УНИВЕРЗИТЕТ У БЕОГРАДУ

ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

**ЗАВРШНИ РАД**

Тема: Софтверски систем за праћење откупа шампињона у Јава окружењу

Ментор: Студент:

Проф. др Синиша Влајић Марко Барачков 118/11

Београд, 2016. Године

Садржај

[1. Увод 4](#_Toc452322944)

[2. Преглед коришћене технологије 5](#_Toc452322945)

[2.1. Јава платформа 5](#_Toc452322946)

[2.1.1. Java 2 EE платформа 7](#_Toc452322947)

[2.2. Алат за изградњу пројекта - Maven 7](#_Toc452322948)

[2.3. Веб сервиси 8](#_Toc452322949)

[2.3.1. REST Web сервиси 9](#_Toc452322950)

[2.4. Клијентска страна – Bootstrap, jQuery 17](#_Toc452322951)

[3. Студијски пример 19](#_Toc452322952)

[3.1. Ларманова метода 19](#_Toc452322953)

[3.2. Кориснички захтеви 22](#_Toc452322954)

[3.2.1. Вербални опис 22](#_Toc452322955)

[3.2.2. Случајеви коришћења 23](#_Toc452322956)

[СК 1: Случај коришћења – Унос добављача 24](#_Toc452322957)

[СК 2: Случај коришћења – Измена добављача 25](#_Toc452322958)

[СК 3: Случај коришћења – Брисање добављача 26](#_Toc452322959)

[СК 4: Случај коришћења – Унос задужења 27](#_Toc452322960)

[СК 5: Случај коришћења – Измена задужења 28](#_Toc452322961)

[СК 6: Случај коришћења – Унос дневне бербе (Сложен) 29](#_Toc452322962)

[СК 7: Случај коришћења – Креирање обрчуна и статистике 30](#_Toc452322963)

[СК 8: Случај коришћења – Пријављивање административног радника 31](#_Toc452322964)

[3.3. Анализа 32](#_Toc452322965)

[3.3.1. Понашање софтверског система – Системски дијаграм секвенци 32](#_Toc452322966)

[ДС 1: Дијаграм секвенци случаја коришћења – Унос добављача 32](#_Toc452322967)

[ДС 2: Дијаграм секвенци случаја коришћења – Измена добављача 34](#_Toc452322968)

[ДС 3: Дијаграм секвенци случаја коришћења – Брисање добављача 36](#_Toc452322969)

[ДС 4: Дијаграм секвенци случаја коришћења – Унос задужења 38](#_Toc452322970)

[ДС 5: Дијаграм секвенци случаја коришћења – Измена задужења 40](#_Toc452322971)

[ДС 6: Дијаграм секвенци случаја коришћења – Унос дневне бербе 42](#_Toc452322972)

[ДС 7: Дијаграм секвенци случаја коришћења – Креирање обрачуна и статистике 45](#_Toc452322973)

[ДС 8: Дијаграм секвенци случаја коришћења – Пријављивање административног радника 47](#_Toc452322974)

[3.3.2. Понашање софтверског система – Дефинисање уговора о системским операцијама 49](#_Toc452322975)

[Уговор УГ1: UcitajListuMesta 49](#_Toc452322976)

[Уговор УГ2: KreirajNovogDobavljaca 49](#_Toc452322977)

[Уговор УГ3: VratiListuDobavljaca 49](#_Toc452322978)

[Уговор УГ4: IzmeniDobavljaca 49](#_Toc452322979)

[Уговор УГ5: ObrisiDobavljaca 49](#_Toc452322980)

[Уговор УГ6: UnosZaduzenja 50](#_Toc452322981)

[Уговор УГ7: IzmeniZaduzenje 50](#_Toc452322982)

[Уговор УГ8: VratiZaduzenja 50](#_Toc452322983)

[Уговор УГ9: UbaciDnevnuBerbu 50](#_Toc452322984)

[Уговор УГ10: VratiListuDnevnihBerbi 50](#_Toc452322985)

[Уговор УГ11: SacuvajStavkeDnevneBerbe 50](#_Toc452322986)

[Уговор УГ12: KreirajObracunStatistiku 51](#_Toc452322987)

[Уговор УГ13: NadjiAdministrativnogRadnika 51](#_Toc452322988)

[3.3.3. Структура софтверског система – Концептуални (доменски) модел 52](#_Toc452322989)

# 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 софтверског система за праћење откупа шампињона 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 прасирање 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 да процес израде иде лако
* 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).

* SistemZaPracenjeDobavljacaSampinjona/rest/dobavljac?search=marko

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.

* SistemZaPracenjeDobavljacaSampinjona/rest/dobavljac/1

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:** SistemZaPracenjeDobavljacaSampinjona/rest/dobavljac | Враћа све добављаче | Мења добављача прослеђеним подацима | Додаје добављача | / |
| **URI који означава појединачни елемент:** SistemZaPracenjeDobavljacaSampinjona/rest/dobavljac/1 | Враћа добављача који има ид 1 | / | / | Брише конретног добављача |

***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 добављаче који постоје у бази:

@GET

@Produces(MediaType.APPLICATION\_JSON)

public Response getDobavljac(@HeaderParam("authorization") String authorization, @QueryParam("search") String search, @QueryParam("place") String place) {

EntityManager em = helper.getEntityManager();

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

StringBuilder query = new StringBuilder("SELECT d FROM Dobavljac d ");

if (search != null) {

query.append("WHERE (d.jmbg LIKE '%")

.append(search)

.append("%' OR d.ime LIKE '%")

.append(search)

.append("%' OR d.prezime LIKE '%")

.append(search)

.append("%' OR d.mesto.naziv LIKE '%")

.append(search)

.append("%')");

}

if (place != null) {

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

.append("d.mesto.naziv = '")

.append(place)

.append("'");

}

System.out.println(query);

List<Dobavljac> dobavljaci = em.createQuery(query.toString()).getResultList();

if (dobavljaci.isEmpty()) {

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

} else {

return Response.ok().entity(dobavljaci).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 рада DVD клуба је пројектован као тронивојска Wеб апликација. На презентационом нивоу налази се wеб претраживач који приказује HTML странице. Апликациона логика је имплментирана 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 пријављивање андмистративног радника, вођење евиденције о добављачима.

Апликација треба да омогући претрагу добављача према одређеним вредностима, као и унос нових добављача, измену података о постојећим добављачима и брисање истих.

За сваког добављача потребно је водити евиденцију о дневној берби као и о задужењима и раздужењима. Треба омогућити унос и измену задужења. Такође је потребно омогућити да се за одређени датум креирају обрачуни свих добављача као и статистика за тај период.

Да би административни радник могао да врши измене над подацима у систему, потребно је да буде пријављен на систем.

### Случајеви коришћења

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

1. Унос добављача
2. Измена добављача
3. Брисање добављача
4. Унос задужења
5. Измена задужења
6. Унос дневне бербе (Сложен)
7. Креирање обрачуна и статистике
8. Пријављивање административног радника



*Дијаграм 1: Дијаграм случајева коришћења*

### СК 1: Случај коришћења – Унос добављача

**Назив СК**

Унос добављача

**Актори СК**

Административни радник

**Учесници СК**

Административни радник и систем (програм)

**Предуслов**: Систем је укључен и андминистративни радник је улогован под својом шифром. Систем приказује форму за рад са добављачима. Доступна је листа места.

**Основни сценарио СК**

1. Административни радник уноси податке о новом добављачу. (АПУСО)
2. Административни радник контролише да ли је коректно унео податке о новом добављачу. (АНСО)
3. Административни радник позива систем да креира новог добављача и запамти податке о новом добављачу. (АПСО)
4. Систем памти податке о новом добављачу. (СО)
5. Систем приказује административном раднику поруку: “Систем је запамтио добављача”. (ИА)

**Алтернативна сценарија:**

* 1. Уколико систем не може да запамти податке о новом добављачу он приказује административном раднику поруку “Систем не може да запамти новог добављача”. (ИА)

### СК 2: Случај коришћења – Измена добављача

**Назив СК**

Измена добављача

**Актори СК**

Административни радник

**Учесници СК**

Административни радник и систем (програм)

**Предуслов**: Систем је укључен и андминистративни радник је улогован под својом шифром. Систем приказује форму за рад са добављачима. Доступна је листа места и листа добављача.

**Основни сценарио СК**

1. Административни радник уноси податке по којима претражује добављаче, бира добављача и уноси (мења) податке о добављачу. (АПУСО)
2. Административни радник контролише да ли је коректно унео податке о добављачу. (АНСО)
3. Административни радник позива систем да измени податке о добављачу. (АПСО)
4. Систем мења податке о добављачу. (СО)
5. Систем приказује адмнинистративном раднику поруку: “Систем је успешно променио податке о добављачу.” (ИА)

**Алтернативна сценарија**:

* 1. Уколико систем не може да промени податке о добављачу он приказује адмнинистративном раднику поруку: “Систем не може да промени добављача”. (ИА)

### СК 3: Случај коришћења – Брисање добављача

**Назив СК**

Брисање добављача

**Актори СК**

Административни радник

**Учесници СК**

Административни радник и систем (програм)

**Предуслов**: Систем је укључен и административни радник је улогован под својом шифром. Систем приказује форму за рад са добављачима. Доступна је листа добављача.

**Основни сценарио СК**

1. Административни радник уноси вредности по којим претражује добављаче и бира добављача којег жели да обрише. (АПУСО)
2. Административни радник позива систем да обрише добављача. (АПСО)
3. Систем брише добављача. (СО)
4. Систем приказује административном раднику поруку: “Систем је обрисао добављача.” (ИА)

**Алтернативна сценарија**

* 1. Уколико систем не може да обрише добављача он приказује административном раднику поруку: “Систем не може да обрише добављача”. (ИА)

### СК 4: Случај коришћења – Унос задужења

**Назив СК**

Унос задужења

**Актори СК**

Административни радник

**Учесници СК**

Административни радник и систем (програм)

**Предуслов:** Систем је укључен и административни радник је улогован под својом шифром. Селектован је добављач. Систем приказује форму за рад са задужењима за тог добављача.

**Основни сценарио СК**

1. Административни радник уноси податке о новом задужењу. (АПУСО)
2. Административни радник контролише да ли је коректно унео податке о новом задужењу. (АНСО)
3. Административни радник позива систем да запамти податке о новом задужењу. (АПСО)
4. Систем памти податке о новом задужењу. (СО)
5. Систем приказује административном раднику поруку:”Систем је запамтио задужење”. (ИА)

**Алтернативна сценарија**

* 1. Уколико систем не може да запамти податке о новом задужењу он приказује административном раднику поруку “Систем не може да запамти ново задужење”. (ИА)

### СК 5: Случај коришћења – Измена задужења

**Назив СК**

Измена задужења

**Актори СК**

Административни радник

**Учесници СК**

Административни радник и систем (програм)

**Предуслов**: Систем је укључен и административни радник је улогован под својом шифром. Селектован је добављач. Систем приказује форму за рад са задужењима за тог добављача. Доступна је листа задужења.

**Основни сценарио СК**

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

**Алтернативна сценарија**

* 1. Уколико систем не може да запамти податке о задужењу он приказује адмнинистративном раднику поруку: “Систем не може да запамти задужење”. (ИА)

### СК 6: Случај коришћења – Унос дневне бербе (Сложен)

**Назив СК**

Унос дневне бербе

**Актори СК**

Административни радник

**Учесници СК**

Административни радник и систем (програм)

**Предуслов**: Систем је укључен и андминистративни радник је улогован под својом шифром. Селектован је добављач. Систем приказује форму за рад са дневном бербом за тог добављача. Доступна је листа дневних берби добвљача.

**Основни сценарио СК**

1. Административни радник позива систем да креира дневну бербу. (АПСО)
2. Систем креира дневну бербу. (СО)
3. Систем приказује административном раднику дневну бербу и поруку “Систем је креирао дневну бербу”. (ИА)
4. Административни радник уноси ставке дневне бербе. (АПУСО)
5. Административни радник контролише да ли је коректно унео ставке. (АНСО)
6. Административни радник позива систем да запамти ставке дневне бербе. (АПСО)
7. Систем памти ставке о дневној берби. (СО)
8. Систем приказује административном раднику поруку: “Систем је запамтио ставке дневне бербе”. (ИА)

**Алтернативна сценарија**

* 1. Уколико систем не може да креира дневну бербу он приказује административном раднику поруку: “Систем не може да креира дневну бербу”. Прекида се извршавање сценарија. (ИА)
  2. Уколико систем не може да запамти ставке дневне бербе он приказује административном раднику поруку “Систем не може да запамти ставке дневне бербе”. (ИА)

### СК 7: Случај коришћења – Креирање обрчуна и статистике

**Назив СК**

Креирање обрчуна и статистике

**Актори СК**

Административни радник

**Учесници СК**

Административни радник и систем (програм)

**Предуслов**: Систем је укључен и андминистративни радник је улогован под својом шифром. Систем приказује форму за уношење датума и креирања обрачуна и статистике.

**Основни сценарио СК**

1. Административни радник уноси датум за креирање. (АПУСО)
2. Административни радник позива систем да креира обрачун и статистику(АПСО)
3. Систем креира обрачун и статистику. (СО)
4. Систем приказује административном раднику обрачун и статистику и поруку “Систем је креирао обрачун и статистику”. (ИА)

**Алтернативна сценарија**

* 1. Уколико систем не може да креира обрачун и статистику приказује поруку “Систем не може да креира обрачун и статистику”. (ИА)

### СК 8: Случај коришћења – Пријављивање административног радника

**Назив СК**

Пријављивање административног радника

**Актори СК**

Административни радник

**Учесници СК**

Административни радник и систем (програм)

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

**Основни сценарио СК**

1. Административни радник уноси податке за аутентификацију административног радника. (АПУСО)
2. Административни радник позива систем да пронађе административног радника са задатим подацима. (АПСО)
3. Систем претражује административне раднике. (СО)
4. Систем приказује поруку да је административни радник успешно пријављен и почетну форму. (ИА)

**Алтернативна сценарија**

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

## 3.3. Анализа

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

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

1. Административни радник позива систем да креира податке о новом добављачу и запамти податке о новом добављачу. (АПСО)
2. Систем приказује административном раднику поруку: “Систем је запамтио добављача”. (ИА)



Дијаграм 2: Унос добављача – основни сценарио

**Алтернативна сценарија**

* 1. Уколико систем не може да креира новог добављача он приказује административном раднику поруку: “Систем не може да креира новог добављача”. Прекида се извршавање сценарија. (ИА)



Дијаграм 3: Унос добављача – алтернативни сценарио

Са наведених секвенцних дијаграма уочавају се 2 системске операције које треба пројектовати:

1. *signal* **UcitajListuMesta**(*List<Mesto>*)
2. *signal* **KreirajNovogDobavljaca**(*Dobavljac*)

### ДС 2: Дијаграм секвенци случаја коришћења – Измена добављача

1. Административни радник позива систем да измени податке о добављачу. (АПСО)
2. Систем приказује адмнинистративном раднику поруку: “Систем је успешно променио податке о добављачу.” (ИА)



Дијаграм 4: Измена добављача – основни сценарио

**Алтернативна сценарија**

* 1. Уколико систем не може да промени податке о добављачу он приказује адмнинистративном раднику поруку: “Систем не може да промени добављача”. (ИА)



Дијаграм 5: Измена добављача – алтернативни сценарио

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

1. *signal* **UcitajListuMesta**(*List<Mesto>*)
2. *signal* **vratiListuDobavljaca**(*List<Dobavljac>*)
3. *signal* **zameniDobavljaca**(*Dobavljac*)

### ДС 3: Дијаграм секвенци случаја коришћења – Брисање добављача

1. Административни радник позива систем да обрише добављача. (АПСО)
2. Систем приказује административном раднику поруку: “Систем је обрисао добављача.” (ИА)



Дијаграм 6: Брисање добављача – основни сценарио

**Алтернативна сценарија**

* 1. Уколико систем не може да обрише добављача он приказује административном раднику поруку: “Систем не може да обрише добављача”. (ИА)



Дијаграм 7: Брисање добављача – алтернативни сценарио

Са наведених секвенцних дијаграма уочавају се 2 системске операције које треба пројектовати:

1. *signal* **vratiListuDobavljaca**(*List<Dobavljac>*)
2. *signal* **obrisiDobavljaca**(*Dobavljac*)

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

1. Административни радник позива систем да запамти податке о новом задужењу. (АПСО)
2. Систем приказује административном раднику поруку:”Систем је запамтио задужење”. (ИА)



Дијаграм 8: Унос задужења – основни сценарио

**Алтернативна сценарија**

* 1. Уколико систем не може да запамти податке о новом задужењу он приказује административном раднику поруку “Систем не може да запамти ново задужење”. (ИА)



Дијаграм 9: Унос задужења – алтернативни сценарио

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

1. *signal* **unosZaduzenja**(*Zaduzenje)*

### ДС 5: Дијаграм секвенци случаја коришћења – Измена задужења

1. Административни радник позива систем да измени податке о изабраном задужењу. (АПСО)
2. Систем приказује административном раднику задужење и поруку “Систем је променио задужење”. (ИА)



Дијаграм 10: Измена задужења – основни сценарио

**Алтернативна сценарија**

* 1. Уколико систем не може да промени податке о задужењу он приказује адмнинистративном раднику поруку: “Систем не може да промени задужење”. (ИА)



Дијаграм 11: Измена задужења – алтернативни сценарио

Са наведених секвенцних дијаграма уочава се 2 системске операције које треба пројектовати:

1. *signal* **vratiZaduzenja**(*List<Zaduzenje>)*
2. *signal* **izmeniZaduzenje**(*Zaduzenje*)

### ДС 6: Дијаграм секвенци случаја коришћења – Унос дневне бербе

1. Административни радник позива систем да креира дневну бербу. (АПСО)
2. Систем приказује административном раднику дневну бербу и поруку “Систем је креирао дневну бербу”. (ИА)
3. Административни радник позива систем да запамти ставке дневне бербе. (АПСО)
4. Систем приказује административном раднику поруку: “Систем је запамтио ставке дневне бербе”. (ИА)



Дијаграм 12: Унос дневне бербе – основни сценарио

**Алтернативна сценарија**

* 1. Уколико систем не може да креира дневну бербу он приказује административном раднику поруку: “Систем не може да креира дневну бербу”. Прекида се извршавање сценарија. (ИА)



Дијаграм 13: Унос дневне бербе – алтернативни сценарио

* 1. Уколико систем не може да запамти ставке дневне бербе он приказује административном раднику поруку “Систем не може да запамти ставке дневне бербе”. (ИА)



Дијаграм 14: Унос дневне бербе – алтернативни сценарио

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

1. *signal* **vratiListuDnevnihBerbi**(*List<DnevnaBerba>)*
2. *signal* **ubaciDnevnuBerbu**(*DnevnaBerba*)
3. *signal* **sacuvajStavkeDnevneBerbe**(List<StavkaDnevneBerbe>)

### ДС 7: Дијаграм секвенци случаја коришћења – Креирање обрачуна и статистике

1. Административни радник позива систем да креира обрачун и статистику(АПСО)
2. Систем приказује административном раднику обрачун и статистику и поруку “Систем је креирао обрачун и статистику”. (ИА)



Дијаграм 15: Креирање обрачуна и статистике – основни сценарио

**Алтернативна сценарија**

* 1. Уколико систем не може да креира обрачун и статистику приказује поруку “Систем не може да креира обрачун и статистику”. (ИА)



Дијаграм 15: Креирање обрачуна и статистике – алтернативни сценарио

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

1. *signal* **KreirajObracunStatistiku**(*datumOd, datumDo)*

### ДС 8: Дијаграм секвенци случаја коришћења – Пријављивање административног радника

1. Административни радник позива систем да пронађе административног радника са задатим подацима. (АПСО)
2. Систем приказује поруку да је административни радник успешно пријављен и почетну форму. (ИА)



Дијаграм 16: Пријављивање административног радника – основни сценарио

**Алтернативна сценарија**

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



Дијаграм 17: Пријављивање административног радника – алтернативни сценарио

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

1. *signal* **NadjiAdministrativnogRadnika**(*AdministrativniRadnik*)

Као резултат анализе сценарија добијено је укупно 11 системских операција које треба пројектовати:

1. *signal* **UcitajListuMesta**(*Mesto)*
2. *signal* **KreirajNovogDobavljaca**(*Dobavljac*)
3. *signal* **VratiListuDobavljaca**(*Dobavljac*)
4. *signal* **IzmeniDobavljaca**(*Dobavljac*)
5. *signal* **ObrisiDobavljaca**(*Dobavljac*)
6. *signal* **UnosZaduzenja**(Zaduzenje)
7. *signal* **IzmeniZaduzenje**(*Zaduzenje*)
8. *signal* **VratiZaduzenja**(*Zaduzenje*)
9. *signal* **UbaciDnevnuBerbu**(*DnevnaBerba*)
10. *signal* **VratiListuDnevnihBerbi**(*DnevnaBerba)*
11. *signal* **SacuvajStavkeDnevneBerbe**(*List<StavkaDnevneBerbe>*)
12. *signal* **KreirajObracunStatistiku**(*datumOd, datumDo)*
13. *signal* **NadjiAdministrativnogRadnika**(*AdministrativniRadnik*)

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

### Уговор УГ1: UcitajListuMesta

Операција: UcitajListuMesta(*List<Mesto>*):signal;

Веза са СК: СК1, СК2

Предуслови: -

Постуслови: -

### Уговор УГ2: KreirajNovogDobavljaca

Операција: KreirajNovogDobavljaca(*Dobavljac*):signal;

Веза са СК: СK1

Предуслови: Вредносна и структурна ограничења морају бити задовољена над објектом Добављац и мора бити доступна листа места. Доступна је и листа добављача.

Постуслови: *Креиран је нови добављач.*

### Уговор УГ3: VratiListuDobavljaca

Операција: VratiListuDobavljaca (*List<Dobavljac>*):signal;

Веза са СК: СК2, СК3

Предуслови: -

Постуслови: -

### Уговор УГ4: IzmeniDobavljaca

Операција: IzmeniDobavljaca (*Dobavljac*):signal;

Веза са СК: СК2

Предуслови: - Вредносна и структурна ограничења морају бити задовољена над објектом Добављач и доступна је листа места. Доступна је и листа добављача.

Постуслови: - Подаци о добављачу су измењени.

### Уговор УГ5: ObrisiDobavljaca

Операција: ObrisiDobavljaca (*Dobavljac*):signal;

Веза са СК: СК3

Предуслови: - Структурно ограничење над објектом Добављац мора бити задовољено. Доступна је листа добављача.

Постуслови: - *Добављач је обрисан.*

### Уговор УГ6: UnosZaduzenja

Операција: KreirajNovoZaduzenje(*Zaduzenje*):signal;

Веза са СК: СК4

Предуслови: - Вредносна и структурна ограничења морају бити задовољена над објектом Задужење. Мора бити селектован добављач.

Постуслови: - *Креирано је ново задужење.*

### Уговор УГ7: IzmeniZaduzenje

Операција: IzmeniZaduzenje (*Zaduzenje*):signal;

Веза са СК: СК5

Предуслови: - Вредносна и структурна ограничења морају бити задовољена над објектом Задузење. Доступна је листа задузења.

Постуслови: - *Подаци о задузењу су запамћени.*

### Уговор УГ8: VratiZaduzenja

Операција: VratiZaduzenja (*Zaduzenje*):signal;

Веза са СК: СК5

Предуслови:-

Постуслови:-

### Уговор УГ9: UbaciDnevnuBerbu

Операција: UbaciDnevnuBerbu (*DnevnaBerba*):signal;

Веза са СК: СК6

Предуслови: Вредносна и структурна ограничења морају бити задовољена над објектом ДневнаБерба. Мора бити селектован добављач.

Постуслови: *Убачена је дневна берба.*

### Уговор УГ10: VratiListuDnevnihBerbi

Операција: **VratiListuDnevnihBerbi** (*DnevnaBerba*):signal;

Веза са СК: СК6

Предуслови: -

Постуслови: -

### Уговор УГ11: SacuvajStavkeDnevneBerbe

Операција: SacuvajStavkeDnevneBerbe (*List<StavkaDnevneBerbe>*):signal;

Веза са СК: СК6

Предуслови: Вредносна и структурна ограничења морају бити задовољена над објектом СтавкаДневнеБербе. Мора бити креирана дневна берба.

Постуслови: *Ставке дневне бербе су запамћене.*

### Уговор УГ12: KreirajObracunStatistiku

Операција: KreirajObracunStatistiku (*datumOd,* datumDo):signal;

Веза са СК: СК7

Предуслови: -

Постуслови: *-*

### Уговор УГ13: NadjiAdministrativnogRadnika

Операција: SacuvajStavkeDnevneBerbe (*List<StavkaDnevneBerbe>*):signal;

Веза са СК: СК8

Предуслови: -

Постуслови: *-*

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



Дијаграм 18: Концептуални модел

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 новембра 2015. [↑](#footnote-ref-2)
3. <https://maven.apache.org/what-is-maven.html> приступано новембра 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)