УНИВЕРЗИТЕТ У БЕОГРАДУ

ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

**ЗАВРШНИ РАД**

Тема: Развој софтверског система за праћење спортских такмичења у JAVA окружењу

Ментор: Студент:

Проф. др Синиша Влајић Арацки Иван 146/11

Београд, 2016. Године

Садржај:

[Попис дијаграма: 3](#_Toc440824128)

[Попис слика: 4](#_Toc440824129)

[Попис табела: 4](#_Toc440824130)

[1. Увод 6](#_Toc440824131)

[2. Преглед коришћене технологије 8](#_Toc440824132)

[2.1. Јава платформа 8](#_Toc440824133)

[2.1.1. Java 2 EE платформа 9](#_Toc440824134)

[2.2. Алат за изградњу пројекта - Maven 9](#_Toc440824135)

[2.3. Веб сервиси 10](#_Toc440824136)

[2.3.1. REST Web сервиси 11](#_Toc440824137)

[2.4. Клијентска страна – Bootstrap, jQuery 19](#_Toc440824138)

[3. Студијски пример 21](#_Toc440824139)

[3.1. Кориснички захтеви 21](#_Toc440824140)

[3.1.1. Вербални опис 21](#_Toc440824141)

[3.1.1. Случајеви коришћења 21](#_Toc440824142)

[СК 1: Случај коришћења – Унос члана DVD клуба 23](#_Toc440824143)

[СК 2: Случај коришћења – Измена члана DVD клуба 24](#_Toc440824144)

[СК 3: Случај коришћења – Брисање члана DVD клуба 25](#_Toc440824145)

[СК 4: Случај коришћења – Унос задужења 26](#_Toc440824146)

[СК 5: Случај коришћења – Измена задужења 27](#_Toc440824147)

[СК 6: Случај коришћења – Претрага задужења 28](#_Toc440824148)

[СК 7: Случај коришћења – Пријављивање радника 29](#_Toc440824149)

[3.2. Анализа 30](#_Toc440824150)

[3.2.1. Понашање софтверског система – Системски дијаграм секвенци 30](#_Toc440824151)

[3.2.2. Понашање софтверског система – Дефинисање уговора о системским операцијама 44](#_Toc440824152)

[3.2.3. Структура софтверског система – Концептуални (доменски) модел 47](#_Toc440824153)

[3.2.4. Структура софтверског система – Релациони модел 47](#_Toc440824154)

[3.3. Пројектовање 53](#_Toc440824155)

[3.3.1. Пројектовање корисничког интерфејса 53](#_Toc440824156)

[3.3.2. Пројектовање контролера корисничког интерфејса 67](#_Toc440824157)

[3.3.3. Пројектовање апликационе логике 67](#_Toc440824158)

[3.3.4. Пројектовање складишта података 80](#_Toc440824159)

[3.3.5. Архитектура софтверског система 82](#_Toc440824160)

[3.5. Имплементација 83](#_Toc440824162)

[3.6. Тестирање 85](#_Toc440824163)

[4. Закључак 86](#_Toc440824164)

[5. Литература 87](#_Toc440824165)

# Попис дијаграма:

[Дијаграм 1: Дијаграм случајева коришћења 22](#_Toc440822714)

[Дијаграм 2: Убацивање новог члана – основни сценарио 30](#_Toc440822715)

[Дијаграм 3: Убацивање новог члана – алтернативни сценарио 31](#_Toc440822716)

[Дијаграм 4: Измена члана – основни сценарио 32](#_Toc440822717)

[Дијаграм 5: Измена члана – алтернативни сценарио 33](#_Toc440822718)

[Дијаграм 7: Брисање члана – алтернативни сценарио 35](#_Toc440822720)

[Дијаграм 8: Унос задужења – основни сценарио 36](#_Toc440822721)

[Дијаграм 9: Унос задужења – алтернативни сценарио 37](#_Toc440822722)

[Дијаграм 10: Измена задужења – основни сценарио 38](#_Toc440822723)

[Дијаграм 11: Измена задужења – алтернативни сценарио 39](#_Toc440822724)

[Дијаграм 13: Претрага задужења – алтернативни сценарио 41](#_Toc440822726)

[Дијаграм 14: Пријављивање радника – основни сценарио 42](#_Toc440822727)

[Дијаграм 15: Пријављивање радника – алтернативни сценарио 42](#_Toc440822728)

[Дијаграм 16: Концептуални модел 47](#_Toc440822729)

[Дијаграм 17: Архитектура система 68](#_Toc440822730)

[Дијаграм 18: Концептуални модел 69](#_Toc440822731)

[Дијаграм 19: СО - ZapamtiClana 70](#_Toc440822732)

[Дијаграм 20: СО - UcitajClana 71](#_Toc440822733)

[Дијаграм 21: СО - ObrisiClana 72](#_Toc440822734)

[Дијаграм 22: СО - IzmeniClana 73](#_Toc440822735)

[Дијаграм 23: СО - UnesiZaduzenje 74](#_Toc440822736)

[Дијаграм 24: СО - IzmeniZaduzenje 75](#_Toc440822737)

[Дијаграм 25: СО – VratiListuMesta 76](#_Toc440822738)

[Дијаграм 26: СО – VratiListuFilmova 77](#_Toc440822739)

[Дијаграм 27: СО - VratiClanove 78](#_Toc440822740)

[Дијаграм 28: СО - NadjiRadnika 79](#_Toc440822741)

[Дијаграм 29: СО - VratiSveZanrove 80](#_Toc440822742)

# Попис слика:

[Слика 1: Форма за унос члана - празна 54](#_Toc440823616)

[Слика 2: Форма за унос члана 55](#_Toc440823617)

[Слика 3: Систем запамтио члана 56](#_Toc440823618)

[Слика 4: Систем не може да запамти члана 56](#_Toc440823619)

[Слика 5: Страна за приказ чланова 56](#_Toc440823620)

[Слика 6: Страна за приказ чланова - претрага 57](#_Toc440823621)

[Слика 7: Измена података о члановима 57](#_Toc440823622)

[Слика 8: Систем је запамтио члана 58](#_Toc440823623)

[Слика 9: Систем није запамтио члана 58](#_Toc440823624)

[Слика 10: Страна за приказ чланова 59](#_Toc440823625)

[Слика 11: Страна за приказ чланова - претражена 59](#_Toc440823626)

[Слика 12: Систем је обрисао члана 60](#_Toc440823627)

[Слика 13: Систем не може да обрише члана 60](#_Toc440823628)

[Слика 14: Страна за задуживање 61](#_Toc440823629)

[Слика 15: Страна за задуживање – претражени филмови 61](#_Toc440823630)

[Слика 16: Систем је запамтио задужење 62](#_Toc440823631)

[Слика 17: Систем не може да запамти задужење 62](#_Toc440823632)

[Слика 18: Страна за приказ задужења 63](#_Toc440823633)

[Слика 19: Систем је променио задужење 63](#_Toc440823634)

[Слика 20: Систем не може да промени изабрано задужење 64](#_Toc440823635)

[Слика 21: Страна за приказ задужења 64](#_Toc440823636)

[Слика 22: Систем не може да прикаже задужења по задатим вредностима 65](#_Toc440823637)

[Слика 23: Страна за пријаву на апликацију 66](#_Toc440823638)

[Слика 24: Систем не може да нађе радника 66](#_Toc440823639)

# Попис табела:

[Tабела 1. Бирање формата одговора 14](#_Toc440824086)

[Табела 2. Резултати позивања HTTP метода 15](#_Toc440824087)

[Табела 3. Табела Radnik 48](#_Toc440824088)

[Табела 4. Табела Clan 48](#_Toc440824089)

[Табела 5. Табела Mesto 49](#_Toc440824090)

[Табела 6. Табела Adresa 49](#_Toc440824091)

[Табела 7. Табела Film 50](#_Toc440824092)

[Табела 8. Табела Zanr 50](#_Toc440824093)

[Табела 9. Табела Zaduzenje 51](#_Toc440824094)

[Табела 10. Табела Clan 80](#_Toc440824095)

[Табела 11. Radnik 81](#_Toc440824096)

[Табела 12. Табела Radnik 81](#_Toc440824097)

[Табела 13. Табела Adresa 81](#_Toc440824098)

[Табела 14. Табела Film 81](#_Toc440824099)

[Табела 15. Табела Zanr 81](#_Toc440824100)

[Табела 16. Табела Zaduzenje 81](#_Toc440824101)

# 1. Увод

Интернет је за само двадесетак година, од свог пробоја до данашњих дана, изменио свет и начин на који људи функционишу. Од простих, статичких веб страна са опскурним информацијама до тога да ако се информације о човеку или компанији не налазе на интернету, они скоро да и не постоје. Те промене су много утицале и на пословање компанија.

Сложени организациони и пословни модели компанија, као и појава јединствених захтева клијената, довели су до потребе за дистрибурианим и преносивим апликацијама, којима се може приступити са различитих уређаја, које имају способност прикупљања, обраде, анализе велике количине информација, на којима се може поуздано одлучивати и водити пословање у турбулентним, економским условима, које омогућавају смањење ризика приликом доношења одлука. Као одговор на промене у савременом пословању, долази до преласка са десктоп апликација на развој дистрибуираних веб апликација.

Основна карактеристика веб апликација јесте извршавање на више различитих машина. Њихов циљ је повећање модуларизације компоненти и сакривање детаља приликом развоја пословних софтверских система. Из угла корисника, повећава се једноставност употребе, доступност и лакоћа приступа. Са програмерске стране, главна предност је лакоћа развоја апликација, њихово одржавање, апстракција и дистрибуција, а све у циљу повећања доступности сервиса крајњим корисницима. Апликације су вишеклијентске, тежи се повећању сигурности трансакција, ефикасности система и доступности података. Веб апликације не захтевају никакве комплексне процедуре приликом инсталације на корисниковом систему, нити приликом имплементације у организацијама. Сва логика се налази на једном месту, стога је потребно изменити систем једном, а промене постају видљиве и доступне свим корисницима. Ово доприноси већем задовољству корисника, али и вишем нивоу квалитета система.

Вeб апликације су програмска решења којима се приступа само путем Интернет претраживача (енг. browser-а) од којих су познатији Internet Explorer, Mozilla Firefox, Opera, итд. Може им се приступати путем Интернета или у оквиру Интранета, тј. интерне мреже. Сваким даном, веб апликације су све заступљеније и популарније, из разлога што су доступне са било ког рачунара и било ког места, па и са мобилних уређаја који су већ сада стандардно опремљени веб претраживачем.

У уводном поглављу описане су коришћене технологије, Јава платформа, са посебним освртом на Java Enterprise Edition платформу, која представља основ за развој веб апликације. Изложен је теоријски осврт на концепт вeb сервиса и детаљније је појашњена архитектура којом се веб сервиси реализују.

У поглављу “Студијски пример” је описан поступак развоја веб апликације за праћење спортских такмичења кроз кораке Ларманове методе – од дефинисања корисничких захтева, преко фазе анализе, фазе пројектовања до фазе имплементације. Дефинисање корисничких захтева је описано кроз случајеве коришћења.

Фаза анализе дата је кроз приказ дијаграма секвенци претходно дефинисаних случајева коришћења, док је у фази пројектовања описано пројектовање корисничког интерфејса, апликационе логике и пројектовање складишта података. Такође, у овом делу је приказана комплетна архитектура овог софтверског система.

Фаза имплементације описује имплементационе компоненте система које представљају реализацију компоненти из фазе пројектовања.

# 2. Преглед коришћене технологије

## 2.1. Јава платформа

Јава платформа представља компјутерску платформу или рачунарско окружење Oracle-а у ком се може извршити апликација која је развијенa помоћу Јава програмског језика и скупа развојних алата[[1]](#footnote-1). Платформа се састоји из следећа две комоненте које су неопходне за њено функционисање:

1. Java Runtime Environment (JRE), која је потребна да би се омогућило покретање Јава апликација и апплета
2. Java Development Kit (JDK), која служи за развој Јава апликација, апплета, и која би требало да садржи JRE.

Најбитнији део ЈRE је је Јаvа Virtual Мachine (ЈVМ), која представља виртуални процесор, који омогућава покретање Јава апликација на било ком оперативном систему на локалном рачунару. Њена основна намена је да обрађује написани и примљени byte-code.

Сваки Јаvа Development Kit користи Јава компајлер, који представља најбитнији део окружења. Наиме, он је потребан како би се покренуо и компајлирао изворни код написан у Јави. Друга битна ставка јесу Јаvа class libraries (ЈCL), које имају тројаку сврху у Јавиној платформи. Прва је омогућавање програмеру да користи скуп већ развијених и имплементираних метода, које помажу при обављању једноставних задатака, као што је парсирање Стрингова или рад са листама. Друго, ове библиотеке омогућавају развој и имплементацију комплексних задатака који зависе од саме платформе при свом извршавању. Неке од тих класа су java.io и java.net које имплементирају комплексну логику, а затим пружају програмеру стандардни интерфејс за рад са методама које конзумирају ове класе. На самом крају, оне симулирају апстрактне концепте и појмове, јер неке платформе не подржавају постојање тих концепата и рад са њима на стандардизован начин.

Јава платформа има следеће верзијe[[2]](#footnote-2):

1. Java 2 Platform, Standard Edition или J2SE
2. Java 2 Platform, Enterprise Edition или J2EE
3. Java 2 Platform, Micro Edition или J2ME

У овом студијском случају фокус ће бити на Јава 2 Платформи, Enterprise Edition, обзиром на чињеницу да је развијана Јавина веб апликација.

### 2.1.1. Java 2 EE платформа

Java 2 Platform, Enterprise Edition представља широко распрострањено развојно окружење, обезбеђује АРI и извршно окружење за развој и покретање великих, дистрибуираних, поуданих, вишенивојских и скалабилних мрежних апликација. Ј2ЕЕ платформа служи за пројектовање и развој великих, пословних апликација, као што и сам назив сугерише. Свака апликација, развијана у овом окружењу, подељена је на више компоненти у складу са функцијама које обављају. Свака компонента може бити инсталирана на различитом систему, у зависности од нивоа и слоја апликације ком припрадају.

Већина Ј2ЕЕ апликација се састоји из три или четири слоја, уобичајено из три слоја, који се простиру на три различита дела: први су клијентске машине, други Ј2ЕЕ серверске машине, а трећи део представља базу података. На овај начин се прешло са старе, двојнивојске клијент-сервер архитектуре, на тронивојски, убацивањем апликационог сервера, који садржи пословну логику, између клијентске апликације и система за управљање базом података.

## 2.2. Алат за изградњу пројекта - Maven

Maven је алат који се користи за изградњу и управљање пројектима и програмима који су засновани на Јава програмском језику.

Примарни циљ је да омогући програмеру да схвати потпуну стање развојног процеса у најкраћем року . Да би се постигао овај циљ постоји неколико области у којима се Maven користи:

* Омогућава процес израде
* Обезбеђује јединственог система изградње
* Пружа информација о квалитету пројекта
* Пружа смернице за најбољи развојни пут пројекта
* Омогућава транспарентне преласке на нове функције

Maven омогућава изградњу пројекта користећи модел самог пројекта и низ компоненти које су заједничке за све пројекте који користе Maven, пружајући јединствени систем изградње пројекта . Maven пружа обиље корисних информација које су делимично преузете из модела пројекта и делом генерисане из извора самог пројекта. Најважније функционалности које пружа *Maven* су:

* Менаџмент зависности (Dependency managment): обезбеђује коришћење централног складиштења ЈАR-ова (библиотеке класа) и других зависности. Maven има механизам којим преузима било који ЈАR потребан за изградњу пројекта. Ово омогућава корисницима Maven-а поновну употребу ЈАR-ова свих пројеката и подстиче комуникацију између пројеката како би се осигурало да су сви проблеми који се тичу компатибилности буду решени.
* Моћан систем за управљање зависностима који подразумева аутоматско ажурирање истих
* Може лако управљати са више пројеката истовремено[[3]](#footnote-3)

## 2.3. Веб сервиси

Постоји огроман број података на интернету па је потребно дефинисати неке стандарде који прописују како је могуће вршити размену података између различитих веб сајтова, између апликација написаним у различитим програмским језицима, а и између свих осталих учесника на интернету. Та поменута размена података се врши помоћу веб сервиса.

Веб сервис омогућује заједнички рад и међусобно размену података између различитих система кроз мрежу. Једна од званичних дефиниција веб сервиса је „ “Веб сервис је софтверски систем који је направљен да подржи интероперабилну интеракцију између машина кроз мрежу.“[[4]](#footnote-4) . Две битне ствари које се истичу из ове дефиниције су да је веб сервис софтверски систем, и да је његова улога да омогући комуникацију између машина или софтверских система.

Вeб сервиси су апликације које комуницирају са другим веб апликацијама у служби размене података. Они могу претворити постојећу апликацију у веб апликацију и тиме је подићи на виши ниво. Користећи веб сервис, апликација може ставити на располагање своје функционалности било којој другој апликацији. У прилог идеји настанка веб сервиса, његово име потиче од енглеске речи „ serve“ – опслуживати, што значи да веб сервис има функцију да ослушкује и опслужује различите захтеве корисника и да одговара на поменуте захтеве.

Кључни принципи који су утицали на развој архитектуре и протокола веб сервиса су:

* Оријентација ка порукама – искључивост коришћења порука за комуникацију
* Поновна употребљивост протокола – избегавање коришћења комплексних протокола; уместо тога коришћење „градивних блокова“ од којих се могу правити разне комбинације
* Аутономност сервиса – идеја да сервиси буду независно креирани, одржавани, надграђивани и обезбеђивани
* Контролисани приступ – могућност контролисања који аспекти једног сервиса су ( и који нису ) видљиви и доступни за јавни приступ неког другог сервиса
* Интеграција базирана на протоколима – интеграција између сервиса треба бити искључиво базирана на протоколима

Данашњи веб сервиси у потпуности испуњавају ове принципе и идеје на којима су настали. Може се рећи својим својствима и начином рада решавају 3 битна проблема:

* Доступност - Користећи веб сервис постојећа функционалност се може учинити доступном и видљивом кроз мрежу те се може користити у више апликација,
* Интероперабилност – исти веб сервис могу користити различите апликације јер је платформски независан и користи стандардизоване протоколе у комуникацији,
* Ефикасност – веб сервиси на веома ефикасан начин чине доступним податке на интернету.

### 2.3.1. REST Web сервиси

REST (Representational State Transfer) представља принцип архитектуре који се заснива на премиси да се сви подаци у оквиру веб сервиса посматрају као јединствени ресурс. Сваки од тих ресурса се може јединствено идентификовати преко URI-а (Uniform Resource Identifier). REST стил је мање формалан начин комуникације у односу на другу врсту веб сервиса, SOAP, а такође је и једноставнији.

REST архитектура се имплементира у форми сервера и клијената који користе информације од сервера. Када говоримо о REST-у, говоримо о размени података при којој нема чувања информација о стању (stateless), и за то се користи HTTP протокол комуникације који је такође stateless протокол комуникације, без информација о стању, мада је могуће користити и друге протоколе. У REST архитектури клијенти и сервери размењују репрезентације ресурса користећи стандардизоване интерфејсе и протокол.

Постоји неколико принципа који утичу на то да REST сервиси, а и сама размена података, буду једноставни, не превише ресурсно захтевни и брзи:

* Идентификовање ресурса преко URI -ја
* Униформни интерфејси
* Самоописне поруке
* Коришћење других техника ако је потребна сесија, сама REST архитектура треба да буде stateless

Такође, једна од предности коју REST има представљају поруке којима се он служи. Наиме, поред стандардних XML порука којима се користе веб cервиси свих архитектура, REST се мозе служити и JSON(JavaScriptObjectNotation) порукама. Карактеристика ових порука је да су једноставније за креирање од одговарајућих XМL порука, а нису стандардима дефинисане и формалне, што их чини читљивијим. Ипак, JSON има нека ограничења, у виду тога да може у себи садржати само текст и нумеричке податке, за разлику од XML-а који може преносити чак и читаве документе. Управо због тог недостатка, JSON поруке не носе у себи велику количину метаподатака, па је њихов пренос кроз мрежу ефикаснији и бржи, што га чини правим избором за REST.

#### 2.3.1.1. Архитектура и принципи RESТ сервиса

Да би се веб cервис сматрао REST мора поседовати одређене особине:

1. Мора постојати одговарајући интерфејс који раздваја клијентски од серверског дела. То значи да клијент не сме бити директно повезан за базом података, већ да о томе брине серверски део. Такође, сервер не треба да размишља о приказу тих података на клијентском делу, што га чини једноставнијим и прилагодљивијим.
2. Мора функционисати као протокол без стања
3. Да би спречио ситуацију у којој клијент користи неодговарајући или измењен садржај/податке, веб сервис мора навести да ли подржава или не могућност кеширања. Ово је директна последица WWW-а, јер на њему клијенти могу кеширати одређене податке.
4. Мора функционисати као вишеслојан систем – клијент има привид да је директно повезан на крајњи извор података, иако између њих има више слојева које он не види.
5. Ресурсима се мора приступати користећи URI

У својој најједноставнијој форми, конкретна имплементација REST веб сервиса прати четири основна принципа:

* Експлицитно коришћење HTTP метода
* Нема коришћења стања
* Излагање URI-ја који су налик структури директоријума
* Трансфер преко XML-а или JSON-а

Два основна принципа REST архитектуре, изведена из њених ограничења, су:

* Перформансе - компоненте комуницирају кроз мрежу много ефикасније од еквивалентних компоненти SOAP сервиса, што чини кључни фактор њиховог рада
* Скалабилност – омогућено је независно мењање клијентског и серверског дела, докле год је интерфејс између њих непромењен

#### 2.3.1.2. HTTP методе

Једна од кључних карактеристика RESTful веб сервиса је коришћење HTTP метода апсолутно у складу са протоколом дефинисаним у РФЦ 2616[[5]](#footnote-5) . Узмимо као пример HTTP GET методу, она је дефинисана као метода која враћа податке и предвиђена је за коришћење од стране клијента за узимање ресурса, за узимање података са сервера или за извршење упита са очекивањем да ће веб сервер пронаћи и вратити тражене податке.

REST захтева од програмера да користе HTTP методе у складу са тим како су методе дефинисане у протоколу. Један од основних REST принципа дефинише један према један пресликавање између CRUD (Create, read, update, delete) операција и HTTP метода. На основу тих дефинисаних пресликавања треба користити:

• POST метода за креирање ресурса на серверу

• GET метода за узимање ресурса са сервера

• PUT метода за измену ресурса на серверу

• DELETE метода за брисање ресурса на серверу

REST сервиси морају бити скалабилни како би одржали перформансе и при повећању броја корисника и/или захтева ка сервису. Користе се разне технике за смањење оптерећења сервера веб сервиса, као што су кластери сервера са балансирањем оптерећења или организовање сервера у одређене топологије како би време одговора на захтеве клијената било што мање. Уколико се користе сервери који само прослеђују захтеве другим серверима или се имплементирају технике контролисања оптерећења неопходно је да клијенти шаљу захтеве који су комплетни и независни. Другим речима, потребно је да клијенти шаљу захтеве који садрже све податке који су потребни серверу за извршење неке операције у самом захтеву, без вођења рачуна о постојању стања или сесије.

Комплетан, независтан захтев не захтева од сервера да, током обраде захтева, захтева било какву врсту стања или контекста апликације. REST апликација која позива REST сервис (или неки други REST клијент) шаље у оквиру заглавља и тела HTTP методе све што је потребно да REST сервис може успешно да одговори на тај захтев. Недостатак потребе да се води рачуна о стању на серверу побољшава перформансе и олакшава и упрошћава пројектовање и имплементацију серверског дела пошто није неопходно водити евиденцију о сесији на серверу REST сервиса.

Са становишта клијентске апликације која захтева ресурсе, URI-ји ће одређивати колико ће интуитиван бити REST веб сервис и да ли ће се користити на начин који је предвиђен.

URI REST сервиса треба да буде толико интуитиван да приступ треба да буде могућ у без проучавања документације сервиса. Потребно је посматрати URI као самоописујући интерфејс за чије коришћење и разумевање програмеру није потребно превише труда и истраживања. Да закључимо, структура URI-ја треба да буде једноставна, предвидива, и лако разумљива.

Један од начина да се ово постигне је да URI има структуру директоријума, или стабла. У оваквој реализацији постојаће хијерархијска структура, са једним главним чвором који се грана на мноштво других (под)чворова.

Неки од савета који се могу применити да би URI био што интуитивнији:

* Сакрити екстензије скрипт фајлова на серверској страни (.jsp, .php, .asp) ако постоје, како би се могле вршити измене без потребе да се мења и URI структура
* Давати називе елементима малим словима
* Заменити размак поља са цртицама или доњим цртама
* Избегавати стрингове са упитима
* Враћати подразумевану (почетну) страну уместо 404 стране у случају погрешног URI-ја

Још једна важна карактеристика URI-ја је да треба да буду статични, тј. када се ресурс промени, или се промени имплементација сервиса, URI мора остати исти.

Репрезентација неког ресурса типично одражава тренутно стање тог ресурса и његових атрибута, у тренутку када га клијент захтева. Дакле, овај последњи захтев се односи на формат података које клијент и веб сервис размењују приликом слања захтева и одговора.

Објекти у моделу података су најчешће повезани на неки начин. Везе којима су објекти повезани треба да буду видљиве и у репрезентацији тих објеката која ће се послати клијенту као одговор на захтев. Ово је могуће урадити на лак, људима читљив начин, представљањем ресурса помоћу XМL-а или JSON-а, и оба начина су подржана у REST архитектури.

Добра је пракса да се клијенту остави могућност избора коју облик података жели да добије са сервера. Ово се најчешће ради на основу вредности Content-Тypе атрибута који се дефинише у заглављу HTTP захтева који клијент шаље REST веб сервису.

Следећа табела показује како треба да изгледа вредност Content-Тypе атрибута уколико клијент жели JSON или XML одговор.

|  |  |
| --- | --- |
| **Format odgovora** | **Content-Type** |
| JSON | aplication/json |
| XML | aplication/xml |

***Tабела 1. Бирање формата одговора***

#### 2*.3.1.3. URI*

У наставку можемо видети објашњења коришћења REST веб сервиса, позивања коришћењем URI-ја као и резултате које веб сервис враћа. Конкретни примери URI-ја састоје се од обавезног дела URL(UniformResourceLocator) и опционих делова након знака питања (?), делова који представљају парове (кључ, вредност).

* tenis/rest/liga?id=2

REST архитектура не препоручује слање параметара преко URI-ја, иако је то очигледно могуће, као што је показано у примерима. Препоручљиво је параметре слати у телу HTTP захтева, а URI треба да служи за хијерархијско досезање ресурса. Следе примери који приказују формирану хијерархијску структуру и интуитивно коришћење URI-ја како би се дошло до жељених елемената.

* tenis/rest/liga/3

У наредној табели ће бити показано који је резултат позивања различитих HTTP метода на појединачним подацима и над скуповима података.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ресурс** | **GET** | **PUT** | **POST** | **DELETE** |
| **URI који означава скуп података:** tenis/rest/liga | Враћа све лиге | / | Додаје лигу | / |
| **URI који означава појединачни елемент:** tenis/rest/liga/2 | Враћа лигу која има ид 2 | Мења лигу са прослеђеним подацима | / | Брише конретну лигу |

***Табела 2. Резултати позивања HTTP метода***

#### 2.3.1.4. JAX-RS

ЈАX-RS (JavaAPIfor RESTful Web Services) је API који служи за креирање веб сервиса у Јава програмском језику у складу са REST архитектуром. Од верзије 1.1 ЈАX-RS је интегрални део ЈАVА ЕЕ и за његово коришћење није потребно никакво додатно конфигурисање. Oва платформа подржава и развој REST веб сервиса. Као што је већ речено, они користе HTTP, али за разлику од SOAP сервиса није им неопходан WSDL, као ни подразумеван XML за размену порука.

ЈАX-RS представља спецификацију сервиса, постоје различите имплементације те спецификације. Најпознатија и највише коришћена имплементација ЈАX-RS спецификације је Јеrsеy, који је такође и референтна имплементација од стране Оraclе-а. Како подржава и анотације и дефинисање ЈАX-RS спецификацијом, Јеrsеy у многоме олакшава креирање REST веб сервиса користећи Јаву и JVM(Јаvа Virtual Маchinе).

REST веб cервиси користе W3C и IETF(Interner Engineering Task Force) стандарде као што су HTTP, XML, URI и MIME. Тиме се добија на упрошћавању разумевања њихове структуре и самим тим лакшег развијања. Користећи неко развојно окружење, као што је NetBeans IDE, сложеност веб cервиса се може додатно смањити, без утицања на функционалности, и самим тим допринети његовом ефикаснијем раду.

Употреба анотација:

ЈАX-RS такође има дефинисане и разне анотације које помажу лаком и једноставном претварању стандардне Јава класе у веб ресурс који може имати улогу сервиса. Најчешће коришћене анотације су:

• @Path – дефинише релативну путању до веб ресурса

• @GET, @PUT, @POST, @DELETE означавају тип HTTP захтева на који ресурс реагује

• @Produces – дефинише формат одговора

• @Consumes – дефинише формат захтева који елемент прихвата

Осим поменутих, постоје и додатне анотације чија је улога да повежу параметре из захтева са параметрима методе, наведени су неки од њих:

• @PathParam – везује улазни параметар методе за део путање

• @Queryparam – везује улазни параметар методе за вредност параметра из путање

• @HederParam – везује улазни параметар методе за вредност HTTP заглавља

Следи пример једноставне Јава класe која, уз помоћ анотација, врши функцију REST вeб сервиса:

import javax.ws.rs.GET;

import javax.ws.rs.Produces;

import javax.ws.rs.Path;

// Putanja za koju Web servis vraća odgovore je "/helloworld"

@Path("/helloworld")

public class HelloWorldResource {

// Ova metoda obrađuje GET zahteve

@GET

// ova metoda vraca običan tekst

@Produces("text/plain")

public String helloWorld() {

// Vrati poruku

return "Hello World";

}

}

Ево и исечка кода из студијског примера који приказује како REST сервис враћа JSON објекат који садржи све такмичаре који постоје у бази:

@GET

@Produces(MediaType.APPLICATION\_JSON)

public Response getTakmicar(@HeaderParam("authorization") String authorization, @QueryParam("search") String search, @QueryParam("takmicar") String takmicar) {

EntityManager em = helper.getEntityManager();

if (helper.isLogged(em, authorization)) {

StringBuilder query = new StringBuilder("SELECT t FROM Takmicar t ");

if (search != null) {

query.append("WHERE t.jmbg LIKE '%")

.append(search)

.append("%' OR t.ime LIKE '%")

.append(search)

.append("%' OR t.prezime LIKE '%")

.append(search)

.append("%' OR t.opis LIKE '%")

.append(search)

.append("%'");

}

if (takmicar != null) {

query.append(search != null ? " AND " : "WHERE ")

.append("t.id = ")

.append(takmicar);

}

System.out.println(query);

List<Takmicar> takmicari = em.createQuery(query.toString())

.getResultList();

if (takmicari .isEmpty()) {

throw new DataNotFoundException("Nema takmičara u bazi");

} else {

return Response.ok().entity(takmicari).build();

}

} else {

throw new NotAuthorizedException("Nemate pristup ovom pozivu!");

}

}

## 2.4. Клијентска страна – Bootstrap, jQuery

За развој клијентске стране апликације (front-end) коришћене су одговарајуће технологије као што су HTML, CSS и JavaScript.

Технологије HTML5 и CSS3 употребљене су кроз Bootstrap front-end оквир, док је за писање JavaScript кôда употребљена бибилиотека jQuery.

jQuery је мултиплатформска JavaScript библиотека направљена како би се олакшало креирање скрипти на клијентској страни веб апликација. Ово је најпопуларнија JavaScript библиотека која се користи на преко 65% најпосећенијих сајтова на вебу. Библиотека је отвореног кода и лиценцирана је под MIT лиценцом.

Синтакса је направљена тако да се олакша кретање кроз документ, селектовање DOM елемената, креирање анимација, управљање догађајима и развој AJAX aпликација. jQuery такође нуди могућност програмерима да на вишем нивоу апстракције изграде своје библиотеке.

У својој сржи jQuery представља библиотеку за управљање DOM садржајем. DOM je структура у облику стабла којом су приказани сви елементи на некој веб страници. У пракси, jQuery може бити коришћен како би се нашао неки елемент и променили његови атрибути.

Предности коришћења jQuery-ја:

* Даје предност раздвајању JavaScript-a i HTML-a: jQuery библиотека омогућава да се на врло једноставан начин управља догађајима који су везани за DOM елементе, уместо да се користе атрибути који се налазе у HTML таговима.
* Кратко и јасно: омогућава да код буде кратак и јасан са могућношћу везивања функција
* Ради једнако на свим претраживачима: JavaScript код може да се понекад другачије понаша у односу на то у ком се претраживачу извршава, али jQuery елиминише овај недостатак
* Проширивост: Нови догађаји, елементи и методе лако се могу додати, а онда и поново користити

jQuery додаје се HTML документу тако што се дода у хијерархију пројекта и наведе следећа линија кода пре свих линкова до осталих JavaScript докумената:

<script type="text/javascript" src="../static/js/jquery.js"></script>

Пример кода написаног помоћу библиотеке jQuery уз помоћу које су додати табови на страницу за измене:

$(document).ready(function () {

$('.tabs .tab-links a').on('click', function (e) {

var currentAttrValue = $(this).attr('href');

$('.tabs ' + currentAttrValue).fadeIn(400).siblings().hide();

$(this).parent('li').addClass('active').siblings()

.removeClass('active');

e.preventDefault();

});

});

**Bootstrap** је најпопуларнији HTML, CSS и JS оквир за развој респонзивних, мобилних пројеката на Интернету. Oн је бесплатан скуп алата за креирање сајтова и веб апликација .

Bootstrap је open-source Јаvа оквир развијен од стране тима на Twitter-у . То је комбинација HTML, CSS и JavaScript кода дизајниран да помогне у изградњи компоненти корисничког интерфејса. Bootstrap је такође програмиран да подржи HTML5 и CSS3 .Назива се и Front-end - оквир .

Као што је речено садржи шаблоне базиране на HTML и CSS – у за дизајн типографије , облике, дугмад, навигацију и друге компоненте интерфејса, као и опционе екстензије JavaScript-a.

Неки од предности Bootstrapa:

* Лак за учење и примену
* Детаљна документација са примерима
* Шаблони за већину HTML елемената
* Дугачку листу компоненти
* Разни JavaScript додаци

Bootstrap у пројекат тако што се документи, односно тема додају хијерархији програма и у HTML документу наведе следећа линија кода:

<script type="text/javascript" src="../static/js/bootstrap.js">

</script>

# 3. Студијски пример

## 3.1. Ларманова метода

Ларманова метода за развој софтвера се заснива на итеративно-инкременталном моделу животног циклуса софтвера. Ларманова метода се састоји од следећих фаза, и то:

1. Спецификација захтева
2. Анализа
3. Пројектовање
4. Имплементација
5. Тестирање.

Спецификација захтева представља прву фазу Ларманове методе. Ова фаза се заснива на блиској сарадњи корисника (оног који креира захтев) и програмера (онај који захтев испуњава).

Модел случаја коришћења се састоји од скупа случајева коришћења, актера и веза између актера и случајева коришћења. Случај коришћења се састоји од једног главног сценарија, а може да садржи и један или више алтернативних сценарија.

Сценарио представља секвенцу акција које описују интеракцију актера и система. Актер је спољни корисник система који захтева извршење једне или више системских операција, према унапред дефинисаном сценарију. Актер може бити особа, рачунарски систем или организација. Један актер може да користи више случајева коришћења, док један случај коришћења може бити коришћен од стране више актера. Једну акцију сценарија може извршити или актер или систем, тако да у зависности од тога ко извршава акције можемо поделити на следећи начин:

Акције које изводи актер су:

* АПУСО – Актер припрема улазне аргументе за системску операцију,  АПСО – Актер позива систем да изврши системску операцију и  АНСО – Актер извршава несистемску операцију.

Акције које изводи систем су:

* СО – Систем извршава системску операцију и  ИА – Излазни аргументи, који представљају резултате извршења системске операције, који се прослеђују до актера.

Након прве фазе следи фаза анализе. Фаза анализе описује логичку структуру и понашање софтверског система (пословну логику софтверског система).

Понашање система подразумева шта систем треба да ради, али не и како то да ради. Оно се описује помоћу системских дијаграма секвенци и уговора о системским операцијама. Системски дијаграм секвенци се прави за сваки претходно утврђени случај коришћења и он приказује догађаје у одређеном редоследу. Као резултат анализе системских дијаграма секвенци добијају се захтеви за извршење системских операција, односно идентификују се системске операције које се требају пројектовати. За сваку од уочених системских операција праве се уговори. Један уговор везан је за једну системску операцију. Уговори се састоје од следећег:

1. Операција - име операције и њени улазни аргументи
2. Веза са СК - имена СК у којима се позива СО
3. Предуслови - пре извршења СО морају бити задовољени одређени предуслови, односно систем мора бити у одговарајућем стању. Предуслови описују шта треба да се деси да би се извршила СО, а не описује како се то десило
4. Постуслови - после извршења системске операције у систему морају бити задовољени одређени постуслови, што значи да систем мора бити у одговарајућем стању или се поништава резултат перације. Постуслови описују ефекте извршења системских операција.

Структура софтверског система се описује помоћу концептуалног модела који описује концептуалне класе домена проблема. Концептуални модел садржи концептуалне класе, које се називају доменски објекти, и асоцијације између концептуалних класа. Концептуалне класе се идентификују на основу атрибута, који описују особине класа.

Према томе, главни дијаграми који се креирају у овој фази су:

1. концептуални модел – служи за опис структуре система
2. секвенцијални дијаграми система – служе за опис понашања система
3. Уговори – служе за опис понашања система.

Трећа фаза је фаза пројектовања. Ова фаза описује физичку структуру и понашање софтверског ситема и њен резултат је тронивојска архитектура софтверског система. Пројектовање архитектуре софтверског система обухвата пројектовање апликационе логике (контролер, пословна логика и брокер базе података), складишта података и корисничког интерфејса. За уговоре о системским операцијама које су уочене у фази анализе, праве се детаљни дијаграми секвенци. За сваку системску операцију се праве софтверске класе које треба да реализују системске операције.

Претпоследња фаза је фаза имплементације. Као што јој и сам назив каже, ова фаза подразумева имплементацију свих компоненти добијених пројектовањем архитектуре система. Софтверски систем који је предмет овог завршног рада развијан је у Јава програмском језику и коришћено је NetBeans развојно окружење.

Овај систем за праћење спортских такмичења је пројектован као тронивојска Wеб апликација. На презентационом нивоу налази се wеб претраживач који приказује HTML странице. Апликациона логика је имплементирана помоћу Jersey радног оквира као REST API-ја који представља контролер пословне логике, и помоћу ЈPA радног оквира за рад са базом података. Осим тога, коришћен је МySQL систем управљања базом података.

Последња фаза ларманове методе је фаза тестирања. Срж ове фазе јесте да се испита и провери да ли добијени систем задовољава почетне захтеве. Како би се валидно истестирао систем неопходно је унети како тачне, тако и погрешне податке. Након уноса оваквих података увиђа се да ли је развијени систем отпоран на грешке, тј. како реагује на погрешне податке, али исто тако и да ли добијени систем радио оно што треба и са коректно унетим подацима. Оно што је неопходно како би систем био квалитетан јесте константно писање тестова.

Тестирање система се састоји од неколико корака:

* функционално тестирање - подразумева тестирање функција система
* тестирање перформанси - тестирају се безбедност, прецизност, брзина и поузданост
* тестирање прихватања - подразумева тестирање од стране корисника
* инсталационо тестирање - подразумева тестирање на лицу места

## 3.2. Кориснички захтеви

Први корак у процесу развоја софтвера Лармановом методом представља прикупљање корисничких захтева. Овај корак, можда је и најважнији у целокупном процесу јер ако се реални систем, односно корисникове потребе не идетификују на прави начин, у каснијим фазама развоја може доћи до изузетних проблема. Од изузетне је важности да пројектант уочи још у овој фази на какве све проблеме може да наиђе у каснијем развоју. Такође потребно је и да уочи прави модел података који би задовољио потребе система, као и начин на који ће се ти подаци приказивати и обрађивати. Захтеви се описују помоћу модела случаја коришћења (Use case).

### 3.2.1. Вербални опис

Потребно је направити апликацију која омогућава прављење спортских такмичења

(нпр. тениска лига). Систем сваког такмичења је “лигашки”. Свако такмичење може да

има једну или више група. У свакој групи сваки од такмичара се такмичи са сваким

једном, два или више пута.

Та апликација би требала да омогућава прављење лига у једном такмичењу са

одговарајућем бројем такмичара у свакој од њих. Апликација треба да

омогући и измену података сваког такмичења и такмичара.

Апликација треба да води евиденцију о свим такмичарима, да омогући претрагу

такмичара према одређеним параметрима, као и унос нових такмичара и измену

података о постојећим такмичарима и брисање истих.

Такође, потребно је водити евиденцију о свим мечевима одиграних у оквиру једног

такмичења. Омогућити унос сваког меча и претрагу мечева по учесницима или неким

другим параметрима.

За сваког такмичара потребно је водити статистику о постигнутим резултатима за време

такмичења и о подацима у међусобним односима са другим такмичарима.

Још једна битна ставка је да омогућимо апликацији да штампа појединачне податке о

сваком такмичару тј. његову статистику, податке међусобних дуела два такмичара

и штампање целокупне лиге (штампање резултујућих табела).

Да би администратор могао да врши неке измене над подацима у систему, потребно

је да буде пријављен на систем.

### 3.2.2. Случајеви коришћења

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

1. Унос такмичења
2. Брисање такмичења
3. Унос нове лиге
4. Унос такмичара
5. Измена такмичара
6. Брисање такмичара
7. Унос меча

Дијаграм 1: Дијаграм случајева коришћења

### СК 1: Случај коришћења – Унос такмичења

**Назив СК**

Унос такмичења

**Актори СК**

Администратор

**Учесници СК**

Администратор и систем (програм)

**Предуслов**: Систем је укључен и администратор је улогован под својом шифром. Систем приказује форму за унос такмичења. Доступни су типови такмичења.

**Основни сценарио СК**

1. Администратор уноси податке о новом такмичењу. (АПУСО)
2. Администратор контролише да ли је коректно унео податке о новом такмичењу. (АНСО)
3. Администратор позива систем да креира ново такмичење и запамти податке о новом такмичењу. (АПСО)
4. Систем памти податке о новом такмичењу. (СО)
5. Систем приказује раднику поруку: “Систем је успешно креирао такмичење”. (ИА)

**Алтернативна сценарија:**

* 1. Уколико систем не може да запамти податке о новом такмичењу он приказује администратору поруку “Систем не може да запамти ново такмичење”. (ИА)

### СК 2: Случај коришћења – Брисање такмичења

**Назив СК**

Брисање такмичења

**Актори СК**

Администратор

**Учесници СК**

Администратор и систем (програм)

**Предуслов**: Систем је укључен и администратор је улогован под својом шифром. Потребно је да у систему постоји макар једно такмичење. Доступна је листа такмичења.

**Основни сценарио СК**

1. Администратор уноси вредности по којима претражује такмичења и бира члана којег жели да измени. (АПУСО)
2. Администратор позива систем да учита податке о изабраном такмичењу. (АПСО)
3. Систем тражи такмичења по задатој вредности. (СО)
4. Систем приказује администратору пронађене такмичења. (ИА)
5. Администратор бира такмичење. (АПУСО)
6. Администратор позива систем да обрише такмичење. (АПСО)
7. Систем брише такмичење. (СО)
8. Систем приказује администратору поруку: “Систем је обрисао такмичење.” (ИА)

**Алтернативна сценарија**:

* 1. Уколико систем не може да нађе такмичења он приказује администратору поруку: “Систем не може да нађе такмичење по задатој вредности”. Прекида се извршење сценарија. (ИА)

Уколико систем не може да обрише такмичење он приказује администратору поруку : „Систем не може да обрише такмичење“. (ИА)

### СК 3: Случај коришћења – Унос нове лиге

**Назив СК**

Унос нове лиге

**Актори СК**

Администратор

**Учесници СК**

Администратор и систем (програм)

**Предуслов**: Систем је укључен и администратор је улогован под својом шифром и у

систему већ постоји такмичење. Систем приказује форму за рад са лигама.

**Основни сценарио СК**

1. Администратор позива систем да креира нову лигу. (АПСО)
2. Систем креира нову лигу. (СО)
3. Систем приказује администратору нову лигу. (ИА)
4. Администратор уноси податке о новој лиги. (АПУСО)
5. Администратор контролише да ли је исправно унео податке о новој лиги. (АНСО)
6. Администратор позива систем да запамти податке о новој лиги. (АПСО)
7. Систем памти податке о новој лиги. (СО)
8. Систем показује администратору поруку: „Систем је креирао нову лигу“. (ИА)

**Алтернативна сценарија**

* 1. Уколико систем не може да креира нову лигу он приказује администратору поруку: “Систем не може да креира лигу”. Прекида се извршење сценарија. (ИА)
  2. Уколико систем не може да запамти податке о новој лиги он приказује администратору поруку: „Систем не може да запамти лигу“. (ИА)

### СК 4: Случај коришћења – Унос такмичара

**Назив СК**

Унос такмичара

**Актори СК**

Администратор

**Учесници СК**

Администратор и систем (програм)

**Предуслов:** Систем је укључен и администратор је улогован под својом шифром и у

систему већ постоји такмичење. Систем приказује форму за рад са такмичарима.

**Основни сценарио СК**

1. Администратор позива систем да креира новог такмичара. (АПСО)
2. Систем креира новог такмичара. (СО)
3. Систем приказује администратору новог такмичара. (ИА)
4. Администратор уноси податке о новом такмичару. (АПУСО)
5. Администратор контролише да ли је коректно унео податке о новом

такмичару. (АНСО)

1. Администратор позива систем да запамти податке о новом такмичару.

(АПСО)

1. Систем памти податке о новом такмичару. (СО)
2. Систем приказује администратору поруку: “Систем је запамтио такмичара”.
3. (ИА)

**Алтернативна сценарија**

4.1. Уколико систем не може да креира новог такмичара он приказује

администратору поруку: “Систем не може да креира новог такмичара”. Прекида се

извршавање сценарија. (ИА)

Уколико систем не може да запамти податке о новом такмичару он приказује

администратору поруку “Систем не може да запамти новог такмичара”. (ИА)

### СК 5: Случај коришћења – Измена такмичара

**Назив СК**

Измена такмичара

**Актори СК**

Администратор

**Учесници СК**

Администратор и систем

**Предуслов**: Систем је укључен и администратор је улогован под својом шифром. Потребно

је да у систему постоји макар једно такмичење. Систем приказује форму за рад са

такмичарима.

**Основни сценарио СК**

1. Администратор уноси вредности по којима претражује такмичаре. (АПУСО)
2. Администратор позива систем да нађе такмичаре по задатим вредностима.

(АПСО)

1. Систем тражи такмичаре по задатим вредностима. (СО)
2. Систем приказује администратору нађене такмичаре. (ИА)
3. Администратор бира такмичара ког жели да измени. (АПУСО)
4. Администратор позива систем да учита податке о изабраном такмичару.

(АПСО)

1. Систем тражи податке о изабраном такмичару. (СО)
2. Систем приказује администратору податке о изабраном такмичару. (ИА)
3. Администратор уноси (мења) податке о такмичару. (АПУСО)
4. Администратор контролише да ли је коректно унео податке о такмичару.

(АНСО)

1. Администратор позива систем да запамти податке о такмичару. (АПСО)
2. Систем памти податке о такмичару. (СО)
3. Систем приказује администратору поруку: “систем је запамтио такмичара.”

(ИА)

**Алтернативна сценарија**

4.1. Уколико систем не може да нађе такмичара он приказује администратору

поруку: “Систем не може да нађе такмичара по задатим вредностима”. Прекида

се извршење сценарија. (ИА)

8.1. Уколико систем не може да нађе такмичара он приказује администратору

поруку: “Систем не може да нађе изабраног такмичара”. Прекида се извршење

сценарија. (ИА)

13.1. Уколико систем не може да запамти податке о такмичару он приказује

администратору поруку: “Систем не може да запамти такмичара”. (ИА)

### СК 6: Случај коришћења – Брисање такмичара

**Назив СК**

Брисање такмичара

**Актори СК**

Администратор

**Учесници СК**

Администратор и систем (програм)

**Предуслов**: Систем је укључен и администратор је улогован под својом шифром.

Потребно је да у систему постоји макар једно такмичење. Систем приказује форму

за рад са такмичарима.

**Основни сценарио СК**

1. Администратор уноси вредност по којој претражује такмичаре. (АПУСО)
2. Администратор позива систем да нађе такмичаре по задатој вредности. (АПСО)
3. Систем тражи такмичаре по задатој вредности. (СО)
4. Систем приказује администратору пронађене такмичаре и поруку: Систем је нашао

такмичаре по задатој вредностиʺ. (ИА)

1. Администратор бира такмичара. (АПУСО)
2. Администратор позива систем да обрише такмичара. (АПСО)
3. Систем брише такмичара. (СО)
4. Систем приказује администратору поруку: “Систем је обрисао такмичара”. (ИА)

**Алтернативна сценарија**

4.1. Уколико систем не може да нађе такмичаре он приказује администратору поруку:

“Систем не може да нађе такмичара по задатој вредности”. Прекида се извршење

сценариа. (ИА)

8.1. Уколико систем не може да обрише такмичара он приказује администратору

поруку: “Систем не може да обрише такмичара”. (ИА)

### СК 7: Случај коришћења – Унос меча

**Назив СК**

Унос меча

**Актори СК**

Администратор

**Учесници СК**

Администратор и систем (програм)

**Предуслов**: Систем је укључен и администратор је улогован под својом шифром.

Потребно је да у систему постоји макар једно такмичење и да се изаберу два

такмичара. Систем приказује листу такмичења и њихових такмичара.

**Основни сценарио СК**

1. Администратор позива систем да креира нов меч. (АПСО)
2. Систем креира нови меч. (СО)
3. Систем приказује администратору нов меч. (ИА)
4. Администратор уноси податке о новом мечу. (АПУСО)
5. Администратор контролише да ли је коректно унео податке о новом мечу.

(АНСО)

1. Администратор позива систем да запамти податке о новом мечу. (АПСО)
2. Систем памти податке о новом мечу. (СО)
3. Систем приказује администратору поруку:” Систем је запамтио меч”. (ИА)

**Алтернативна сценарија**

3.1. Уколико систем не може да креира нов меч он приказује администратору

поруку: “Систем не може да креира нов меч”. Прекида се извршење сценарија. (ИА)

8.1. Уколико систем не може да запамти податке о новом мечу он приказује

администратору поруку “Систем не може да запамти нов меч”. (ИА)

## 3.3. Анализа

### 3.3.1. Понашање софтверског система – Системски дијаграм секвенци

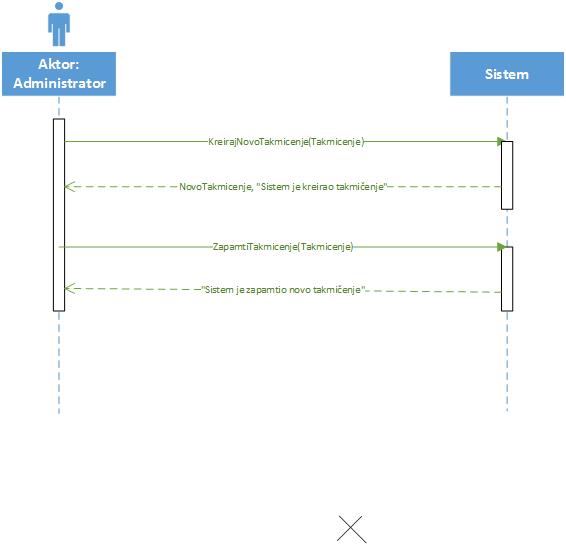
#### ДС 1: Дијаграм секвенци случаја коришћења – Унос такмичења

1. Администратор позива систем да креира ново такмичење. (АПСО)
2. Систем приказује администратору ново такмичење. (ИА)
3. Администратор позива систем да запамти податке о новом такмичењу.

(АПСО)

1. Систем приказује администратору поруку: “систем је запамтио такмичење”.

(ИА)



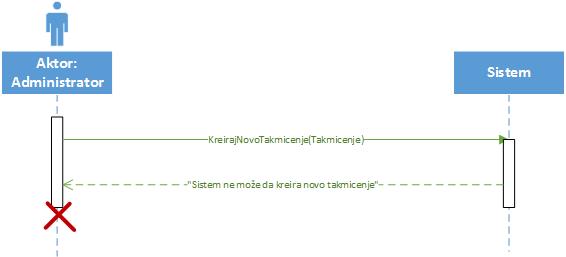
Дијаграм 2: Убацивање новог takmičenja – основни сценарио

**Алтернативна сценарија:**

Уколико систем не може да креира ново такмичење он приказује

администратору поруку: “Систем не може да креира ново такмичење”. Прекида се

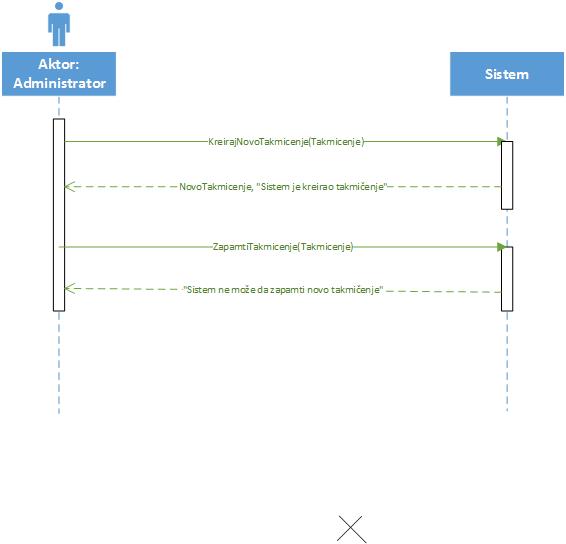
извршавање сценарија. (ИА)



* 1. Уколико систем не може да запамти податке о новом такмичену он

приказује администратору поруку “Систем не може да запамти ново такмичење”.

(ИА)



Са наведених секвенцних дијаграма уочавају се системске операције које треба

пројектовати:

1. signal **KreirajNovoTakmicenje**(*Takmicenje*)

2. signal **ZapamtiTakmicenje** (*Takmicenje*)

#### ДС 2: Дијаграм секвенци случаја коришћења – Брисање такмичења

1. Администратор позива систем да учита податке о изабраном такмичењу. (АПСО)
2. Систем приказује администратору пронађене такмичења. (ИА)
3. Администратор позива систем да обрише такмичење. (АПСО)
4. Систем приказује администратору поруку: “Систем је обрисао такмичење.” (ИА)



Дијаграм 4: Брисање такмичења – основни сценарио

**Алтернативна сценарија**:

Уколико систем не може да нађе такмичења он приказује администратору

поруку: “Систем не може да нађе такмичења по задатим вредностима”. Прекида

се извршење сценарија. (ИА)



* 1. Уколико систем не може да нађе такмичења он приказује администратору

поруку: “Систем не може да нађе изабрано такмичења”. Прекида се извршење

сценарија. (ИА)



* 1. Уколико систем не може да запамти податке о такмичењу он приказује

администратору поруку: “Систем не може да запамти податке о такмичењу”. (ИА)



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

1. signal **PretraziTakmicenja**(KriterijumPretrage, List<*Takmicenje*>)
2. signal **ObrisiTakmicenje**(*Takmicenje*)

#### ДС 3: Дијаграм секвенци случаја коришћења – Унос лиге

1. Administratorpoziva sistem da kreira *novu ligu*. (APSO)
2. Sistem prikazuje administratoru*novu ligu*. (IA)
3. Administratorpoziva sistem da zapamti podatke o *novoj ligu*. (APSO)
4. Sistem prikazuje administratoruporuku: “Sistem je zapamtio *ligu*”. (IA)



**Alternativna scenarija:**

2.1. Ukoliko sistem ne može da kreira *novu ligu* on prikazuje administratoruporuku: “Sistem ne može da kreira *novu ligu*”. Prekida se izvršavanje scenarija. (IA)



* 1. Ukoliko sistem ne može da zapamti podatke o *novom takmičaru* on prikazuje administratoruporuku “Sistem ne može da zapamti *novog takmičara*”. (IA)



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

1. *signal* **KreirajNovuLigu**(*Takmicar*)
2. *signal* **ZapamtiLigu**(*Takmicar*)
3. *signal* ***UcitajListuTakmicara****(List<Takmicar>)*

#### ДС 4: Дијаграм секвенци случаја коришћења – Унос такмичара

1. Администратор позива систем да креира новог такмичара. (АПСО)
2. Систем приказује администратору новог такмичара. (ИА)
3. Администратор позива систем да запамти податке о новом такмичару.

(АПСО)

1. Систем приказује администратору поруку: “Систем је запамтио такмичара”.

(ИА)



**Алтернативна сценарија**

* 1. Уколико систем не може да креира новог такмичара он приказује

администратору поруку: “Систем не може да креира новог такмичара”. Прекида се

извршавање сценарија. (ИА)



* 1. Уколико систем не може да запамти податке о новом такмичару он приказује

администратору поруку “Систем не може да запамти новог такмичара”. (ИА)



Sa navedenih sekvencnih dijagrama uočavaju se 3 sistemske operacije koje treba projektovati:

1. *signal* **KreirajNovogTakmicara**(*Takmicar*)
2. *signal* **ZapamtiTakmicara**(*Takmicar*)
3. *signal* **UcitajListuLiga***(List<Liga>)*

#### DS 5: Dijagram sekvenci slučaja korišćenja – Измена такмичара

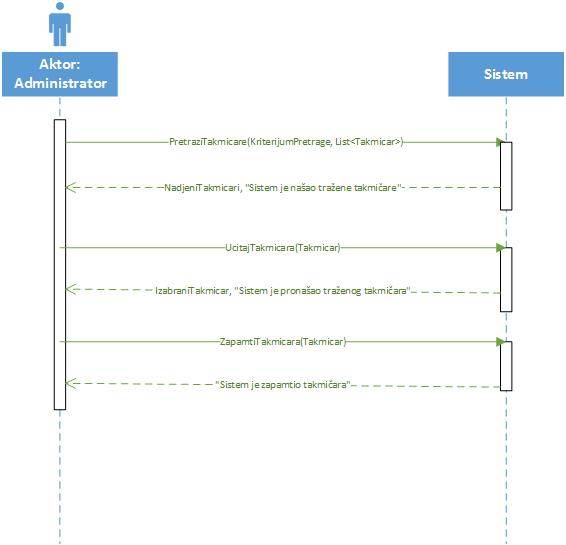
1. Администратор позива систем да нађе такмичаре по задатим вредностима.

(АПСО)

1. Систем приказује администратору нађене такмичаре. (ИА)
2. Администратор позива систем да учита податке о изабраном такмичару.

(АПСО)

1. Систем приказује администратору податке о изабраном такмичару. (ИА)
2. Администратор позива систем да запамти податке о такмичару. (АПСО)
3. Систем приказује администратору поруку: “систем је запамтио такмичара.”(ИА)

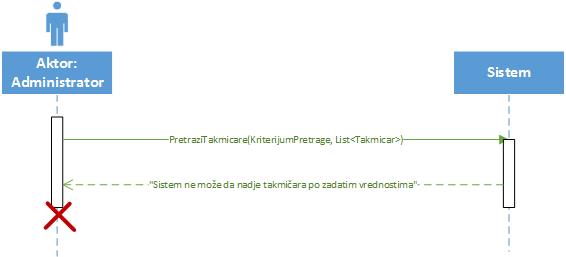
****

**Алтернативна сценарија**

2.1. Уколико систем не може да нађе такмичара он приказује администратору

поруку: “Систем не може да нађе такмичара по задатим вредностима”. Прекида

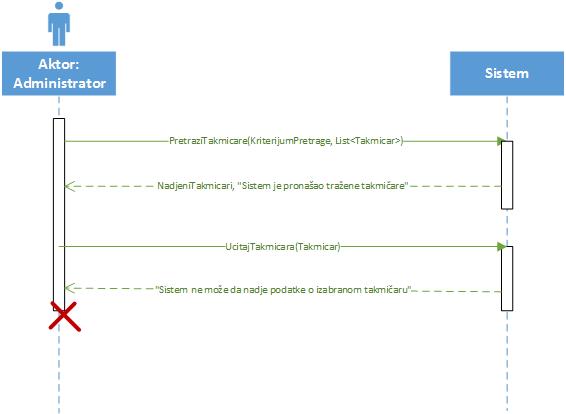
се извршење сценарија. (ИА)



4.1. Уколико систем не може да нађе такмичара он приказује администратору

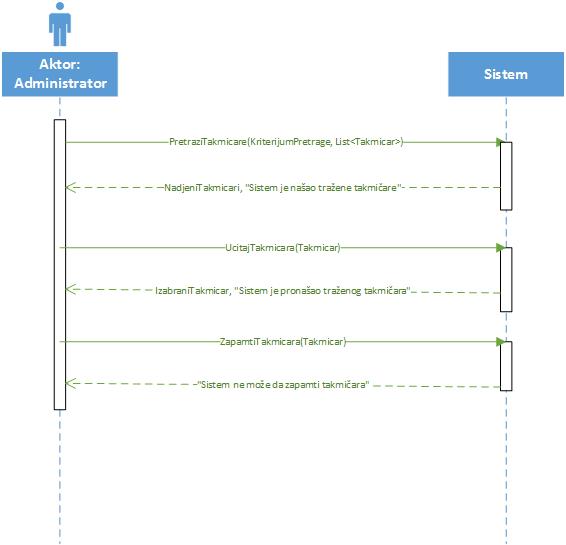
поруку: “Систем не може да нађе изабраног такмичара”. Прекида се извршење

сценарија. (ИА)



6.1. Уколико систем не може да запамти податке о такмичару он приказује

администратору поруку: “Систем не може да запамти такмичара”. (ИА)



Sa navedenih sekvencnih dijagrama uočavaju se 3 sistemske operacije koje treba projektovati:

1. *signal* **PretraziTakmicare**(*KriterijumPretrage*, *List*<*Takmicar>*)
2. *signal* **UcitajTakmicara**(*Takmicar*)
3. *signal* **ZapamtiTakmicara***(Takmicar)*

#### ДС 6: Дијаграм секвенци случаја коришћења – Брисање такмичара

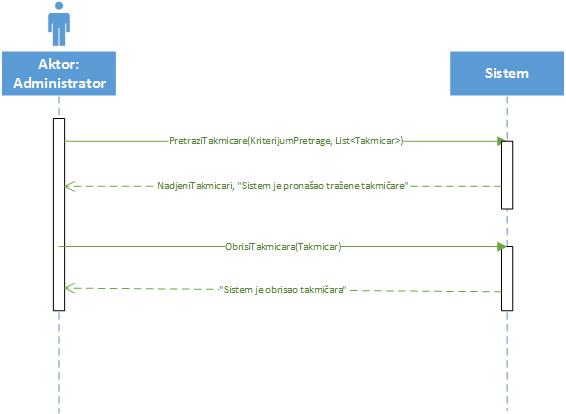
1. Администратор позива систем да нађе такмичаре по задатој вредности.

(АПСО)

1. Систем приказује администратору пронађене такмичаре и поруку: Систем је

нашао такмичаре по задатој вредностиʺ. (ИА)

1. Администратор позива систем да обрише такмичара. (АПСО)
2. Систем приказује администратору поруку: “Систем је обрисао такмичара”. (ИА)

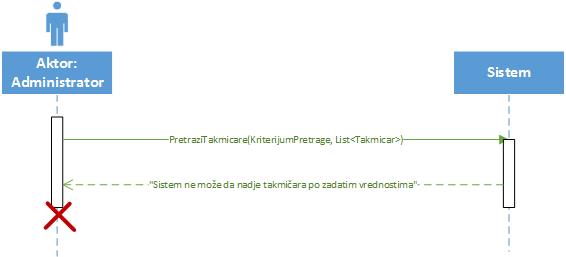
****

**Алтернативна сценарија**

2.1. Уколико систем не може да нађе такмичаре он приказује администратору поруку:

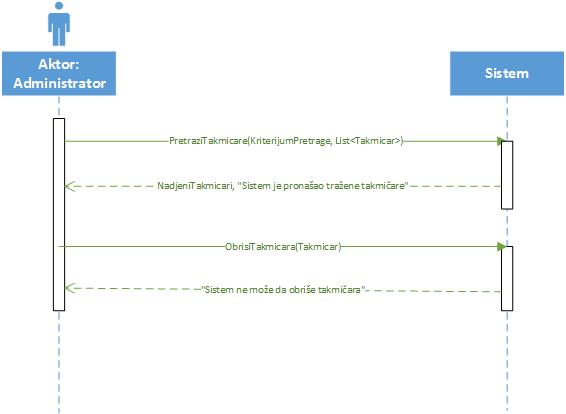
“Систем не може да нађе такмичара по задатој вредности”. Прекида се извршење

сценариа. (ИА)



4.1. Уколико систем не може да обрише такмичара он приказује администратору

поруку: “Систем не може да обрише такмичара”. (ИА)



Sa navedenih sekvencnih dijagrama uočavaju se sistemske operacije koje treba projektovati:

1. *signal* **PretraziTakmicare**(*KriterijumPretrage*, *List*<*Takmicar>*)
2. *signal* **ObrisiTakmicara**(*Takmicar*)

#### DS 7: Dijagram sekvenci slučaja korišćenja – Унос меча

1. Администратор позива систем да креира нов меч. (АПСО)
2. Систем приказује администратору нов меч. (ИА)
3. Администратор позива систем да запамти податке о новом мечу. (АПСО)
4. Систем приказује администратору поруку:” Систем је запамтио меч”. (ИА)



**Алтернативна сценарија**

2.1. Уколико систем не може да креира нов меч он приказује администратору

поруку: “Систем не може да креира нов меч”. Прекида се извршење сценарија. (ИА)



4.1. Уколико систем не може да запамти податке о новом мечу он приказује

администратору поруку “Систем не може да запамти нов меч”. (ИА)



Sa navedenih sekvencnih dijagrama uočavaju se 3 sistemske operacije koje treba projektovati:

1. *signal* **KreirajNoviMec**(*Mec*)
2. *signal* **ZapamtiMec**(*Mec*)
3. *signal* **UcitajListuTakmicara***(List<Takmicar>)*

Као резултат анализе сценарија добијено је укупно 16 системских операција које треба пројектовати:

1. signal***KreirajNovoTakmicenje****(Takmicenje)*
2. signal***ZapamtiTakmicenje*** *(Takmicenje)*
3. signal***PretraziTakmicenja****(KriterijumPretrage, List<Takmicenje>)*
4. signal **ObrisiTakmicenje**(*Takmicenje*)
5. signal **KreirajNovuLigu**(*Liga*)
6. signal **ZapamtiLigu**(*Liga*)
7. signal **UcitajListuTakmicara**(List<Takmicar>)
8. signal **UcitajListuLiga**(List<*Liga* >)
9. signal**KreirajNovogTakmicara***(Takmicar)*
10. signal **ZapamtiTakmicara**(*Takmicar*)
11. signal **PretraziTakmicare**(KriterijumPretrage, List<*Takmicar*>)
12. signal **UcitajTakmicara**(*Takmicar*)
13. signal **ObrisiTakmicara**(*Takmicar*)
14. signal **KreirajNoviMec**(*Mec*)
15. signal **ZapamtiMec**(*Mec*)

### 3.3.2. Понашање софтверског система – Дефинисање уговора о системским операцијама

**Ugovor UG1: KreirajNovoTakmicenje**

Operacija: KreirajNovoTakmicenje (Takmicenje):signal;

Veza sa SK: SK1

Preduslovi: *Prosto vrednosno ograničenje nad objektom Takmicenje mora biti zadovoljeno.*

Postuslovi: *Kreirano je novo takmičenje.*

**Ugovor UG2: ZapamtiTakmicenje**

Operacija: ZapamtiTakmicenje (Takmicenje):signal;

Veza sa SK: SK1, SK2

Preduslovi: *Prosto vrednosno ograničenje nad objektom Takmicenje mora biti zadovoljeno. Strukturno ograničenje nad objektom Takmicenje mora biti zadovoljeno.*

Postuslovi: *Podaci o takmičenju su zapamćeni.*

**Ugovor UG3: PretraziTakmicenja**

Operacija: PretraziTakmicenja(KriterijumPretrage, List<Takmicenje>):signal;

Veza sa SK: SK1, SK2, SK3, SK4

Preduslovi: -

Postuslovi: -

**Ugovor UG4: ObrisiTakmicenje**

Operacija: ObrisiTakmicenje(Takmicenje):signal;

Veza sa SK: SK2

Preduslovi: Strukturna ograničenja nad objektom Takmicar moraju biti zadovoljena.

Postuslovi: *Takmičar je obrisan.*

**Ugovor UG5: KreirajNovuLigu**

Operacija: KreirajNovuLigu (Liga):signal;

Veza sa SK: SK1

Preduslovi: *Prosto vrednosno ograničenje nad objektom Liga mora biti zadovoljeno.*

Postuslovi: *Kreirana je nova liga.*

**Ugovor UG6: ZapamtiLigu**

Operacija: **ZapamtiLigu** (Liga):signal;

Veza sa SK: SK4

Preduslovi: - *Prosto vrednosno ograničenje nad objektom* Liga *mora biti zadovoljeno. Strukturno ograničenje nad objektom* Liga *mora biti zadovoljeno*

Postuslovi: *- Podaci o* Ligi *su zapamćeni*

**Ugovor UG7: UcitajListuTakmicara**

Operacija: UcitajListuTakmicara (List<Takmicar>):signal;

Veza sa SK: SK 4

Preduslovi: -

Postuslovi: -

**Ugovor UG8: UcitajListuLiga**

Operacija: UcitajListuLiga (List<Liga>):signal;

Veza sa SK: SK5

Preduslovi: - *Prosto vrednosno ograničenje nad objektom liga mora biti zadovoljeno. Strukturno ograničenje nad objektom liga mora biti zadovoljeno*

Postuslovi: *- Podaci o ligi su zapamćeni*

**Ugovor UG9: KreirajNovogTakmicara**

Operacija: KreirajNovogTakmicara (Takmicar):signal;

Veza sa SK: SK4

Preduslovi: -

Postuslovi: -

**Ugovor UG10:** **ZapamtiTakmicara**

Operacija: ZapamtiTakmicara (Takmicar):signal;

Veza sa SK: SK5

Preduslovi*: - Prosto vrednosno ograničenje nad objektom takmicar mora biti zadovoljeno. Strukturno ograničenje nad objektom takmicar mora biti zadovoljeno*

Postuslovi*: Takmičar je zapamcen.*

**Ugovor UG11: PretraziTakmicare**

Operacija: PretraziTakmicare(KriterijumPretrage, List<Takmicar>):signal;

Veza sa SK: SK4, SK5, SK6, SK7

Preduslovi: -

Postuslovi: -

**Ugovor UG12: UcitajTakmicara**

Operacija: UcitajTakmicara (List<Takmicar>):signal;

Veza sa SK: SK 5

Preduslovi: -

Postuslovi: -

**Ugovor UG13: ObrisiTakmicara**

Operacija: ObrisiTakmicara(Takmicar):signal;

Veza sa SK: SK6

Preduslovi: Strukturna ograničenja nad objektom Takmicar moraju biti zadovoljena.

Postuslovi: *Takmičar je obrisan.*

**Ugovor UG14: KreirajNoviMec**

Operacija: KreirajNoviMec(Mec):signal;

Veza sa SK: SK7

Preduslovi: - Prosta vrednosna ograničenja nad objektom Meč moraju biti zadovoljena. Strukturna ograničenja isto moraju biti zadovoljena. Meč ne može da bude odigran od strane dva takmičara koja ne postoje u sistemu.

Postuslovi: - Meč je kreiran.

**Ugovor UG15: Zapamti Mec**

Operacija: ZapamtiMec (Mec):signal;

Veza sa SK: SK7

Preduslovi: - Prosta vrednosna ograničenja nad objektom mec moraju biti zadovoljena. Strukturna ograničenja isto moraju biti zadovoljena.

Postuslovi: - Podaci o meču su zapamćeni.

### 3.3.3. Структура софтверског система – Концептуални (доменски) модел



Дијаграм 16: Концептуални модел

### 3.3.4. Структура софтверског система – Релациони модел

1. **TipTakmicenja** (TipTakmicenjaID, NazivTipa, VrstaKruznogSistema)
2. **Takmicenje**(TakmicenjeID, Naziv, DatumPocetka, *TipTakmicenjaID*)
3. **Liga**(LigaID, Naziv, BrojMaxTakmicara, *TakmicenjeID*)
4. **Mesto**(Ptt, Naziv)
5. **Takmicar**(TakmicarID, Ime, Prezime, FBLink, Opis, PozicijaULigi, SetPlus, SetMinus, GemPlus, GemMinus, BrojPoena, BrojPobeda, BrojIzgubljenih, *Ptt, LigaID*)
6. **Mec**(MecID, Rezultat, *TakmicarDID*, *TakmicarGID*)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1. **Tabela TipTakmicenja** | | **Prosto vrednosno ograničenje** | | **Složeno vrednosno ograničenje** | | **Strukturno ograničenje** |
| **Atributi** | **Ime** | **Tip atributa** | **Vrednost atributa** | **Međuzav. atributa jedne tabele** | **Međuzav. atributa više tabela** | INSERT /  UPDATE CASCADES  DELETE RESTRICTED |
| TipTakmicenjaID | Long | not null and >0 |  |  |
| NazivTipa | String | not null |  |  |
| VrstaKruznogSistema | String | not null |  |  |

***Tabela 1. Tabela TipTakmicenja***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Tabela Takmicenje** | | **Prosto vrednosno ograničenje** | | **Složeno vrednosno ograničenje** | | **Strukturno ograničenje** |
| **Atributi** | **Ime** | **Tip atributa** | **Vrednost atributa** | **Međuzav. atributa jedne tabele** | **Međuzav. atributa više tabela** | INSERT  RESTRICTED  TipTakmicenja  UPDATE  CASCADES  DELETE CASCADES |
| TakmicenjeID | Long | not null and >0 |  |  |
| Naziv | Naziv | not null |  |  |
| DatumPocetka | Date | not null |  |  |
| *TipTakmicenjaID* | Long | not null |  |  |

***Tabela 2. Takmicenje***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Tabela Liga** | | **Prosto vrednosno ograničenje** | | **Složeno vrednosno ograničenje** | | **Strukturno ograničenje** |
| **Atributi** | **Ime** | **Tip atributa** | **Vrednost atributa** | **Međuzav. atributa jedne tabele** | **Međuzav. atributa više tabela** | INSERT /  UPDATE RESTRICTED Takmicenje  DELETE  RESTRICTED |
| LigaID | Integer | not null and >0 |  |  |
| Naziv | String | not null |  |  |
| BrojMaxTakmicara | String | not null |  |  |
|  | *TakmicenjeID* | Long | Not null |  |  |  |

***Tabela 3.Tabela Liga***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Tabela Mesto** | | **Prosto vrednosno ograničenje** | | **Složeno vrednosno ograničenje** | | **Strukturno ograničenje** |
| **Atributi** | **Ime** | **Tip atributa** | **Vrednost atributa** | **Međuzav. atributa jedne tabele** | **Međuzav. atributa više tabela** | INSERT  RESTRICTED  UPDATE RESTRICTED  DELETE RESTRICTED |
| Ptt | Long | not null and >0 |  |  |
| Naziv | String | not null |  |  |

***Tabela 4. Tabela Mesto***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Tabela Zanr** | | **Prosto vrednosno ograničenje** | | **Složeno vrednosno ograničenje** | | **Strukturno ograničenje** |
| **Atributi** | **Ime** | **Tip atributa** | **Vrednost atributa** | **Međuzav. atributa jedne tabele** | **Međuzav. atributa više tabela** | INSERT /  UPDATE RESTRICTED  DELETE RESTRICTED |
| TakmicarID | Long | not null and >0 |  |  |
| Ime | String | not null |  |  |
|  | Prezime | String |  |  |  |  |
|  | FBLink | String |  |  |  |  |
|  | Opis | String |  |  |  |  |
|  | PozicijaULigi | int |  |  |  |  |
|  | SetPlus | int |  |  |  |  |
|  | SetMinus | int |  |  |  |  |
|  | GemPlus | int |  |  |  |  |
|  | GemMinus | int |  |  |  |  |
|  | BrojPobeda | int |  |  |  |  |
|  | BrojIzgubljenih | int |  |  |  |  |
|  | Broj poena | int |  |  |  |  |
|  | Ptt | Long | Not null |  |  |  |
|  | LigaId | Long | Not null |  |  |  |

***Tabela 5. Tabela Takmicar***

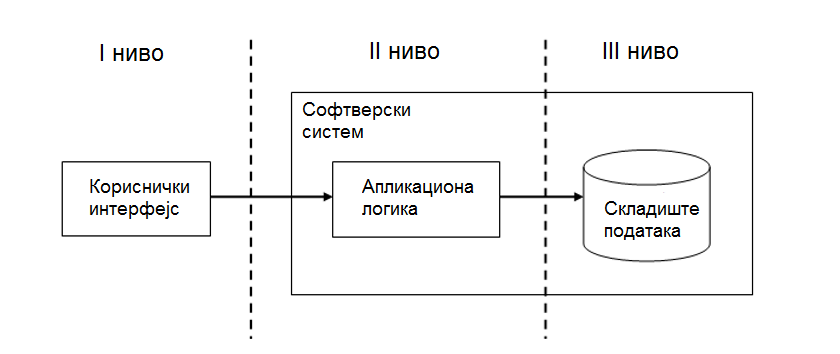
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Tabela Mec** | | **Prosto vrednosno ograničenje** | | **Složeno vrednosno ograničenje** | | **Strukturno ograničenje** |
| **Atributi** | **Ime** | **Tip atributa** | **Vrednost atributa** | **Međuzav. atributa jedne tabele** | **Međuzav. atributa više tabela** | INSERT  RESTRICTED  UPDATE RESTRICTED  DELETE / |
| MecID | Long | not null and >0 |  |  |
| Rezultat | String | not null |  |  |
| *TakmicarDID* | Long | not null  and > 0 |  |  |
| *TakmicarGID* | Long | not null  and > 0 |  |  |

***Tabela 6. Tabela Mec***

## 3.4. Пројектовање

Фаза пројектовања описује физичку структуру и понашање софтверског система (архитектуру софтверског система). Архитектура система се састоји од следћих нивоа:

* Kориснички интерфејс – преставља улазно-излазну репрезентацију софтверског система
* Aпликациона логика – описује структуру и понашање софтверског система
* Складиште података – чува стање атрибута софтверског система



##### **Слика 1: Тронивојска архитектура**

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

У фази пројектовања се описује архитектура софтверског система. Овај софтверски систем се састоји две одвојене целине. Прва целина представља серверску апликацију која комуницира са складиштем података. Такође садржи REST интерфејс помоћу кога пружа могућност различитим клијентским апликацијама да захтевају и добију разне податке из базе података.

Друга одвојена целина је клијентска веб апликација која је задужена за приказивање података корисницима веб апликације. Кориснички интерфејс прихвата улазне податке из пословног система и приказује резултате операција које су вршене над подацима.

### 3.4.1. Пројектовање корисничког интерфејса

Кориснички интерфејс представља реализацију улаза и/или излаза софтверског система[[6]](#footnote-6). У овом раду се кориснички интерфејс налази на веб страницама, HTML страницама. Веб стране прихватају податке које уноси корисник, догађаје које он прави, затим позивају контролера корисничког интерфејса, прослеђују му податке, а затим те податке приказују након што их систем обради.

##### **Слика 2: Структура корисничког интерфејса**

#### 3.4.1.1. СК 1: Случај коришћења – Унос tакмичења

**Назив СК**

Унос такмичења

**Актори СК**

Администратор

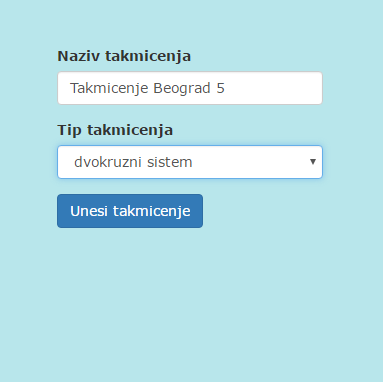
**Учесници СК**

Администратор и систем (програм)

**Предуслов**: Систем је укључен и администратор је улогован под својом шифром. Систем приказује форму за унос такмичења. Доступни су типови такмичења.

**Основни сценарио СК**

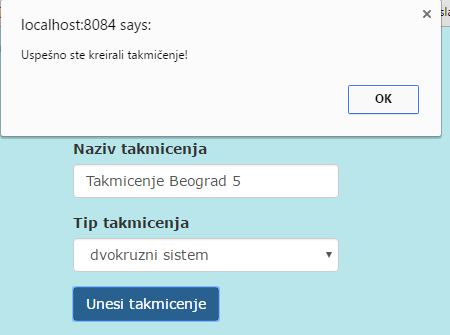
1. Администратор уноси податке о новом такмичењу. (АПУСО)



##### **Слика 3: Форма за унос такмичења**

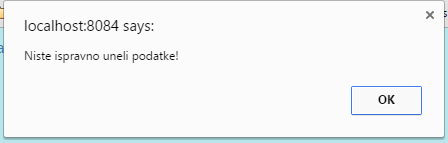
1. Администратор контролише да ли је коректно унео податке о новом такмичењу. (АНСО)
2. Администратор позива систем да креира ново такмичење и запамти податке о новом такмичењу. (АПСО)
3. Систем памти податке о новом такмичењу. (СО)
4. Систем приказује раднику поруку: “Систем је успешно креирао такмичење”. (ИА)

##### **Слика 4: Систем запамтио такмичење**



**Алтернативна сценарија:**

* 1. Уколико систем не може да запамти податке о новом такмичењу он приказује администратору поруку “Систем не може да запамти ново такмичење”. (ИА)



##### **Слика 5: Систем не може да запамти такмичење**

#### 3.4.1.2. СК 2: Случај коришћења – Брисање такмичења

**Назив СК**

Брисање такмичења

**Актори СК**

Администратор

**Учесници СК**

Администратор и систем (програм)

**Предуслов**: Систем је укључен и администратор је улогован под својом шифром. Потребно је да у систему постоји макар једно такмичење. Доступна је листа такмичења.

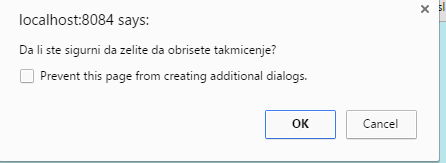
##### **Слика 6: Страна за приказ такмичења**



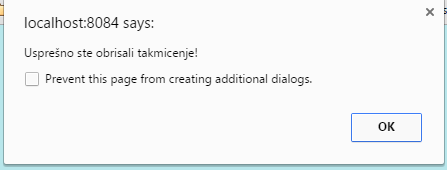
**Основни сценарио СК**

1. Администратор уноси вредности по којима претражује такмичења и бира члана којег жели да измени. (АПУСО)
2. Администратор позива систем да учита податке о изабраном такмичењу. (АПСО)
3. Систем тражи такмичења по задатој вредности. (СО)
4. Систем приказује администратору пронађене такмичења. (ИА)
5. Администратор бира такмичење. (АПУСО)

##### **Слика 7: Страна за приказ брисања такмичења**



1. Администратор позива систем да обрише такмичење. (АПСО)
2. Систем брише такмичење. (СО)
3. Систем приказује администратору поруку: “Систем је обрисао такмичење.” (ИА)



##### **Слика 8: Систем је обрисао такмичење**

**Алтернативна сценарија**:

* 1. Уколико систем не може да нађе такмичења он приказује администратору поруку: “Систем не може да нађе такмичење по задатој вредности”. Прекида се извршење сценарија. (ИА)
  2. Уколико систем не може да обрише такмичење он приказује администратору поруку : „Систем не може да обрише такмичење“. (ИА)

#### 3.4.1.3. СК 3: Случај коришћења – Унос нове лиге

**Назив СК**

Унос нове лиге

**Актори СК**

Администратор

**Учесници СК**

Администратор и систем (програм)

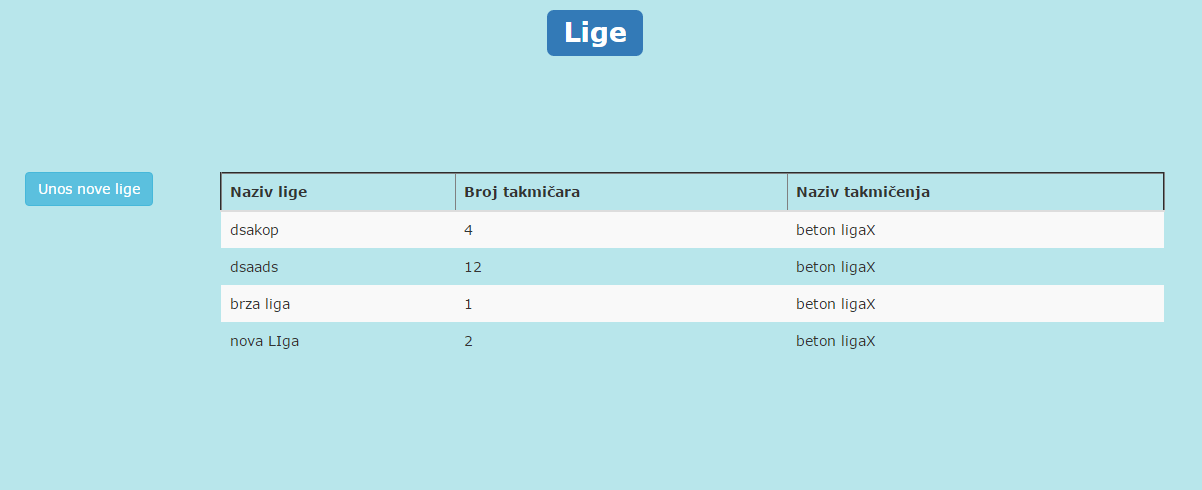
**Предуслов**: Систем је укључен и администратор је улогован под својом шифром и у

систему већ постоји такмичење. Систем приказује форму за рад са лигама.

**Основни сценарио СК**

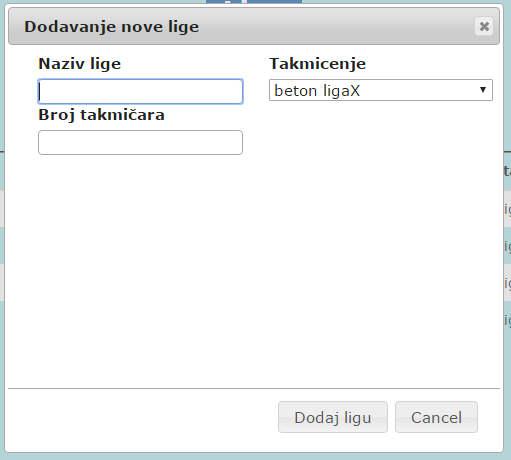
1. Администратор позива систем да креира нову лигу. (АПСО)

##### **Слика 9: Приказ свих лига**



1. Систем креира нову лигу. (СО)
2. Систем приказује администратору нову лигу. (ИА)
3. Администратор уноси податке о новој лиги. (АПУСО)

##### **Слика 9: Приказ форме за унос лиге**



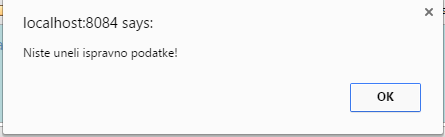
1. Администратор контролише да ли је исправно унео податке о новој лиги. (АНСО)
2. Администратор позива систем да запамти податке о новој лиги. (АПСО)
3. Систем памти податке о новој лиги. (СО)
4. Систем показује администратору поруку: „Систем је креирао нову лигу“. (ИА)



##### **Слика 10: Систем је креирао лигу**

**Алтернативна сценарија**

* 1. Уколико систем не може да креира нову лигу он приказује администратору поруку: “Систем не може да креира лигу”. Прекида се извршење сценарија. (ИА)



##### **Слика 11: Систем не може да обрише лигу**

* 1. Уколико систем не може да запамти податке о новој лиги он приказује администратору поруку: „Систем не може да запамти лигу“. (ИА)

#### 3.4.1.4. СК 4: Случај коришћења – Унос такмичара

**Назив СК**

Унос такмичара

**Актори СК**

Администратор

**Учесници СК**

Администратор и систем (програм)

**Предуслов:** Систем је укључен и администратор је улогован под својом шифром и у

систему већ постоји такмичење. Систем приказује форму за рад са такмичарима.

**Основни сценарио СК**

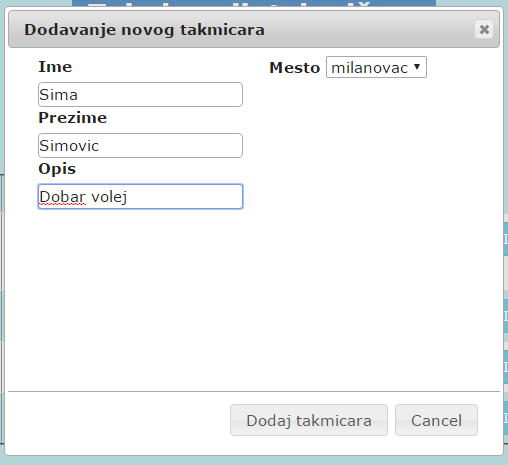
1. Администратор позива систем да креира новог такмичара. (АПСО)

##### **Слика 12: Приказ табеле свих такмичара**

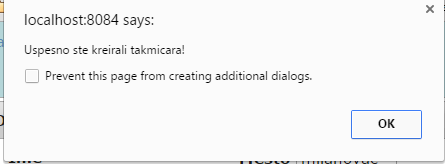


1. Систем креира новог такмичара. (СО)
2. Систем приказује администратору новог такмичара. (ИА)
3. Администратор уноси податке о новом такмичару. (АПУСО)

##### **Слика 13: Приказ форме за унос такмичара**



1. Администратор контролише да ли је коректно унео податке о новом такмичару. (АНСО)
2. Администратор позива систем да запамти податке о новом такмичару. (АПСО)
3. Систем памти податке о новом такмичару. (СО)
4. Систем приказује администратору поруку: “Систем је запамтио такмичара”.(ИА)



##### **Слика 14: Систем је запамтио такмичење**

**Алтернативна сценарија**

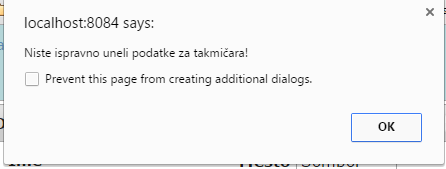
4.1. Уколико систем не може да креира новог такмичара он приказује

администратору поруку: “Систем не може да креира новог такмичара”. Прекида се

извршавање сценарија. (ИА)

Уколико систем не може да запамти податке о новом такмичару он приказује

администратору поруку “Систем не може да запамти новог такмичара”. (ИА)



##### **Слика 15: Систем не може да запамти такмичара**

#### 3.4.1.5. СК 5: Случај коришћења – Измена задужења

**Назив СК**

Измена задужења

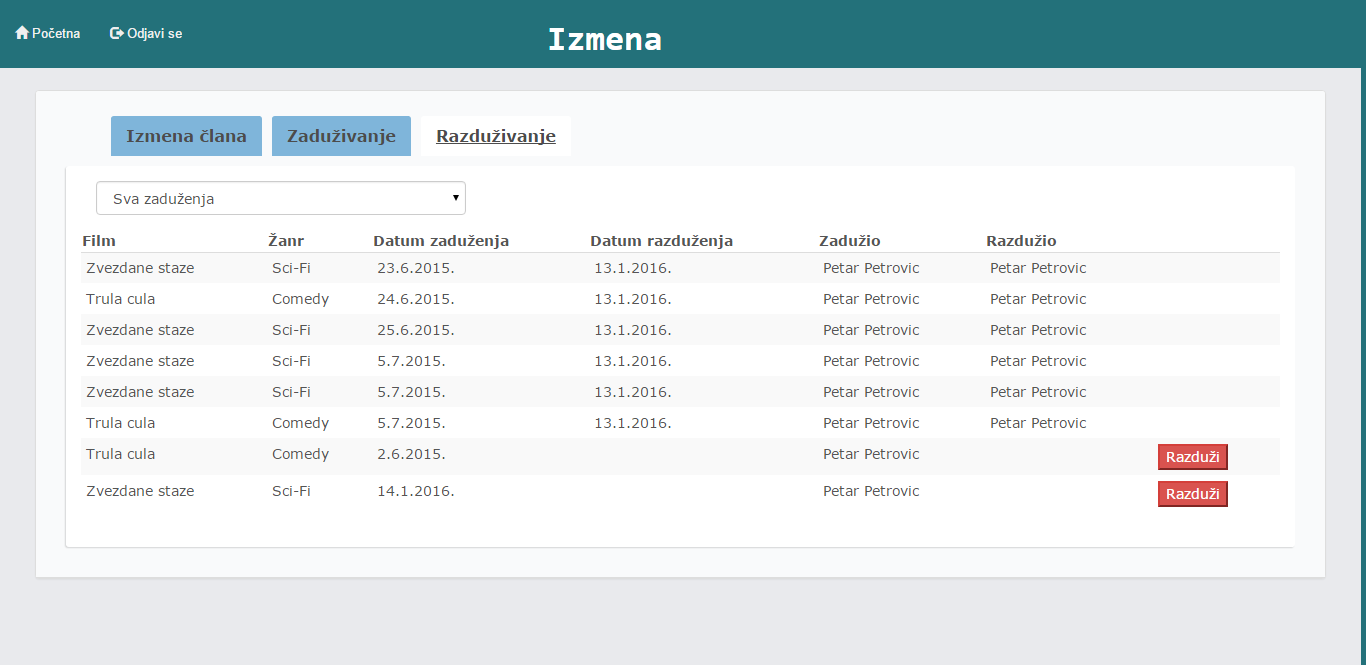
Актори СК

**Радник**

Учесници СК

**Радник и систем (програм)**

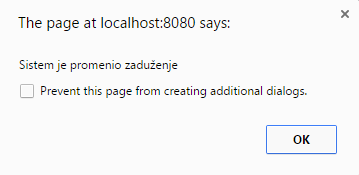
**Предуслов**: Систем је укључен и радник је улогован под својом шифром. Систем приказује форму за рад са задужењима.



##### **Слика 20: Страна за приказ задужења**

**Основни сценарио СК**

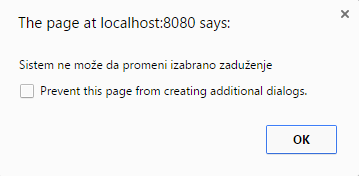
1. Радник бира задужење које жели да измени. (АПУСО)
2. Радник позива систем да измени податке о изабраном задужењу. (АПСО)
3. Систем мења и памти податке о изабраном задужењу. (СО)
4. Систем приказује раднику задужење и поруку: “Систем је променио задужење.” (ИА)



##### **Слика 21: Систем је променио задужење**

**Алтернативна сценарија**

4.1. Уколико систем не може да промени задужење он приказује раднику поруку: “Систем не може да промени изабрано задужење”. (ИА)



##### **Слика 22: Систем не може да промени изабрано задужење**

#### 3.4.1.6. СК 6: Случај коришћења – Претрага задужења

**Назив СК**

Претрага задужења

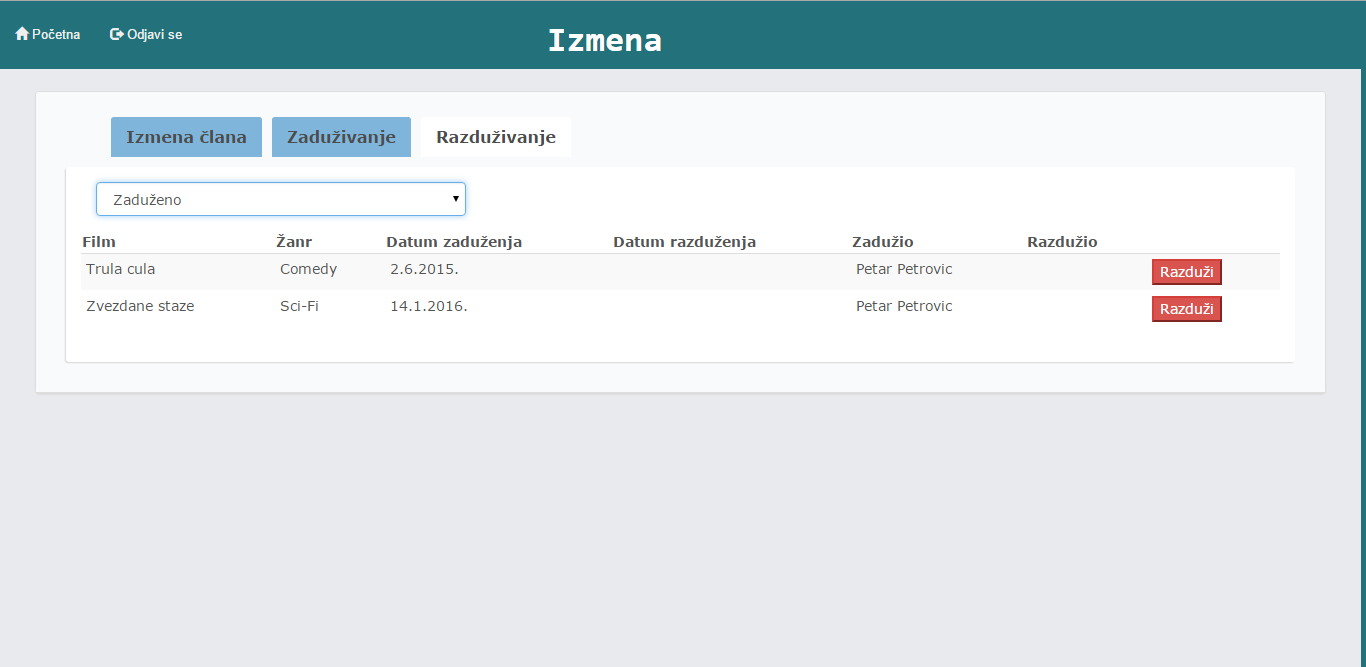
**Актори СК**

Радник

**Учесници СК**

Радник и систем (програм)

**Предуслов**: Систем је укључен и радник је улогован под својом шифром. Систем приказује форму за рад са задужењима.



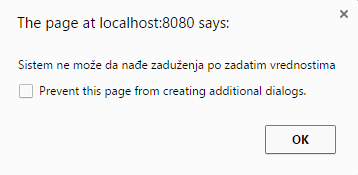
##### **Слика 23: Страна за приказ задужења**

**Основни сценарио СК**

1. Радник бира критеријум претраге. (АПУСО)
2. Радник позива систем да нађе задужења по задатим вредностима. (АПСО)
3. Систем тражи задужења по задатим вредностима. (СО)
4. Систем приказује раднику податке о задужењима. (ИА)

**Алтернативна сценарија**

4.1. Уколико систем не може да нађе задужења он приказује раднику поруку: “Систем не може да нађе задужења по задатим вредностима”. (ИА)



##### **Слика 24: Систем не може да прикаже задужења по задатим вредностима**

#### 3.4.1.7. СК 7: Случај коришћења – Пријављивање радника

**Назив СК**

Пријављивање радника

**Актори СК**

Радник

**Учесници СК**

Радник и систем (програм)

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



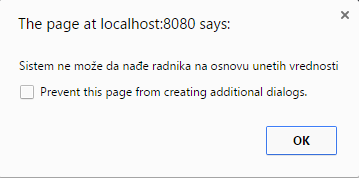
##### **Слика 25: Страна за пријаву на апликацију**

**Основни сценарио СК**

1. Радник уноси податке за аутентификацију радника. (АПУСО)
2. Радник позива систем да пронађе радника са задатим подацима. (АПСО)
3. Систем претражује раднике. (СО)
4. Систем приказује почетну форму”. (ИА)

**Алтернативна сценарија**

4.1. Уколико систем не може да нађе радника приказује поруку “Систем не може да нађе радника на основу унетих вредности”. (ИА)



##### **Слика 26: Систем не може да нађе радника**

### 3.4.2. Пројектовање контролера корисничког интерфејса

Контролер корисничког интерфејса је одговоран за:

* прихватање графичких објеката од екранске форме,
* конвертовање података који се налазе у графичким објектима у доменске објекте који ће бити прослеђени кроз мрежу до апликационог сервера
* конвертовање доменских објеката у графичке објекте и прослеђује их до екранске форме.

Екранска форма прослеђује податке и приказује податке добијене од контролера корисничког интерфејса.

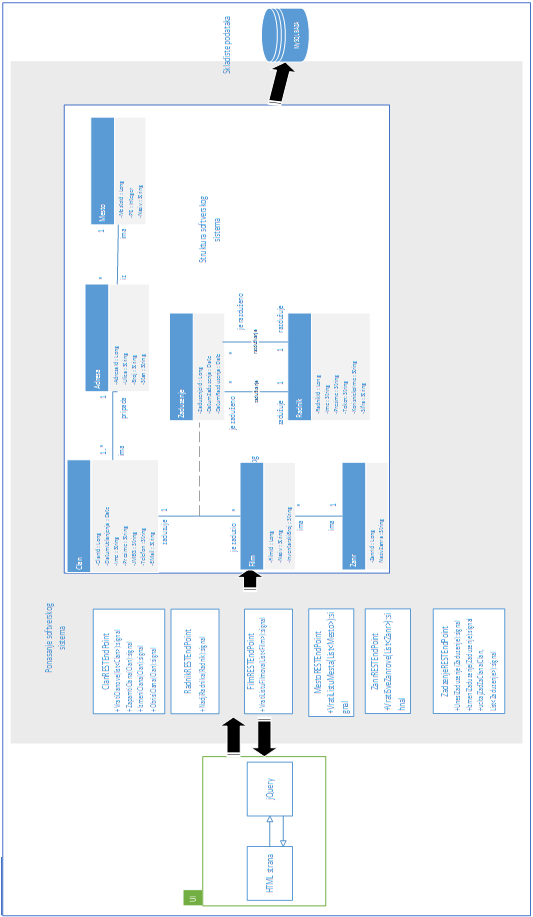
### 3.4.3. Пројектовање апликационе логике

Апликациони сервери треба да обезбеде сервисе који ће да омогуће реализацију апликационе логике софтверског система. Пројектовани апликациони сервер у овом примеру имплементиран је као REST вeб cервис и садржи:

* класе веб сервиса за комуникацију са клијентима – контролер апликационе логике

#### 3.4.3.1. Контролер апликационе логике

У фазама прикупљања захтева и анализе дата је спецификација структуре и понашања софтверског система, односно спецификација пословне логике софтверског система. Наредна слика даје опис система након фаза пројектовања комуникације са клијентима и контролера апликационе логике, кога у овом конкретном случају чине одређене класе веб сервиса.



Дијаграм 17: Архитектура система

#### **3.4.3.2. Пројектовање структуре софтверског система**

Софтверске класе структуре праве се на основу концептуалних класа које су раније дефинисане. Свака класа се састоји из приватних поља атрибута, гетера и сетера за те атрибуте, параметарског и непараметарског конструктора.

Дијаграм 18: Концептуални модел

**Слика 45: Структура софтверског система**

#### 3.4.3.3. Пројектовање понашања софтверског система

Системске операције, односно методе које су одговорне за њихово извршење имплементиране су у оквиру класа веб сервиса, односно у оквиру његових конкретних метода.

Свака системска операција представља посебну методу у одговарајућој класи веб сервиса, а по потреби позива и неке помоћне методе. Ове методе проверавају потребне предуслове, су задужене за операције са базом везане за конкретну класу.

#### 3.4.3.4. Уговор УГ1: ZapamtiClana

Операција: ZapamtiClana(*Clan*)::сигнал;

Веза са СК: СК1

Предуслови: *Просто вредносно ограничење над објектом Clan мора бити задовољено*.

*Структурно ограничење над објектом Clan мора бити задовољено.*

Постуслови: *Подаци о члану су запамћени.*

Дијаграм 19: СО - ZapamtiClana

#### 3.4.3.5. Уговор УГ2: UcitajClana

Операција: UcitajClana(*Clan*):сигнал;

Веза са СК: СК2, СК6

Предуслови: -

Постуслови: -

Дијаграм 20: СО - UcitajClana

#### 3.4.3.6. Уговор УГ3: ObrisiClana

Операција: ObrisiClana(*Clan*):сигнал;

Веза са СК: СК3

Предуслови: -

Постуслови: *Члан је обрисан. Структурно ограничење над објектом Clan мора бити задовољено.*

Дијаграм 21: СО - ObrisiClana

#### 3.4.3.7. Уговор УГ4: IzmeniClana

Операција: ZapamtiZaduzenje(*Zaduzenje*):сигнал;

Веза са СК: СК2

Предуслови: *Просто вредносно ограничење над објектом Zaduzenje мора бити задовољено. Структурно ограничење над објектом Zaduzenje мора бити задовољено.*

Постуслови: *Подаци о задужењу су запамћени.*

Дијаграм 22: СО - IzmeniClana

#### 3.4.3.8. Уговор УГ5: UnesiZaduzenje

Операција: UcitajZaduzenje(*Zaduzenje*):сигнал;

Веза са СК: СК4

Предуслови: -

Постуслови: -

Дијаграм 23: СО - UnesiZaduzenje

#### 3.4.3.9. Уговор УГ6: IzmeniZaduzenje

Операција: UcitajZaduzenje(*Zaduzenje*):сигнал;

Веза са СК: СК5

Предуслови: -

Постуслови: -

Дијаграм 24: СО - IzmeniZaduzenje

#### 3.4.3.10. Уговор УГ7: VratiListuMesta

Операција: UcitajZaduzenje(*Zaduzenje*):сигнал;

Веза са СК: СК1, СК2

Предуслови: -

Постуслови: -

Дијаграм 25: СО – VratiListuMesta

#### 3.4.3.11. Уговор УГ8: VratiListuFilmova

Операција: UcitajZaduzenje(*Zaduzenje*):сигнал;

Веза са СК: СК4

Предуслови: -

Постуслови: -

Дијаграм 26: СО – VratiListuFilmova

#### 3.4.3.12. Уговор УГ9: VratiClanove

Операција: UcitajZaduzenje(*Zaduzenje*):сигнал;

Веза са СК: СК2, СК3

Предуслови: -

Постуслови: -

Дијаграм 27: СО - VratiClanove

#### 3.4.3.13. Уговор УГ10: NadjiRadnika

Операција: NadjiRadnika(*Radnik*):сигнал;

Веза са СК: СК7

Предуслови: -

Постуслови: -

Дијаграм 28: СО - NadjiRadnika

#### 3.4.3.14. Уговор УГ11: VratiSveZanrove

Операција: VratiSveZanrove(*List<Zanr>*):сигнал;

Веза са СК: СК4, СК5

Предуслови: -

Постуслови: -

Дијаграм 29: СО - VratiSveZanrove

### 3.4.4. Пројектовање складишта података

У раду је коришћен МySQL систем за управљање базом података. Пројектоване су следеће табеле:

|  |  |  |
| --- | --- | --- |
| **Table: Clan** | | |
| ID | Bigint | 20 |
| datum\_uclanjenja | Date |  |
| Ime | Varchar | 100 |
| Prezime | Varchar | 100 |
| JMBG | Varchar | 13 |
| Telefon | Varchar | 100 |
| Mail | Varchar | 100 |
| Ulica | Varchar | 100 |
| Stan | Varchar | 100 |
| Broj | Varchar | 100 |
| Mesto | Bigint | 20 |

***Табела 9. Табела Clan***

|  |  |  |
| --- | --- | --- |
| **Table: Radnik** | | |
| ID | Bigint | 20 |
| Ime | Varchar | 100 |
| Prezime | Varchar | 100 |
| Token | Varchar | 255 |
| Korisnicko\_Ime | Varchar | 100 |
| Sifra | Varchar | 25 |
| ***Табела 10. Radnik*** | | |
| **Table: Mesto** | | |
| ID | Bigint | 20 |
| PttBroj | Int | 11 |
| Naziv | Varchar | 100 |

***Табела 11. Табела Radnik***

|  |  |  |
| --- | --- | --- |
| **Table: Film** | | |
| ID | Bigint | 20 |
| Naziv | Varchar | 100 |
| Inventarski\_Broj | Varchar | 100 |
| Zanr | Bigint | 20 |

***Табела 12. Табела Film***

|  |  |  |
| --- | --- | --- |
| **Table: Zanr** | | |
| ID | Bigint | 20 |
| Naziv\_zanra | Varchar | 100 |

***Табела 13. Табела Zanr***

|  |  |  |
| --- | --- | --- |
| **Table: Zaduzenje** | | |
| ID | Bigint | 20 |
| Datum\_Zaduzenja | Date |  |
| Datum\_Razduzenja | Date |  |
| Zaduzio | Bigint | 20 |
| Razduzio | Bigint | 20 |
| Clan | Bigint | 20 |
| Film | Bigint | 20 |

***Табела 14. Табела Zaduzenje***

### 3.4.5. Архитектура софтверског система

Архитектура софтверског система је тронивојска. Као што се може видети са дијаграма испод, систем је подељен у 3 целине. Прву целину чини презентациони део, односно корисничи интерфејс који ће бити имплементиран у стандардним Web технологијама као што су HTML и jQuery (библиотека за JavaScript). На нивоу апликационе логике налазе се REST сервиси који управљају променама над доменом, односно подацима у бази података, али и враћају захтеване податке. REST сервиси биће имплементирани у Jersey радном оквиру, док ће база података бити имплементирана у MySQL-у.

### 

Дијаграм 30: Архитектура система

## 3.6. Имплементација

Све наведене компоненте које су добијене у фази пројектовања улазе у фазу имплементације где се и реализују у некој од постојећих платформи. За имплементацију ове тронивојске архитектуре коришћена је Јава ЕЕ технологија са подршком Apache Tomcat 8.0.27 сервера, у развојном окружењу NetBeans 8.1., док је MySql служио као систем за управљање базом података, у оквиру Wamp сервера 2.5. Кориснички интерфејс је имплементиран помоћу jQuery-ja у HTML формату.

Доменске класе у пакету com.romb.njt.domain:

* Administrator.java
* Liga.java
* Mec.java
* Mesto.java
* Takmicar.java
* Takmicenje.java
* TipTakmicenja.java

Сопствени изузеци у пакету com.romb.njt.exception:

* AbstractException.java.java
* BasicAuthenticationException.java
* DataNotFoundException.java
* ErrorMessage.java
* MyRollbackException.java
* NotAuthorizedException.java
* ValidationException.java

Помоћне класе у пакету com.romb.njt.helper

* AbstractTokenCreator.java
* Base64Token.java
* EMF.java
* JsonToken.java
* RestHelpClass.java

REST сервиси у пакету com.romb.njt.service

* AdministratorREST.java
* LigaRESTEndpoint.java
* MatchRESTEndpoint.java
* MestoRESTEndpoint.java
* TakmicarRESTEndpoint.java
* TakmicenjeREST.java
* TipTakmicenjaREST

HTML стране:

* index.html
* pocetna.html
* lige.html
* takmicari.html
* takmicenje.html
* unosTakmicenja.html

Остали статички фајлови налазе се у **assets** фолдеру.

## 3.7. Тестирање

Сваки од имплементираних случајева коришћења је тестиран. Приликом тестирања сваког случаја коришћења, поред унетих правилних података, уношени су и неправилни подаци да би се утврдило какав ће резултат извршења бити.На основу извршених тестирања отклоњени су уочени недостаци.

# 4. Закључак

Овај завршни рад прати развој информационог система за поједине функције службе кућног лечења. У уводном делу дипломског рада дато је детаљно упознавање са технологијом веб сервиса.

У раду су приказане фазе животног циклуса развоја софтверског система, а коришћена је упрошћена Ларманова метода развоја софтвера, која предвиђа следеће фазе развоја:

* Фаза прикупљања захтева је представљена преко модела случајева коришћења.
* Фаза анализе је описана помоћу системских дијаграма секвенци и уговора (понашање софтверског система), концептуалног и релационог модела (структура софтверског система).
* Фаза пројектовања представљена је помоћу дијаграма класа и дијаграма секвенци, а садржи и пројектовање корисничког интерфејса.
* Фаза имплементације се заснива на резултатима фазе пројектовања, а извршена је коришћењем NetBeans развојног окружења
* Фаза тестирања није детаљно разматрана, пошто нису писани аутоматски тестови.

Радећи на овој апликацији увидео сам неке предности и мане у односу на неке друге технологије које сам у свом досадашњем раду користио. Почећу са манама, односно са стварима које су мене конкретно мучиле кроз овај рад, јер сам увек размишљао о томе шта би се десило да апликација много нарасте. Ту наилазим на први проблем, JPA ми је био од изузетне користи јер је по завршетку пројектовања базе омогућио да се изузетном брзином генерише модел и скоро све што је везано за рад са базом података, међутим, приметио сам да је ефикасан када је потребно обрадити не тако велике количине података, јер је ипак на далеко вишем нивоу апстракције него што је потребно за рад са базом. Друго, приметио сам да ми је на крају доста више времена одузео рад са jQuery библиотеком, него што би то било да сам користио неки радни оквир, као што je AngularJS.

Са друге стране, од самог почетка рада са REST архитектуром, био сам веома заинтригиран тиме како функционише, како се на врло једноставан начин може решити проблем проширивости и у смислу функционалности апликација, али и у смислу додавања нових сервера који ће опслуживати кориснике када за то буде било потребе. Bootstrap радни оквир још једна је ставка која ме је изузетно привукла што због своје једноставности, што због изузетне лакоће коришћења и изузетног ефекта који гради.

Изабрао сам овај сет технологија за свој рад јер сматрам да су веома модерне и због тога сам хтео да их изучим боље.

# 5. Литература

1. Др Синиша Влајић, Душан Савић, Војислав Станојевић, Илија Антовић, Милош Милић, Пројектовање софтвера-Напредне Јава технологије,Београд 2008
2. Java WebServices, <http://docs.oracle.com/javaee/6/tutorial/doc>, приступано јануара 2016.
3. Jersey 2.7 User Guide, <https://jersey.java.net/documentation/2.7/user-guide.html>, приступано јануара 2016.
4. REST with Java (JAX-RS) using Jersey, <http://www.vogella.com/tutorials/REST/article.html>, приступано јануара 2016.
5. Maven build tool, <https://maven.apache.org/what-is-maven.html>, приступано јануара 2016.
6. Web Services Glossary, <http://www.w3.org/TR/ws-gloss/>, приступано јануара 2016.
7. RESTful Web services: The basics, <http://www.ibm.com/developerworks/library/ws-restful/>, приступано јануара 2016.
8. What is Bootstrap, <http://stackoverflow.com/questions/14546709/what-is-bootstrap>, приступано децембра 2015.

1. Др Синиша Влајић, Душан Савић, Војислав Станојевић, Илија Антовић, Милош Милић, Пројектовање софтвера-Напредне Јава технологије,Београд 2008 [↑](#footnote-ref-1)
2. Java WebServices, <http://docs.oracle.com/javaee/6/tutorial/doc>, приступано децембра 2015. [↑](#footnote-ref-2)
3. <https://maven.apache.org/what-is-maven.html> приступано децембра 2015. [↑](#footnote-ref-3)
4. <http://www.w3.org/TR/ws-gloss/>, приступано новембра 2015. [↑](#footnote-ref-4)
5. <http://www.ibm.com/developerworks/library/ws-restful/>, приступано новембра 2015. [↑](#footnote-ref-5)
6. Др Синиша Влајић, Душан Савић, Војислав Станојевић, Илија Антовић, Милош Милић, Пројектовање софтвера-Напредне Јава технологије,Београд 2008 [↑](#footnote-ref-6)