Универзитет у Београду

Факултет Организационих наука

Лабораторија за софтверско инжењерство

Семинарски рад из предмета

Софтверски процес

Тема: Икс-Окс игра

Професор:

др Синиша Влајић

Студент:

Александар Ристић 2020/3716

Београд, 2021.

Садржај

1 Кориснички захтеви

1.1 Вербални опис

Потребно је направити софтверски систем који ће омогућити играчима играње популарне друштвене игре „Икс-Окс“.

Како би играч могао да користи дате услуге неопходно је да се претходно региструје, уписивање корисничког имена и шифре.

Играч се приликом коришћења апликације прво пријављује на софтверски систем, уписивањем сопственог корисничког имена и шифре.

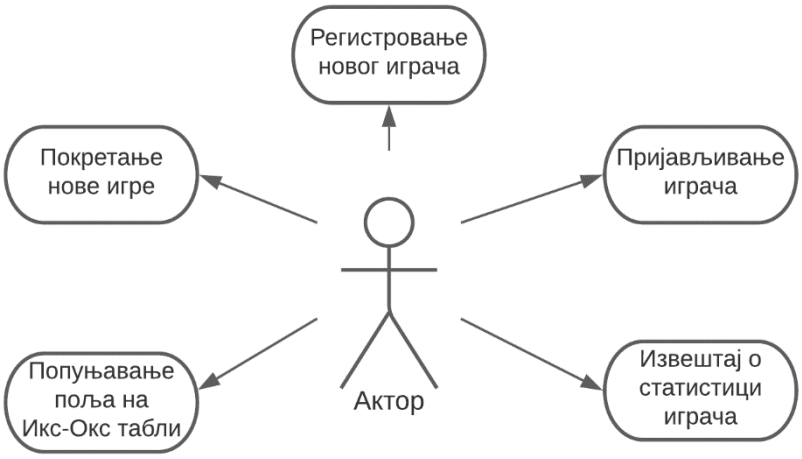
Софтверски ситем затим омогућава играчу приступ пољима на табли у које може да уписује X или O. Неопходно је да два играча буду пријављења на систем. Након завршене партије, играч има опцију за покретање нове игре.

Софтверски систем води евиденцију о играчима, уписивањем броја победа и пораза након сваке одигране партије.

1.2 Случајеви коришћења

У конкретном случају идентификовани су следећи случајеви коришћења који су приказани и на слици 1:

1. Регистровање новог играча
2. Пријављивање играча
3. Попуњавање поља на Икс-Окс табли
4. Извештај о статистици играча
5. Покретање нове игре



Слика 1. Дијаграм случајева коришћења

СК 1: Случај коришћења – Регистровање новог играча

Назив СК:
Регистровање новог играча

Актор:
Играч

Учесници СК:
Играч и систем (програм)

Предуслов: Систем је укључен и приказује форму за регистровање играча.

Основни сценарио СК:

1. **Играч** уноси податке за регистровање **играча**. (АПУСО)
2. **Играч** контролише да ли је исправно унео податке за регистровање. (АНСО)
3. **Играч** позива систем да запамти новог **играча** са унетим подацима. (АПСО)
4. **Систем** памти податке о новом **играчу**. (СО)
5. **Систем** приказује **играчу** поруку: „Регистровање успешно!“ . (ИА)

Алтернативна сценарија:

5.1. Уколико **играч** није попунио поља за регистровање, **систем** приказује **играчу** поруку: „Неисправно унети подаци! Поља за унос података не смеју бити празна.“. (ИА)

5.2. Уколико **играч** са унетим корисничким именом већ постоји у бази, **систем** приказује **играчу** поруку: „Играч са унетим корисничким именом већ постоји! Изаберите друго.“ (ИА)

СК 2: Случај коришћења – Пријављивање играча

Назив СК:
Пријављивање играча

Актор:
Играч

Учесници СК:
Играч и систем (програм)

Предуслов: Систем је укључен и приказује форму за пријављивање играча.

Основни сценарио СК:

1. **Играч** уноси податке за пријављивање **играча**. (АПУСО)
2. **Играч** контролише да ли је исправно унео податке за пријављивање. (АНСО)

3. **Играч** позива систем да пронађе **играча** са задатим подацима. (АПСО)
4. **Систем** претражује **играче**. (СО)
5. **Систем** приказује **играчу** поруку: „**Играч** је успешно улогован на систем!“ . (ИА)

Алтернативна сценарија:

5.1 Уколико **систем** не може да пронађе **играча**, приказује **играчу** поруку: „Погрешно корисничко име и/или шифра!“.

СК 3: Случај коришћења – Попуњавање поља на Икс-Окс табли

Назив СК:
Попуњавање поља на Икс-Окс табли

Актор:
Играч

Учесници СК:
Играч и систем (програм)

Предуслов: Систем је укључен и **играч** је пријављен са својом шифром. **Систем** приказује форму за унос поља на Икс-Окс табли.

Основни сценарио СК:

1. **Играч** бира поље на табли у које жели да унесе X или O знак. (АНСО)
2. **Играч** позива систем да попуни поље знаком. (АПСО)
3. **Систем** проверава да ли је играч на потезу. (СО)
4. **Систем** врши ажурирање Икс-Окс табле. (СО)
5. **Систем** проверава да ли постоји добитна комбинација. (СО)
6. **Систем** приказује **играчу** поруку: „Крај игре! Победник је: <<име играча>>“ (ИА)
7. **Систем** врши ажурирање статистике играча. (СО)

Алтернативна сценарија:

4.1. Уколико **играч** није на потезу, **систем** приказује **играчу** поруку: „Противник је на потезу!“. Прекида се извршење сценарија. (ИА)

5.1. Уколико је поље већ попуњено, **систем** приказује играчу поруку: „Поље је већ попуњено, изаберите друго!“. Прекида се извршење сценарија. (ИА)

6.1. Уколико не постоји добитна комбинација **систем** приказује ажурирани изглед Икс-Окс табле, са попуњеним пољем. Прекида се извршење сценарија. (ИА)

СК 4: Случај коришћења – Извештај о статистици играча

Назив СК:
Извештај о статистици играча

Актор:
Играч

Учесници СК:
Играч и систем (програм)

Предуслов: Систем је укључен и **играч** је пријављен са својом шифром.

Основни сценарио СК:

1. **Играч** позива систем да прикаже извештај о статистици. (АПСО)
2. **Систем** претражује статистику у бази на основу корисничког имена пријављеног играча. (СО)
3. **Систем** приказује **играчу** број остварених победа и пораза. (ИА)

СК 5: Случај коришћења – Покретање нове игре

Назив СК:
Покретање нове игре

Актор:
Играч

Учесници СК:
Играч и систем (програм)

Предуслов: Систем је укључен и **играч** је пријављен са својом шифром.

Основни сценарио СК:

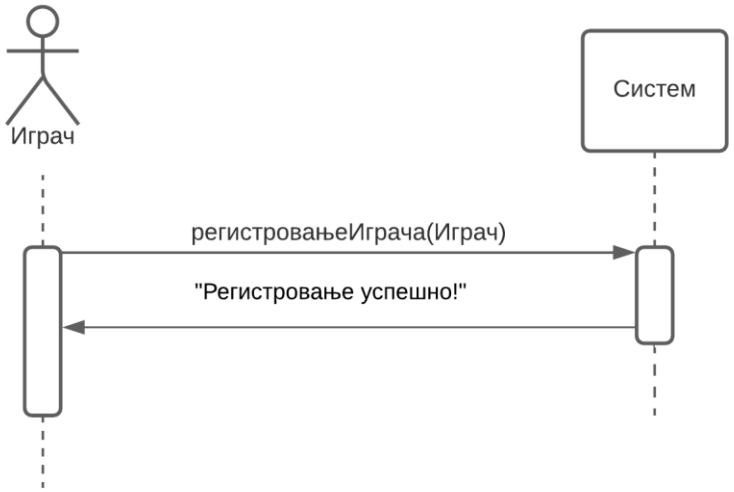
1. **Играч** позива систем да покрене нову игру. (АПСО)
2. **Систем** покреће нову игру. (СО)
3. **Систем** приказује **играчу** празну Икс-Окс таблу и поруку: „Нова игра креирана!“. (ИА)

2 Анализа

2.1 Понашање софтверског система – Системски дијаграм секвенци

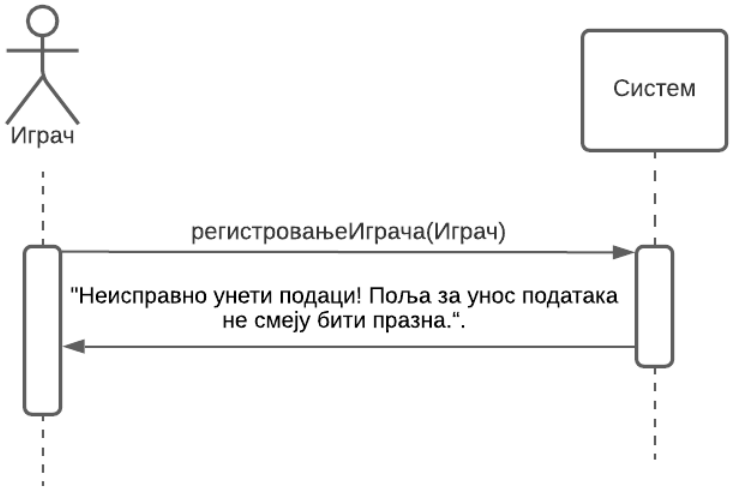
ДС 1: Дијаграм секвенци случаја коришћења – Регистровање новог играча

1. **Играч** позива **систем** да запамти новог играча са унетим подацима. (АПСО)
2. **Систем** приказује **играчу** поруку: „Регистровање успешно!“ . (ИА)

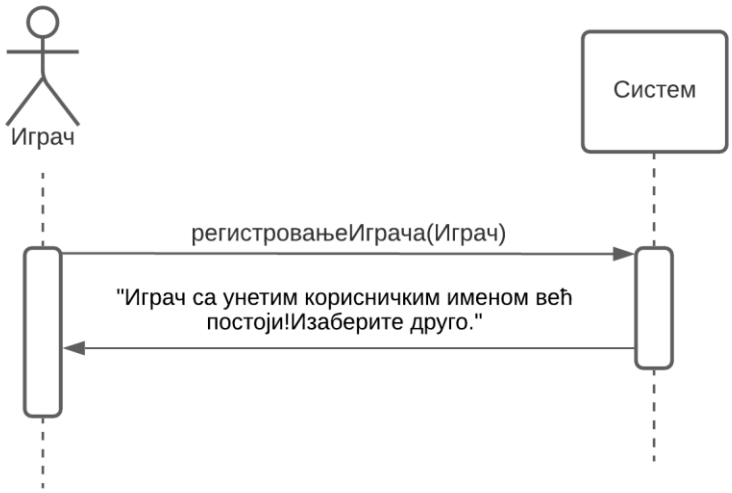


Алтернативни сценарио:

2.1. Уколико **играч** није попунио поља за регистровање, **систем** приказује **играчу** поруку: „Неисправно унети подаци! Поља за унос података не смеју бити празна.“. (ИА)



2.2. Уколико **играч** са унетим корисничким именом већ постоји у бази, **систем** приказује **играчу** поруку: „Играч са унетим корисничким именом већ постоји! Изаберите друго.“ (ИА)

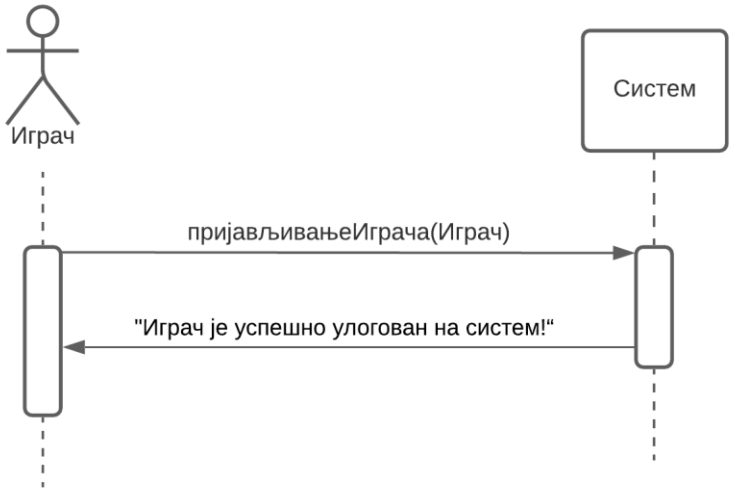


Са наведених секвенцних дијаграма уочава се 1 системска операција коју треба пројектовати:

1. *сигнал* регистровањеИграча(Играч)

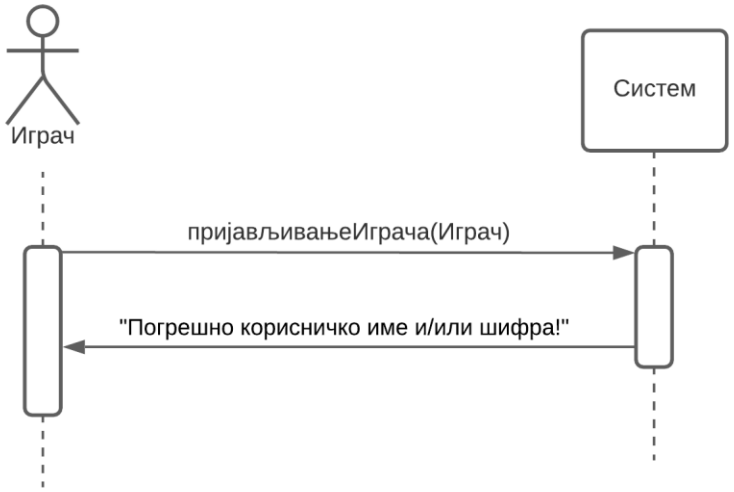
ДС 2: Дијаграм секвенци случаја коришћења - Пријављивање играча

1. **Играч** позива **систем** да запамти новог **играча** са унетим подацима. (АПСО)
2. **Систем** приказује **играчу** поруку: „**Играч** је успешно улогован на систем!“ . (ИА)



Алтернативни сценарио

2.1. Уколико **систем** не може да пронађе **играча**, приказује **играчу** поруку: „Погрешно корисничко име и/или шифра!“.

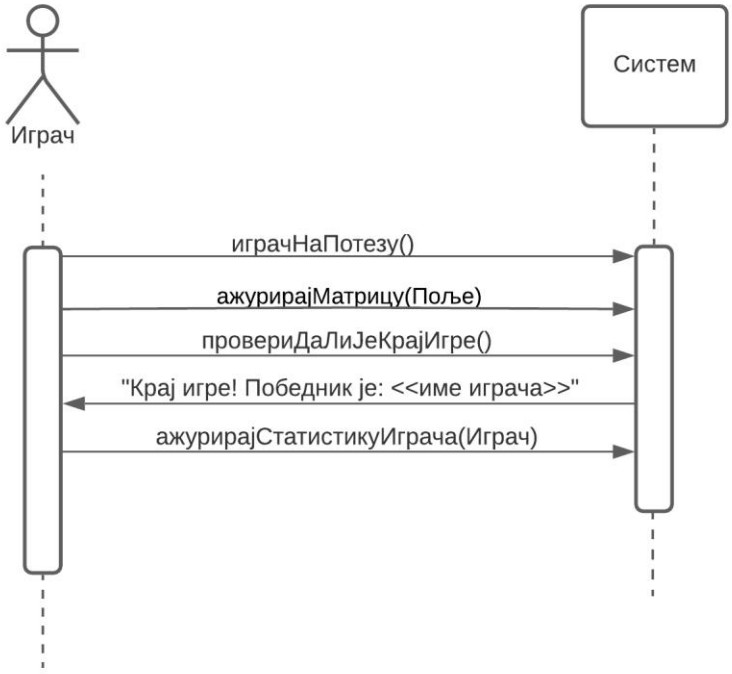


Са наведених секвенцних дијаграма уочава се 1 системска операција коју треба пројектовати:

1. *сигнал* пријављивањеИграча(Играч)

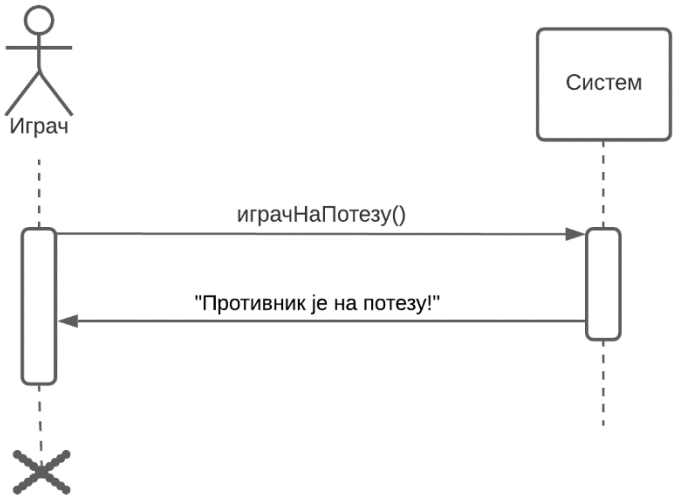
ДС 3: Дијаграм секвенци случаја коришћења – Попуњавање поља на Икс-Окс табли

1. **Играч** кликом на поље позива систем да попуни поље. (АПСО)
2. **Систем** приказује **играчу** поруку: „Крај игре! Победник је: <<име играча>>“ (ИА)

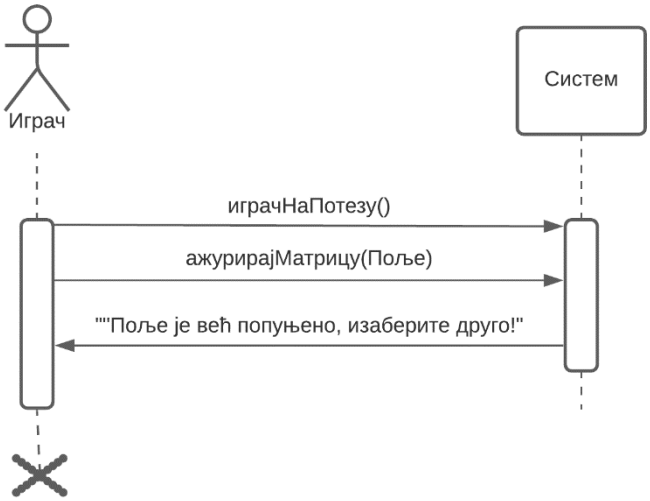


Алтернативни сценарио:

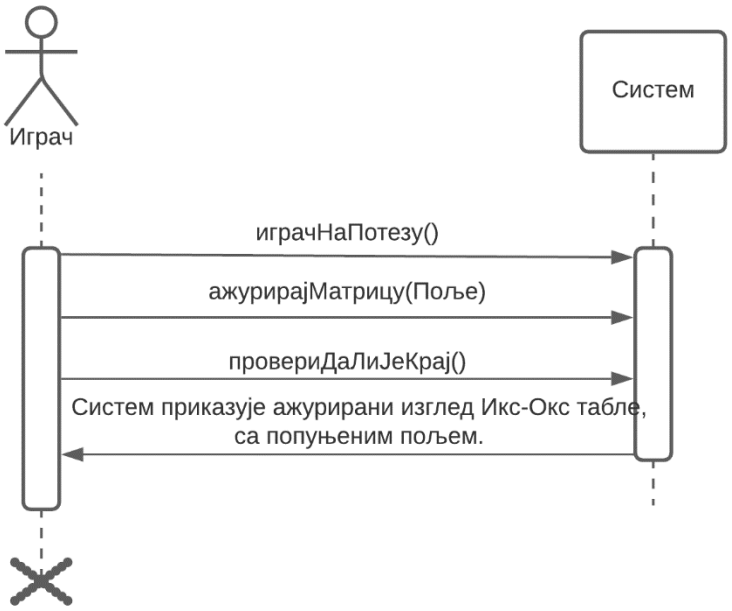
2.1. Уколико **играч** није на потезу, **систем** приказује играчу поруку: „Противник је на потезу!“. Прекида се извршење сценарија.



2.2. Уколико је поље већ попуњено, **систем** приказује **играчу** поруку: „Поље је већ попуњено, изаберите друго!“. Прекида се извршење сценарија.



2.3. Уколико не постоји добитна комбинација **систем** приказује ажурирани изглед Икс-Окс табле, са попуњеним пољем. Прекида се извршење сценарија. (ИА)

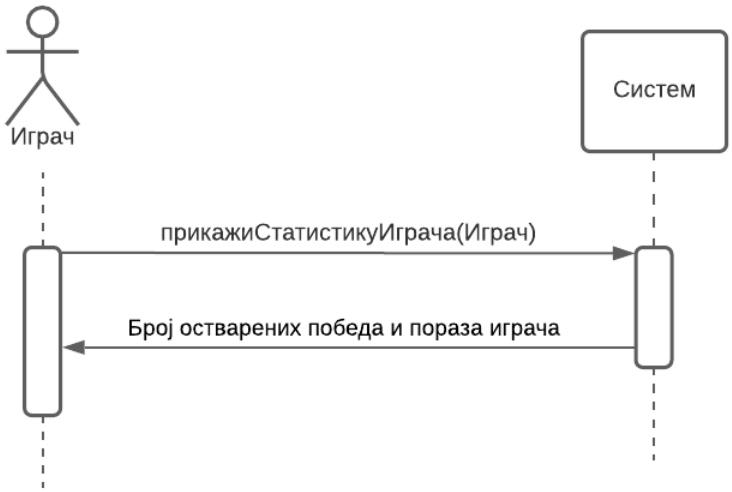


Са наведених секвенцних дијаграма уочавају се 3 системске операције које треба пројектовати:

1. *сигнал* играчНаПотезу()
2. *сигнал* ажурирајМатрицу(Поље)
3. *сигнал* провериДаЛиЈеКрај()
4. ажурирајСтатистикуИграча(Играч)

ДС 4: Дијаграм секвенци случаја коришћења – Извештај о статистици играча

1. **Играч** позива систем да прикаже извештај о статистици. (АПСО)
2. **Систем** приказује **играчу** број остварених победа и пораза. (ИА)

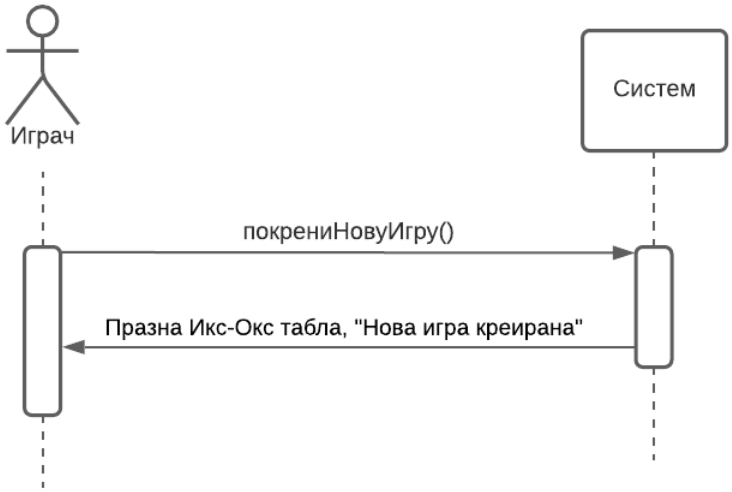


Са наведених секвенцних дијаграма уочава се 1 системска операција коју треба пројектовати:

1. *сигнал* прикажиСтатистикуИграча(Играч)

ДС 5: Дијаграм секвенци случаја коришћења – Покретање нове игре

1. **Играч** позива **систем** да покрене нову игру. (АПСО)
2. **Систем** приказује **играчу** празну Икс-Окс таблу и поруку: „Нова игра креирана!“. (ИА)



2.2 Понашање софтверског система – Дефинисање уговора о системским операцијама

Уговор УГ1: РегистровањеИграча

Операција: регистровањеИграча(Играч):сигнал
Веза са СК: СК1
Предуслови: Вредносна и структурна ограничења над објектом Играч су задовољена.
Постуслови: Подаци о играчу су запамћени.

Уговор УГ2: ПријављивањеИграча

Операција: пријављивањеИграча(Играч):сигнал
Веза са СК: СК2
Предуслови: -
Постуслови: -

Уговор УГ3: ИграчНаПотезу

Операција: играчНаПотезу():сигнал;
Веза са СК: СК3
Предуслови: -
Постуслови: -

Уговор УГ4: АжурирањеМатрице

Операција: ажурирајМатрицу():сигнал;
Веза са СК: СК3
Предуслови: -
Постуслови: Икс-Окс матрица је ажурирана.

Уговор УГ5: ПроверaКрајИгре

Операција: провериДаЛиЈеКрај():сигнал;
Веза са СК: СК3
Предуслови: -
Постуслови: -

Уговор УГ6: АжурирањеСтатистикеИграча

Операција: ажурирајСтатистикуИграча(Играч):сигнал;
Веза са СК: СК3
Предуслови: Игра је завршена – постоји добитна комбинација.
Постуслови: Статистика играча је ажурирана.

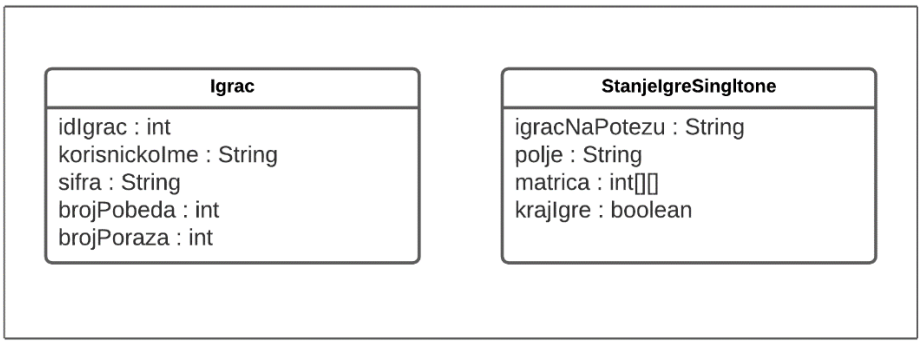
Уговор УГ7: ПрикажиСтатистикуИграча

Операција: прикажиСтатистикуИграча(Играч):сигнал;
Веза са СК: СК4
Предуслови: -
Постуслови: -

Уговор УГ8: ПокрениНовуИгру

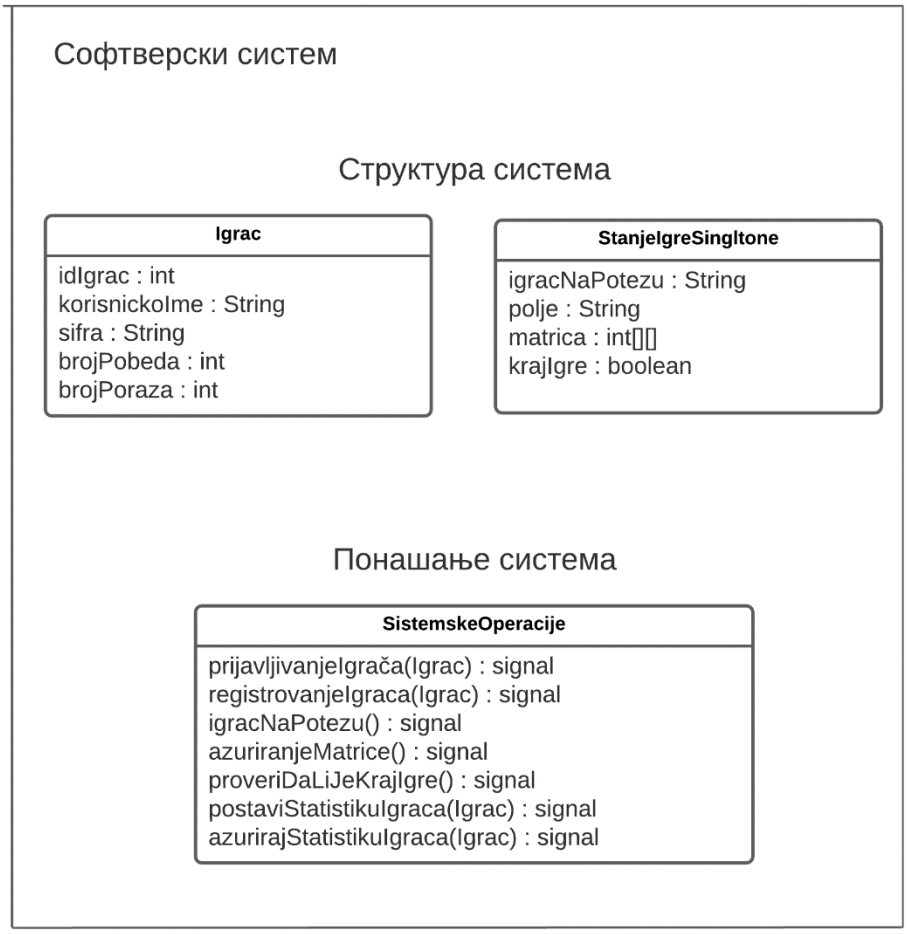
Операција: покрениНовуИгру():сигнал;
Веза са СК: СК5
Предуслови: -
Постуслови: Икс-Окс матрица је ажурирана.

2.3 Структура софтверског система – концептуални (доменски) модел



Слика 2. Концептуални (доменски) модел

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



Слика 3. Структура и понашање софтверског система

2.4 Структура софтверског система – релациони модел

Igrac(idIgrac, korisnickoIme, sifra, brojPobeda, brojPoraza)

Табела Igrac		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	INSERT / UPDATE / DELETE /
	idIgrac	int	not null			

			and >0			
	korisnickoIme	String	not null and unique			
	sifra	String	not null			
	brojPobeda	int				
	brojPoraza	int				

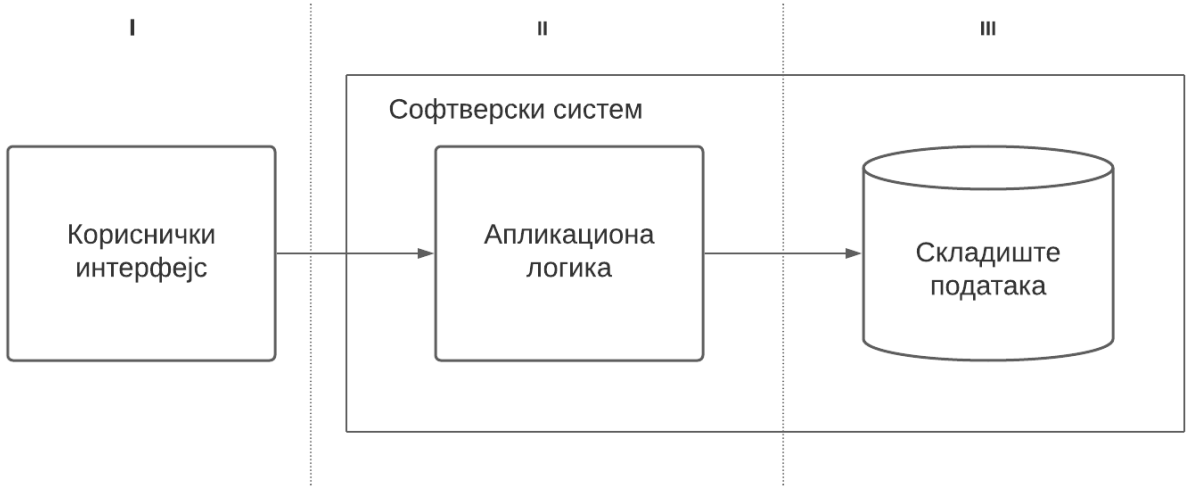
3 Пројектовање

У фази пројектовања описују се физичка структура и понашање софтверског система (архитектура софтверског система). Пројектовање архитектуре подразумева пројектовање корисничког интерфејса, апликационе логике и складишта података. Под пројектовање корисничког интерфејса се подразумева пројектовање екранских форми и контролера корисничког интерфејса. Апликациона логика обухвата пројектовање контролера апликационе логике, пословне логике (логичка структура и понашање софтверског система) и брокера базе података.

3.1 Архитектура софтверског система

У оквиру пројекта примењена је тронивојска архитектура која се састоји од корисничког интерфејса, апликационе логике и складишта података.

Ниво корисничког интерфејса налази се на страни клијента, док су на серверској страни смештени апликациона логика и складиште података.



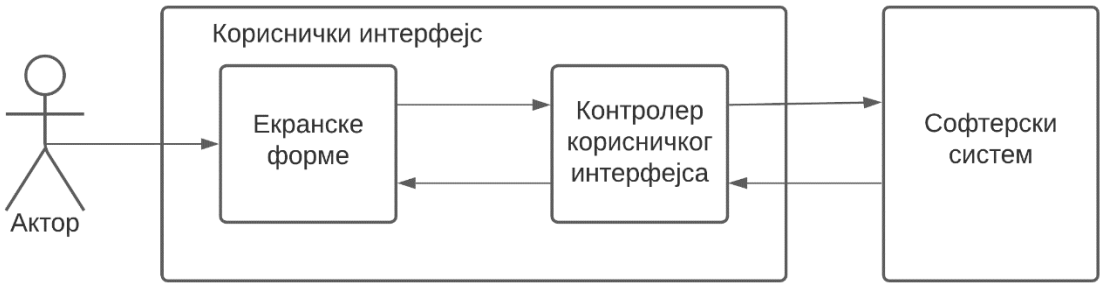
Слика 4. Тронивојска архитектура

3.2 Пројектовање корисничког интерфејса

Кориснички интерфејс се састоји од екранских форми и контролера корисничког интерфејса.

Екрансе форме служе да прихвате податке и догађаје које актор уноси или прави, проследи их до контролера корисничког интерфејса и на крају прикаже податке које је добио као одговор од контролера.

Контролери корисничког интерфејса служе да од екранске форме прихвате графичке објекте, конвертују их у доменске, затим да проследе захтеве ка контролеру апликационе логике, прихвате одговор од истог и на крају доменске објекте претворе у графичке.



Слика 5. Структура корисничког интерфејса

3.2.1 Пројектовање екранских форми

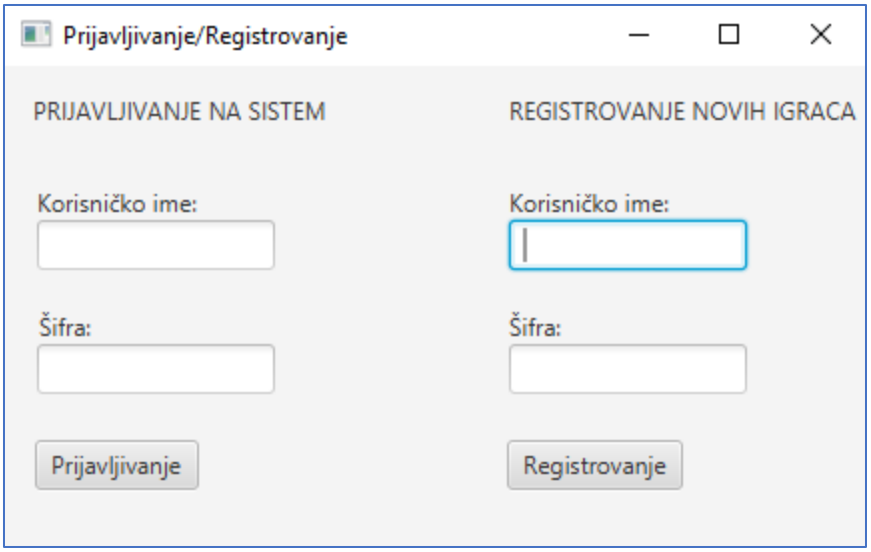
СК1: Случај коришћења – новог играча

Назив СК:
Регистровање новог играча

Актор:
Играч

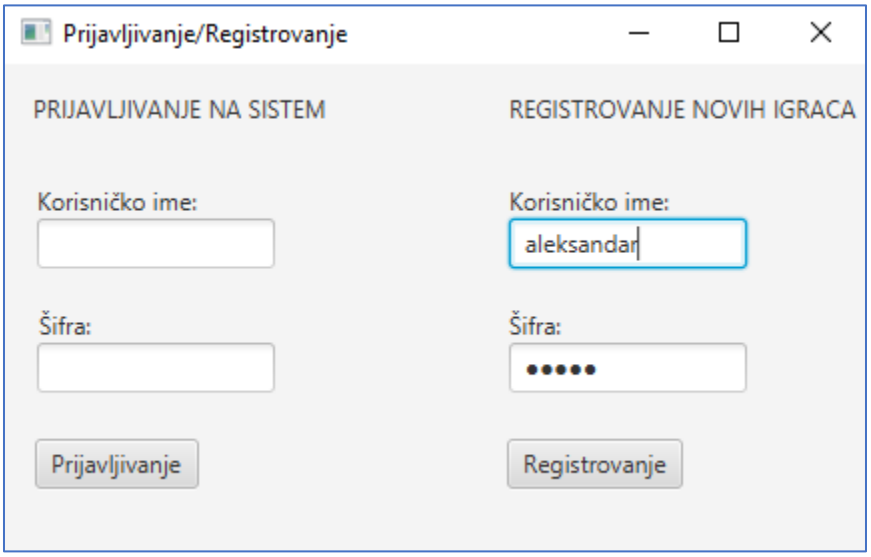
Учесници СК:
Играч и систем (програм)

Предуслов: Систем је укључен и приказује форму за регистровање играча.

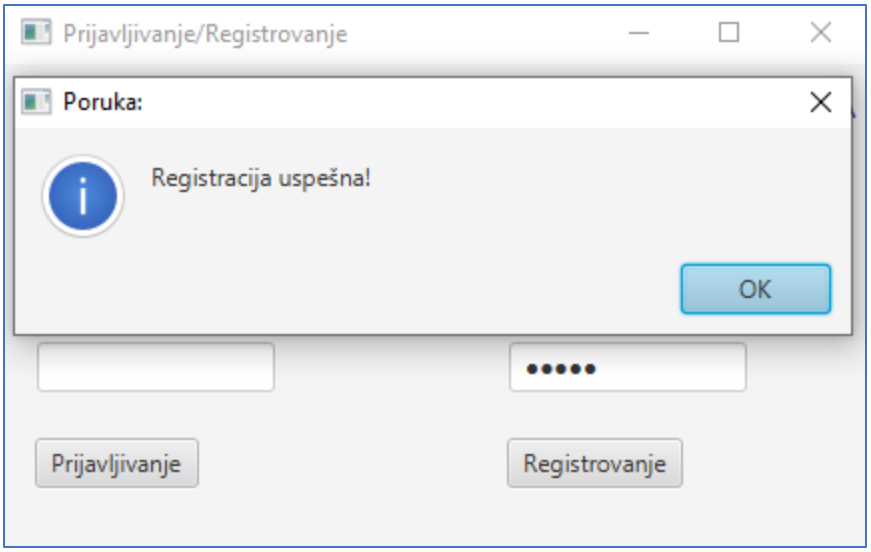


Основни сценарио СК:

1. **Играч** уноси податке за регистровање играча. (АПУСО)

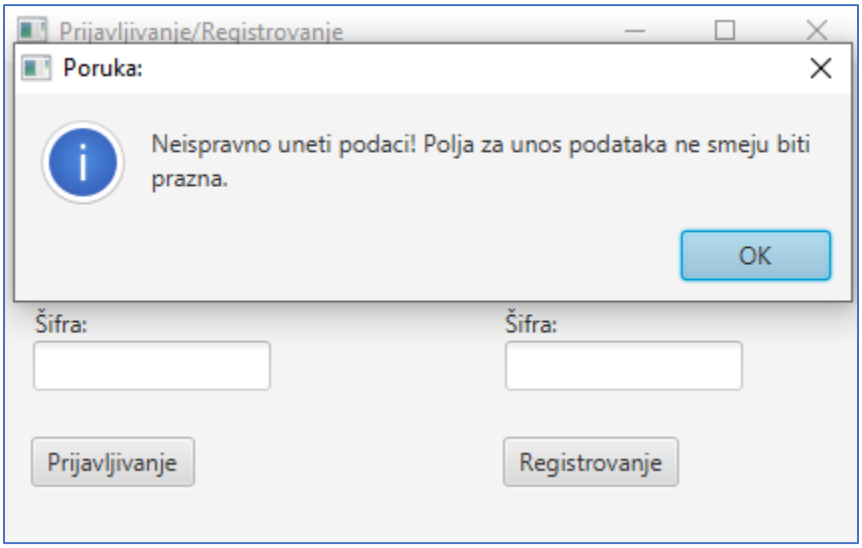


2. **Играч** контролише да ли је исправно унео податке за регистровање. (АНСО)
3. **Играч** позива систем да запамти новог **играча** са унетим подацима. (АПСО)
4. **Систем** памти податке о новом **играчу**. (СО)
5. **Систем** приказује **играчу** поруку: „Регистровање успешно!“ . (ИА)

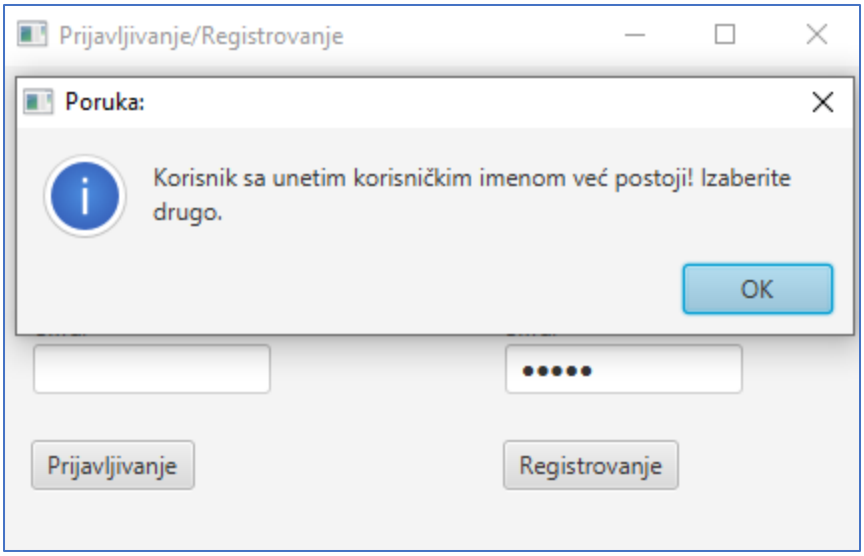


Алтернативна сценарија:

5.1. Уколико **играч** није попунио поља за регистровање, **систем** приказује **играчу** поруку: „Неисправно унети подаци! Поља за унос података не смеју бити празна.“.(ИА)



5.2. Уколико **играч** са унетим корисничким именом већ постоји у бази, **систем** приказује **играчу** поруку: „Играч са унетим корисничким именом већ постоји! Изаберите друго.“ (ИА)



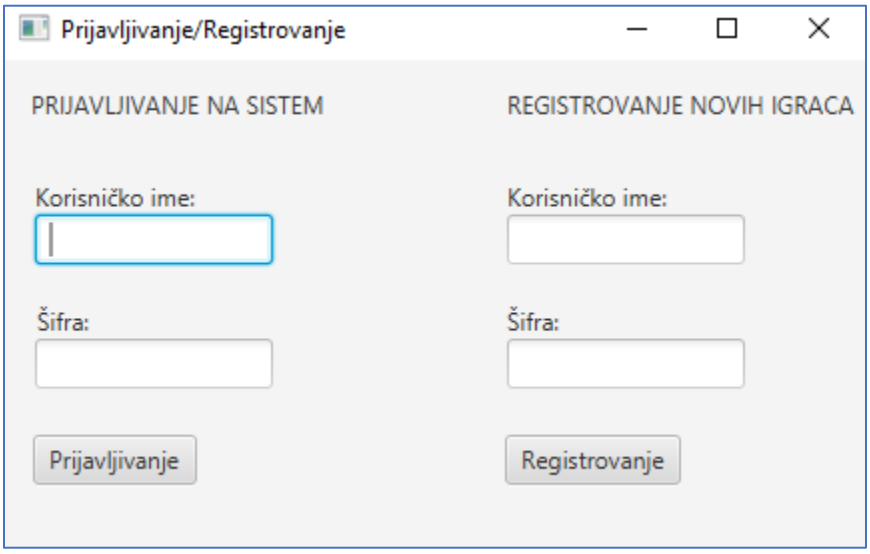
СК 2: Случај коришћења – Пријављивање играча

Назив СК:
Пријављивање играча

Актор:
Играч

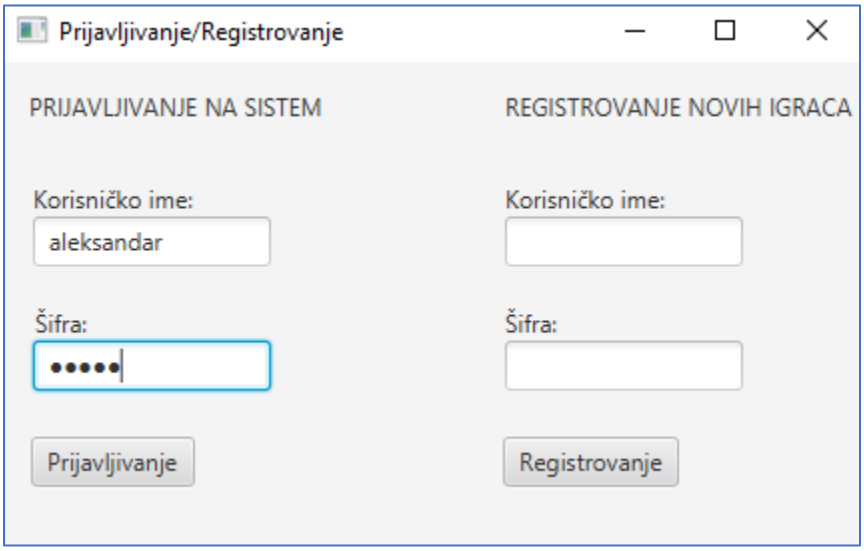
Учесници СК:
Играч и систем (програм)

Предуслов: Систем је укључен и приказује форму за пријављивање играча.

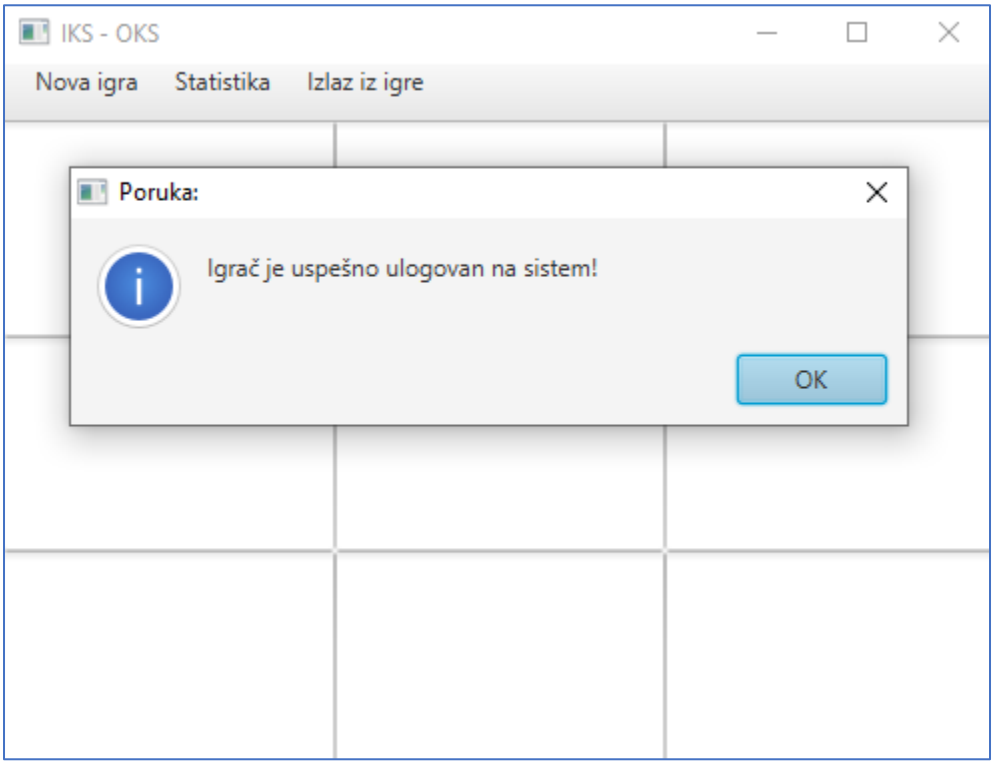


Основни сценарио СК:

1. **Играч** уноси податке за пријављивање **играча**. (АПУСО)

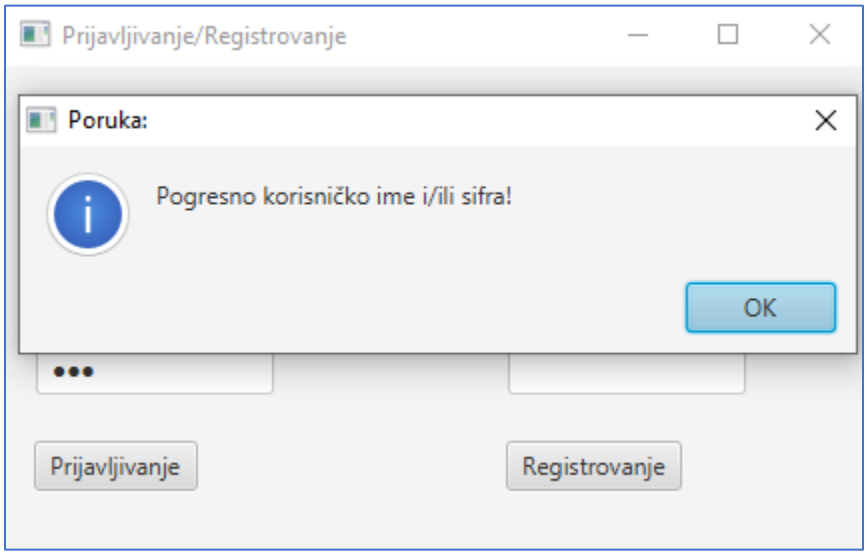


2. **Играч** контролише да ли је исправно унео податке за пријављивање. (АНСО)
3. **Играч** позива систем да пронађе **играча** са задатим подацима. (АПСО)
4. **Систем** претражује **играче**. (СО)
5. **Систем** приказује **играчу** поруку: „**Играч** је успешно улогован на систем!“ . (ИА)



Алтернативна сценарија:

5.1 Уколико **систем** не може да пронађе **играча**, приказује **играчу** поруку: „Погрешно корисничко име и/или шифра!“.



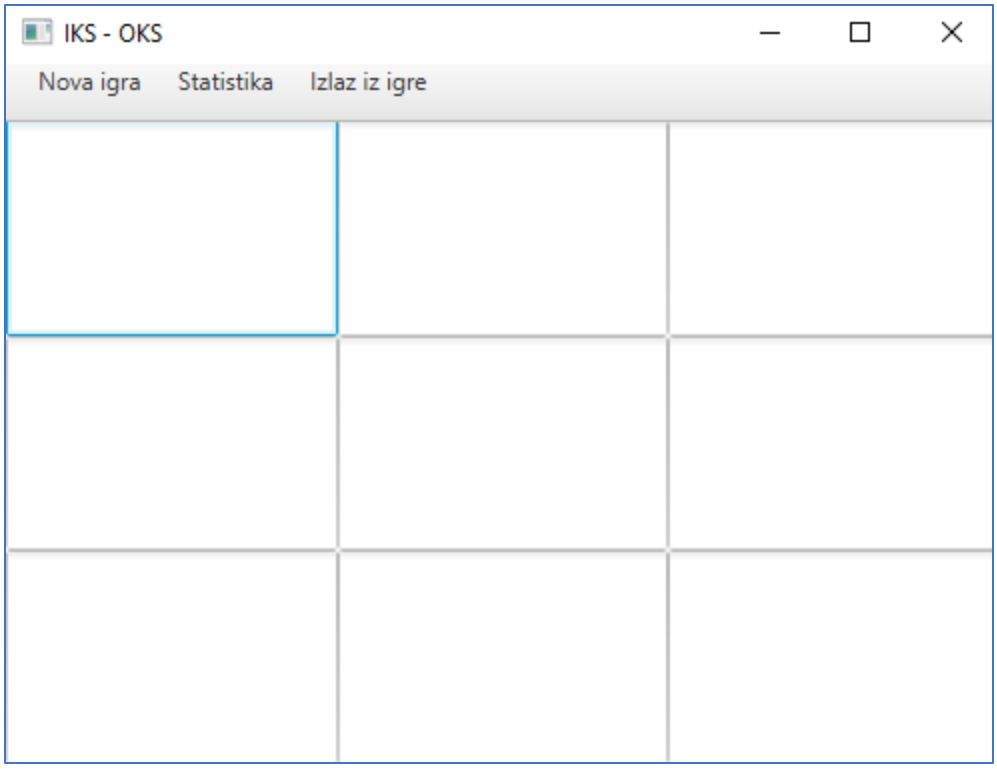
СК 3: Случај коришћења – Попуњавање поља на Икс-Окс табли

Назив СК:
Попуњавање поља на Икс-Окс табли

Актор:
Играч

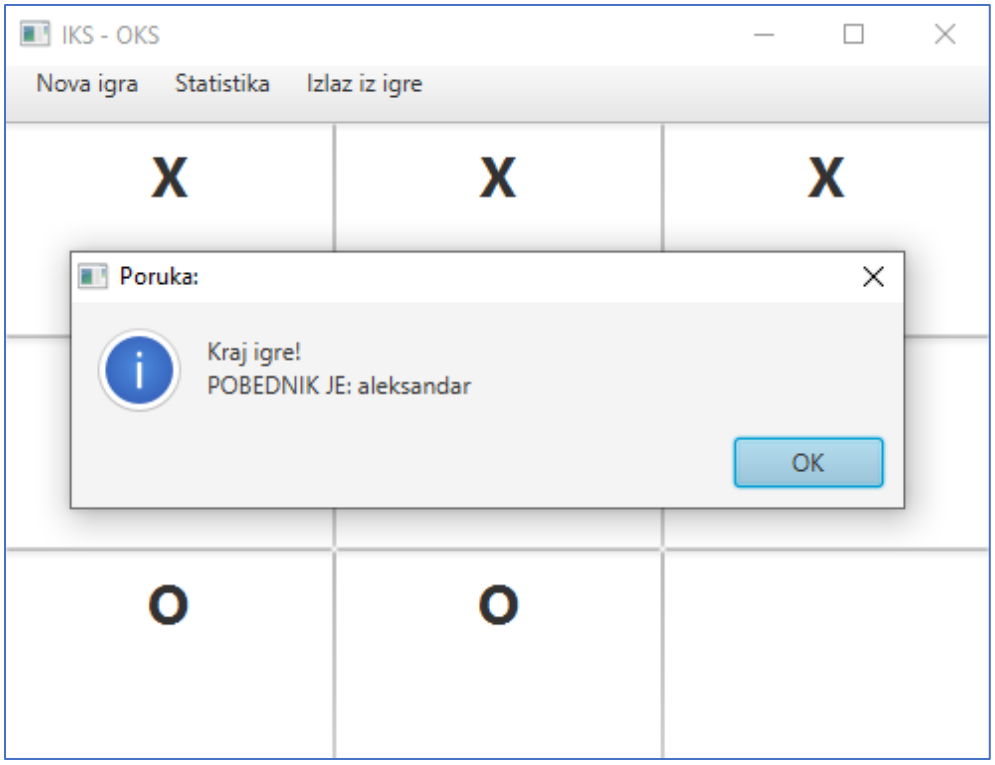
Учесници СК:
Играч и систем (програм)

Предуслов: Систем је укључен и **играч** је пријављен са својом шифром. **Систем** приказује форму за унос поља на Икс-Окс табли.



Основни сценарио СК:

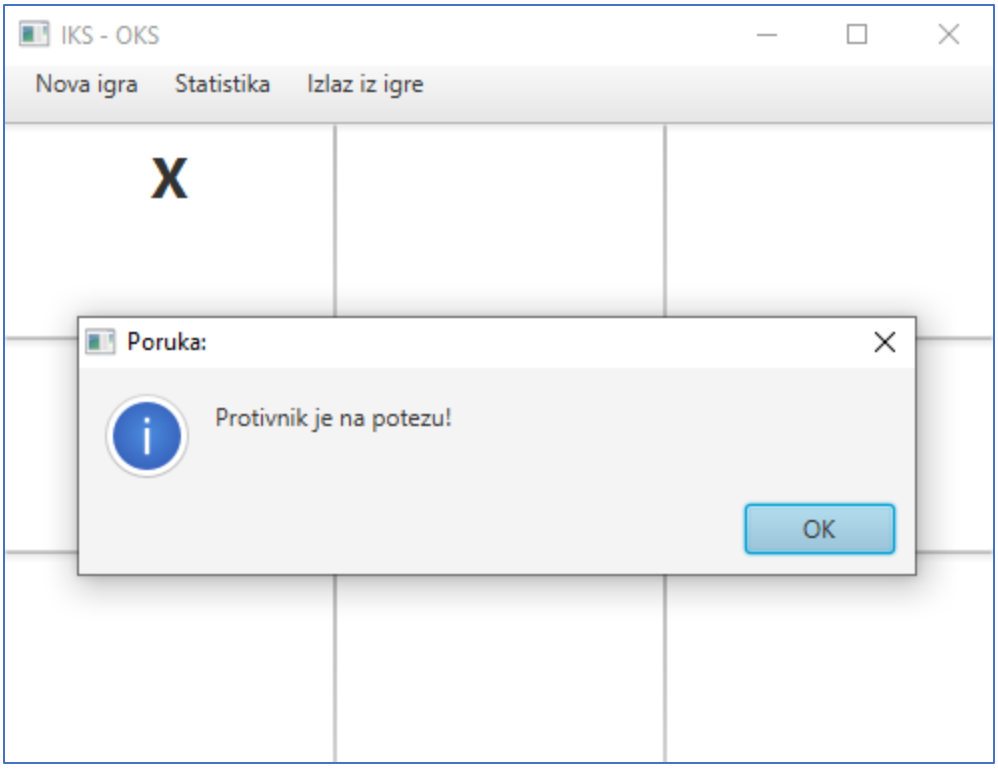
1. **Играч** бира поље на табли у које жели да унесе X или O знак. (АНСО)
2. **Играч** позива систем да попуни поље. (АПСО)
3. **Играч** позива систем да попуни поље знаком. (АПСО)
4. **Систем** проверава да ли је играч на потезу. (СО)
5. **Систем** врши ажурирање Икс-Окс табле. (СО)
6. **Систем** проверава да ли постоји добитна комбинација. (СО)
7. **Систем** приказује **играчу** поруку: „Крај игре! Победник је: <<име играча>>“ (ИА)



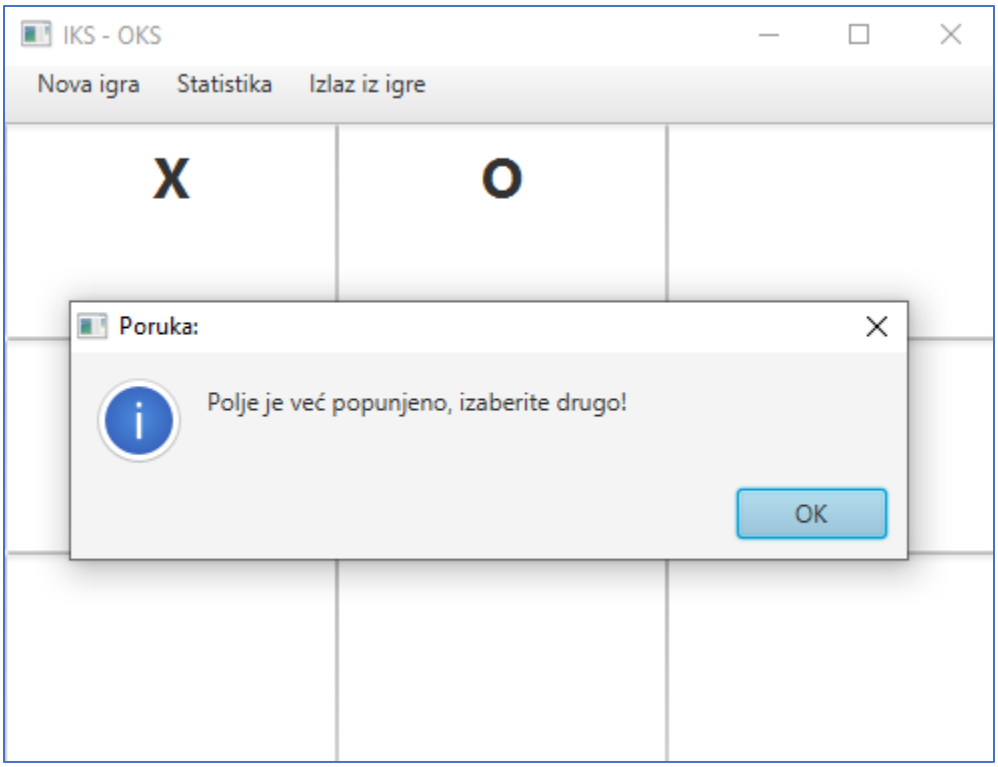
8. **Систем** врши ажурирање статистике играча. (СО)

Алтернативна сценарија:

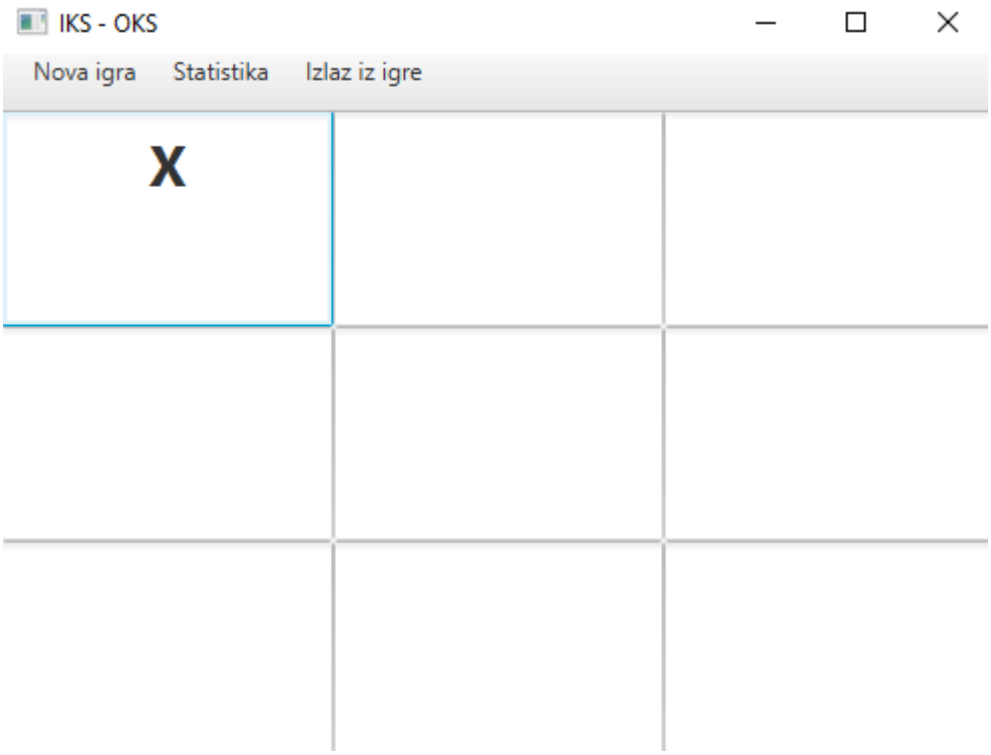
4.1. Уколико **играч** није на потезу, **систем** приказује играчу поруку: „Противник је на потезу!“. Прекида се извршење сценарија. (ИА)



5.1. Уколико је поље већ попуњено, **систем** приказује **играчу** поруку: „Поље је већ попуњено, изаберите друго!“. Прекида се извршење сценарија. (ИА)



6.1 Уколико не постоји добитна комбинација **систем** приказује ажурирани изглед Икс-Окс табле, са попуњеним пољем. Прекида се извршење сценарија. (ИА)



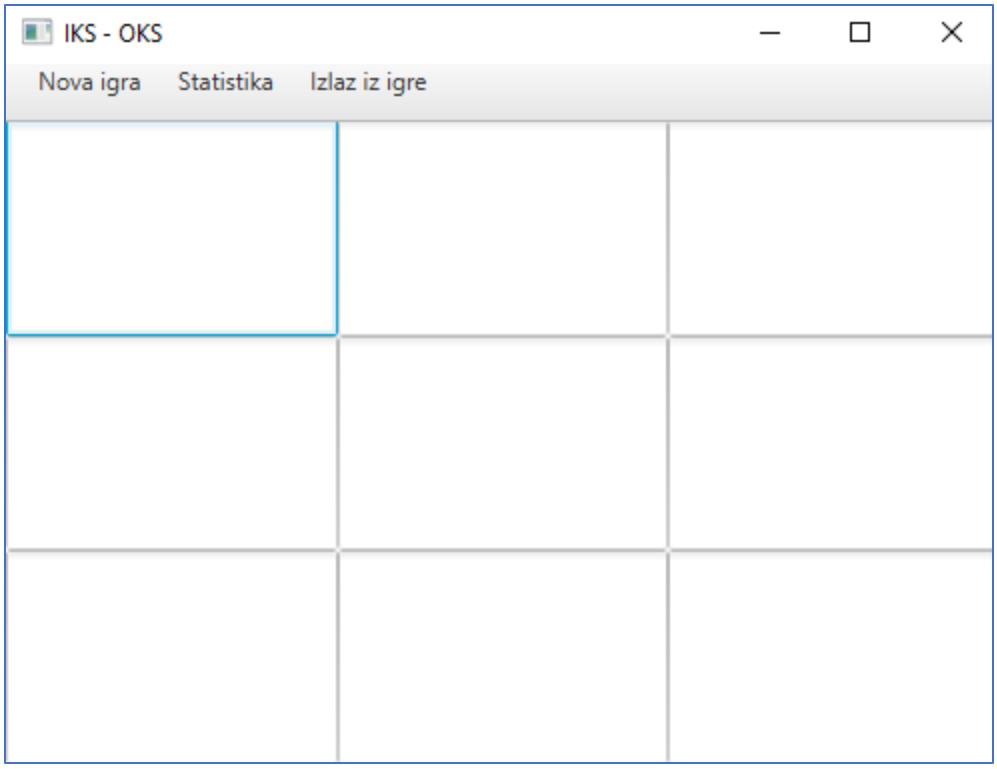
СК 4: Случај коришћења – Извештај о статистици играча

Назив СК:
Извештај о статистици играча

Актор:
Играч

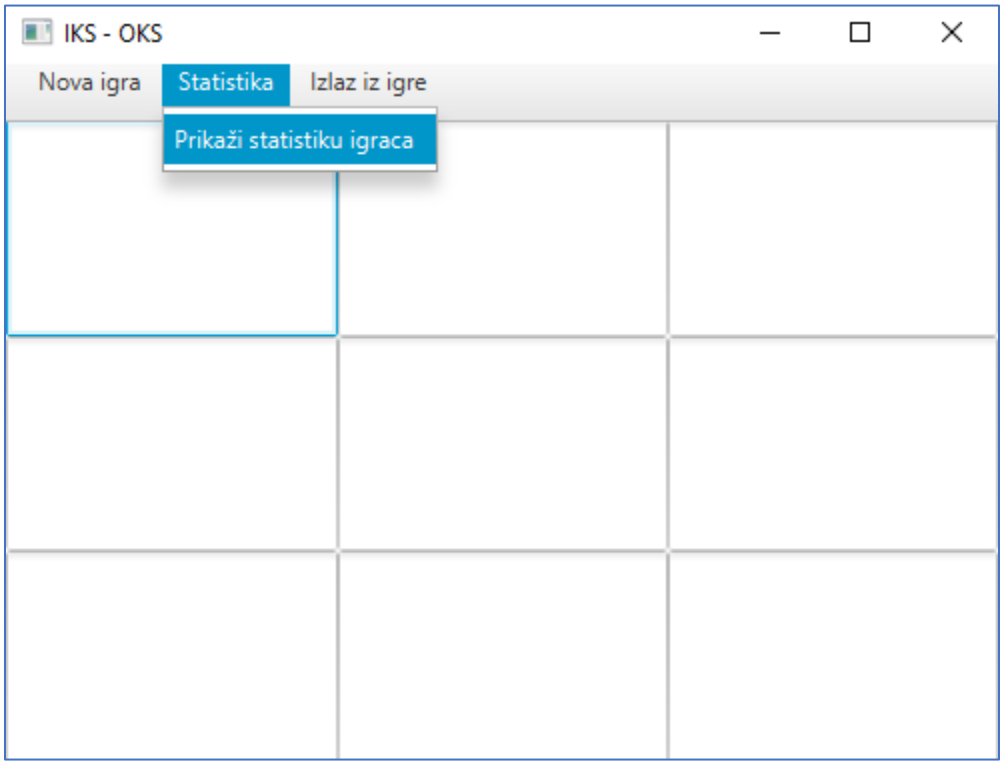
Учесници СК:
Играч и систем (програм)

Предуслов: Систем је укључен и **играч** је пријављен са својом шифром.

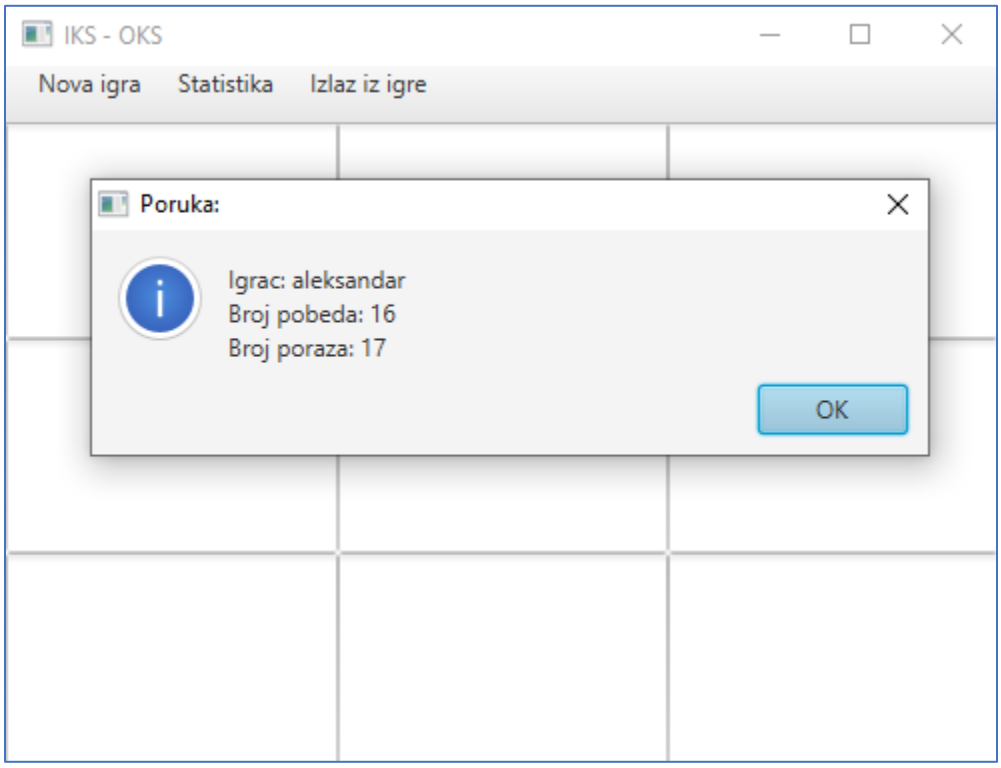


Основни сценарио СК:

1. **Играч** позива систем да прикаже извештај о статистици. (АПСО)



2. **Систем** претражује статистику у бази на основу корисничког имена пријављеног играча. (СО)
3. **Систем** приказује **играчу** број остварених победа и пораза. (ИА)



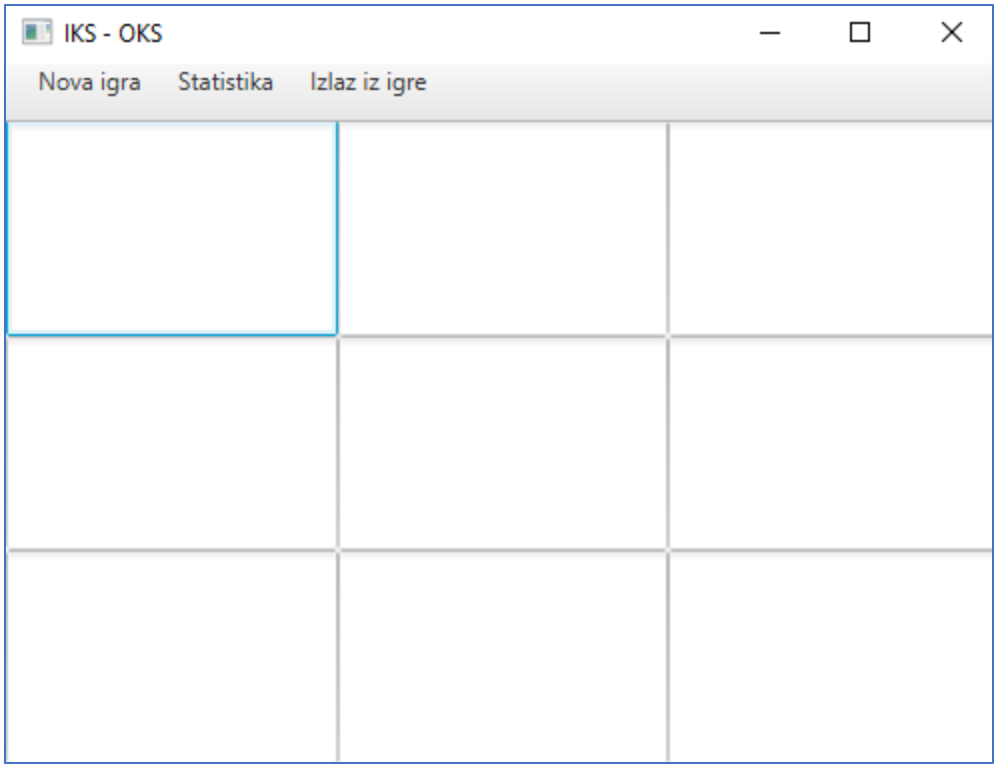
СК 5: Случај коришћења – Покретање нове игре

Назив СК:
Покретање нове игре

Актор:
Играч

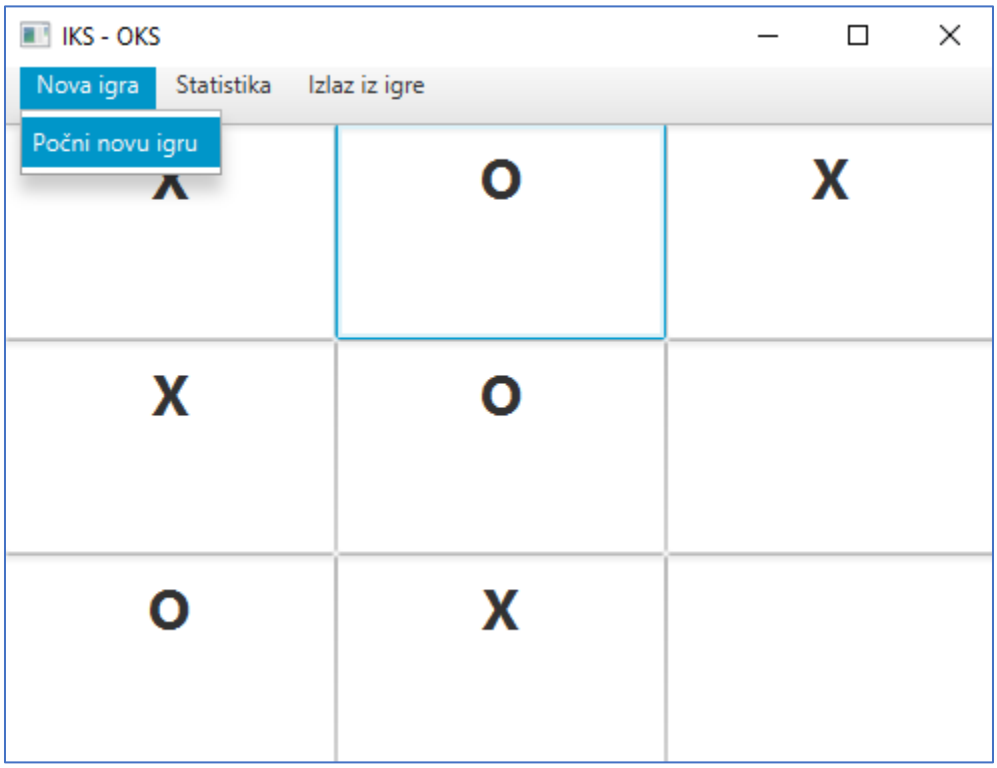
Учесници СК:
Играч и систем (програм)

Предуслов: Систем је укључен и **играч** је пријављен са својом шифром.

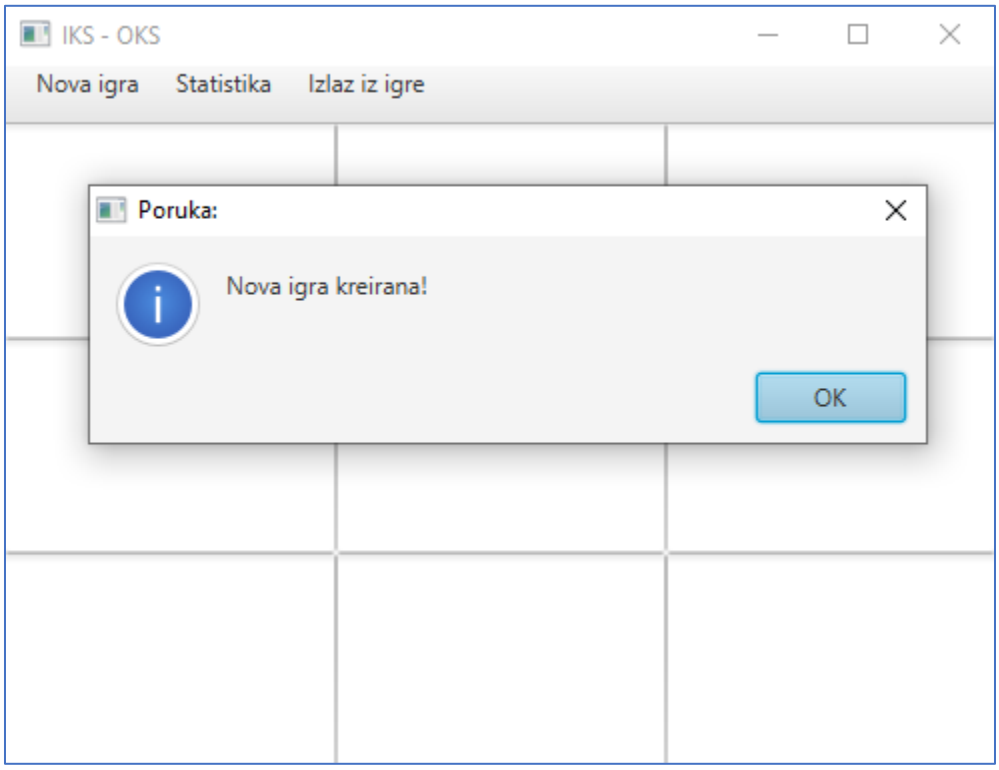


Основни сценарио СК:

1. **Играч** позива систем да покрене нову игру. (АПСО)



2. **Систем** покреће нову игру. (СО)
3. **Систем** приказује **играчу** празну Икс-Окс таблу и поруку: „Нова игра креирана!“. (ИА)



3.2.2 Пројектовање контролера корисничког интерфејса

Као што је већ раније речено, контролери корисничког интерфејса имају улогу да прихвате податке и догађаје од екранских форми, конвертују их у доменске објекте и проследе до контролера апликационе логике који се налази на серверској страни, и на крају прихвате одговор и изврше конвертовање доменског објекта у графичке објекте.

У оквиру пројекта направљена су два контролера:

1. KontrolerGUIPrijavljivanje - реализован над екранском формом за пријављивање и регистровање играча
2. KontrolerGUIIksOKs - реализован над екранском формом за играње Икс-Окс игре

```
public class KontrolerGUIPrijavljivanje {

    FXMLDocumentController fxcon;
```

```
    KontrolerServer_Service service;
    KontrolerServer kal;
    public TransferObjekat gto;
    public KontrolerGUIPrijavljivanje(FXMLDocumentController fxcon) throws
NoSuchFieldException, IllegalArgumentException, IllegalAccessException {
        this.fxcon = fxcon;
        service = new KontrolerServer_Service();
        kal = service.getKontrolerServerPort();
        gto = new TransferObjekat();
        this.fxcon.prijavljivanje.setOnAction(new OsluskivacPrijavljivanje(this));
        this.fxcon.registrovanje.setOnAction(new OsluskivacRegistrovanje(this));
    }
    public void prijavljivanje() {
        Igrac igrac=new Igrac();
        KonverterGUIDK.konvertujGUIUDK(fxcon, igrac, "Prijavljivanje");
        gto.setGdo(igrac);
        igrac = kal.proveraPrijavljivanja(gto).getGdo();
        if (igrac != null) {
            GUIIksOks.IksOksKlijent iksOksKlijent;
            Stage s;
            iksOksKlijent = new GUIIksOks.IksOksKlijent();
            s = new Stage();
            try {
                iksOksKlijent.start(s);
                iksOksKlijent.vratiFXMLkon().kngui.postaviIgraca(igrac);
                fxcon.zatvoriFormu();
                poruka("Igrač je uspešno ulogovan na sistem!");
```

```
            } catch (Exception ex) {
                Logger.getLogger(KontrolerGUIPrijavljivanje.class.getName()).log(Level.SEVERE, null,
ex);
            }
        } else {
            poruka("Pogresno korisničko ime i/ili sifra!");
        }
    }
    public void registrovanje() {
        Igrac igrac=new Igrac();
        KonverterGUIDK.konvertujGUIUDK(fxcon, igrac, "Registrovanje");
        gto.setGdo(igrac);
        if (igrac.getKorisnickoIme().equals("") || igrac.getSifra().equals("")) {
            poruka("Neispravno uneti podaci! Polja za unos podataka ne smeju biti prazna.");
        } else {
            gto = kal.registrovanjeIgraca(gto);
            if (gto.getGdo() == null) {
                poruka("Korisnik sa unetim korisničkim imenom već postoji! Izaberite drugo.");
            } else {
                poruka("Registracija uspešna!");
            }
        }
    }
    public void poruka(String poruka) {
        Alert infoAlert = new Alert(Alert.AlertType.INFORMATION);
        infoAlert.setTitle("Poruka:");
        infoAlert.setHeaderText(null);
```

```
        infoAlert.setContentText(poruka);
        infoAlert.showAndWait();
    }
```

Класа KontrolerGUIIksOks:

```
public class KontrolerGUIIksOks {
    public FXMLDocumentController fxdc;
    public OsluskivacXO xo;
    public OsluskivacPromena op;
    public TransferObjekat to;
    public Igrac igrac;
    public int brojac = 0;
    public String redniBrojIgraca;
    public KontrolerServer_Service service;
    public KontrolerServer kal;
    public KontrolerGUIIksOks(FXMLDocumentController fxdc) throws NoSuchFieldException,
IllegalArgumentException, IllegalAccessException, IOException, FileNotFoundException,
ClassNotFoundException {
        this.fxdc = fxdc;
        service = new KontrolerServer_Service();
        kal = service.getKontrolerServerPort();
        to = new TransferObjekat();
        to = kal.kreirajMatricu(to);
        this.fxdc.izlaz.setOnAction(new OsluskivacIzlaz(this));
        this.fxdc.informacijeOIgracu.setOnAction(new OsluskivacStatistikaIgraca(this));
        this.xo = new OsluskivacXO(this);
        this.fxdc.p00.addEventFilter(MouseEvent.MOUSE_PRESSED, xo);
        this.fxdc.p01.addEventFilter(MouseEvent.MOUSE_PRESSED, xo);
```

```
    this.fxdc.p02.addEventFilter(MouseEvent.MOUSE_PRESSED, xo);
    this.fxdc.p10.addEventFilter(MouseEvent.MOUSE_PRESSED, xo);
    this.fxdc.p11.addEventFilter(MouseEvent.MOUSE_PRESSED, xo);
    this.fxdc.p12.addEventFilter(MouseEvent.MOUSE_PRESSED, xo);
    this.fxdc.p20.addEventFilter(MouseEvent.MOUSE_PRESSED, xo);
    this.fxdc.p21.addEventFilter(MouseEvent.MOUSE_PRESSED, xo);
    this.fxdc.p22.addEventFilter(MouseEvent.MOUSE_PRESSED, xo);
    this.fxdc.novaIgra.setOnAction((new OsluskivacNovaIgra(this)));
    op = new OsluskivacPromena(this);
    op.start();
}
public void postaviXO(TextArea polje) throws InterruptedException_Exception {
    if (polje.getText().equals("")) {
        to = kal.igracNaPotezu(to);
        if (to.getIgracNaPotezu().equals("") && brojac == 0) {
            redniBrojIgraca = "igrac1";
            brojac++;
            polje.setText("X");
            to.setIgracNaPotezu("X");
            azurirajMatricu("X" + polje.getId());
        } else {
            if (brojac == 0) {
                redniBrojIgraca = "igrac2";
                brojac++;
            }
            if (to.getIgracNaPotezu().equals("X") && redniBrojIgraca.equals("igrac2")
                    || to.getIgracNaPotezu().equals("O") && redniBrojIgraca.equals("igrac1")) {
```

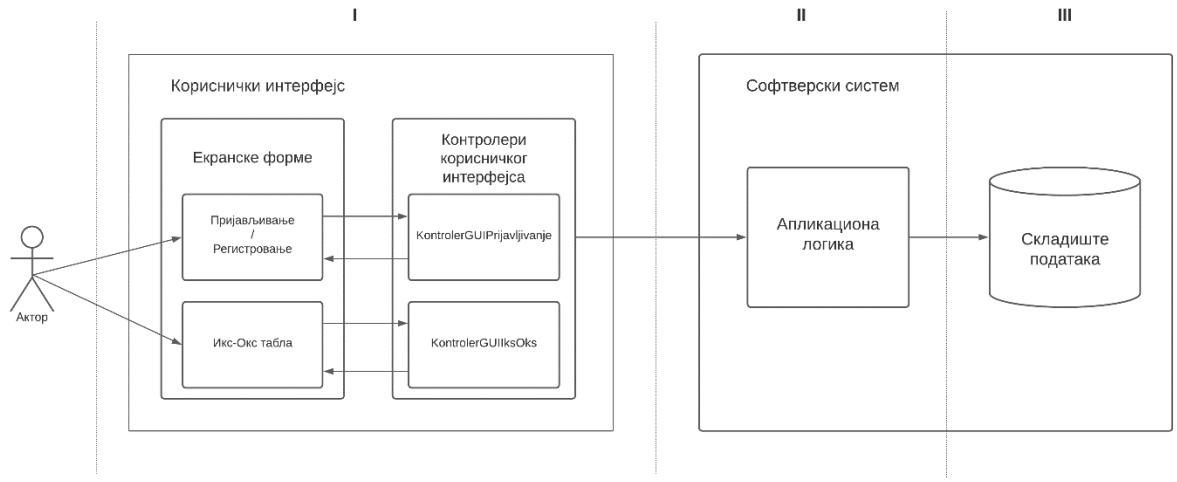
```
                poruka("Protivnik je na potezu!");
            } else {
                polje.setText(to.getIgracNaPotezu());
                azurirajMatricu(to.getIgracNaPotezu() + polje.getId());
            }
        }
    } else {
        poruka("Polje je već popunjeno, izaberite drugo!");
    }
}
public void azurirajMatricu(String polje) {
    to.setPolje(polje);
    kal.azurirajMatricu(to);
}
public TransferObjekat proveriDaLiJeKrajIgre() throws InterruptedException_Exception {
    to = kal.proveriDaLiJeKrajIgre(to);
    return to;
}
public void porukaZaKrajIgre(char c) {
    Platform.runLater(new Runnable() {
        @Override
        public void run() {
            Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
            if (c == 'X' && redniBrojIgraca.equals("igrac1")) {
                poruka("Kraj igre!" + "\n" + "POBEDNIK JE: " + igrac.getKorisnickoIme());
            }
            if (c == 'O' && redniBrojIgraca.equals("igrac2")) {
```

```
                poruka("Kraj igre!" + "\n" + "POBEDNIK JE: " + igrac.getKorisnickoIme());
            }
        }
    });
}
public void postaviStatistikuIgraca() {
    to.setGdo(igrac);
    igrac=kal.postaviStatistikuIgraca(to).getGdo();
    poruka("Igrac: " + igrac.getKorisnickoIme() + "\n" + "Broj pobeda: " + igrac.getBrojPobeda()
+ "\n" + "Broj poraza: " + igrac.getBrojPoraza());
}
public void apdejtujStatistikuIgraca(char c) {
    if (redniBrojIgraca.equals("igrac1")) {
        if (c == 'X') {
            postaviBrojPobeda();
            kal.apdejtujStatistikuIgraca(to);
        } else if (c == 'O') {
            postaviBrojPoraza();
            kal.apdejtujStatistikuIgraca(to);
        }
    }
    if (redniBrojIgraca.equals("igrac2")) {
        if (c == 'O') {
            postaviBrojPobeda();
            kal.apdejtujStatistikuIgraca(to);
        } else if (c == 'X') {
            postaviBrojPoraza();
```

```
            kal.apdejtujStatistikuIgraca(to);
        }
    }
}
 public void pokreniNovuIgru() throws InterruptedException {
    brojac = 0;
    for(Field f : fxdc.getClass().getDeclaredFields()) {
        if(f.getType().getName().equals("javafx.scene.control.TextArea")) {
            try {
                ((javafx.scene.control.TextArea) f.get(fxdc)).setText("");
            } catch (IllegalArgumentException ex) {
                Logger.getLogger(KontrolerGUIIksOks.class.getName()).log(Level.SEVERE, null, ex);
            } catch (IllegalAccessException ex) {
                Logger.getLogger(KontrolerGUIIksOks.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
   }
    to = new TransferObjekat();
    to = kal.kreirajMatricu(to);
    op.resume();
    poruka("Nova igra kreirana!");
}
public void postaviBrojPobeda() {
    int brojPobeda = igrac.getBrojPobeda() + 1;
    igrac.setBrojPobeda(brojPobeda);
    to.setGdo(igrac);
}
```

```
    public void postaviBrojPoraza() {
        int brojPoraza = igrac.getBrojPoraza() + 1;
        igrac.setBrojPoraza(brojPoraza);
        to.setGdo(igrac);
    }
    public void postaviIgraca(Igrac igrac) {
        this.igrac = igrac;
    }
    public void poruka(String poruka) {
        Alert infoAlert = new Alert(Alert.AlertType.INFORMATION);
        infoAlert.setTitle("Poruka:");
        infoAlert.setHeaderText(null);
        infoAlert.setContentText(poruka);
        infoAlert.showAndWait();
    }
    public void zatvoriProgram() {
        Platform.exit();
        System.exit(0);
    }
}
```

Након пројектовања екранских форми и контролера корисничког интерфејса тронивојска архитектура се може представити на следећи начин:



Слика 6. Изглед тронивојске архитектуре након пројектовања екранских форми и контролера корисничког интерфејса

3.3 Комуникација са клијентима

Серверски и клијентски део апликације су повезани преко Веб сервиса, односно комуникација се врши помоћу SOAP протокола.

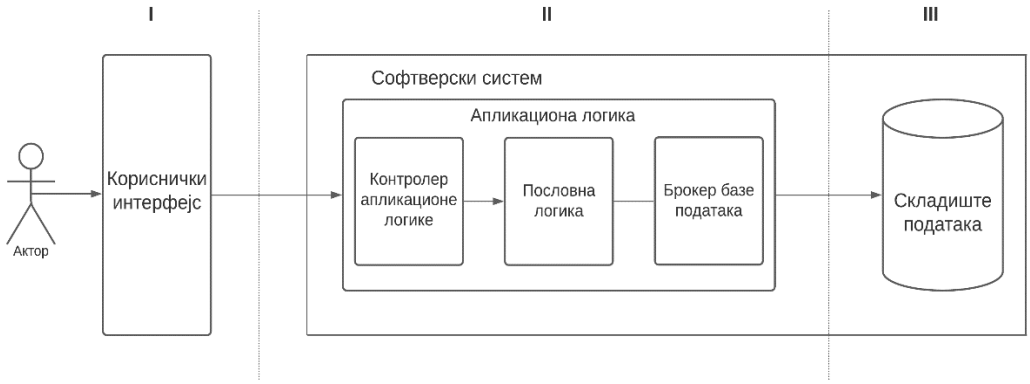
Са клијентског дела апликације позивају се системске операције које се налазе на серверском делу, при чему се за слање и примање података користи трансфер објекат класе ТрансферОбјекат.

```
public class TransferObjekat implements Serializable{

  public Igrac gdo;

  public String polje;

  public String igracNaPotezu;

  public boolean krajIgre;

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

  public String vratiPolje(){return polje;}

  public GeneralDObject vratiDK(){return (GeneralDObject) gdo;}

  public void postaviIgracNaPotezu(String igracNaPotezu) {this.igracNaPotezu=igracNaPotezu;}

  public String vratiIgracNaPotezu() {return igracNaPotezu;}
```

```
  public String postaviPolje(String polje) {return this.polje=polje;}
  public void postaviKrajIgre(boolean krajIgre) {this.krajIgre=krajIgre;}
}
```

3.4 Пројектовање апликационе логике

Апликациона логика представља средњи ниво тронивојске архитектуре. Састоји се из контролера апликационе логике, пословне логике и брокера базе података.



Слика 7. Структура апликационе логике унутар тронивојске архитектуре

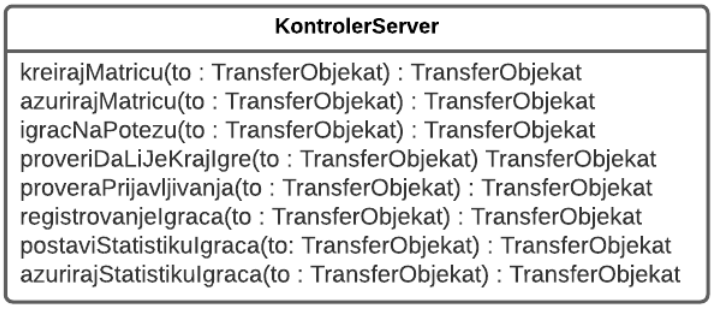
3.4.1 Контролер апликационе логике

Контролер апликационе логике представља контролер који се налази на серверској страни и служи да прихвати захтеве од клијената, проследи их системским операцијима на обраду, и на крају врати одговарајући резултат.

```
public class KontrolerServer {
   public TransferObjekat igracNaPotezu(TransferObjekat to) {
      new IgracNaPotezu(to).igracNaPotezu();
      return to;
   }
   public TransferObjekat kreirajMatricu(TransferObjekat to) {
```

```
        new KreiranjeMatrice(to).kreirajMatricu();
        return to;
    }
    public TransferObjekat azurirajMatricu(TransferObjekat to) {
        new AzuriranjeMatrice(to).azurirajMatricu();
        return to;
    }
    public TransferObjekat proveriDaLiJeKrajIgre(TransferObjekat to) throws
InterruptedException {
        new ProveraKrajIgre(to).proveriDaLiJeKrajIgre();
        return to;
    }
    public TransferObjekat proveraPrijavljivanja(TransferObjekat to) {
        new ProveraPrijavljivanja(to).proveraPrijavljivanja();
        return to;
    }
    public TransferObjekat registrovanjeIgraca(TransferObjekat to) {
        new RegistrovanjeIgraca(to).registrovanjeIgraca();
        return to;
    }
    public TransferObjekat postaviStatistikuIgraca(TransferObjekat to) {
        new PostaviStatistikuIgraca(to).postaviStatistikuIgraca();
        return to;
    }
    public TransferObjekat apdejtujStatistikuIgraca(TransferObjekat to) {
        new AzuriranjeStatistikeIgraca(to).azurirajStatistikuIgraca();
        return to;
```

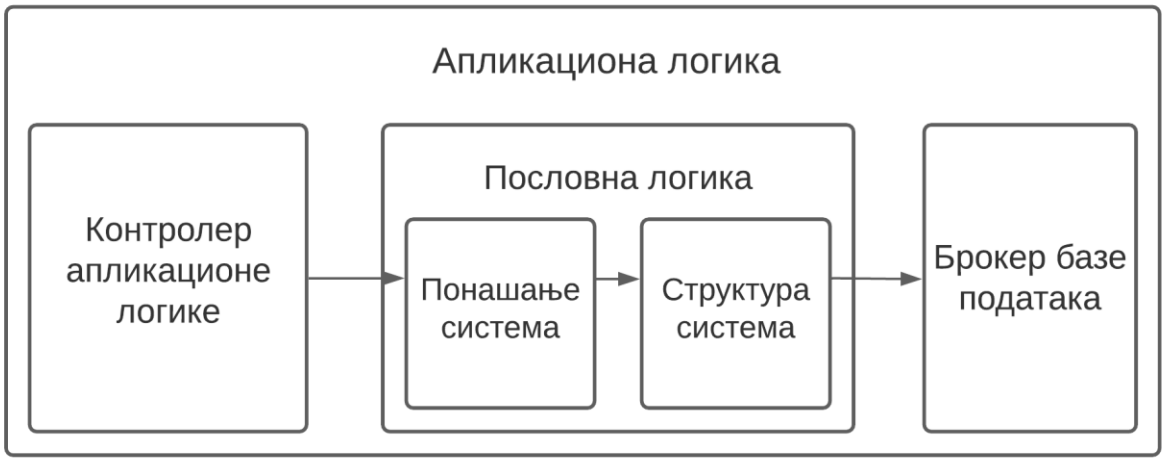
```
    }
}
```



Слика 8. Контролер апликационе логике

3.4.2 Пословна логика

Пословна логика софтверског система се састоји од логичке структуре и понашања система.



Слика 9. Структура пословне логике софтверског система

3.4.2.1 Пројектовање понашања софтверског система – системске операције

Класе које су одговорне за понашање софтверског система наслеђују једну од две апстрактне класе, у завиности од тога да ли учествују у обради перзистентних или неперзистентних података.

Класе системских операција које се одвијају над објектом класе StanjeIgreSingltone наслеђују апстрактну класу OpstaPromenaStanjaIgre, док класе системских операцију које се извршавају над објектима класе Igrac наслеђују апстрактну класу OpstaPromenaStanjaBaze.

Класа OpstaPromenaStanjaIgre:

```
public abstract class OpstaPromenaStanjaIgre {
   StanjeIgreSingltone si;
   TransferObjekat to;
   OpstaPromenaStanjaIgre(TransferObjekat to) {
      this.si=StanjeIgreSingltone.getInctanceOfSingltoneClass();
      this.to=to;
   }
```

Класа OpstaPromenaStanjaBaze:

```
public abstract class OpstaPromenaStanjaBaze {
   public BrokerBazePodataka bbp = new BrokerBazePodataka1();
   GeneralDObject gdo;
   TransferObjekat to;
   public OpstaPromenaStanjaBaze(TransferObjekat to) {
      this.to=to;
   }
   public void opstaPromenaStanjaBaze (){
      bbp.makeConnection();
       boolean signal = izvrsiSO();
      if (signal==true)
```

```
        bbp.commitTransation();
      else
       bbp.rollbackTransation();
      bbp.closeConnection();
     }
  abstract public boolean izvrsiSO();
}
```

3.4.2.2 Пројектовање концептуалних решења СО

За сваки од уговора пројектује се концептуално решење.

Уговор УГ1: РегистровањеИграча

Операција: регистровањеИграча(Играч):сигнал
Веза са СК: СК1
Предуслови: Вредносна ограничења над објектом Играч су задовољена.
Постуслови: Подаци о играчу су запамћени.

```
public class RegistrovanjeIgraca extends OpstaPromenaStanjaBaze {
   Igrac igrac;
   public RegistrovanjeIgraca(TransferObjekat to) {
      super(to);
   }
   public void registrovanjeIgraca() {
      opstaPromenaStanjaBaze();
      to.postaviDK(igrac);
   }
   @Override
   public boolean izvrsiSO() {
      List<GeneralDObject> l = bbp.findRecord(new Igrac(), "WHERE korisnickoIme = '" +
to.gdo.korisnickoIme + "'");
```

```
        if (l.size() > 0) {
            return false;
        }
        bbp.insertRecord(to.vratiDK());
        igrac = (Igrac) to.vratiDK();
        return true;
    }
}
```

Уговор УГ2: ПријављивањеИграча

Операција: пријављивањеИграча(Играч):сигнал
Веза са СК: СК2
Предуслови: -
Постуслови: -

```
public class ProveraPrijavljivanja extends OpstaPromenaStanjaBaze {
    Igrac igrac;
    public ProveraPrijavljivanja(TransferObjekat to) {
        super(to);
    }
    public void proveraPrijavljivanja() {
        opstaPromenaStanjaBaze();
        to.postaviDK(igrac);
    }
    @Override
    public boolean izvrsiSO() {
        List<GeneralDObject> l = bbp.findRecord(new Igrac(), "WHERE korisnickoIme = '" +
to.gdo.korisnickoIme + "' and sifra='" + to.gdo.sifra + "'");
        if (l.isEmpty()) {
            return false;
```

```
        }
        igrac = (Igrac) l.get(0);
        return true;
    }
}
```

Уговор УГ3: ИграчНаПотезу

Операција: играчНаПотезу():сигнал;
Веза са СК: СК3
Предуслови: -
Постуслови: -

```
public class IgracNaPotezu extends OpstaPromenaStanjaIgre {
    public IgracNaPotezu(TransferObjekat to) {
        super(to);
    }
    public void igracNaPotezu() {
        to.postaviIgracNaPotezu(si.getIgracNaPotezu());
        si.setIgracNaPotezu(to.vratiIgracNaPotezu());
    }
}
```

Уговор УГ4: АжурирањеМатрице

Операција: ажурирајМатрицу():сигнал;
Веза са СК: СК3
Предуслови: -
Постуслови: Икс-Окс матрица је ажурирана.

```
public class AzuriranjeMatrice extends OpstaPromenaStanjaIgre {
    public AzuriranjeMatrice(TransferObjekat to) {
        super(to);
    }
    public void azurirajMatricu() {
```

```
        si.setPolje(to.vratiPolje());
        int red = Integer.parseInt(to.vratiPolje().substring(3));
        int kolona = Integer.parseInt(to.vratiPolje().substring(2, 3));
        int matrica[][] = si.getMatrica();
        String igrac = to.vratiPolje().substring(0, 1);
        if (igrac.equals("X")) {
            matrica[red][kolona] = 1; // Za X se u matrici upisuje vrednost 1
        }
        if (igrac.equals("O")) {
            matrica[red][kolona] = 2; // Za O se u matrici upisuje vrednost 2
        }
        si.setMatrica(matrica);
    }
}
```

Уговор УГ5: ПроверaКрајИгре

Операција: провериДаЛиЈеКрај():сигнал;
Веза са СК: СК3
Предуслови: -
Постуслови: -

```
public class ProveraKrajIgre extends OpstaPromenaStanjaIgre {
    public ProveraKrajIgre(TransferObjekat to) {
        super(to);
    }
    public void proveriDaLiJeKrajIgre() throws InterruptedException {
        to.postaviPolje(si.getPolje());
        if (si.getMatrica() != null) {
            int[][] matrica = si.getMatrica();
            if (matrica[0][0] == matrica[0][1] && matrica[0][0] == matrica[0][2]
```

```
        && matrica[0][0] != 0 && matrica[0][1] != 0 && matrica[0][2] != 0) {
    si.setKrajIgre(true);
    to.postaviKrajIgre(true);
}
if (matrica[1][0] == matrica[1][1] && matrica[1][0] == matrica[1][2]
        && matrica[1][0] != 0 && matrica[1][1] != 0 && matrica[1][2] != 0) {
    si.setKrajIgre(true);
    to.postaviKrajIgre(true);
}
if (matrica[2][0] == matrica[2][1] && matrica[2][0] == matrica[2][2]
        && matrica[2][0] != 0 && matrica[2][1] != 0 && matrica[2][2] != 0) {
    si.setKrajIgre(true);
    to.postaviKrajIgre(true);
}
if (matrica[0][0] == matrica[1][0] && matrica[0][0] == matrica[2][0]
        && matrica[0][0] != 0 && matrica[1][0] != 0 && matrica[2][0] != 0) {
    si.setKrajIgre(true);
    to.postaviKrajIgre(true);
}
if (matrica[0][1] == matrica[1][1] && matrica[0][1] == matrica[2][1]
        && matrica[0][1] != 0 && matrica[1][1] != 0 && matrica[2][1] != 0) {
    si.setKrajIgre(true);
    to.postaviKrajIgre(true);
}
if (matrica[0][2] == matrica[1][2] && matrica[0][2] == matrica[2][2]
        && matrica[0][2] != 0 && matrica[1][2] != 0 && matrica[2][2] != 0) {
    si.setKrajIgre(true);
```

```
            to.postaviKrajIgre(true);
        }
        if (matrica[0][0] == matrica[1][1] && matrica[0][0] == matrica[2][2]
            && matrica[0][0] != 0 && matrica[1][1] != 0 && matrica[2][2] != 0) {
            System.out.println("PRVA DIJAGONALA");
            si.setKrajIgre(true);
            to.postaviKrajIgre(true);
        }
        if (matrica[2][0] == matrica[1][1] && matrica[2][0] == matrica[0][2]
            && matrica[2][0] != 0 && matrica[1][1] != 0 && matrica[0][2] != 0) {
            si.setKrajIgre(true);
            to.postaviKrajIgre(true);
        }
      }
   }
}
```

Уговор УГ6: АжурирањеСтатистикеИграча

Операција: ажурирајСтатистикуИграча(Играч):сигнал;
Веза са СК: СКЗ
Предуслови: Игра је завршена – постоји добитна комбинација.
Постуслови: Статистика играча је ажурирана.

```
public class AzuriranjeStatistikeIgraca extends OpstaPromenaStanjaBaze {
   public AzuriranjeStatistikeIgraca(TransferObjekat to) {
      super(to);
   }
   public void azurirajStatistikuIgraca() {
      opstaPromenaStanjaBaze();
   }
```

```
    @Override
    public boolean izvrsiSO() {
        bbp.updateRecord(to.vratiDK());
        return true;
    }
}
```

Уговор УГ7: ПрикажиСтатистикуИграча

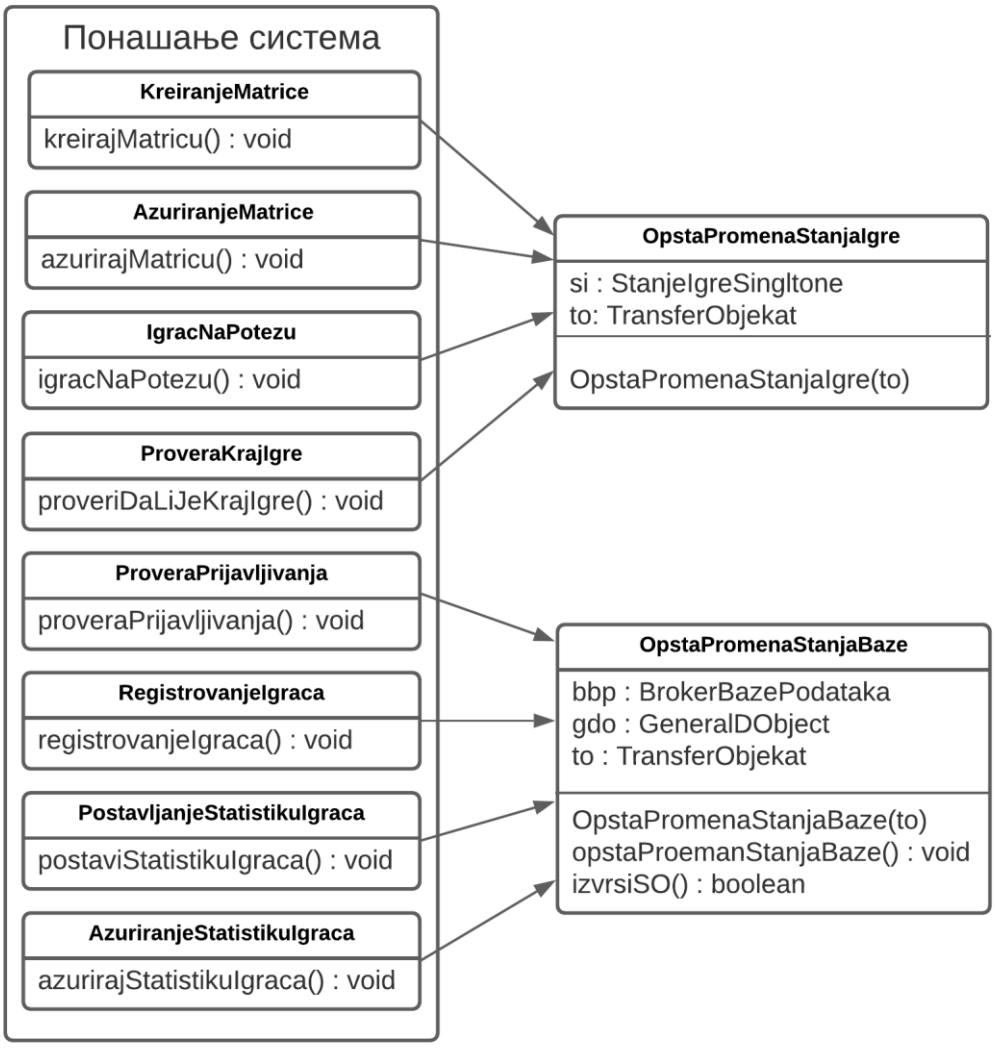
Операција: прикажиСтатистикуИграча(Играч):сигнал;
Веза са СК: СК4
Предуслови: -
Постуслови: -

```
public class PostaviStatistikuIgraca extends OpstaPromenaStanjaBaze {
    Igrac igrac;
    public PostaviStatistikuIgraca(TransferObjekat to) {
        super(to);
    }
    public void postaviStatistikuIgraca() {
        opstaPromenaStanjaBaze();
        to.postaviDK(igrac);
    }
    @Override
    public boolean izvrsiSO() {
        List<GeneralDObject> l=bbp.findRecord(new Igrac(), "WHERE korisnickoIme = '" +
to.gdo.korisnickoIme+"'");
        igrac = (Igrac) l.get(0);
        return true;
    }
}
```

Уговор УГ8: ПокрениНовуИгру

Операција: покрениНовуИгру():сигнал;
Веза са СК: СК5
Предуслови: -
Постуслови: Икс-Окс матрица је ажурирана.

```
public class KreiranjeMatrice extends OpstaPromenaStanjaIgre{
   public KreiranjeMatrice(TransferObjekat to) {
      super(to);
   }
    public void kreirajMatricu() {
      int[][] matrica = new int[3][3];
      for (int i = 0; i < matrica.length; i++) {
         for (int y = 0; y < matrica[0].length; y++) {
            matrica[i][y] = 0;
         }
         si.setMatrica(matrica);
         si.setPolje("");
         si.setIgracNaPotezu("nova-igra");
         si.setKrajIgre(false);
      }
   }
}
```



Слика 10. Системске операције

3.4.2.3 Пројектовање логичке структуре система

Структура софтверског система класа StanjeIgreSingltone и класе Igrac која наслеђује апстрактну класу GeneralDObject.

Класа StanjeIgreSingltone представља Синглтон класу (имплементирана уз помоћ Синглтон патерна) чија је улога да чува тренутно стање игре (неперзистентни подаци), док класа Igrac садржи неопходне податке о играчу, који се чувају и у складишту података (перзистентни подаци).

```
public class StanjeIgreSingltone {
    private String igracNaPotezu = "";
    private String polje = "";
    private int[][] matrica = null;
    private boolean krajIgre = false;
    static StanjeIgreSingltone sc = new StanjeIgreSingltone();
    private StanjeIgreSingltone() {
    }
    public static StanjeIgreSingltone getInctanceOfSingltoneClass() {
        return sc;
    }
    public String getIgracNaPotezu() {
        return igracNaPotezu;
    }
    public void setIgracNaPotezu(String s) {
       if (s.equals("X")) {
            igracNaPotezu = "O";
        }
        if (s.equals("O")) {
            igracNaPotezu = "X";
        }
        if (s.equals("")) {
            igracNaPotezu = "O";
        }
        if (s.equals("nova-igra")) {
            igracNaPotezu = "";
        }
```

```
    }
    public String getPolje() {
        return polje;
    }
    public void setPolje(String polje) {
        this.polje = polje;
    }
    public int[][] getMatrica() {
        return matrica;
    }
    public void setMatrica(int[][] matrica) {
        this.matrica = matrica;
    }
    public boolean getKrajIgre() {
        return krajIgre;
    }
    public void setKrajIgre(boolean krajIgre) {
        this.krajIgre = krajIgre;
    }
}
```

Класа GeneralDObject:

```
public abstract class GeneralDObject implements Serializable
{ abstract public String getAtrValue();
  abstract public String setAtrValue();
  abstract public String getClassName();
  abstract public String getWhereCondition();
```

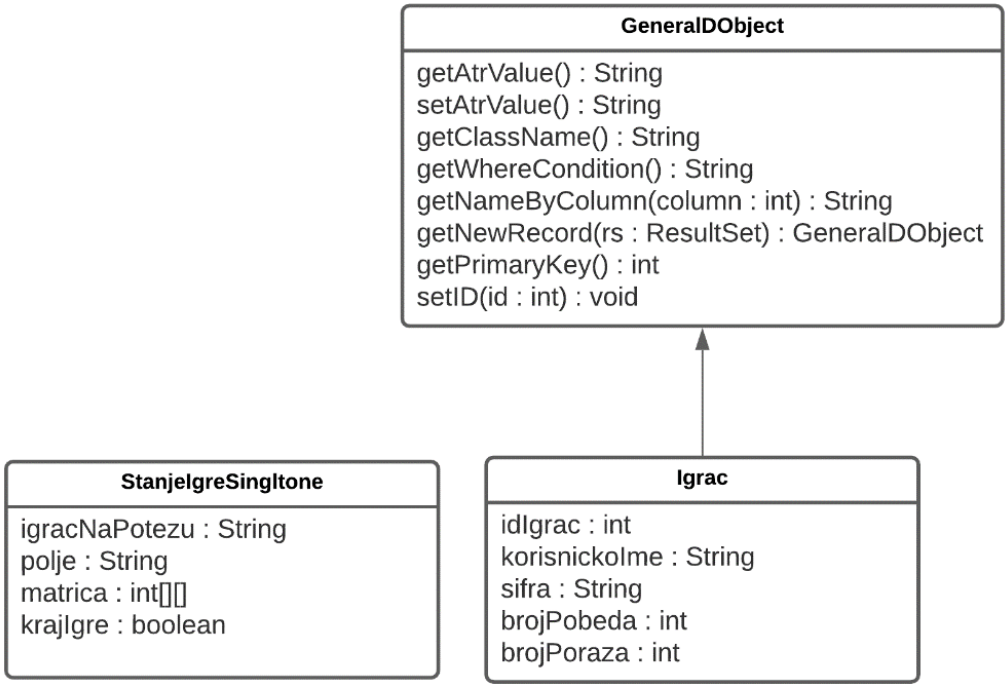
```
  abstract public String getNameByColumn(int column);
  abstract public GeneralDObject getNewRecord(ResultSet rs) throws SQLException;
  abstract public int getPrimaryKey();
  abstract public void setID(int id);
}
```

Класа Igrac:

```
public class Igrac extends GeneralDObject implements Serializable{
    public int idIgrac;
    public String korisnickoIme;
    public String sifra;
    public int brojPobeda;
    public int brojPoraza;
    public Igrac() {
        korisnickoIme = "";
        sifra = "";
        brojPobeda = 0;
        brojPoraza = 0;
    }
    public Igrac(int idIgrac, String korisnickoIme, String sifra, int brojPobeda, int brojPoraza) {
        this.idIgrac = idIgrac;
        this.korisnickoIme = korisnickoIme;
        this.sifra = sifra;
        this.brojPobeda = brojPobeda;
        this.brojPoraza = brojPoraza;
    }
    // primarni kljuc
```

```
    public Igrac(int idKorisnik) {
        this.idIgrac = idKorisnik;
    }
    public void setID(int id) {
        this.idIgrac = id;
    }
    public int getPrimaryKey() {
        return this.idIgrac;
    }
    @Override
    public GeneralDObject getNewRecord(ResultSet rs) throws SQLException {
        return new Igrac(rs.getInt("idIgrac"), rs.getString("korisnickoIme"), rs.getString("sifra"),
rs.getInt("brojPobeda"), rs.getInt("brojPoraza"));
    }
    @Override
    public String getAtrValue() {
        return idIgrac + ", " + (korisnickoIme == null ? null : "'" + korisnickoIme + "'") + ", " + (sifra ==
null ? null : "'" + sifra + "') +  ", " + brojPobeda + ", " + brojPoraza;
    }
    @Override
    public String setAtrValue() {
        return "idIgrac=" + idIgrac + ", " + "korisnickoIme=" + (korisnickoIme == null ? null : "'" +
korisnickoIme + "'") + ", " + "sifra=" + (sifra == null ? null : "'" + sifra + "'") + ", " + "brojPobeda="
+ brojPobeda + ", " + "brojPoraza=" + brojPoraza;
    }
    @Override
    public String getClassName() {
        return "Igrac";
```

```
    }
    @Override
    public String getWhereCondition() {
        return "idIgrac = " + idIgrac;
    }
    @Override
    public String getNameByColumn(int column) {
        String names[] = {"idIgrac", "korisnickoIme", "sifra", "brojPobeda", "brojPoraza",
"datumRodjenja"};
        return names[column];
    }
    public String[] getNameAtributes() {
     String names[] = {"idIgrac", "korisnickoIme", "sifra", "brojPobeda", "brojPoraza",
"datumRodjenja"};
        return names;
    }
}
```



Слика 11. Логичка структура система

3.4.3 Брокер базе података

Брокер базе података има улогу да омогући перзистентност података односно да омогући трансформацију објеката у слогове базе (дематеријализација) и трансформација слогова у објекте (материјализација).

У оквиру пројекта имплементирана је класа BrokerBazePodataka која наслеђује генерички брокер базе података назван OpstiBrokerBazePodataka.

Интерфејс OpstiBrokerBazePodataka:

```
public interface OpstiBrokerBazePodataka
{  boolean makeConnection();
   boolean insertRecord(GeneralDObject odo);
   boolean updateRecord(GeneralDObject odo);
   List<GeneralDObject> findRecord(GeneralDObject odo,String where);
   boolean commitTransation();
```

```
    boolean rollbackTransation();
    void closeConnection();
}
```

Класа BrokerBazePodataka:

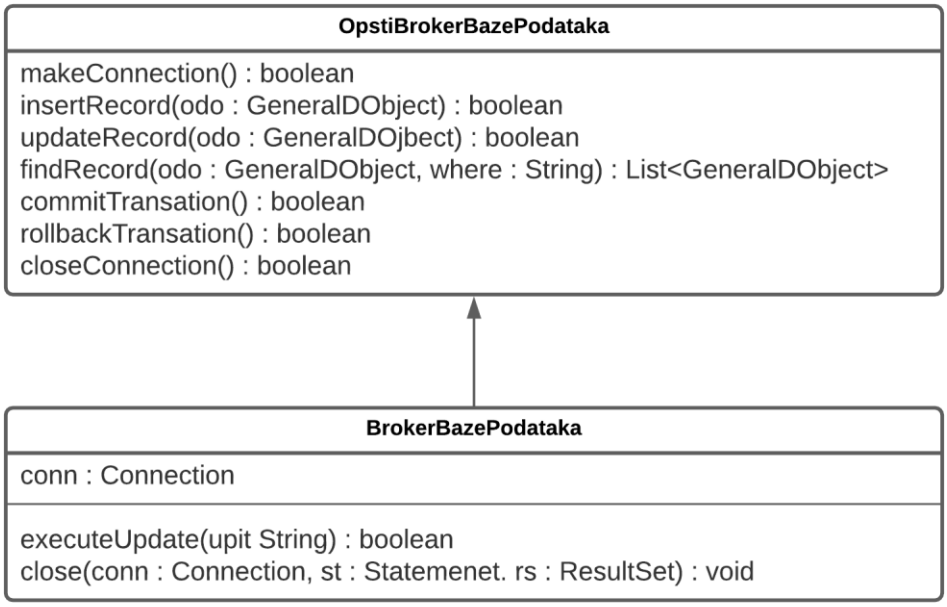
```
public class BrokerBazePodataka implements OpstiBrokerBazePodataka {
    Connection conn = null;
    @Override
    public boolean makeConnection() {
        String Url;
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Url = "jdbc:mysql://127.0.0.1:3306/IksOks";
            conn = DriverManager.getConnection(Url, "root", "root");
            conn.setAutoCommit(false);
        } catch (SQLException | ClassNotFoundException ex) {
            Logger.getLogger(BrokerBazePodataka.class.getName()).log(Level.SEVERE, null, ex);
            return false;
        }
        return true;
    }
    @Override
    public boolean insertRecord(GeneralDObject odo) {
        String upit = "INSERT INTO " + odo.getClassName() + " VALUES (" + odo.getAtrValue() + ")";
        return executeUpdate(upit);
    }
    @Override
    public boolean updateRecord(GeneralDObject odo) {
```

```
    String upit = "UPDATE " + odo.getClassName() + " SET " + odo.setAtrValue() + " WHERE " +
odo.getWhereCondition();
    return executeUpdate(upit);
  }
  public boolean executeUpdate(String upit) {
    Statement st = null;
    boolean signal = false;
    try {
      st = conn.prepareStatement(upit);
      int rowcount = st.executeUpdate(upit);
      if (rowcount > 0) {
        signal = true;
      }
    } catch (SQLException ex) {
      Logger.getLogger(BrokerBazePodataka.class.getName()).log(Level.SEVERE, null, ex);
      signal = false;
    } finally {
      close(null, st, null);
    }
    return signal;
  }
  @Override
  public List<GeneralDObject> findRecord(GeneralDObject odo, String where) {
    ResultSet rs = null;
    Statement st = null;
    String upit = "SELECT * FROM " + odo.getClassName() + " " + where;
    List<GeneralDObject> ls = new ArrayList<>();
```

```
    boolean signal;
    try {
        st = conn.prepareStatement(upit);
        rs = st.executeQuery(upit);
        while (rs.next()) {
            ls.add(odo.getNewRecord(rs));
        }
    } catch (SQLException ex) {
        Logger.getLogger(BrokerBazePodataka.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        close(null, st, rs);
    }
    return ls;
}
@Override
public boolean commitTransation() {
    try {
        conn.commit();
    } catch (SQLException esql) {
        return false;
    }
    return true;
}
@Override
public boolean rollbackTransation() {
    try {
        conn.rollback();
```

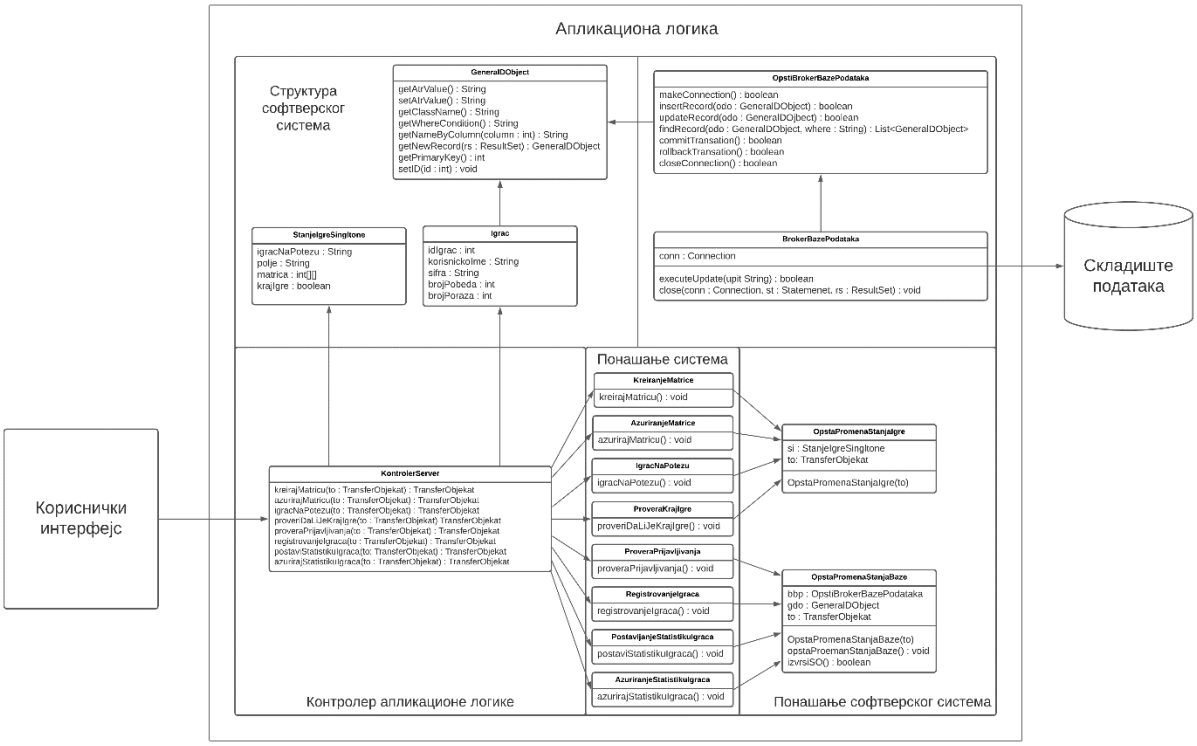
```
    } catch (SQLException esql) {
        return false;
    }
    return true;
}
@Override
public void closeConnection() {
    close(conn, null, null);
}
public void close(Connection conn, Statement st, ResultSet rs) {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            Logger.getLogger(BrokerBazePodataka.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    if (st != null) {
        try {
            st.close();
        } catch (SQLException ex) {
            Logger.getLogger(BrokerBazePodataka.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    if (conn != null) {
        try {
            conn.close();
```

```
        } catch (SQLException ex) {
            Logger.getLogger(BrokerBazePodataka.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
  }
}
```



Слика 1. Брокер базе података

Након пројектовања комплетне апликационе логике добија се следећи дијаграм:



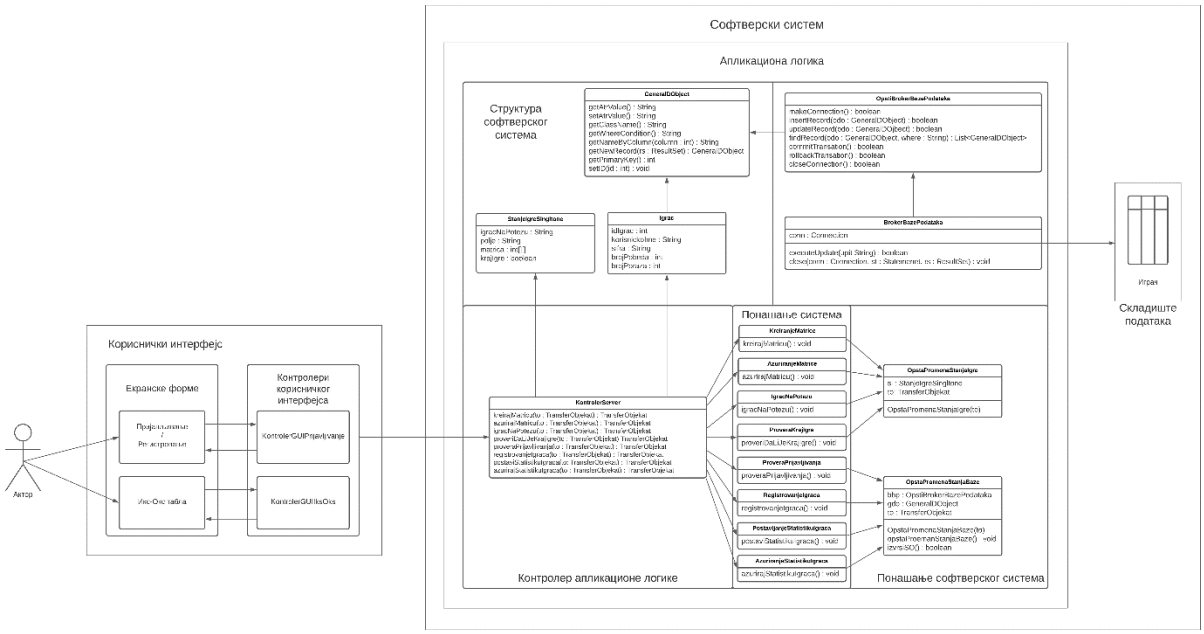
Слика 13. Архитектура софтверског система након пројектовања апликационе логике

3.5 Пројектовање складишта података

На основу софтверске класе Igrac пројектована је табела Igrac у релационој бази података. У раду је коришћен „MySQL“.

Табела Igrac		
Име поља	Тип	Величина
idIgrac	INT	
korisnickoIme	VARCHAR	45
sifra	VARCHAR	45
brojPobeda	INT	
brojPoraza	INT	

Након што су пројектовани сви нивои тронивојске софтверске архитектуре, могуће је представити коначну архитектуру система.



Слика 14. Коначна архитектура

3.6 Принципи, методе и стратегије пројектовања софтвера

3.6.1 Принципи (технике) пројектовања софтвера

Апстракција

Апстракција подразумева издвајање општих информација од специфичних. Опште информације се свесно издвајају како би се нагласила сама суштина неке појаве, без улажења у њене специфичности и различита могућа појављивања.

У контексту софтверског пројектовања разликују се два механизма апстракције:

1. параметризација
2. спецификација

Спецификација даље води до три главне врсте апстракција:

1. процедурална апстракција
2. апстракција података
3. апстракција контролом

Параметризација

Параметризација је механизам апстракције који издваја нека општа својства појаве из скупа елемената, при чему та општа својства представља помоћу параметара.

Разликују се 5 случајева параметризације:

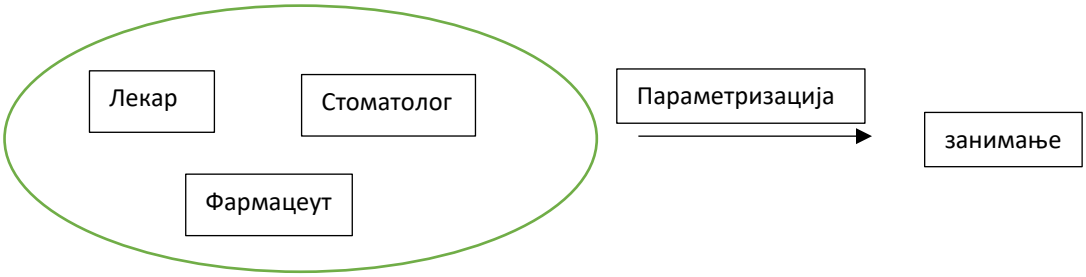
1. параметризација скупа елемената простог тима
2. параметризација скупа елемената сложеног типа
3. параметризација скупа операција
4. параметризација скупа процедура
5. параметризација скупа наредби

Параметризација скупа елемената простог типа

Уколико имамо скуп разлучитих занимања (послова), њих можемо представити преко параметра простог типа, који ће имати улогу општег представника свих занимања.

Програмским кодом то би могло да се представи на следећи начин:

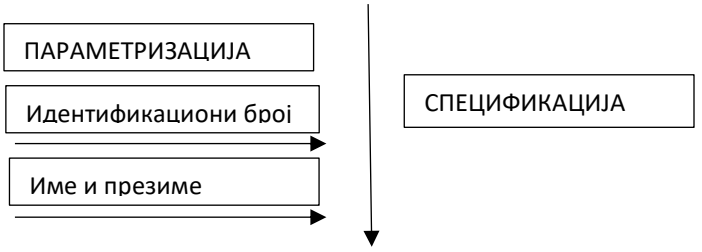
String zanimanje;



Параметризација скупа елемената сложеног типа

Уколико имамо скуп фармацеута, при чему сваки фармацеут поседује идентификациони број, као и име и презиме, тада се параметризацијом добијаји њихова општа својства, односно атрибути. Параметризација се ради за сваки скуп вредности атрибута, при чему се скуп идентификационих бројева фармацеута параметризује преко својства *Идентификациони број*, док се име и презиме параметризују преко својства *Име и презиме.* Наведени скуп фармацеута је означен са ф1, ф2, ф3.

ф1	ф2	ф3
1111	2222	3333
Петар Петровић	Милош Милошевић	Милица Милићевић

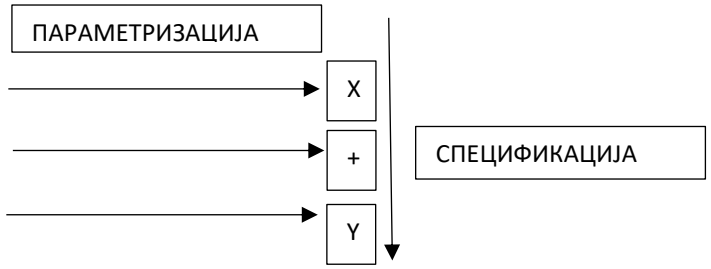


Параметризацијом долазимо до општих својстава елеманата скупа, а тиме заправо вршимо спецификацију скупа.

Параметризација скупа података

Уколико имамо скуп неких конкретних операција, тада се параметризацијом добијају општа својства наведеник операција. То се може објаснити на примеру операције сабирања два броја, чији су параметри операнди бројеви, а оператор знак сабирања који нам говори шта та операција ради. Наведени скуп конкретних операција наведен је са оп1, оп2, оп3.

оп1	оп2	оп3
2	9	11
+	+	+
5	7	2

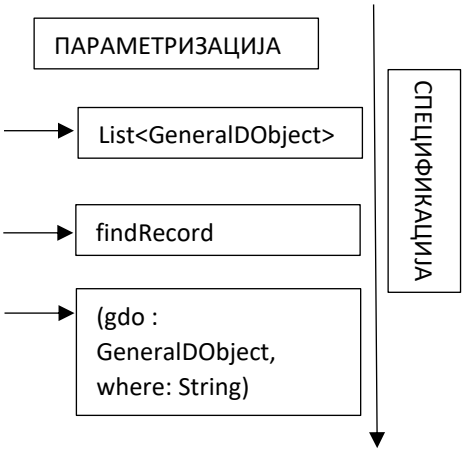


Параметризацијом, односно навођењем општих својстава операција врши се спецификација операција, при чему се добија општа операција, а у наведеном примеру општа операција за сабирање два броја.

Параметризација скупа процедура

Параметризацијом скупа процедура се добијају општа својства процедура. Елементи процедуре су: назив процедуре, шта процедура враћа, аргументи процедуре и тело процедуре. Наведени скуп процедура је означен са пр1, пр2, пр3.

	пр1	пр2	пр3
шта процедура враћа	List<GeneralDObject>	List<GeneralDObject>	List<GeneralDObject>
назив процедуре	findRecord	findRecord	findRecord
аргументи процедуре	(gdo : GeneralDObject, where: String)	(gdo : GeneralDObject, where: String)	(gdo : GeneralDObject, where: String)
тело процедуре	метода која враћа листу играча на основу корисничког имена	метода која враћа листу играча на основу ID играча	метода која враћа листу играча на основу броја победа



Параметризација скупа наредби

Уколико имамо скуп наредби:

System.out.print(1 + “. red ”+ matrica[0]);

System.out.print(2+ “. red ”+ matrica[1]);

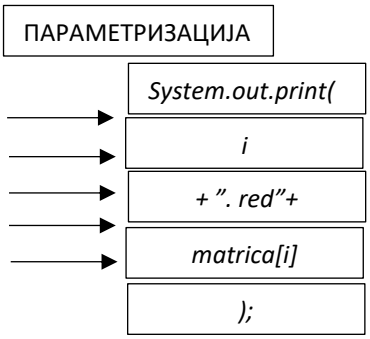
System.out.print(3+ “. red ”+ matrica[2]);

које приказују елементе низа, тада се параметризацијом добијају општа својства наведених наредби, односно параметри

1. *System.out.print(*
2. *i*
3. *+ ”. red”+*
4. *matrica[i]*
5. *);*

Наведени скуп наредби означен је са на1, на2, на3.

на1	на2	на2
System.out.print(	*System.out.print(*	*System.out.print(*
1	2	3
+ ”. red”+	*+ ”. red”+*	*+ ”. red”+*
matrica[1]	*matrica[2]*	*matrica[3]*
);	);	);



Спецификација

Спецификација представља апстракцију која издваја издваја општа својства из скупа елемената, која могу бити представљена преко процедуре, податка или контроле.

Процедурална апстракција

Из скупа процедура процедуларном апстракцијом се издвајају њихова општа својства.

1. тип онога што враћа процедура
2. име процедуре
3. аргументи процедуре

Процедуралном апстракцијом се добија потпис процедуре, а поред потписа процедуре може да обухвати и детаљни опис шта ради и који су услови извршења процедуре. Према

упрошћеној Ларманової методи, општа процедурална апстракција садржи: потпис процедуре, веза са СК, предуслови и постуслови.

Као пример општег случаја процедуралне апстракције може се навести уговор о системској операцији РегистровањеИграча(Играч).

Потпис процедуре: регистровањеИграча(Играч) : сигнал;
Веза са СК: СК1
Предуслови: Вредносна и структурна ограничења над објектом Играч су задовољена.
Постуслови: Подаци о играчу су запамћени.

Апстракција података

Апстракцијом података се из скупа података издвајају њихова општа својства.

Као што је у ранијем примеру већ показану, ако имамо скуп фармацеута, тада се параметризацијом добијају њихова општа својства, ондосно атрибути: *Идентификациони број и Име и презиме.*

У пројекту је апстракција података извршена над објектом Играч, који такође садржи Име и презиме као својство (атрибут). Апстракција података је дефинисана именом (Играч) скупа и спецификацијом скупа (Име и презиме).

Уколико објекти, односно елементи скупа, садрже и понашање поред структура, онда се над њима ради и процедурална апстракција, а као резултат тога се добија класа која садржи атрибуте и процедуре (методе).

Уколико елементи скупа не садрже структуру, већ само понашање, онда се над њима врши само процедурална апстракција, а као резултат тога се добија интерфејс.

Спојеност и кохезија

Два циља која је потребно постићи у развоју објектно-оријентисаног софтвера су:

1. класе треба да имају велику кохезију
2. класе треба да буду слабо повезане

Кохезија

Кохезија представља меру којом се утврђује колико су методе унутар неке класе повезане. За класу се каже да има високу кохезију ако у реализацији њеног понашања метода1(), учествују остале методе класе (метода11(), метода12()...). Губитак кохезије значи да је одређена класа одговорна да обезбеди више различитих понашања метода1(), метода2(), метода3(), која нису међусобно повезана. Таква класа је тешка за одржавање и надоградњу.

Као пример класе са високом класом навешћу апстрактну класу *OpstaPromenaStanjaBaze* и њено понашање *opstaPromenaStanjabaze()* које обезбеђује. Поред овог понашања, поменута класа садржи још једну методу која је у функцији основног понашања класе.

```
public abstract class OpstaPromenaStanjaBaze {
   public OpstiBrokerBazePodataka bbp = new BrokerBazePodataka();
   GeneralDObject gdo;
   TransferObjekat to;
   public OpstaPromenaStanjaBaze(TransferObjekat to) {
      this.to=to;
   }
   public void opstaPromenaStanjaBaze (){
      bbp.makeConnection();
       boolean signal = izvrsiSO();
      if (signal==true)
        bbp.commitTransation();
      else
       bbp.rollbackTransation();
      bbp.closeConnection();
     }
   abstract public boolean izvrsiSO();
}
```

Спојеност

Спојеност (Куповање) представља меру повезаности, односно зависности класа. Спојеношћу се одређује колико је једна класа зависна од друге класе, а приликом пројектовања и имплементирања класа потребно је правити класе са што мањом повезаношћу. Класе које имају велику спојеност је тешко посматрати као изоловане, промене у једној класи утицаће и на класу са којом је повезана и тешко је такве класе поново користити.

Као пример класе са слабом спојеношћу може се навести класа *ProveraPrijavljivanja* која је зависна само од једне друге класе. Слично је и са осталим класама које наслеђују апстрактну класу OpstaPromenaStanjaBaze.

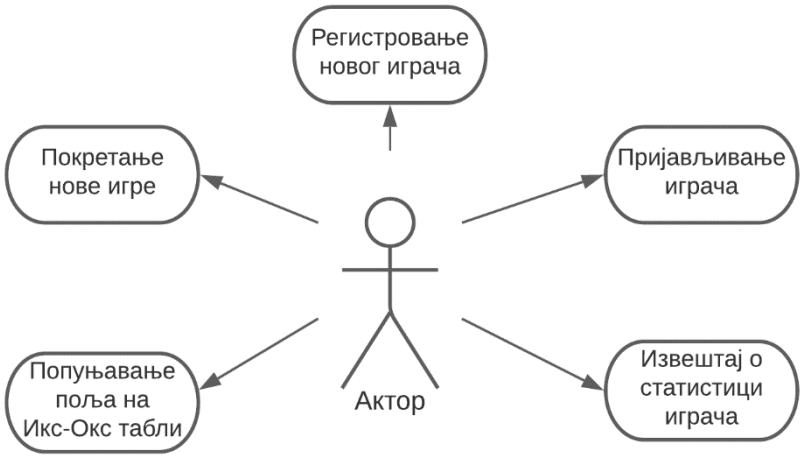
```
public class ProveraPrijavljivanja extends OpstaPromenaStanjaBaze {
   Igrac igrac;
   public ProveraPrijavljivanja(TransferObjekat to) {
      super(to);
   }
   public void proveraPrijavljivanja() {
      opstaPromenaStanjaBaze();
      to.postaviDK(igrac);
   }
   @Override
   public boolean izvrsiSO() {
      List<GeneralDObject> l = bbp.findRecord(new Igrac(), "WHERE korisnickoIme = '" +
to.gdo.korisnickoIme + "' and sifra='" + to.gdo.sifra + "'");
      if (l.isEmpty()) {
         return false;
      }
      igrac = (Igrac) l.get(0);
      return true;
   }
}
```

Декомпозиција и модуларизација

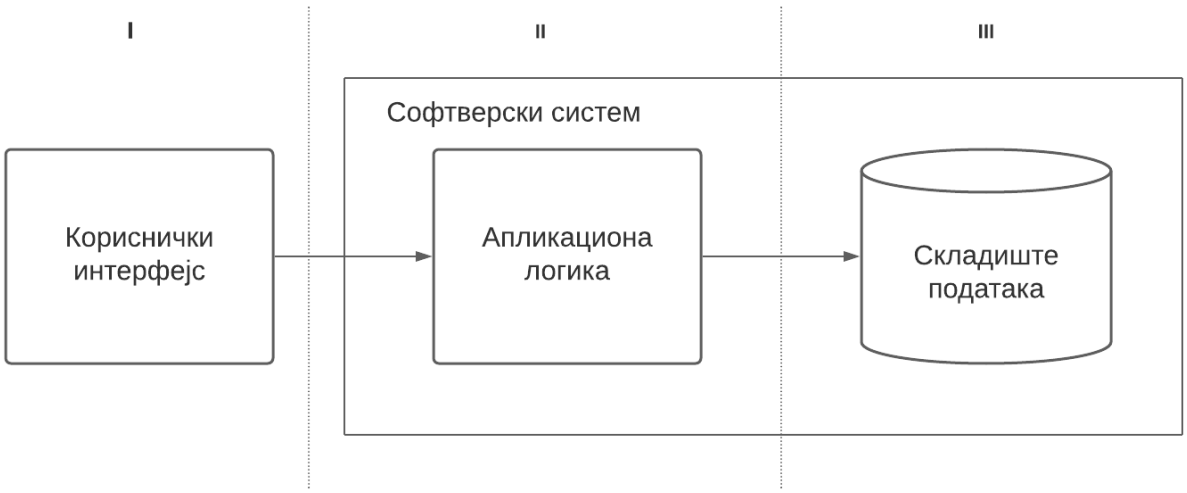
Декомпозиција представља процес рашчлањивања почетног проблема на скуп подпроблема који се независно решавају. На овај начин се олакшава решавање почетног проблема. Као резултат процеса декомпозиције софтверског система настаје модуларизација јер се систем дели у више модула.

Декомпозиција се може посматрати из неколико различитих аспеката развоја софтвера:

1. Декомпозиција код прикључупљања захтева – кориснички захтеви се разлажу на мање скупове захтева који се затим представљају преко случајева коришћења



2. Декомпозиција код пројектовања софтвера – тринивојска архитектура је пример разлагања софтвера на три одвојена модула (кориснички интерфејс, апликациона логика, складиште података)



3. Декомпозиција функција (метода)

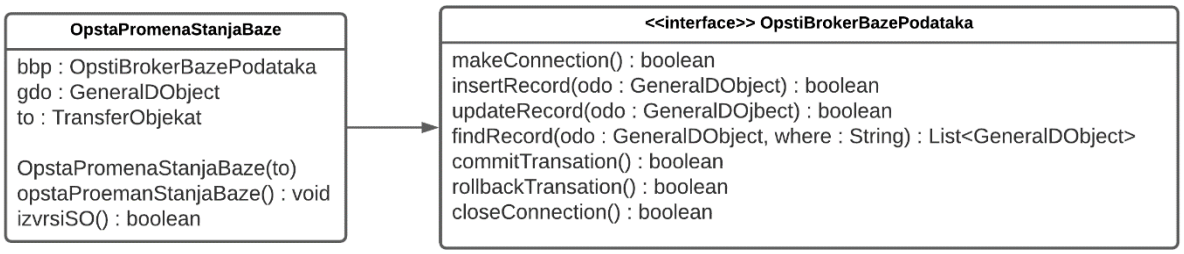
Учаурење/Сакривање информација

Учаурење је процес у коме се врши раздвајање особина модула које су доступне другим модулима, од особина које су приватне и могу се користити само унутар датог модула. Сакривање информација настаје као резултат процеса учаурења.

```
public class StanjeIgreSingltone {
...
//приватна особина класе
   private boolean krajIgre = false;
//јавна особина класе
public boolean getKrajIgre() {
      return krajIgre;
...
   }
```

Одвајање интерфејса и имплементације

Интерфејс се одваја од имплементације и излаже кориснику, при чему корисник не зна како су операције интерфејса имплементиране. Као пример може се навести интерфејс OpstiBrokerBazePodataka, а како корисник апстрактна класа OpstaPromenaStanjaBaze.



3.6.2 Стратегије пројектовања софтвера

Неке од стратегија пројектовања софтвера су:

1. Подели и победи
2. С врха на доле
3. Одоздо нагоре
4. Итеративно – инкрементални приступ

Подели и победи

Стратегија заснована на принципу декомпозиције, односно разлагању почетног проблема на подскуп проблема како би се лакше решио почетни проблем.

С врха на доле

Стратегија заснована на принципу декомпозиције функција, односно разлагању почетне функције на подскуп функција које се независно извршавају, а које се на крају интегришу у целину како би се реализовала почетна функција.

Одоздо на горе

Стратегија заснована на принципу генерализације. У једној сложеној функцији уочава се целина која се затим изолује у засебну функцију, чиме се сложена функција декомпонује на више независних целина.

Итеративно – инкерементални приступ

Стратегија заснована на дељењу пројекта на мини пројекте који се затим независно развијају и пролазе кроз све фазе пројектовања софтвера. Мини пројекти пролазе кроз више итерација, а као резултат итерација се добија интерно издање (*release*) или build који представља инкремент за систем. На крају се тако имплементирани мини пројекти спајају у целину (софтверски систем).

3.6.3 Методе пројектовања софтвера

Неке од најважнијих метода пројектовања софтвера су:

1. Функционално оријентисано програмирање – засновано на функцијама. Систем се посматра из угла његовог понашања.
2. Објектно оријентисано програмирање – засновано на објектима, при чему објекти могу да репрезентују и структуру и понашање система.
3. Пројектовање засновано на структури података – засновано на структури, односно прво се уочава структуру система, а тек онда функције које се врше на том структуром.
4. Пројектовање засновано на компонентама – уочавају се већ постојеће компоненте које се могу поново искористити за решање проблема, а тек онда се имплементирају компоненте за које није постојало већ готово решење.

Примена патерна у пројектовању

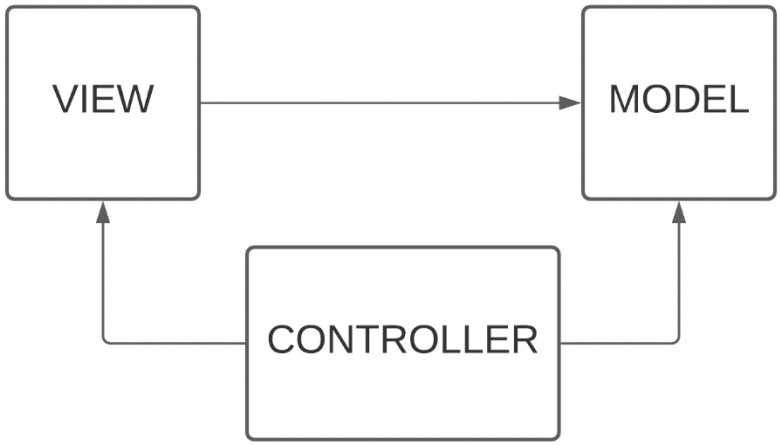
Патерни представљају решење неког проблема, које се може поново искористити за решавање неких нових проблема. Једна од основних карактеристика патерна јесте поновна употребљивост.

Улога патерна је да нам помогну у одржавању и надоградњи софтверског система. Архитектура софтверског система се састоји од компонентни које су међусобно повезане преко њихових интерфејса. Разликују се макро и микро архитектура. Макро архитектура је реализована преко EFC и MVC патерна, док је микро архитектура реализована преко патерна пројектовања и то: креационих, структурних и патерна понашања. Поред ових патерна, постоје и патерни који покривају друге фазе развоја софтверског система. Патерни пројектовања су независни од технологије у којој се имплементирају.

MVC (Model – View – Controller) патерн

MVC патерн представља макроархитектурни патерн, а који дели софтверски систем на три дела, и то како се и из самог имена наслућује, на:

1. view – представља кориснички интерфејс, односно екранске форме, које обезбеђују кориснику унос података и позивање одговарајућих системских операција, као и визуелни приказ стања модела
2. controller – прима захтеве од клијента и прослеђује их на даље извршење и обраду. У случају да дође до промене стање модела он обавештава view да је стање промењено.
3. model – односи се на стање система, које мењају системске операције које корисник позива.



Слика 2. MVC архитектура

Патерни пројектовања микро архитектуре

1. Креациони патерни – помажу да се изгради систем независно од тога како су објекти креирани, компоновани и репрезентовани
2. Структурни патерни – описују сложене структуре међусобно повезаних класа и објеката
3. Патерни понашања – описују начин на који класе или објекти сарађују

У оквиру пројекта је коришћен креациони патерн *Singltone* за креирање класе StanjeIgreSingltone како би се омогућило само једно појављивање класе и глобални приступ до ње, како би све класе системских операција приступале идентичном објекту, односно тренутном стању игре.

```
public class StanjeIgreSingltone {
   private String igracNaPotezu = "";
   private String polje = "";
   private int[][] matrica = null;
   private boolean krajIgre = false;
   static StanjeIgreSingltone sc = new StanjeIgreSingltone();
   private StanjeIgreSingltone() {
   }
   public static StanjeIgreSingltone getInctanceOfSingltoneClass() {
      return sc;
   }
   public String getIgracNaPotezu() {
      return igracNaPotezu;
   }
   public void setIgracNaPotezu(String s) {
     if (s.equals("X")) {
         igracNaPotezu = "O";
      }
```

```
    if (s.equals("O")) {
        igracNaPotezu = "X";
    }
    if (s.equals("")) {
        igracNaPotezu = "O";
    }
    if (s.equals("nova-igra")) {
        igracNaPotezu = "";
    }
}
public String getPolje() {
    return polje;
}
public void setPolje(String polje) {
    this.polje = polje;
}
public int[][] getMatrica() {
    return matrica;
}
public void setMatrica(int[][] matrica) {
    this.matrica = matrica;
}
public boolean getKrajIgre() {
    return krajIgre;
}
public void setKrajIgre(boolean krajIgre) {
    this.krajIgre = krajIgre;
```

```
    }
}
```

Како би се избегло вишеструко понављање кода приликом извршења системских операција које мењају стање базе података, искоришћен је и *Template* патерн који садржи све оне делове кода који се понављају у свакој од системских операција задужених за промену стања базе (отварање конекције ка бази, потврда трансакције, поништење трансакције).

```
public abstract class OpstaPromenaStanjaBaze {
    public OpstiBrokerBazePodataka bbp = new BrokerBazePodataka();
    GeneralDObject gdo;
    TransferObjekat to;
    public OpstaPromenaStanjaBaze(TransferObjekat to) {
        this.to=to;
    }
    public void opstaPromenaStanjaBaze (){
        bbp.makeConnection();
         boolean signal = izvrsiSO();
        if (signal==true)
          bbp.commitTransation();
        else
         bbp.rollbackTransation();
        bbp.closeConnection();
       }
    abstract public boolean izvrsiSO();
}
```

4 Генеричко програмирање

4.1 Рефлексија

Рефлексија је механизам који који омогућава добијање основних информација о класи и свим њеним чланицама, при чему се дате информације називају метаподаци који се у случају објектно оријентисаног програмирања чувају у мета-објектима.

Пример:

```
public static boolean konvertujGUIUDK(FXMLDocumentController fxcon, GeneralDObject gdo) {
    for (Field f : fxcon.getClass().getDeclaredFields()) {
        for (Field dk : gdo.getClass().getDeclaredFields()) {
            dk.setAccessible(true);
            if (dk.getName().equals(f.getName())) {
                if (f.getType().getName().equals("javafx.scene.control.TextArea") &&
dk.getType().getName().equals("java.lang.String")) {
                    try {
                        dk.set(gdo, ((javafx.scene.control.TextArea) f.get(fxcon)).getText());
                    } catch (IllegalArgumentException | IllegalAccessException ex) {
                        Logger.getLogger(KonverterGUIDK.class.getName()).log(Level.SEVERE, null, ex);
                        return false;
                    }
                }
            }
        }
    }
    return true;
}
```

5 Литература

1. др Синиша Влајић, *Софтверски процес (скрипта),* Београд, 2016