

Алтернативна сценарија

- 3.1 Уколико **систем** не може да прикаже тражену форму, он приказује **кориснику** поруку: “Систем не може да прикаже тражену форму.”.(ИА)



Слика 20 Неуспешни приказ форме правила игре

СК5: Случај коришћења- Покретање игре

Назив СК

Покретање игре

Актори СК

Корисник

Учесници СК

Корисник и **систем** (програм)

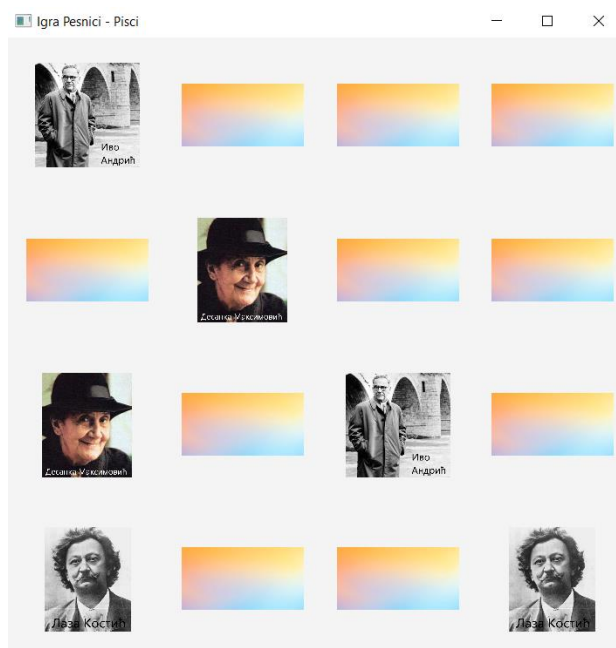
Предуслов: **Систем** је укључен и **корисник** је улогован под својом шифром. **Систем** приказује форму игрице из главног менија. Учитан је улоговани корисник.



Слика 21 Форма игрице

Основни сценарио СК

1. **Корисник** бира картицу са талона. (АПУСО)
2. **Корисник** контролише да ли је изабрао одговарајућу картицу како би остварио победу. (АНСО)

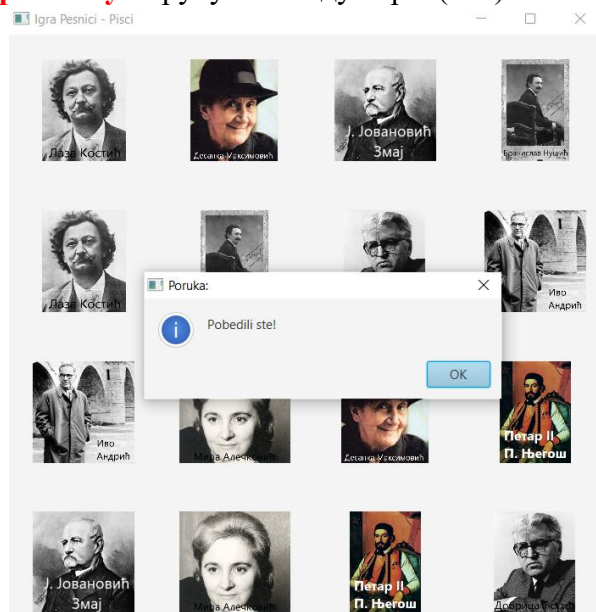


Слика 22 Корисник бира картице

3. **Корисник** позива **систем** да запамти податке о одиграној партији. (АПСО)

Опис акције: Корисник по проналаску свих картица и завршетку игре, имплицитно позива системску операцију **dodajPartiju(Partija)**

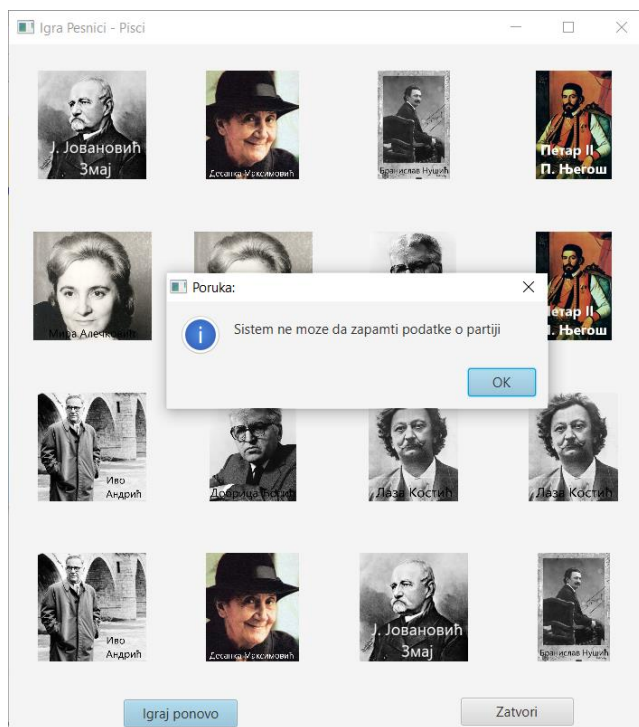
4. **Систем** памти податке о партији. (СО)
5. **Систем** приказује **кориснику** поруку о исходу игре. (ИА)



Слика 23 Кориснику се приказује исход игре

Алтернативна сценарија:

5.1 Уколико **систем** не може да запамти податке о одиграној **партији**, он приказује **кориснику** поруку: “Систем не може да запамти податке о **партији**”.



Слика 24 Неуспешно чување података о партији

СК6: Случај коришћења- Преглед активности и профила

Назив СК

Преглед активности и профила

Актори СК

Корисник

Учесници СК

Корисник и **систем** (програм)

Предуслов: **Систем** је укључен и **корисник** је улогован под својом шифром. **Систем** приказује форму главни мени. Учитан је улоговани корисник и листа његових одиграних партија.



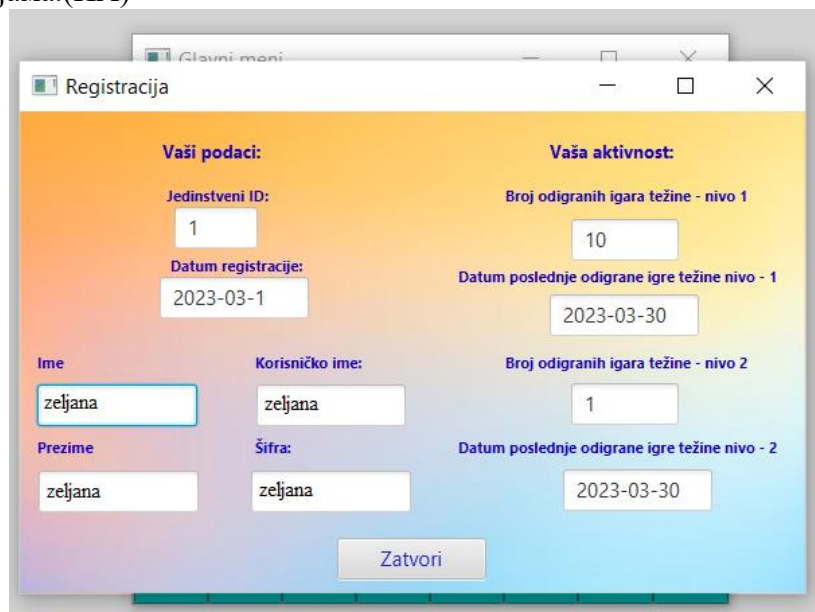
Слика 25 Форма главни мени

Основни сценарио СК

1. **Корисник** позива **систем** да прикаже форму са подацима о профилу и активностима одиграних партија.(АПСО)

Опис акције: Корисник кликом на опцију из падајућег менија „3. Мој профил > Мој профил“ позива системску операцију **prikaziMojProfil()**

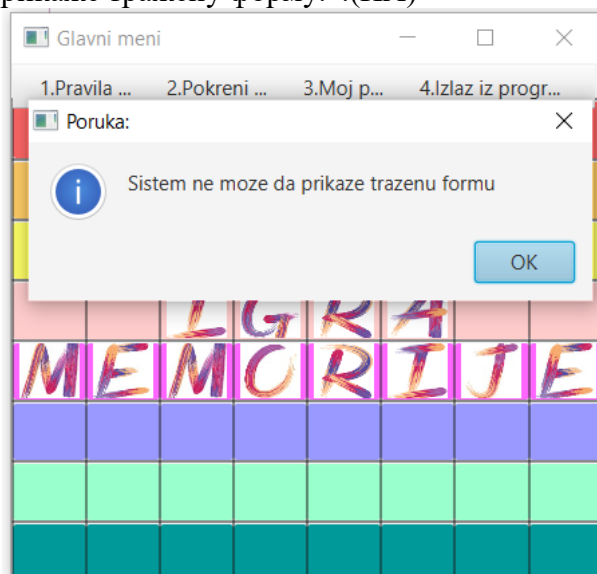
2. **Систем** проналази тражену форму.(СО)
3. **Систем** приказује **кориснику** форму са информацијама о профилу и одиграним партијама.(ИА)



Слика 26 Форма мој профил

Алтернативна сценарија

- 3.1 Уколико **систем** не може да прикаже тражену форму, он приказује **кориснику** поруку: “Систем не може да прикаже тражену форму.”.(ИА)



Слика 27 Неуспешни приказ форме правила игре

СК7: Случај коришћења- Излаз из програма

Назив СК

Излаз из програма

Актери СК

Корисник

Учесници СК

Корисник и **систем** (програм)

Предуслов: **Систем** је укључен и **корисник** је улогован под својом шифром. **Систем** приказује главни мени.



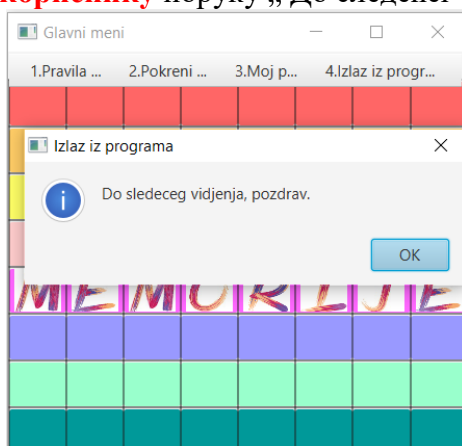
Слика 28 Форма главни мени

Основни сценарио СК

1. **Корисник** позива **систем** да изврши опцију излаз из програма.(АПСО)

Опис акције: Корисник кликом на опцију из падајућег менија „4. Излаз из програма“ позива системску операцију **izlazIzPrograma()**

2. **Систем** проналази тражену опцију.(СО)
3. **Систем** приказује **кориснику** поруку „ До следећег виђења, поздрав.“.(ИА)



Слика 29 Излазна порука

3.1.2 Пројектовање контролера корисничког интерфејса

Контролер корисничког интерфејса је одговоран да:

1. прихвати податке које шаље екранска форма
2. конвертује податке (који се налазе у графичким елементима) у објекат који представља улазни аргумент СО која ће бити позвана
3. шаље захтев за извршење системске операције до апликационог сервера (софтверског система)
4. прихвата објекат (излаз) софтверског система настао као резултат извршења системске операције
5. конвертује објекат у податке графичких елемената

3.2 Пројектовање апликационе логике

Апликациона логика описује структуру и понашање система. Апликациони сервер се састоји из:

1. **Контролера апликационе логике** – треба да подигне серверски сокет који ће да послушкује мрежу. Служи за комуникацију са клијентом и одговоран је да прихвати захтев за извршење системске операције од клијента и проследи га до пословне логике која је одговорна за извршење СО
2. **Пословна логика** – описана је структуром (доменске класе) и понашањем (системске операције)
3. **Брокер базе података** – служи за комуникацију између пословне логике и базе података

3.2.1 Контрорлер апликационе логике

Део за комуникацију подиже серверски сокет који послушкује мрежу. Када клијентски сокет успостави конекцију са серверским сокетом, тада сервер генерише нит која ће успоставити двосмерну комуникацију са клијентом.

Класа *GenerickiKontrolerServer* подиже серверски сокет који послушкује мрежу и чека клијента који ће се повезати и успоставити комуникацију. Након успостављања комуникације, за сваког клијента се генерише посебна нит *Klijent*, а затим се нит клијента чува у листи повезаних клијената. Класа *Klijent* која представља нит која је додељена једном клијенту, је одговорна за прихватање захтева од корисника. Корисник путем форме шаље захтев за извршење неке од системских операција до одговарајуће нити повезане са тим клијентом, та нит *Klijent* прима захтев и преусмерава га до класа које су одговорне за извршење захтеване системске операције на серверској страни. Након извршења системске операције, одговор се враћа до конкретне нити *Klijent* која даље тај одговор враћа назад кориснику на клијентској страни. Комуникација између клијента и сервера се обавља разменом објекта *GenerickiTransferObjekat*, који се шаље путем сокета.

Форма одговорна за регистрацију комуницира са сервером путем сокета, док форма одговорна за пријаву корисника комуницира са сервером путем Web сервиса који доста поједностављује начин успостављања конекције између клијента и сервера.

3.2.2 Пословна логика

3.2.2.1 Пројектовање понашања софтверског система (системске операције)

За сваки од претходно дефинисаних уговора правимо системску операцију, што заправо представља пројектовање понашања. Класа *OpstelZvršenjeSO* представља апстрактну класу која садржи методу *opstelZvršenjeSO()*, која представља шаблон извршавања сваке операције над базом података. Класа као атрибут има *BrokerBazePodataka* преко кога је могуће позивати операције читања, уписивања, брисања и ажурирања базе података, и методу *opstelZvršenjeSO()* путем које се успоставља конекција са базом података, позива специфична *izvršiSO()* метода, уколико није нарушен интегритет базе позивају се методе *commitTransaction()/rollbackTransaction()*, а на крају се затвара конекција са базом.

```
public abstract class OpstelZvršenjeSO {

    public BrokerBazePodataka bbp = new BrokerBazePodatakaImpl();

    GeneralDObject gdo;

    public boolean opstelZvršenjeSO() {

        bbp.makeConnection();

        boolean signal = izvršiSO();

        if (signal == true) {

            bbp.commitTransation();

        } else {

            bbp.rollbackTransation();

        }

        bbp.closeConnection();
        return signal;
    }

    abstract public boolean izvršiSO();
}
```

Свака системска операција даје своју конкретну имплементацију апстрактне методе *izvršiSO()*. За сваку системску операцију треба направити концептуална решења која су директно повезана са логиком проблема.

За сваки уговор пројектује се концептуално решење.

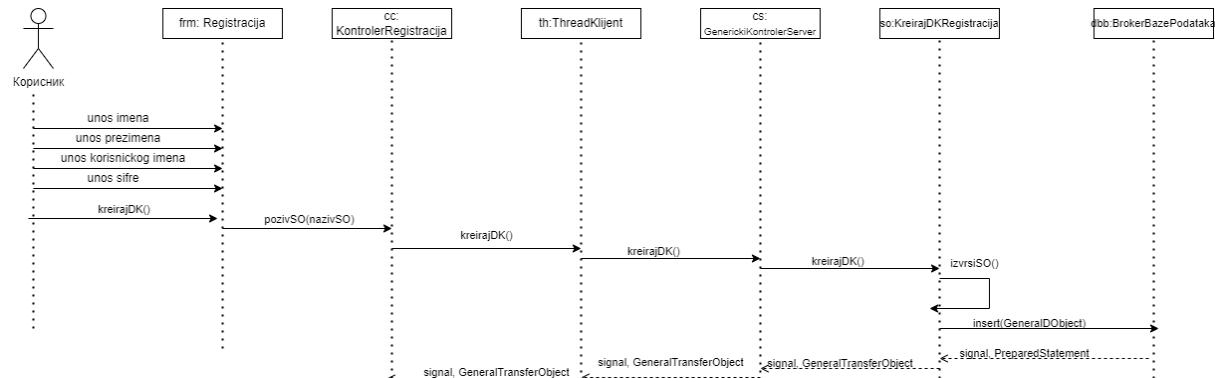
Уговор UG1: kreirajDK

Операција: **kreirajDK():signal;**

Веза са СК: CK1

Предуслови: Вредносна и структурна ограничења над објектом Корисник морају бити задовољена.

Постуслови: Креиран је нови корисник.



Слика 30 Дијаграм секвенци за уговор **kreirajDK**

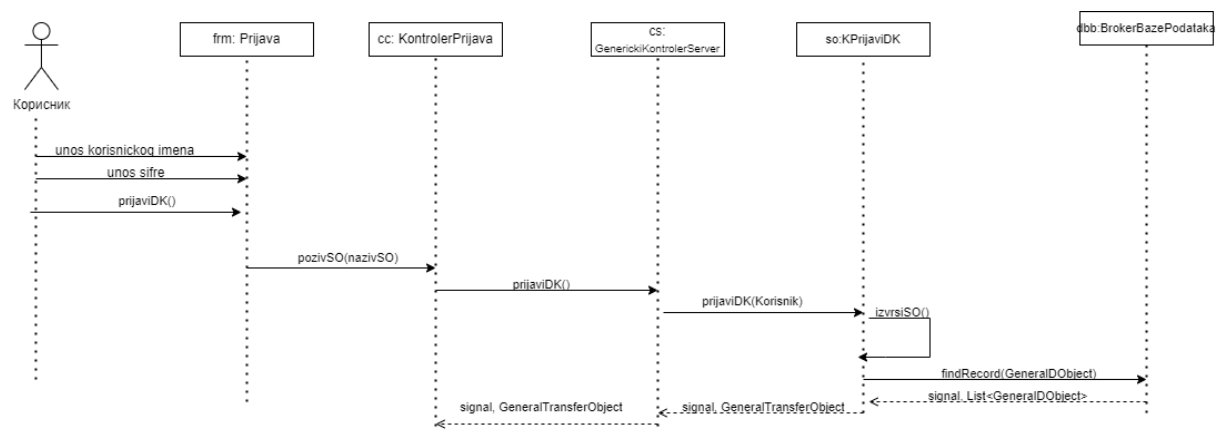
Уговор UG2: prijavidK

Операција: **prijaviDK(): signal;**

Веза са СК: CK2

Предуслови: /

Постуслови: /



Слика 31 Дијаграм секвенци за уговор **prijavidK**

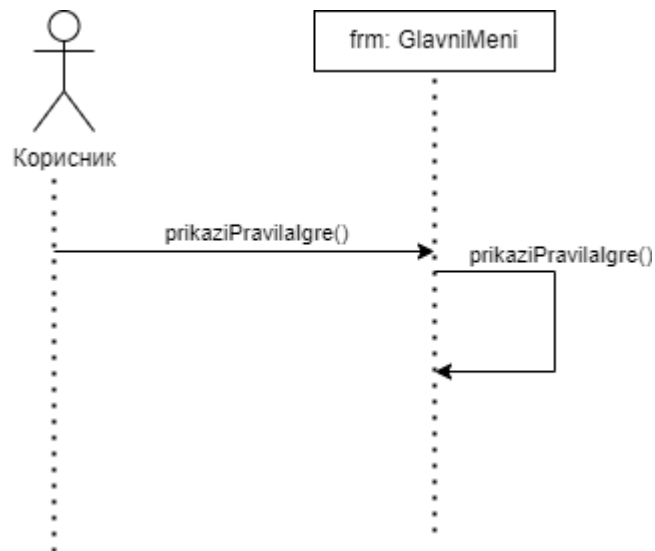
Уговор UG3: prikaziPravilaIgre

Операција: `prikaziPravilaIgre()`: signal;

Веза са СК: СК3

Предуслови: /

Постуслови: /



Слика 32 Дијаграм секвенци за уговор prikaziPravilaIgre

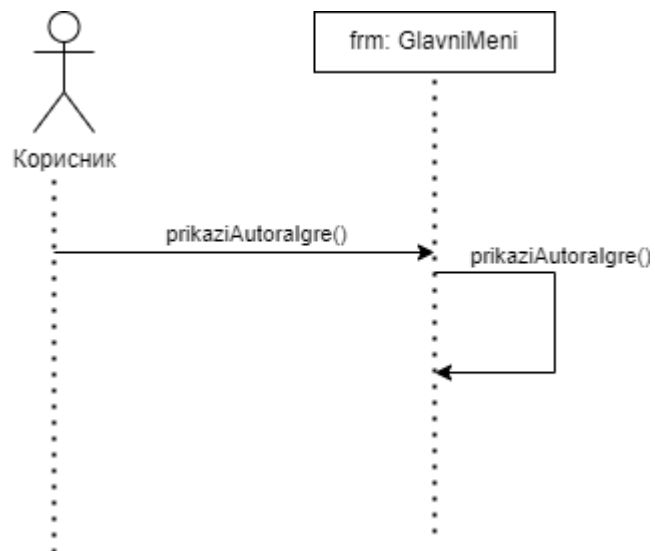
Уговор UG4: prikaziAutorialgre

Операција: `prikaziAutorialgre()`: signal;

Веза са СК: СК4

Предуслови: /

Постуслови: /



Слика 33 Дијаграм секвенци за уговор prikaziAutorialgre

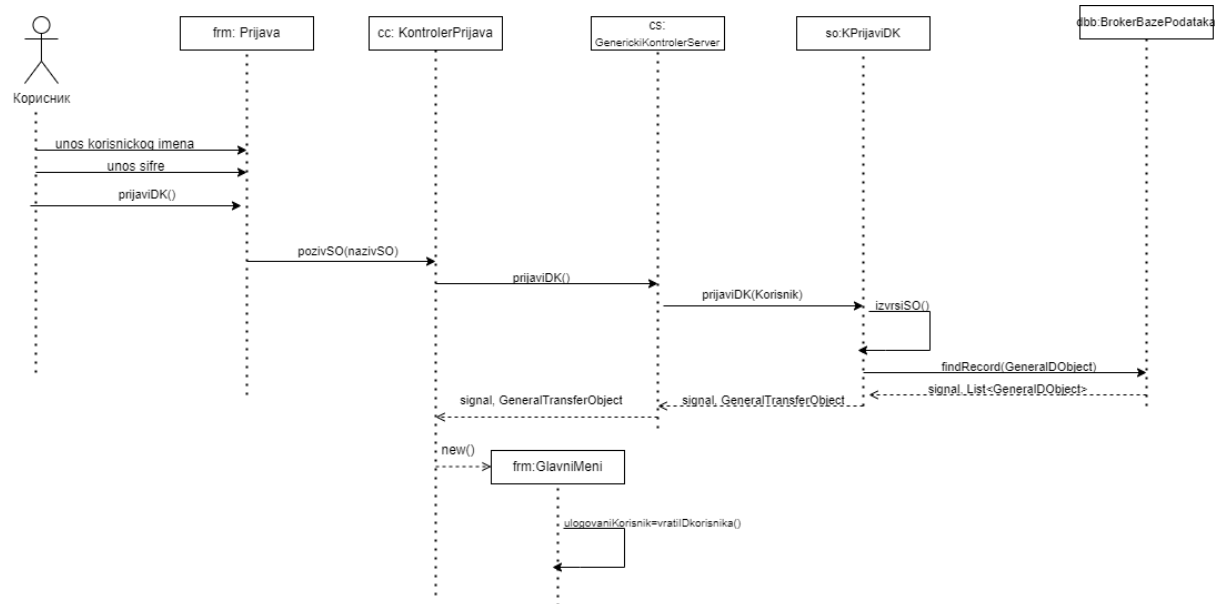
Уговор UG5: vratiIDKorisnika

Операција: `vratiIDKorisnika()`: signal;

Вежа са СК: СК5

Предуслови: /

Постуслови: /



Слика 34 Дијаграм секвенци за уговор vratiIDKorisnika.

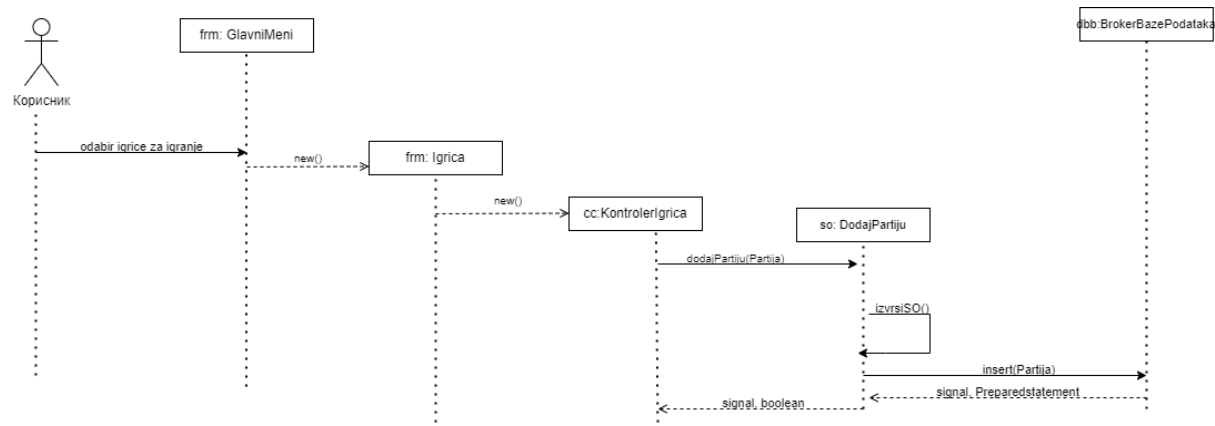
Уговор UG6: dodajPartiju

Операција: `dodajPartiju(Partija)`: signal;

Вежа са СК: СК5

Предуслови: Вредносна и структурна ограничења над објектом Партија морају бити задовољена.

Постуслови: Креирана је нова партија.



Слика 35 Дијаграм секвенци за уговор dodajPartiju

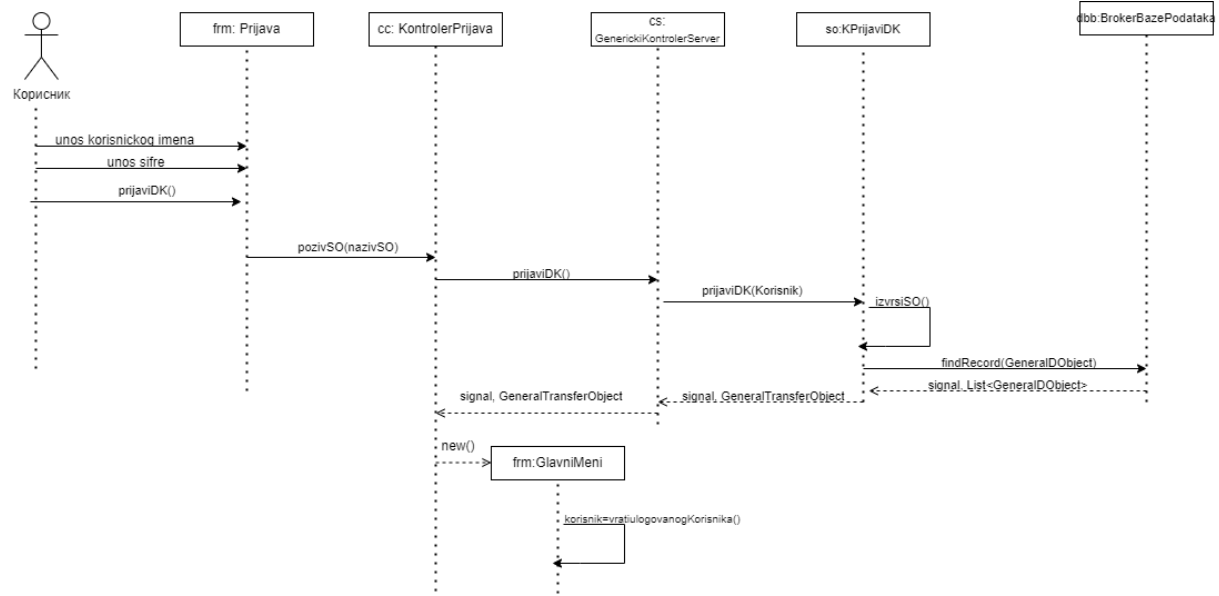
Уговор UG7: vratiUlogovanogKorisnika

Операција: vratiUlogovanogKorisnika(): signal;

Вежа са СК: СК6

Предуслови: /

Постуслови: /



Слика 36 Дијаграм секвенци за уговор vratiUlogovanogKorisnika

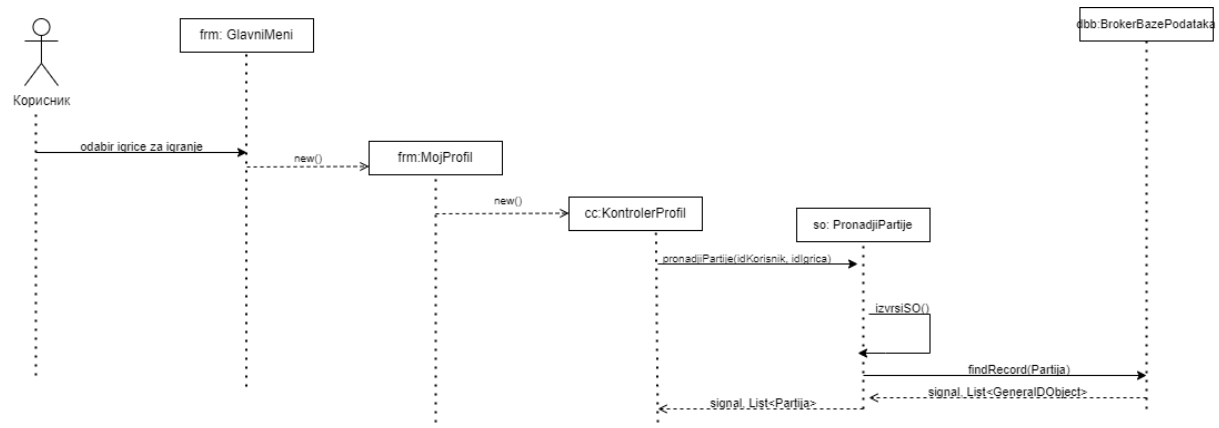
Уговор UG8: pronadjiPartije

Операција: pronadjiPartije(Long idKorisnik, int idIgrica): signal;

Вежа са СК: СК6

Предуслови: /

Постуслови: /



Слика 37 Дијаграм секвенци за уговор pronadjiPartije

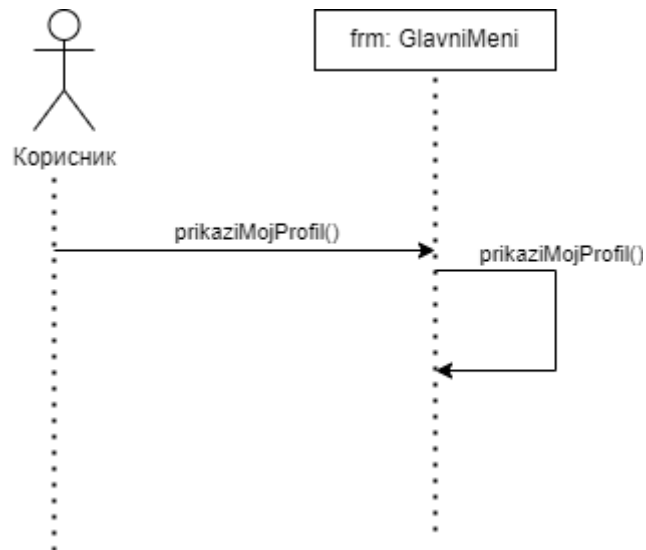
Уговор UG9: prikaziMojProfil

Операција: **prikaziMojProfil()**: signal;

Веза са СК: СК6

Предуслови: /

Постуслови: /



Слика 38 Дијаграм секвенци за уговор prikaziMojProfil

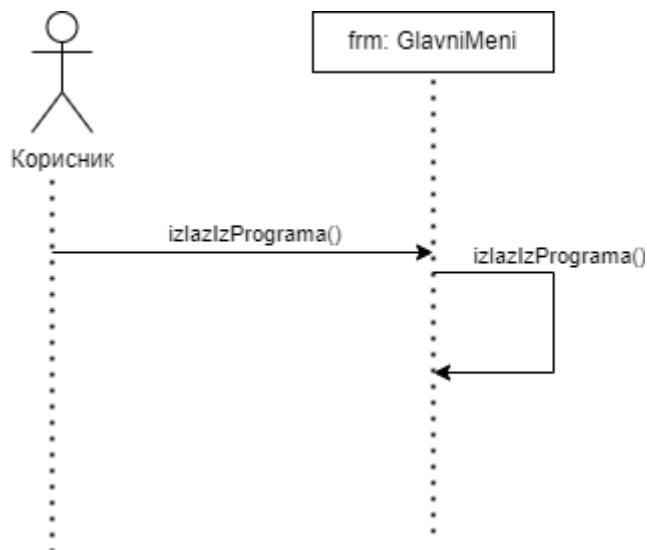
Уговор UG10: izlazIzPrograma

Операција: **izlazIzPrograma()**: signal;

Веза са СК: СК7

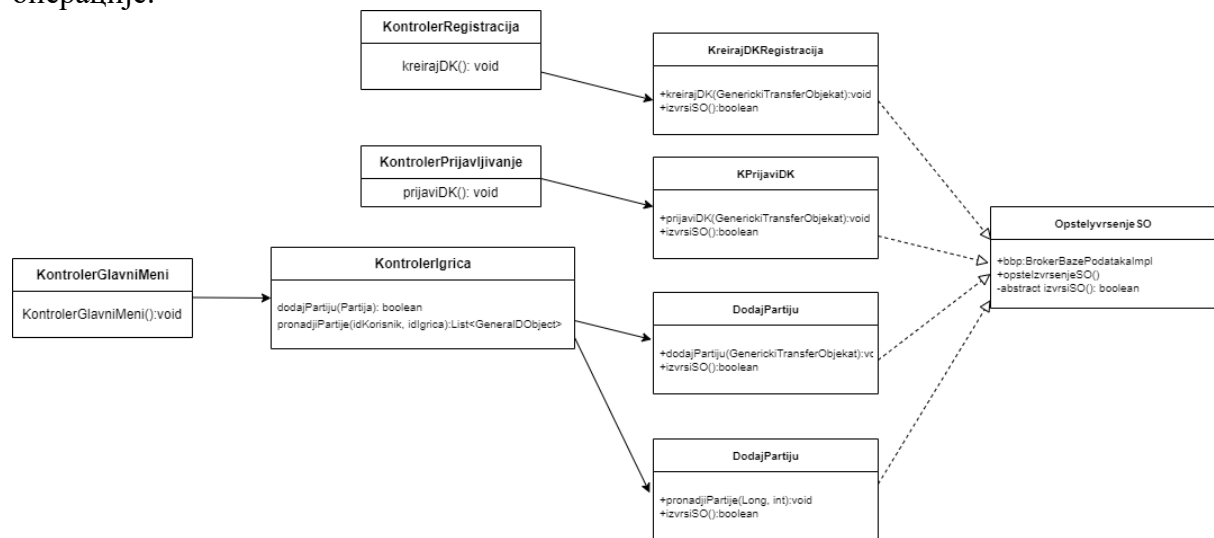
Предуслови: /

Постуслови: /



Слика 39 Дијаграм секвенци за уговор izlazIzPrograma

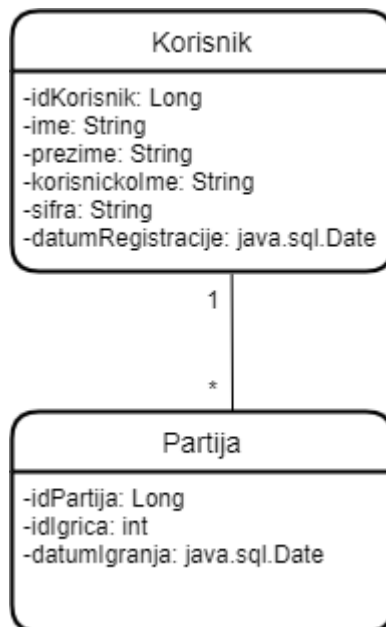
На следећој слици је приказан однос контролера са класама које реализују системске операције.



Слика 40 Однос контролера и класа које реализују СО

3.2.2.2 Пројектовање структуре софтверског система (доменске класе)

На основу концептуалних класа креирају се софтверске класе структуре.



Слика 41 Концептуални дијаграм класа

Како би било могуће реализовати операције читања, уписивања, брисања и ажурирања базе података, свака класа мора наследити апстрактну класу *GeneralDObject* и имплементирати њене методе.

```
public abstract class GeneralDObject 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.GeneralDObject getNewRecord(ResultSet rs)
    throws SQLException;

    public abstract String columnsForInsert();
}
```

```

public class Korisnik extends GeneralDObject implements Serializable {

    Long idKorisnik;
    String korisnickoIme;
    String sifra;
    String ime;
    String prezime;
    java.sql.Date datumRegistracije;

    public Korisnik(Long idKorisnik, String korisnickoIme, String sifra, String ime, String
    prezime, java.sql.Date datumRegistracije) {

        this.idKorisnik = idKorisnik;
        this.korisnickoIme = korisnickoIme;
        this.sifra = sifra;
        this.ime = ime;
        this.prezime = prezime;
        this.datumRegistracije = datumRegistracije;

    }
    /.../

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

    @Override
    public String getAtrValue2() {
        return "" + korisnickoIme + ", " + sifra + ", " + ime + ", " + prezime + ", " +
        datumRegistracije + "";
    }

    @Override
    public String setAtrValue() {
        return "idKorisnik=" + idKorisnik + ", " + "korisnickoIme=" + korisnickoIme + ",
        " + "sifra=" + sifra + ", ime=" + ime + ", prezime=" + prezime + ", datumRodjenja="
        + datumRegistracije + "";
    }

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

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

```

```

@Override
public String getNameByColumn(int column) {
    String names[] = {"idKorisnik", "korisnickoIme", "sifra", "ime", "prezime",
"datumRegistracije"};
    return names[column];
}

@Override
public DomenskiObjekat.GeneralDObject getNewRecord(ResultSet rs) throws
SQLException {
    return new DomenskiObjekat.Korisnik(rs.getLong("idKorisnik"),
rs.getString("korisnickoIme"), rs.getString("sifra"), rs.getString("ime"),
rs.getString("prezime"), rs.getDate("datumRegistracije"));
}

@Override
public String columnsForInsert() {

    return " (korisnickoIme, sifra, ime, prezime, datumRegistracije) ";
}
}

```

```

public class Partija extends GeneralDObject {

```

```

    Long idPartija;
    Long idKorisnik;
    int idIgrica;
    java.sql.Date datumIgranja;

    public Partija(Long idPartija, Long idKorisnik, int idIgrica) {

        this.idPartija = idPartija;
        this.idKorisnik = idKorisnik;
        this.idIgrica = 0;
        SimpleDateFormat sm = new SimpleDateFormat("yyyy-MM-dd");
        this.datumIgranja = java.sql.Date.valueOf(sm.format(datumIgranja));
    }

```

```

/.../

```

```

@Override
public String getAtrValue() {

    return idPartija + ", " + idKorisnik + ", " + idIgrica + ", " + datumIgranja + "";
}

@Override
public String getAtrValue2() {

```



```

        return idKorisnik + ", " + idIgrica + ", " + datumIgranja + "";
    }

    @Override
    public String setAttrValue() {

        return "idPartija=" + idPartija + ", " + "idKorisnik=" + idKorisnik + ", idIgrica=" +
idIgrica + ", datumIgranja=" + datumIgranja + "";

    }

    @Override
    public String getClassName() {

        return "Partija";
    }

    @Override
    public String getWhereCondition() {

        // return "idKorisnik=" + idKorisnik; mada cu ja pretrazivati po idkorisnik i id igra
        return "idPartija=" + idPartija;

    }

    @Override
    public String getNameByColumn(int column) {
        String names[] = {"idPartija", "idKorisnik", "idIgrica", "datumIgranja"};
        return names[column];
    }

    @Override
    public GeneralDBObject getNewRecord(ResultSet rs) throws SQLException {
        return new DomenskiObjekat.Partija(rs.getLong("idPartija"),
rs.getLong("idKorisnik"), rs.getInt("idIgrica"), rs.getDate("datumIgranja"));
    }

    @Override
    public String columnsForInsert() {
        return " (idKorisnik, idIgrica, datumIgranja) ";
    }
}

```

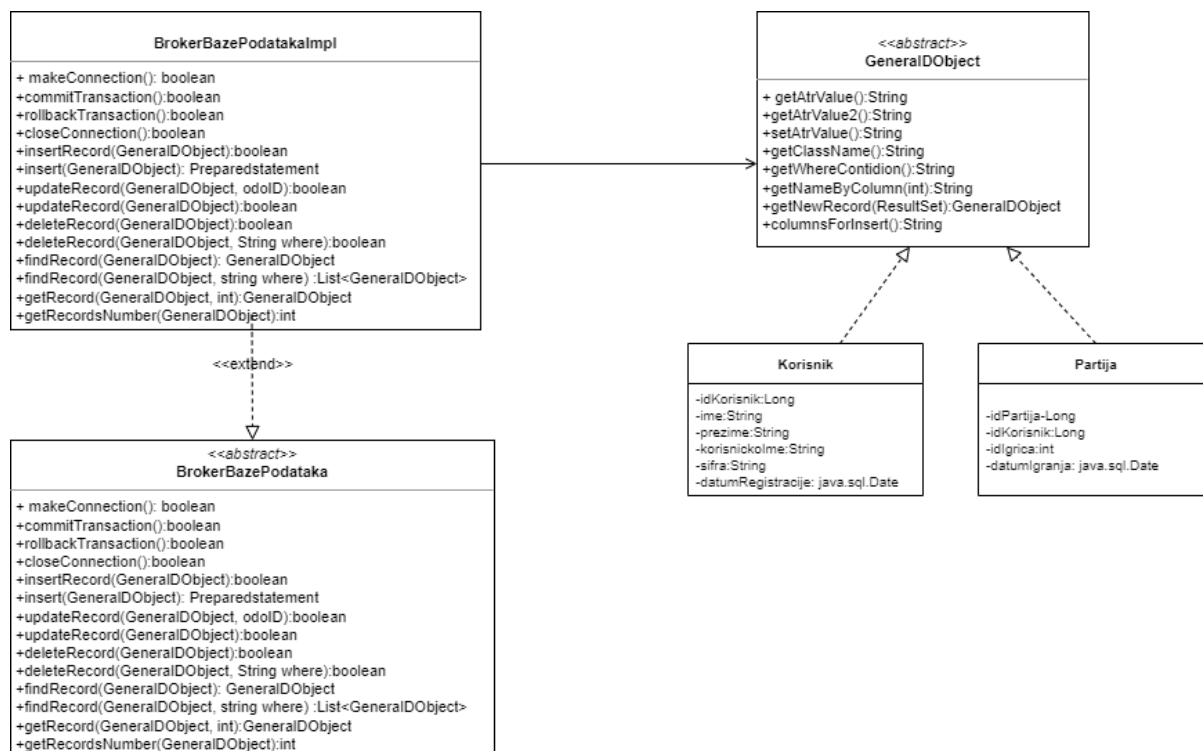
3.2.2.3 Брокер базе података

Класа *BrokerBazePodataka* пројектује се са циљем да обезбеди презистентни сервис доменским објектима које је потребно сачувати у бази података. Према томе класа *BrokerBazePodataka* представља перзистентни оквир који посредује у свим операцијама над базом података, а реализована је на следећи начин.

```
public abstract class BrokerBazePodataka {  
  
    public abstract boolean makeConnection();  
  
    public abstract boolean insertRecord(GeneralDObject odo);  
  
    public abstract boolean updateRecord(GeneralDObject odo, GeneralDObject odoold);  
  
    public abstract boolean updateRecord(GeneralDObject odo);  
  
    public abstract boolean deleteRecord(GeneralDObject odo);  
  
    public abstract boolean deleteRecords(GeneralDObject odo, String where);  
  
    public abstract GeneralDObject findRecord(GeneralDObject odo);  
  
    public abstract List<GeneralDObject> findRecord(GeneralDObject odo, String where);  
  
    public abstract boolean commitTransation();  
  
    public abstract boolean rollbackTransation();  
  
    public abstract void closeConnection();  
  
    public abstract GeneralDObject getRecord(GeneralDObject odo, int index);  
  
    public abstract int getRecordsNumber(GeneralDObject odo);  
  
    public abstract PreparedStatement insert(GeneralDObject gdo) throws SQLException;  
}
```

Класа *BrokerBazePodataka* је апстрактна класа, а све њене методе су пројектоване као генеричке, што значи да ће моћи да прихвате различите доменске објекте који наследе класи *GeneralDObject*. На овај начин избегавамо да имплементирамо појединачне методе које би биле везане за сваку конкретну класу, и тако умножавали један исти код.

Класа *BrokerBazePodatakaImpl* наслеђује абстрактну класу *BrokerBazePodataka* и даје конкретну реализацију горе поменутих метода.



Слика 42 Однос доменских класа са класом *BrokerBazePodatakaImpl*

3.2.3 Пројектовање складишта података

На основу релационог модела и ограничења пројектоване су табеле базе података које користи наш софтверски систем:

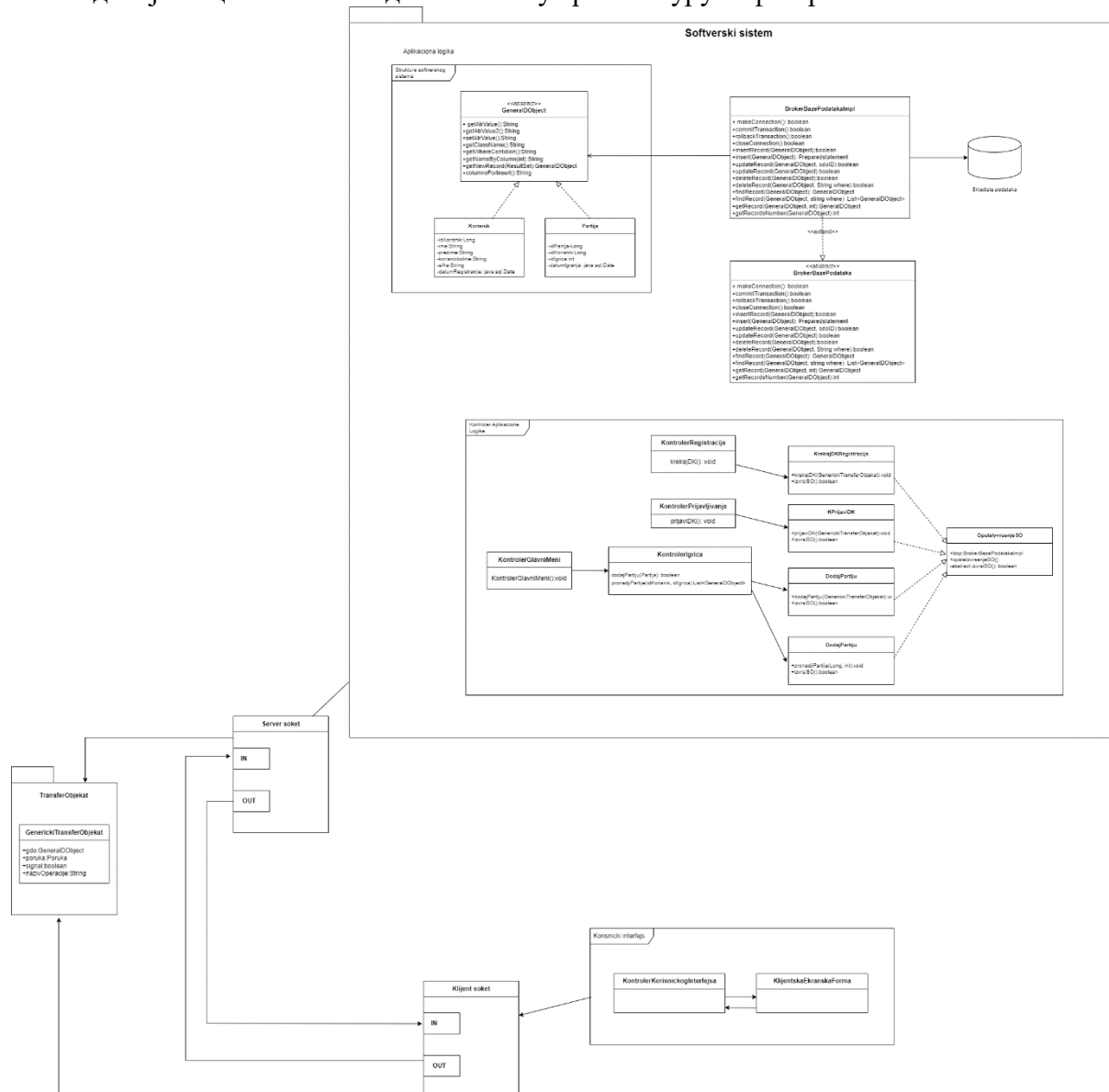
<input type="checkbox"/>	Column Name	Data Type	Length	Default	PK?	Not Null?	Unsigned?	Auto Incr?
<input type="checkbox"/>	idKorisnik	int	11		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	korisnickoIme	varchar	15		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	sifra	varchar	10		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	ime	varchar	20		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	prezime	varchar	30		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	datumRegistracije	date			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Слика 43 Табела Корисник

<input type="checkbox"/>	Column Name	Data Type	Length	Default	PK?	Not Null?	Unsigned?	Auto Incr?
<input type="checkbox"/>	idPartija	int	11		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	idKorisnik	int	11		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	idIgrica	int	11		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	datumIgranja	date			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>					<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Слика 44 Табела Партија

На следећој слици можемо видети коначну архитектуру софтверског система

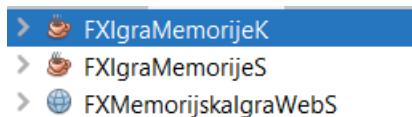


Слика 45 Коначна архитектура софтверског система

4. Фаза имплементације

Софтверски систем је развијан у програмском језику “Java”, развојно окружење NetBeans. Као систем за управљање базом података коришћен је MySQL.

Организација пројеката је приказана на следећој слици.



Слика 46 Организација пројекта

Пројекат *FXIgraMemorijeS* садржи нити за покретање комуникације са клијентом, генерички контролер сервер, брокер базе података и његову имплементацију, *opstelzvrjenjeSO* и системску операцију *KreirajDKRegistracija* задужену за регистровање нових корисника, доменски објекат *GeneralDObject* као и класу *Korisnik* која га реализује, као и трансфер објекат који се шаље назад ка клијенту.

Пројекат *FXMemorijskalgraWebS* садржи генерички контролер сервер одговоран за комуникацију са клијентом, брокер базе података и његову имплементацију, *opstelzvrjenjeSO* и системску операцију *KPrijavaDK* задужену за пријаву корисника, доменски објекат *GeneralDObject* као и класу *Korisnik* која га реализује, као и трансфер објекат који се шаље назад ка клијенту.

Пројекат *FXIgraMemorijeK* садржи форме које се приказују кориснику, укључујући и логику игрице, сокет који служи за комуникацију са сервером као и потребан пакет за успостављање комуникације са Web сервисом.

5. Тестирање

У фази тестирања, тестиран је сваки од имплементираних случајева коришћења. Приликом тестирања сваког случаја коришћења, поред унетих правилних података, уношени су и неправилни подаци да би се утврдио резултат извршења. Тестирање је спроведено на најједноставнији начин, ручним покретањем, уношењем података, и бележењем одговора система. Приликом тестирања није коришћен аутоматизовани приступ.

6. Принципи методе и стратегије пројектовања софтвера

Развој софтверског систем „ИГРА МЕМОРИЈЕ“ је у овом раду описан путем упрошћене Ларманове методе, а детаљи сваке фазе ове методе:

1. Прикупљање захтева од корисника
2. Анализа
3. Пројектовање
4. Имплементација
5. Тестирање

су разрађене у претходним, истоименим, поглављима овог документа.

6.1 Принципи (технике) пројектовања софтвера

Апстракција

Апстракција подразумева свесно издвајање општих информација како би се истакла суштина појаве, док се специфичне информације избегавају како би се избегли „неважни“ детаљи. Према наведеном и коришћена Ларманова метода је базирана на концепту апстракције. Преласком у сваку наредну фазу развоја све се више разрађује и улази у више детаља претходне фазе. На пример један случај коришћења из фазе прикупљања захтева је детаљније разрађен пројектовањем екранских форми и детаљним описивањем сваке од акција сценарија у фази пројектовања.

У контексту пројектовања софтвера постоје два кључна механизма апстракције: параметризација и спецификација.

Апстракција спецификацијом води до три главне врсте апстракција: процедурална апстракција, апстракција података, апстракција контролом.

Параметризација

Параметризација је апстракција која из неког скупа елемената, издваја њихова општа својства која су представљена преко параметара. Разликујемо пет случаја параметризације:

1. Параметризација скупа елемената простог типа
2. Параметризација скупа елемената сложеног типа
3. Параметризација скупа операција
4. Параметризација скупа процедура
5. Параметризација скупа наредби

На следећим примерима ћемо објаснити сваку од њих.

Пример параметризације скупа елемената простог типа би био случај када скуп целих бројева 1,2,3...10,11,... представимо преко општег представника x , односно у програмерском запису то би било **int x** .

Пример параметризације скупа сложеног типа би био случај када скуп корисника нашег система (1, Петар, Петровић, Пера, пера), (2, Ана, Анић, Анчи, ана), (3, Јован, Јовић, јоле, јоле)... параметризујемо, и представимо их преко њихових општих својстава (атрибута): ИД, Име, Презиме, Корисничко име, шифра. Параметризација се ради за сваки скуп вредности атрибута. Скуп ИД-јева (1, 2, 3) се параметризује преко својства ИД, скуп имена (Петар, Ана, Јован) и презимена (Перић, Анчић, Јовић) се параметризује преко својстава Име и Презиме, а скуп корисничких имена (Пера, Анчи, јоле) и шифри (пера, ана, јоле) се параметризује преко својстава Корисничко име и Шифра.

Параметризацијом долазимо до општих својстава елемената скупа. Навођење општих својстава елемената скупа представља спецификацију скупа.

Апстракција података је дефинисана именом и спецификацијом скупа: Корисник (ИД, име, презиме, корисничко име, шифра).

Из овога се закључује да параметризација претходи спецификацији, јер спецификација користи добијене резултате параметризације.

Пример параметризације скупа операција би био када бисмо скуп операција (5+5), (7+3), (9+1).. представили општом операцијом $x+y$, где су x и y операнди над којима се изводи операција, док оператор $+$ указује на то шта операција ради.

Пример параметризације скупа процедура:

Ако посматрамо скуп процедура: List findAll(Result Set) {метода која враћа листу типа Корисник}, List findAll(Result Set) {метода која враћа листу типа Партије} параметризацијом добијамо општу процедуру: List findAll(Result Set). Елементи неке процедуре су: назив процедуре, улазни аргументи, повратна вредност и тело процедуре. Параметризацијом долазимо до општих својстава процедура, њиховим навођењем долазимо до потписа процедура. Потпис процедуре је једна од главних делова спецификације процедуре (процедурална апстракција).

Пример параметризације скупа наредби би био:

Ако посматрамо скуп наредби: System.out.println("Korisnicko ime" +korisnik[1].getKorisnickoIme()) , System.out.println("Korisnicko ime" +korisnik[2].getKorisnickoIme())... параметризацијом добијамо општу наредбу: System.out.println("Korisnicko ime" +korisnik[i].getKorisnickoIme()).

Параметризацијом долазимо до општих својстава наредби, њиховим навођењем долази се до спецификације наредбе која постаје општа наредба. Наведена спецификација је заправо апстракција наредбе (апстракција контролом).

Спецификација

Спецификација је апстракција која из неког скупа елемената, издваја њихова општа својства која могу бити представљена преко процедуре, податка или контроле.

Процедуралном апстракцијом се издваја из неког скупа процедура оно што су њихова општа својства:

1. Тип онога што враћа процедура
2. Име процедуре
3. Аргументи процедуре

Процедуралном апстракцијом се добија потпис процедуре. Поред потписа, процедурална апстракција обухвата и допунске секције које детаљно описују шта ради процедура и који су услови извршења процедуре.

Као пример процедуралне апстракције прикажаћемо уговор о системској операцији из фазе анализе, упрошћене Ларманове методе.

Уговор UG1: kreirajDK

Операција: kreirajDK():signal;

Веза са СК: СК1

Предуслови: *Вредносна и структурна ограничења над објектом Корисник морају бити задовољена.*

Постуслови: *Креиран је нови корисник.*

Апстракцијом података се из неког скупа података издваја оно што су њихова општа својства. На претходном примеру показано је да се посматрањем скупа Корисника (1, Петар, Петровић, Пера, пера), (2, Ана, Анић, Анчи, ана), (3, Јован, Јовић, јоле, јоле) параметризацијом добијају атрибути самих корисника, ИД, Име, Презиме, Корисничко име, Шифра. Навођење атрибута представља спецификацију тог скупа.

Апстракција података јесте дефинисана именом скупа (Корисник) и спецификацијом скупа (ИД, Име, Презиме, Корисничко име, Шифра).

Уколико елементи скупа имају и структуру и понашање тада се над њима може применити и процедурална и апстракција података. Као резултат наведених апстракција се добија класа, која се састоји из атрибута и процедура, који су учаурени у класи. Уколико елементи скупа имају само понашање тада се над њима ради процедурална апстракција која као резултат даје интерфејс.

Када смо код упрошћене Ларманове методе радили концептуални модел (структура) тада смо радили апстракцију података, а када смо радили индентификацију системских операција тада смо радили процедуралну апстракцију.

Постоје два облика апстракције контроле:

1. Апстракција наредби, којом се из неког скупа наредби, издваја оно што су њихова општа својства и представљају се преко контролне структуре и опште наредбе.
2. Апстракција структура података, којом се из неког скупа структура података, издваја оно што су њихова општа својства и представљају се преко итератора, који у општем смислу контролише кретање кроз структуру података.

Спојеност (coupling) и Кохезија (cohesion)

У развоју објектно-оријентисаног софтвера потребно је истовремено постићи два циља. треба изградити класе које имају високу кохезију (high cohesion), а да те класе буду слабо повезане (weak coupling).

Спојеност (coupling) се односи на везе између класа, док се кохезија (cohesion) односи на повезаност метода унутар једне класе.

Постизање високе кохезије унутар класе можемо видети на следећем примеру:

```
public abstract class OpsteIzvršenjeSO {

    public BrokerBazePodataka bbp = new BrokerBazePodatakaImpl();

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

    abstract public boolean izvršiSO();
}
```

Све методе унутар класе *OpsteIzvršenjeSO* су јако повезане и раде на заједничком циљу да обезбеде комуникацију са базом, а да се при томе не наруши конзистентно стање базе података.

Класе које су одговорне за реализацију системске операције и које наслеђују класу *OpsteIzvršenjeSO* нису међусобно повезане. На овај начин се постиже слабо купловање између класа, директна веза између ових класа не постоји, оне немају међусобну комуникацију (једина комуникација је посредна, преко базе података и понашања које је прописано у класи *OpsteIzvršenjeSO*). Постизање слабе спојености можемо видети на следећем примеру класе *KreirajDKRegistracija* која је задужена за реализацију системске операције регистрације корисника.

```
public class KreirajDKRegistracija extends OpsteIzvršenjeSO {

    GenerickiTransferObjekat gto;

    public void kreirajDK(GenerickiTransferObjekat gto) {
        this.gto = gto;
        opsteIzvršenjeSO();
    }

    @Override
    public boolean izvršiSO() {
        gto.setSignal(false);
        if (gto.getDK() == null) {
            gto.setPoruka("Poslati korisnik je null");
        }

        try {
            PreparedStatement ps = bbp.insert(gto.getDK());
        }
    }
}
```

```

ResultSet tableKeys = ps.getGeneratedKeys();
tableKeys.next();
Long korisnikID = tableKeys.getLong(1);

Korisnik sacuvaniKor = (Korisnik) gto.getDK();
sacuvaniKor.setIDKorisnika(korisnikID);
gto.setDK(sacuvaniKor);
gto.setSignal(true);

} catch (SQLException ex) {
    System.out.println("SO.KKreirajDK.izvrsiSO() nije uspesno izvorsena! " +
ex.getMessage());
    gto.setPoruka("Sistem ne moze da kreira korisnika na osnovu unetih podataka!");
}

return gto.getSignal();
}
}

```

Декомпозиција и модуларизација

Декомпозиција је, у најопштијем случају, процес којим се почетни проблем дели на скуп подпроблема који се независно решавају. Расчлањивање почетног проблема на мање подпроблеме олакшава решавање почетног проблема. У контексту развоја софтверског система декомпозицијом софтверски систем делимо на више модула. Из овога се може закључити да је модуларизација резултат декомпозиције. У нашем случају у примењеној упрошћеној Лармановој методи, са декомпозицијом се сусрећемо већ у првој фази прикупљања захтева од корисника, где почетни кориснички захтев декомпонујемо на више случаја коришћења: Регистрација, Пријава на систем, Преглед правила игре, Покретање игре и остали случајеви описани у првом поглављу. Затим ппримену декомпозиције поново можемо видети у фази пројектовања, где је архитектура софтверског система декомпонована у три модула: кориснички интерфејс, апликациона логика и складиште података. Сваки од добијених модула се може независно пројектовати и имплементирати. Добијене модуле треба пројектовати тако да они имају јаку кохезију, али да су што слабије међусобно повезани, и тако да сваки од модула сакрива своје интерне информације.

Са применом декомпозиције се сусрећемо и приликом писања функција. Једна функција се расчлањује на више подфункција које су одговорне за њену успешну реализацију. Па тако на примеру претходно приложене класе *KreirajDK* одговорне за извршење системске операције регистрације, позива се метода *opstelIzvrjenjeSO()* која је на једном вишем нивоу састављена од више подфункција, задужених за успостављање/прекид конекције са базом *makeConnection()/closeConnection()*, конкретне имплементације *izvrsiSO()*, а затим *commitTransaction()/rollbackTransaction()*. Свака од подфункција је задужена за извршење свог задатка, а опет све оне раде у циљу обезбеђивања главне системске операције регистрације. Функционалном декомпозицијом смо избегли проблем да нам се комплетан код налази у оквиру само једне велике функције, где би се измешали разни аспекти развоја софтвера.

Енкапсулација и сакривање информација

Енкапсулацији или учаурење јесте процес у коме се раздвајају особине модула нпр. класе које су јавне за друге модуле (класе) од особина које су сакривене од других модула (класа) система. Сакривање информације настаје као резултат процеса енкапсулације. На пример у класи *GUIKontrolerRegistracija* метода *kreirajDK()* јесте јавна за све остале класе, док је метода која се позива у оквиру ње *svePopunjeno()* приватна и доступна је само класи *GUIKontrolerRegistracija*. Учаурењем ми заправо вршимо апстракцију, јер приказујемо само суштинску методу *kreirajDK()* док са друге стране сакривамо све њене детаље и оно шта се све позива у оквиру ње. Овиме истичемо шта метода ради, а не како то она ради.

```
public void kreirajDK() {  
  
    if (svePopunjeno()) {  
  
        Korisnik kor = popuniKorisnika();  
  
        gto.setDK(kor);  
  
        pozivSO(nazivSOKreiraj());  
  
        poruka(gto.getPoruka().getPoruka());  
  
    } else {  
  
        poruka("Sva polja moraju biti popunjena!");  
  
    }  
  
}  
  
private boolean svePopunjeno() {  
  
    String korIme = this.fxcon.korisnickoIme.getText();  
    String sifra = this.fxcon.sifra.getText();  
    String ime = this.fxcon.ime.getText();  
    String prezime = this.fxcon.prezime.getText();  
  
    if (!korIme.isEmpty() && !sifra.isEmpty() && !ime.isEmpty() && !prezime.isEmpty())  
    {  
        return true;  
    }  
    return false;  
}
```

Одвајање интерфејса и имплементације

Интерфејс се одваја од имплементације и излаже се кориснику, који не зна како су операције интерфејса имплементирани. У нашем случају пример би била класа *BrokerBazePodatakaImpl* која представља корисника, док би класа *GeneralDObject* била интерфејс који излаже своје операције кориснику. *BrokerBazePodatakaImpl* не зна како су операције класе *GeneralDObject* имплементирани (пример у методи *deleteRecord(GeneralDObject)*), брокер базе података не зна како су имплементирани методе *getClassName()* и *getWhereCondition()*. За конкретну имплементацију метода *getClassName()* и *getWhereCondition()* су одговорне конкретне класе (Корисник, Партија) које наслеђују класу *GeneralDObject*. На овај начин смо постигли одвајање интерфејса од имплементације.

@Override

```
public boolean deleteRecord(GeneralDObject odo) {  
    String upit = "DELETE FROM " + odo.getClassName() + " WHERE " +  
odo.getWhereCondition();  
    return executeUpdate(upit);  
}
```

6.2 Стратегије пројектовања софтвера

Неке од познатих стратегије које ћемо размотрити јесу:

- 1 Подели и владај (Divide and conquer)
- 2 Приступ са врха на доле (Top down)
- 3 Приступ одоздо на горе (Bottom up)
- 4 Итеративно инкрементални приступ

Подели и владај (Divide and conquer)

Подели и победи стратегија заснована је на принципу декомпозиције која почетни проблем дели у скуп потпроблема који се независно решавају, а у циљу лакшег решавања почетног проблема. Можемо уочити да смо ову стратегију применили у следећим фазама упрошћене Ларманове методе:

1. Прикупљање захтева – Комплексан кориснички захтев је подељен и описан преко скупа независних случајева коришћења.
2. Анализа – Структуру смо описали преко скупа концепата који чине концептуални модел. Понашање смо описали преко скупа независних системских операција.
3. Пројектовање – Архитектура система подељена је у три нивоа: кориснички интерфејс, апликациона логика и складиште података. Кориснички интерфејс се даље може поделити на два дела: екранске форме и контролер КИ. Апликациона логика се даље може поделити у три дела: контролер апликационе логике, пословна логика и брокер базе података.

Пристап са врха на доле (Top down) и Пристап одоздо на горе (Bottom up)

Стратегија с врха на доле је заснована на принципу декомпозиције функција која почетну функцију декомпонује на више подфункција које се решавају независно, а њиховом поновном интеграцијом у целину се реализује почетна функција.

Стратегија одозго на горе је заснована на принципу генерализације, где у некој сложеној функцији уочавамо једну или више логичких целина које проглашавамо за подфункције. На тај начин је једна сложена функција декомпонована у више независних функција, а композицијом тих функција ћемо обезбедити исту функционалност као и сложена функција.

На пример функцију *opsteIzvršenjeSO()* смо поделили на више логичких подфункција, свака од њих је имплементирана на свој начин (дата је на пример конкретна имплементација једне од подфункција, *makeConnection()*), а када их интегришемо, све оне јесу у функцији *opsteIzvršenjeS()*.

```
public boolean opsteIzvršenjeSO() {
    bbp.makeConnection();
    boolean signal = izvrsiSO();
    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/igramemorije";
        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;
}
```

Итеративно инкрементални пристап

Систем се дели у више мини пројеката који се независно развијају и пролазе кроз све фазе развоја софтвера. Мини пројекат може бити везан за случај коришћења или за сложену системску операцију. Сваки мини пројекат пролази кроз више итерација, чији резултат, интерно издање (*release*) представља инкремент за систем. На крају се

имплементирани мини пројекти интегришу у софтверски систем. На Пример: У примеру развоја овог софтверског система имали смо неколико независних случајева коришћења из којих смо добили неколико независних системских операција. Развој појединачне системске операције није зависио од развоја било које друге системске операције, односно нека системска операција може бити имплементирана и интегрисана у систем док друга системска операција може тек бити идентификована у фази анализе.

6.3 Методе пројектовања софтвера

Неке од најважнијих метода пројектовања софтвера јесу:

- 1 Функционо оријентисано пројектовање
- 2 Објектно оријентисано пројектовање
- 3 Пројектовање засновано на структури података
- 4 Пројектовање засновано на компонентама

Метода пројектовања софтвера коришћена за развој овог софтвера јесте објектно оријентисано пројектовање (*Object - Oriented Design*).

Објектно оријентисано пројектовање је засновано на објектима (класама). Објекти могу да репрезентују и структуру (атрибуте) и понашање (методе) софтверског система. Код објектно оријентисаног пројектовања паралелно се развијају и структура и понашање система.

У овом примеру као резултат фазе анализе добија се логичка структура и понашање софтверског система. Структура је описана преко концептуалног модела, док је понашање описано преко дијаграма секвенци. Концептуални модел се састоји од скупа међусобно повезаних концепата (класа, односно њихових појављивања објеката), док секвенчни дијаграми описују интеракцију између објеката (актера и софтверског система). То значи да су у основи логичке структуре и понашања софтверског система објекти. Па тако добијамо име ове анализе, објектно оријентисана анализа.

Касније у фази пројектовања и имплементације, одређујемо класе које ће реализовати структуру (доменске класе) и понашање (класе које ће реализовати системске операције). То значи да су у основи физичке структуре и понашања софтверског система такође објекти, односно класе. Према томе овакво пројектовање добија назив објектно оријентисано пројектовање.

Принципи објекто-оријентисаног пројектовања

Постоје следећи принципи код објектно-оријентисаног пројектовања класа:

1. Принцип Отворено - Затворено
2. Принцип замене Барбаре Лисков
3. Принцип инверзије зависности
4. Принцип уметања зависности
5. Принцип одвајања интерфејса

Принцип Отворено – Затворено каже да модул треба да буде отворен за проширење али и затворен за модификацију. У нашем примеру класа *opstelIzvršenjeSO* је сама по себи затворена за промену на једном бишем нивоу апстракције, а она такође садржи и апстрактну методу *izvrsiSO()*, па самим тим је она и отворена за промену у свим осталим класама које је наслеђују и које на конкретан начин имплементирају методу *izvrsiSO()*, на пример класа *KreirajDKRegistracija* задужена за извршење системске операције.

Принцип замене Барбаре Лисков каже да подкласе треба да буду заменљиве надкласама. У нашем примеру можемо видети да су све доменске класе, Корисник и Партија, заменљиве надкласом *GeneralDObject* из које су изведене.

Принцип инверзије зависности каже да требамо зависити од апстракције а не од конкретизације. Модули највишег нивоа не треба да зависе од модула најнижег нивоа, већ оба треба да зависе искључиво од апстракције и омогућава нам да пројектујемо класе које су међу собом слабо повезане. Апстракција не треба да зависи од детаља, док детаљи треба да зависе од апстракције. На пример у класи *BrokerBazePodataka* и *BrokerBazePodatakaImpl* избегавамо да направимо директну повезаност са конкретним доменским класама, већ између њих унећемо апстракцију од које сви зависе, а то је класа *GeneralDObject*. *BrokerBazePodataka* тј. *BrokerBazePodatakaImpl* се са неком доменском класом повезује преко класе *GeneralDObject* коју реализује та конкретна класа.

Принцип уметања зависности каже да се зависност између две компоненте програма успостављају у време извршења програма преко неке треће компоненте. У нашем случају зависност између брокера базе података и неке доменске класе успоставља у време извршавања програма преко класе *GeneralDObject*.

Принцип одвајања интерфејса каже да је боље да постоји више специфичних интерфејса него један генерални интерфејс опште намене. Интерфејс се одваја од имплементације и излаже се кориснику који може да позива његове операције, али корисник неће знати како су операције интерфејса имплементирани. У нашем случају пример би била класа *BrokerBazePodatakaImpl* која представља корисника, док би класа *GeneralDObject* била интерфејс који излаже своје операције кориснику. *BrokerBazePodatakaImpl* не зна како су операције класе *GeneralDObject* имплементирани.

7. Примена патерна у пројектовању софтвера

Приликом пројектовања реалног софтверског система, као и било ког другог великог система, веома је битно унапред размишљати о одржавању. У пракси се често дешава да добар део софтверског кода може да се искористи више пута, и да на тај начин, употребом дела кода постојећег пројекта скратимо време и трошкове реализације новог пројекта. Проблеми који се јаве током развоја софтвера се не морају увек решавати од почетка, већ можемо користити решења и добре праксе које су успешно коришћене раније. Такође свако ново, добро решење које се нађе, треба се укључити и користити поново. Сходно овоме, у многим објектно оријентисаним системима се налази на исте узоре класа и објеката који комуницирају. Управо ти узорци решавају проблеме пројектовања и омогућавају флексибилност и поновну употребљивост објектно оријентисаног дизајна. Ти узорци помажу да се поново искористе делови успешних

пројеката, и тиме нови пројекат базирају на претходно приказаном добром искуству. Ови узорни називају се патерни. Патерни имају за циљ да нам помогну у одржавању и надоградњи софтверског система.

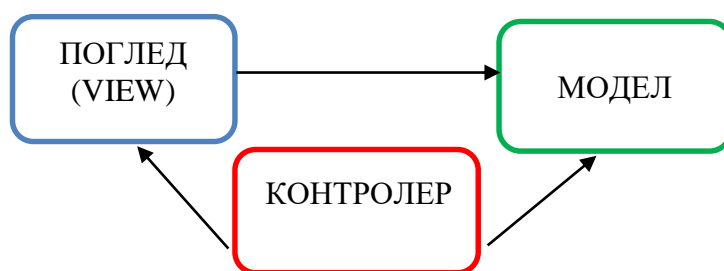
Макроархитектура описује структуру и организацију софтверског система на највишем нивоу. Макропатерни су EFC (Enterprise Component Framework) и MVC (Model View Controller) патерн који је коришћен у нашем случају.

MCV (Model – View – Controller)

Овај патерн посматра на софтверски систем кроз три дела:

1. Поглед (View) - обезбеђује кориснику интерфејс путем кога он уноси и/или позива одговарајуће операције које треба да се изврше над моделом. Поглед приказује кориснику стање модела. У нашем случају је то на пример форма за регистрацију корисника.
2. Контролер - ослушкује и прихвата захтев од клијента за извршење операције. Након тога позива операцију која је дефинисана у моделу. Уколико модел промени стање, контролер обавештава поглед да је стање промењено. У нашем случају би то на пример био *GUIKontrolerRegistracija* који са форме прихвата захтев за извршењем неке перације, након чега позива задужену операцију дефинисану у моделу (или захтев за перацијом делегира до серверске стране, која је обрађује и резултат враћа назад до овог контролера, а он затим тај резултат приказује клијенту путем форме).
3. Модел – представља стање система. Стање могу мењати неке од операција модела. Модел не мора да зна ко је поглед, а ко контролер, што и јесте суштина овог патерна.

Следећи графикон илуструје MVC патерн.



Опште о патернима

Патерни (узори) су готова решеља која се примењују у пројектовању софтвера.

„Сваки узор описује проблеме који се стано понавља у нашем окружењу и затим описује суштину решења проблема тако да се то решење може употребити милион пута, а да се два пута не понови исти начин.“ (Cristopher Alexander)

Сваки патерн се састоји од три дела: проблема који се решава, контекста у смислу одређених ограничења и решења проблема. У одговарајућем контексту, иста правила се могу променити да би се проблем решио у том конкретном случају. Патерни су развијени како би подржали изградњу великих, сложених система, како би се документовао сваки потенцијални проблем и његово решење, а затим стечено знање програмери у будућности поново користили и промењивали. Такође, патерни дефинишу

комуникацију између објеката и класа, прилагођену за решавање општих проблема у одређеном контексту. Имена патерна, апстракције и идентификација кључних аспеката заједничке структуре чине их корисним за креирање објектно оријентисаног дизајна који се може поново користити. Постоје три категорије патерна преко којих се реализује микроархитектура софтверског система: патерни за креирање објеката, структурни патерни и патерни понашања.

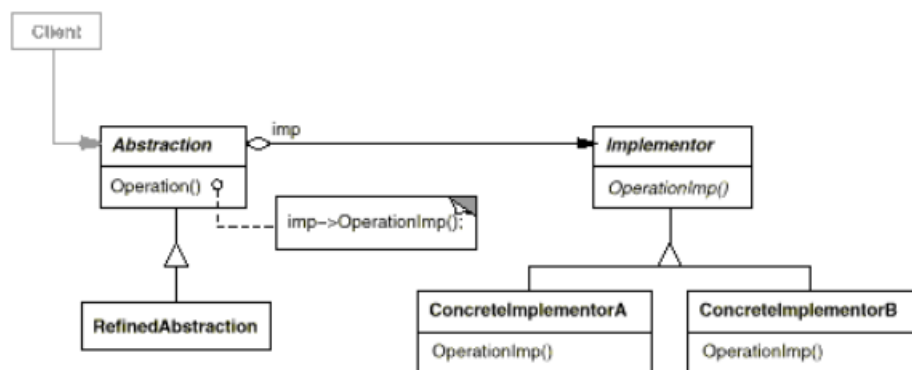
Патерни за креирање објеката апстрахују процес креирања објеката. Дају велику прилагодљивост у томе шта ће бити креирано, ко ће то креирати, како и када ће бити креирано. Они помажу да се изгради систем независно од тога како су објекти креирани, компоновани или репрезентовани.

Структурни патерни описују сложене структуре међусобно повезаних класа и објеката. Патерни понашања описују начин на који класе или објекти сарађују и распоређују одговорности.

Патерни као решење проблема пројектовања

Bridge патерн

Концепт овог структурног патерна јесте одвајање (декупловање) апстракције од њене имплементације тако да се оне могу мењати независно. На следећој слици је приказана структура *Bridge* патерна.



Слика 47 Структура Bridge патерна

У нашем случају класа *BrokerBazePodatakaImpl* је реализована помоћу овог патерна. *Abstraction* класи са слике би одговарала наша *BrokerBayePodatakaImpl* класа, која има референцу на *Implementor* класу, а у нашем случају би овој класи одговарала класа *GeneralDObject*. Класама *ConcreteImplementorA* и *ConcreteImplementorB* би одговарале конкретне класе Корисник и Партија које реализују класу *GeneralDObject*.

На примеру методе *deleteRecord(GeneralDObject)* извучене из класе *BrokerBazePodatakaImpl* можемо видети да на једном апстрактном нивоу ми треба да извршимо методу *deleteRecord* (која би на слици одговарала *Operation()* методи *Abstraction* класе), док се тек у време извршења програма одређује који ће тачно имплементор (у нашем случају или класа Корисник или класа Партија) бити задужен да реализује операције *getClassname()*, *getWhereCondition()*.

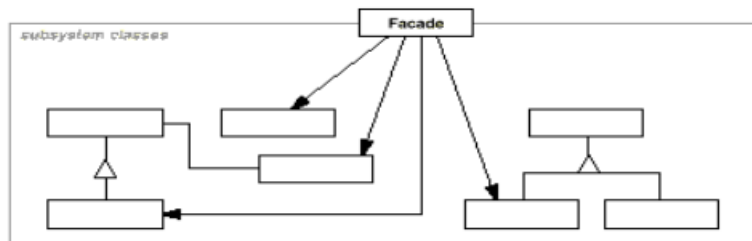
@Override

```

public boolean deleteRecord(GeneralDObject odo) {
    String upit = "DELETE FROM " + odo.getClassname() + " WHERE " +
odo.getWhereCondition();
    return executeUpdate(upit);
}
  
```

Facade патерн

Концепт овог структурног патерна јесте обезбеђивање јединственог интерфејса за скуп интерфејса неког подсистема. Facade патерн дефинише интерфејс високог нивоа који омогућава да се систем лакше користи. На следећој слици је приказана структура Facade патерна.



Слика 48 Структура Facade патерна

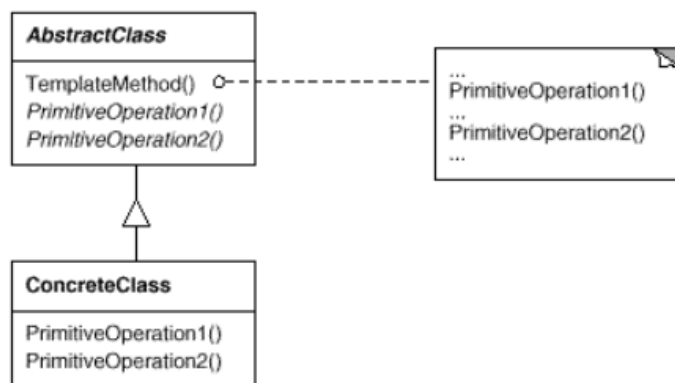
Како бисмо клијенту поједноставили коришћење софтверског система, ми за њега правимо једну улазну тачку, тачније фасаду, преко које он може једноставно да приступа и да позива одговарајуће операције, без потребе да зна шта је испод те фасадe. Задатак фасадe јесте да прихвати, а затим преусмери прихваћени захтев до подсистема или компонената задуженим за извршење тог захтева, према томе она тачно мора да зна која класа/компонента подсистема је одговорна за извршење ког захтева. У нашем случају фасада би била класа *GUIKontrolerGlavniMeni* која кориснику излаже све опције са главног менија и прихвата захтев за извршењем позване операције од корисника, али истовремено она од корисника заклања шта се све дешава када она тај захтев треба да преусмери до конкретне класе задужене за извршење позване операције (да ли ће захтев послати преко сокета, преко Web сервиса, до које класе задужене за извршење операције проследити захтев, помоћу којих технологија је ово реализовано).

Template method патерн

Концепт овог патерна понашања јесте дефинисање скелет алгоритма у операцији, препуштајући извршење неких корака операција подкласама. *Template method* омогућава подкласама да редефинишу неке од корака алгоритама без промене алгоритамске структуре.

Применом овог патерна избегавамо велики проблем вишеструког понављања једног истог кода специфицираног за конкретни проблем, и стварамо могућност да генерализован део кода можемо вишеструко употребљавати.

На следећој слици је приказана структура *Template method* патерна.



Слика 49 Структура Template method патерна

У нашем случају класа *opsteIzvršenjeSO* би одговарала *AbstractClass* са слике, док би класа задужена да реализује системску операцију, на пример *KreirajDKRegistracija* одговарала *ConcreteClass*. Метода *opsteIzvršenjeSO()* одговара *TemplateMethod()* са слике, и она прописује шаблонско понашање сваке класе која наследи класу *opsteIzvršenjeSO*, док би абстрактна метода *izvrsiSO()* одговарала методама *PrimitiveOperation1()* и/или *PrimitiveOperation2()* са слике које своју конкретну имплементацију добијају у конкретној класи (нпр. класа *KreirajDKRegistracija*). На овом примеру видимо да је *Template method* патерн директно повезан са претходно поменутиим принципом Отворено – Затворено.

```
public abstract class OpsteIzvršenjeSO {

    public BrokerBazePodataka bbp = new BrokerBazePodatakaImpl();

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

    abstract public boolean izvrsiSO();
}
```

8. Закључак

За развој софтверског система „ИГРА МЕМОРИЈЕ“, коришћена је поједностављена Ларманова метода за развој софтвера. Тренутно развијено софтверско решење јесте применљиво, али такође оставља пуно простора, да се коришћењем савремених технологија, побољшају и отклоне потенцијални недостаци, као и простора да се надограде нове функционалности које би задовољиле потребе корисника.

Коришћена литература

„СОФТВЕРСКИ ПРОЦЕС СКРИПТА- радни материјал, вер. 1.2“, др Синиша Влајић, Београд 2016

”ПРОЈЕКТОВАЊЕ СОФТВЕРА СКРИПТА- радни материјал, вер. 1.3”, др Синиша Влајић, Београд 2020