УНИВЕРЗИТЕТ У БЕОГРАДУ

ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

ЗАВРШНИ РАД

Тема: Развој софтверског система за праћење рада посластичарнице употребом ASP.NET Core оквира

Ментор:

проф. др Саша Лазаревић

Студент:

Александра Марковић 2017/0006

Београд, 2021. године

Развој софтверског система за праћење рада посластичарнице употребом ASP.NET Core оквира

У овом раду је описан начин израде веб апликације за праћење рада посластичарнице употребом ASP.NET Core оквира и њена практична примена. Апликација је у документацији представљена проласком кроз пет фаза упрошћене Ларманове методе, а то су: спецификација корисничких захтева, анализа, пројектовање, имплементација и тестирање. Оне ће детаљно бити представљене у даљем раду.

Апликација одговара захтевима савременог друштва, јер су веб апликације постале његов незаобилазни део, без кога се не може замислити свакодневица презапосленог човека. Корисници јој приступају преко интернет мреже, наручују колаче са места на којем живе и раде, са свог рачунара. Тако штеде време савременог човека, који живи убрзано, у условима опште глобализације. Управо то је једна од предности веб апликација у односу на десктоп апликације, које се инсталирају на личном рачунару. За коришћење је потребна само интернет мрежа и претраживач.

У складу са научно-технолошком револуцијом и развојем потрошачког друштва, развијале су се и веб апликације, а интернет је постао један од главних начина куповине, па је тако онлајн продаја потиснула традиционалне начине куповине, који захтевају одлазак у удаљене продавнице и губљење времена савременог потрошача. Овај софтвер наменски треба да користе посластичарнице које се баве производњом различитих врста колача, како би онлајн продајом побољшале своје пословање.

Садржај

Списак слика

Списак табела

1. Увод

Све већа потреба за веб апликацијама је последица убрзаног живота савременог човека и његових потреба које су условљене недостатком слободног времена и жељом за променама. Док је пандемија вируса *COVID-19* која је потресла свет прошле године у енормној мери ослабила туризам, узрокујући туристичким агенцијама економске губитке до 75% веће него што су били раније, на онлајн куповину је утицала веома повољно. Дакле, значај веб апликација које омогућавају куповину путем Интернета је све већи и представља начин да се побољша ефикасност и профитабилност многих компанија различитих делатности.

Апликација је реализована употребом *ASP.NET Core-a, Entity Framework Core-a 3.1.9, MVC* архитектуре и других технологија које су описане у самом раду. У сврхе развоја софтвера коришћен је програмски језик *C#* и *MS Visual Studio Code 2019* развојно окружење. Коришћени су и објашњени *Repository* и *UnitOfWork* патерни.

Рад је организован у седам поглавља. У другом поглављу ближе је објашњена клијент-сервер архитектура која се имплементира при изради веб апликација и технологије које коришћене у раду. Описан је начин функционисања веб апликације кроз пет корака.

У трећем поглављу ради се о програмским језицима – шта су, како је дошло до њиховог развоја, како је извршена подела и које су основне разлике између објектно-оријентисаног и процедуралног програмирања. Описан је језик C#, а такође је дат и преглед општих карактеристика објектно-оријентисаног програмирања и његових основних концепата.

У четвртом поглављу описане су три *.NET* платформе, при чему је фокус на *.NET Core* платформи која је коришћена. Објашњена је *MVC* архитектура и основна намена сваке од њених компоненти – погледа, контролера и модела. Представљен је *Entity Framework Core,* алат за објектно-релационо мапирање и употреба *LINQ* језика.

У петом поглављу, помоћу пет фаза упрошћене Ларманове методе дат је приказ апликације и њених основних функционалности. Приказани су случајеви коришћења, екранске форме, уговори, дијаграми секвенци, архитектура софтверског система, објашњен је начин комуницирања са базом и још много ставки које ће бити наведене у даљем раду.

Шесто поглавље се односи на закључак, а седмо на литературу коришћену за израду документације.

2. Клијент-сервер архитектура

Технологије које су коришћене се могу посматрати са два аспекта:

1) Клијентски аспект – *Bootstrap, JavaScript, AJAX, JQuery*
2) Серверски аспект – *ASP.NET Core, MVC, Entity Framework Core 3.1.9*

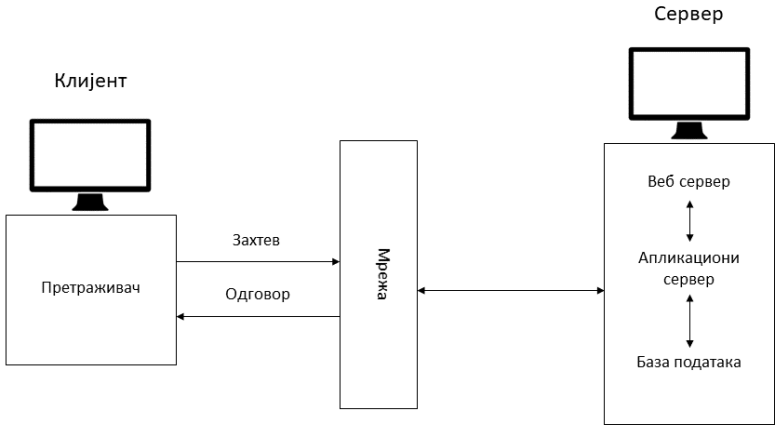
Серверске технологије се могу поделити на [21]:

1) Самосталне програме
2) Серверске скрипте
3) Специјалне језике

Што се тиче клијентских технологија, оне се деле на [21]:

1) Технологије за презентацију садржаја (*HTML, CSS* и др.)
2) Скриптне језике (*JavaScript* и др.)
3) DOM (*Document Object Model*)
4) Додатне објекте (*GIF, JPG, PNG* датотеке и др.)

Веб апликације се, као и већина апликација, заснивају на усаглашеном раду клијентског и серверског уређаја. Помоћу њих, милиони корисника могу приступити одређеном веб сајту са било које локације у свету. Сам појам веб апликација се односи на део кода, програм или комплетно софтверско решење за одређени пословни сценарио у којем се интеракција врши између више корисника, или између корисника и сервера употребом претраживача. [4]
Сервер је задужен за управљање складиштем података, док се на клијентској страни учитавају информације које су потребне корисницима. Захтеви и одговори се приказују коришћењем *HTTP* (*Hyper-Text Transfer Protocol*)[1] протокола.



Слика 1 – Приказ начина функционисања веб апликације

[1] *HTTP Protocol* је мрежни протокол који представља најчешћи метод за пренос информација на вебу. Користи се у апликацијама у којима је имплементирана клијент-сервер архитектура и функционише по принципу захтев/одговор.

Радни процес веб апликације у пет корака је следећи [4]:

1) Клијент шаље захтев кроз веб претраживач или кориснички интерфејс апликације
2) Захтев стиже на веб сервер[2], а он га потом прослеђује серверу апликације
3) Извршавање захтева на серверу апликације (извршавање упита над базом података, преузимање информација из базе података, обрада информација, формулисање резултата)
4) Добијени и форматирани резултати се шаљу назад на веб сервер
5) Клијент прима одговор од веб сервера, а тражене информације се приказују на корисничком екрану

[2] Веб сервер (као што су IIS и Apache) је серверски софтвер или хардвер захтеван од стране веб апликација ради управљања клијентским захтевима, проналажења тражених ресурса и пружања одговора клијентима.

3. Развој програмских језика

Средство комуницирања између програмера (корисника) и рачунара назива се програмски језик. Он омогућава програмеру да преда рачунару извесне податке и инструкције које том рачунару пошаљу захтев да изврши планиране операције.

Програмски језици се према степену зависности језика од рачунара деле на две врсте :

1) језици ниског нивоа
2) језици високог нивоа

Први програмски језик био је машински. Он је морао бити усклађен са електронским компонентама рачунара на ком се користио. Програмери су писали бинарни код за бележење инструкција, адресирање меморије и бележење података, што је био тежак посао, а грешке су биле веома честе. Инструкције се у оквиру њега директно извршавају, јер их рачунар одмах разуме и претвара у електричне сигнале потребне за његов рад (не треба му програм преводилац). Због тога се постиже велика брзина у раду рачунара и повећава ефикасност. Главни недостатак машинског језика је слаба разумљивост од стране човека, зато што се код састоји од нула и јединица и неопходно је добро познавање архитектуре рачунара на ком се обавља посао. [26]

Асемблерски језик представља корак у побољшању структуре програмирања. У њему се користи комбинација бројева и слова која може да замени неке инструкције у машинском коду. Преводилац који га преводи у машински код зове се асемблер. Овим језиком су писани програми који су данас познати као драјвери. [26]

И машински и асемблерски језик припадају групи програмских језика нижег нивоа. Временом се јавља потреба за повећањем продуктивности програмера и ефикасности кода, па се развијају програмски језици вишег нивоа. За разлику од програмских језика ниског нивоа, програмски језици високог нивоа су разумљивији кориснику, читљивији и мање зависни од архитектуре рачунара на ком се користе. Ближи су природном језику и могу сакрити значајна подручја рачунарског система (нпр.управљање меморијом), па је и сам процес развоја програма знатно олакшан. Да би рачунар уопште могао да изврши наредбе из било ког програма написаног у вишем програмском језику, оне се морају превести у машински језик који рачунар препознаје, за шта се користе специјални програми преводиоци: компајлер, интерпретатор или хибридни преводиоци.

Постоји још много подела програмских језика према различитим критеријумима, а неки од њих су: начин решавања проблема, начин расподеле меморије, количина апстракције и друге. Једна од најчешћих је на процедуралне, објектно-оријентисане, функцијске и логичке језике. [11]

Процедурални језици су почели да се користе у шездесетим и седамдесетим годинама прошлог века. Назив потиче од чињенице да се ослањају на процедуре, које садрже низ корака које треба извршити редом, одозго на доле. Они дефинишу како ће нешто бити извршено, за разлику од непроцедуралних који специфицирају само шта ће бити извршено, без залажења у детаље. И даље су у употреби, али се временом јавила потреба за

коришћењем језика који су лакши за управљање и праћење и у којима је садржан виши ниво апстракције (скривања детаља). Тако се јављају објектно-оријентисани језици.

Објектно-оријентисани језици, како сам назив каже, користе објекте и омогућавају разлагање програмерског кода на мање логичке целине. Објекат представља инстанцу класе, која дефинише његове атрибуте и понашање. Једна од већих предности које нуди овај приступ програмирању је то што је програмерима омогућено да креирају модуле који не морају да се мењају када се креира нова врста објекта. У том случају, програмер само креира објекат који наслеђује особине постојећег и управо због тога се програми који су објектно-оријентисани могу лако мењати. Временом је објектно-оријентисано програмирање нашло начин да заузме место у свим новим програмским језицима, којима припада и C#, језик који је коришћен за израду завршног рада. У даљем раду биће више речи о настанку и главним карактеристикама овог језика.

3.1 Програмски језик C#

C# спада у младе објектно-оријентисане програмске језике и један је од најзаступљенијих у свету информационих технологија. Настао је као унапређена верзија програмских језика Java, C и C++. Као саставни део Microsoft развојног окружења, његова прва верзија је приказана 2002. године, а касније је унапређиван додавањем бројних функција које су олакшале програмерима да пишу кодове без велике употребе изворног кода. Са аспекта зависности програмског језика од рачунара, C# спада у језике вишег нивоа. Као језик опште примене, намењен је развоју софтвера у оквиру .NET окружења. [8]

Припада групи *case sensitive* и *type safe* језика, што значи да прави разлику између малих и великих слова и захтева експлицитно кастовање уколико неку променљиву или податак одређеног типа треба заменити другим типом. То значи да је потребно придржавати се строгих правила уколико се извршава конверзија типова. [7] Његова синтакса није сложена и има око осамдесет резервисаних речи. Поседује особине неопходне за успешно креирање сваког програма: једноставност, брзину, сигурност, објектну оријентисаност и усмереност ка интернету. Готово да нема апликације која се не може направити коришћењем овог језика, а најчешће развијане јесу напредне десктоп и веб апликације.

3.1.1 Историјат

Главни дизајнер C#-а, Андерс Хејлсберг је 1999. основао тим стручњака са идејом да створи нови објектно-оријентисани програмски језик C који се у почетку звао Кул (енг. *„C-like Object Oriented Language"*). Његова замисао је била да отклони недостатке постојећих програмских језика. Иако је *Microsoft* желео да задржи првобитни назив, то није било могуће због заштитног знака. Зато је језик преименован у *C#*, а библиотеке ASP.NET пренесене у њега у јулу 2000. године на Конференцији професионалних програмера (енг. *Professional Developers Conference*). Добио је име „шарп“ (енг. *C sharp*), повишено це или цис, што у музици означава ноту повишену за пола тона више. Фајлови писани у овом језику имају екстензију *.cs*. [7]

Многи програмери су оспоравали значај C#-а као новог програмског језика. Међу њима су креатор програмског језика *Java*, Џејмс Гослинг, који је сматрао да је C# имитација Јаве. Суоснивач *Sun Mycrosistems-a*, Били Џој, делио је његово мишљење. Од 2005. године, од издања C# 2.0 језици C# и Јава почињу да се развијају у различитим правцима, а сличности међу њима постају све мање. [7] Од 2002. године он постаје први програмски језик прилагођен за рад у .NET Framework окружењу и бива непрестано унапређиван кроз различите верзије.

3.1.2 Изузеци

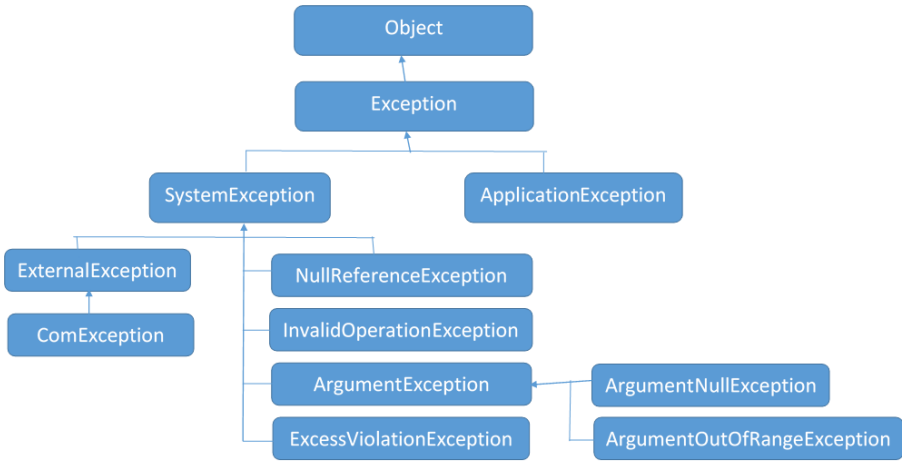
Изузеци представљају примарни механизам за откривање грешака у C#-у и ради несметаног одвијања програма неопходна је њихова обрада. Они се односе на непредвиђене догађаје који спречавају програм да настави рад. Могу у основи да буду хардверска грешка или последица програмске грешке, коју претходно није открио преводилац. [7]

Обрадом изузетака се одговора на грешке. CLR (енг. *Common Language Runtime*), извршно окружење, јавља да се грешка догодила. Она се у програму може обрадити, отклонити или прекинути програм на коректан начин. Ако програм не обради изузетак, извршно окружење ће га прекинути. [7]

Изузеци се у C#-у представљају класом *Exception* и они су објекти за себе. За сваки изузетак, класа *System.Exception* представља базну класу и садржи информације о узроку проблема, као што је на пример порука (енг. *Message*) која садржи текстуални опис проблема. Поред поруке, ту су и *StackTrace* и *InnerException*, који представљају снимак стека у моменту дешавања грешке и изузетак који је проузроковао тренутни изузетак (ако постоји такав), респективно. Коришћење *try-catch-finally* блока омогућава успешну обраду изузетака, при чему је *finally* блок опциони – не мора да постоји. Уколико постоји, он се увек пише на крају и увек се извршава, без обзира на то да ли је изузетак настао или не. У програму може да се напише више *catch* блокова, при чему сваки треба да хвата одређени тип изузетка.

```
try{
// ispitni blok
}
catch(<Prvi_tip_izuzetaka>)
{
// iskazi za obradu prvog tipa izuzetaka
}
catch(<Drugi_tip_izuzetaka>)
{
// iskazi za obradu drugog tipa izuzetaka
}
catch(Exception e)
{
// iskazi za obradu opsteg tipa izuzetaka
}
finally{
// iskazi koji se izvrsavaju
}
```

На следећој слици приказан је хијерархијски поредак изузетака у C#-у:



Слика 2 Хијерархијски приказ изузетака
https://www.csharpnaija.com.ng/2020/03/exceptions-and-exception-handling-in-c.html

При руковању изузецима, веома је важно познавати ову хијерархију јер се при обради, тј. хватању одређеног изузетка, обрађују и сви изузеци који су на нижем нивоу у поменутој хијерархији.

У наредном одељку биће објашњене особине и основни концепти објектно-оријентисаног програмирања кроз примере написане у C#-у.

3.2 Карактеристике објектно-оријентисаног програмирања

Објектно оријентисано програмирање је почело је да се развија шездесетих година двадесетог века, јер је тада постало јасно да рачунари имају непроцењиву примену у свим животним областима. Да би се максимално искористио њихов потенцијал требало је пронаћи начин да се сложени подаци у рачунарским програмима представе на прегледнији и једноставнији начин.

У следећој табели приказане су главне разлике између ранијег процедуралног програмирања и напредног објектно-оријентисаног програмирања [22]:

ПРОЦЕДУРАЛНО ПРОГРАМИРАЊЕ	ОБЈЕКТНО-ОРИЈЕНТИСАНО ПРОГРАМИРАЊЕ
Користи приступ одозго према доле у програмирању	Користи приступ одоздо према горе у осмишљавању програма

Програм је подељен на мале целине на основу функција	Програм је подељен на објекте у зависности од проблема
Свака функција садржи различите податке	Сваки објекат контролише своје податке
Прати системски приступ решавања проблема	Фокусира се на сигурност података без обзира на алгоритам
Функције су важније од података у програму	Главни приоритет су подаци, а не функције у програму
Нема једноставног начина за скривање података	Скривање података је могуће у ООП-у
Не постоји концепт наслеђивања	Наслеђивање је дозвољено

Табела 1 Разлике између процедуралног и објектно-оријентисаног програмирања

Класа је нека врста шаблона која садржи атрибуте и методе. Атрибути представљају карактеристике објекта, а методе су операције које се извршавају над објектом. Објекат је дефинисан класом. Он се без класе не може креирати, а сама класа може имати више објеката. Објекат је, дакле, логичка или апстрактна структура, скуп информација које чине јединствени ентитет, у који програмер уграђује код. Помоћу њега програмер може да прегледа делове неког решења појединачно и уклапа их у целокупно решење. Он је целина у програмском коду. Настаје инстанцирањем класа. [22]

Нека је, на пример, класа робот, тада би атрибути били његова величина, боја и друге особине. Методе су операције које објекат може да уради, на пример да се окрене, стане, или изврши било коју другу наредбу.

Основни концепти (три стуба) објектно-оријентисаног програмирања су [4]:

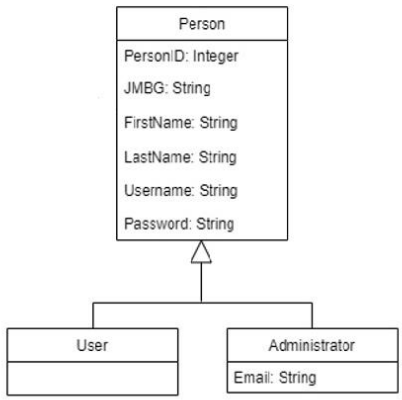
1. Наслеђивање
2. Енкапсулација
3. Полиморфизам

3.2.1 Наслеђивање

Наслеђивање је један од најважнијих концепата у објектно оријентисаном програмирању. Оно омогућава да се дефинишу односи између класа. Тако објекти истог типа могу да деле сличне функције и имају заједничке атрибуте. Основна класа, која се назива и „родитељска“ је она из које се наслеђује, а изведена класа је „потомак“ који наслеђује из основне. Управо она се дефинише уз помоћ наслеђивања. У C# се наслеђивање дефинише са две тачке (:). [4] Постоје различити типови наслеђивања [4]:

1. Једноструко наслеђивање - описује једну класу изведену из друге класе. Оно представља најчешће примењивани метод наслеђивања.

Помоћу овог наслеђивања дефинишу се атрибути који припадају обема класама, а специфичности се дефинишу у специфичној класи. На слици 3 дат је пример једноструког наслеђивања кроз *UML* дијаграм, а испод се налази део кода којим је у апликацији оно дефинисано.



Слика 3 Илустрација једноструког наслеђивања

```
public class Person
   {
        public int PersonID { get; set; }
        public string JMBG { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Username { get; set; }
        public string Password { get; set; }
        public City City { get; set; }
        public int CityID { get; set; }
    }
public class User: Person
    {
    }
public class Administrator: Person
    {
        public string Email { get; set;}
    }
```

У наведеном примеру, базна класа је класа Особа (енг. *Person*), а класе Администратор (енг. *Administrator*), и Корисник (енг. *User*) су изведене из ње и наслеђују све њене атрибуте приказане на сликама, с тим што класа Администратор има сопствени атрибут Мејл (енг. *Email*).

2. Вишеструко наслеђивање – користи се када изведена класа наслеђује више основних класа. C# га не подржава, али се у њему се овај тип наслеђивања може постићи помоћу интерфејса.
3. Хијерархијско наслеђивање – када више класа наслеђује другу класу

4. Вишеслојно наслеђивање – када се класа изводи из класе која је већ изведена
5. Хибридно наслеђивање – комбинација више наслеђивања
6. Имплицитно наслеђивање – сви типови у *.NET Core-y* су имплицитно потомци класе *System Object* и њених изведених класа.

3.2.2 Енкапсулација

Енкапсулација или учаурење омогућава да атрибути и методе класе могу да буду видљиви или невидљиви и ван објекта. Помоћу ње програмер спречава некоректно руковање класом. Једна класа може садржати атрибуте и функције који имају различите нивое приступа. Користи се у сигурносне сврхе. [4]

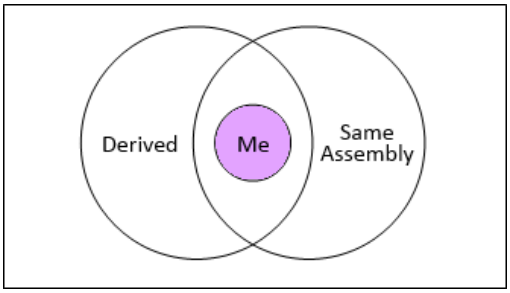
Једноставно речено, енкапсулација у објектно-оријентисаном програмирању представља принцип који налаже да се приступ подацима мора регулисати тако да „унутрашњи механизми“ класа треба да буду скривени од крајњих корисника, али – тако да се корисницима класе омогући неометано коришћење свих неопходних функционалности. [17]

Овим концептом се свим ставкама у оквиру класе одређује опсег видљивости (приватни, јавни, заштићени или интерни). Како би се омогућио приступ приватним пољима, користе се get и set методе у оквиру такозваних својстава (енг. *property*).

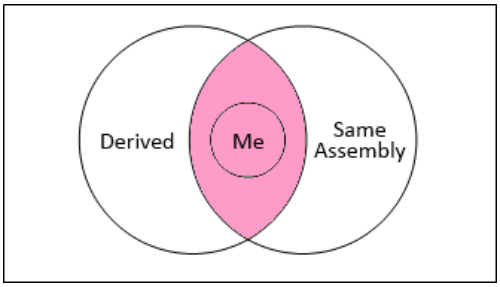
За сваку класу, атрибут, метод или набрајање (енг. *enum*) C# подржава следеће нивое заштите [4]:

1. *Public* - приступ доступан ван ставке, било где у оквиру програма
2. *Private* - приступ ставци има само објекат
3. *Protected* - приступ атрибуту или методу имају само објекат и објекти изведених класа
4. *Internal* - приступ ставци имају само објекти унутар истог склопа
5. *Protected Internal* – метод или атрибут су доступни класама које се налазе у истом склопу или класама које су изведене из дате класе
6. *Private Protected* – приступ доступан из изведене класе у истом склопу

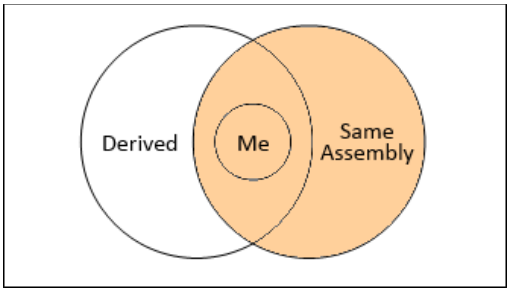
На сликама које следе су илустровани наведени нивои приступа.



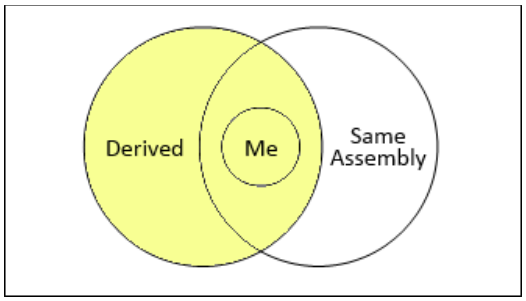
Слика 4 Енкапсулација – Private
https://stackoverflow.com/questions/614818/in-c-what-is-the-difference-between-public-private-protected-and-having-no



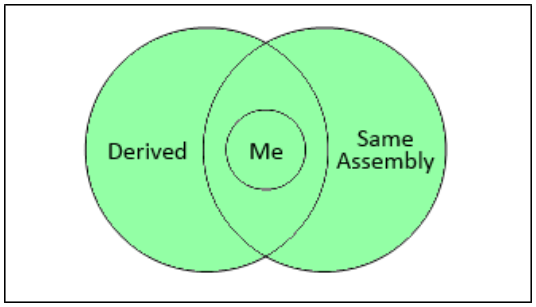
Слика 5 Енкапсулација - Private protected
https://stackoverflow.com/questions/614818/in-c-what-is-the-difference-between-public-private-protected-and-having-no



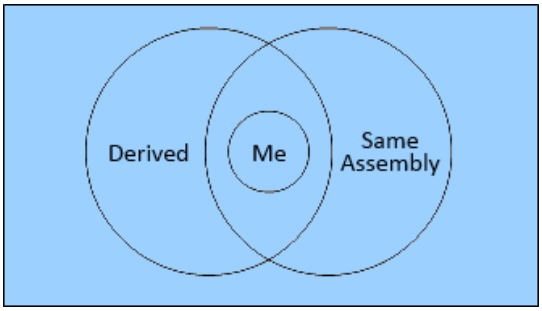
Слика 6 Енкапсулација – Internal
https://stackoverflow.com/questions/614818/in-c-what-is-the-difference-between-public-private-protected-and-having-no



Слика 7 Енкапсулација – Protected
https://stackoverflow.com/questions/614818/in-c-what-is-the-difference-between-public-private-protected-and-having-no



Слика 8 Енкапсулација – Protected Internal
https://stackoverflow.com/questions/614818/in-c-what-is-the-difference-between-public-private-protected-and-having-no



Слика 9 Енкапсулација – Public
https://stackoverflow.com/questions/614818/in-c-what-is-the-difference-between-public-private-protected-and-having-no

3.2.3 Полиморфизам

У објектно-оријентисаном програмирању полиморфизам представља концепт који омогућава да се редефинише начин функционисања неке методе. Постиже се коришћењем виртуелних метода, које се дефинишу и имплементирају у основним класама, а које је могуће користити на различите начине у изведеним класама, након што се у њима редефинишу. Ово се постиже техникама преклапања (енг. *overload*) и преписивања (енг. *override*), као и преклапањем оператора. [23]

Класа која је изведена наслеђује понашање базне класе, међутим, у програмирању постоје проблеми који захтевају да одређена метода у изведеној класи има исти потпис, а другачију имплементацију него у базној. Зато је настао концепт виртуелних метода. Базна класа треба да има дефинисану виртуелну методу, чије понашање у изведеној класи желимо да променимо и прилагодимо потребама.

Постоји неколико правила када је у питању коришћење виртуелних метода [18]:

1) За њихову декларацију користи се кључна реч *virtual*
2) Виртуелне методе се морају имплементирати када се дефинишу (тело методе не може бити празно)
3) Приватне и статичке методе не могу да буду виртуелне
4) Реимплементиране методе такође морају да имају своју имплементацију
5) Виртуелна и реимплементирана метода морају да имају исти потпис, што подразумева исти назив, модификатор приступа, тип резултата и исте параметре

У наставку је дат пример преписивања метода [18]:

```
public class Racun
{
    public virtual void Prikazi()
    {
        Console.WriteLine(“Racun: {0}”, brojRacuna);
    }
}
public class TekuciRacun : Racun
{
    public override void Prikazi()
    {
        Console.WriteLine(“Tekuci racun: {0}”, brojRacuna);
    }
}
Racun racun = new Racun();
racun.Prikazi(); // Racun; 11456
racun = new TekuciRacun();
racun.Prikazi(); // Tekuci racun: 11456
```

Два главна типа полиформизма су [4]:

1. статичко (рано) повезивање – дешава се у току компајлирања апликације, када је апликација преведена
2. динамичко (касно) повезивање – дешава се у току њеног извршавања, када је апликација покренута

Статички полиморфизам се састоји од преклапања метода истог назива у некој класи, а са различитим параметрима. Тако се поједностављује код.

Динамички полиморфизам се у C# јавља у три уобичајене форме:
1. интерфејс
2. наслеђивање
3. генеричка класа

Важно је знати да интерфејс сам по себи не може да буде инстанциран, али описује шта нека инстанца мора да имплементира. Помоћу генеричких класа се описује општи случај без употребе конкретних типова.

Следећи код представља пример употребе интерфејса и генеричких класа:

```
public interface IRepository<T>
    {
        void Add(T s);
        List<T> GetAll();
        T FindById(T id);
        void Delete(T s);
        void Update(T s);
        List<T> Search(Expression<Func<T, bool>> pred);
    }
public interface IRepositoryCity: IRepository<City>
    {
    }
public class RepositoryCity : IRepositoryCity
    {
        private Context context;
        public RepositoryCity(Context context)
        {
            this.context = context;
        }
        public void Add(City s)
        {
            context.Cities.Add(s);
        }
        public void Delete(City s)
        {
            context.Cities.Remove(s);
        }
        public City FindById(City city)
        {
            return context.Cities.SingleOrDefault(k => k.CityID == city.CityID);
        }
        public List<City> GetAll()
        {
            return context.Cities.ToList();
        }
        public List<City> Search(Expression<Func<City, bool>> pred)
        {
            return context.Cities.Where(pred).ToList();
        }
        public void Update(City s)
        {
            City city = context.Cities.SingleOrDefault(c => c.CityID == s.CityID);
            context.Entry(city).CurrentValues.SetValues(s);
        }
    }
```

4. .NET платформе

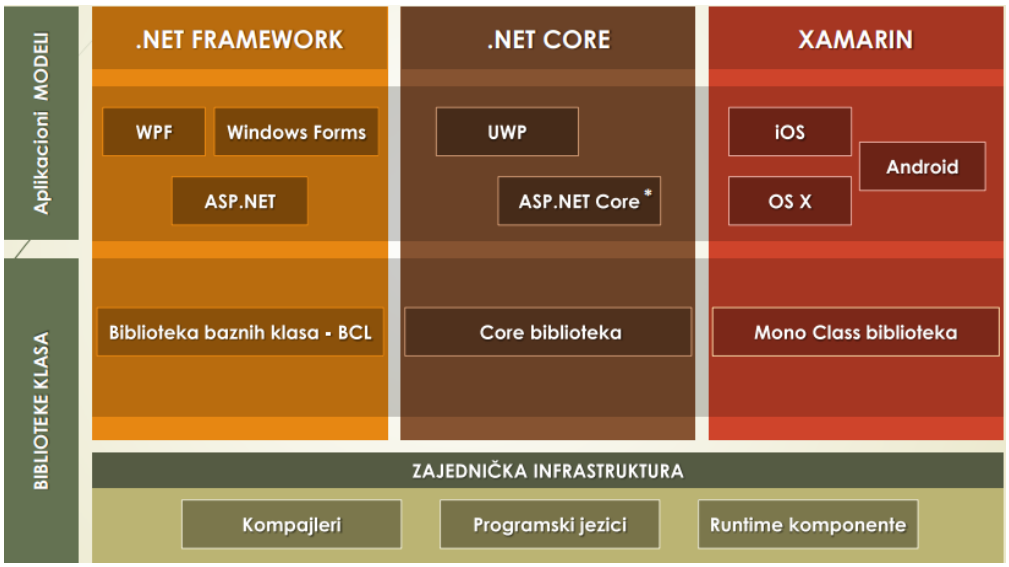
.NET је данас име најмодерније развојне платформе отвореног кода за изградњу свих типова апликација коју *Microsoft* везује за софрверске технологије будућности. Она настаје као најважнија иницијатива у историји Microsofta и визија рачунарства у будућности која долази. Бесплатна је и доступна на сваком месту и у свако време. Представља скуп сервиса и производа који омогућавају ефикасан рад у развоју модерних апликација. У току стварања *.NET* платформе *Microsoft* је предвидео да ће основа нових технологија будућности бити веб сервиси, као основни градивни блокови дистрибуираних апликација ка интернету.
.NET платформа са собом носи много предности [24]:

1) дистрибуиране софтверске апликације су више објектно оријентисане
2) повећана је ефикасност у развоју софтвера
3) софтверске целине су преко интерфејса расположиве различитим уређајима
4) рад апликације се лакше контролише
5) на претходним искуствима развијена је библиотека класа, доследно су дефинисани основни типови
6) на било ком језику да се напише код , ако има подршку *.NET-a* , биће преведен на код разумљив међујезику *IL* (енг. *Intermediate Language*)
7) На *ASP.NET* технологији је унапређен приступ динамичким веб страницама
8) развијени су алати за поједностављено писање *XML* сервиса
9) уведен је концепт склопова (енг. *assembly*) за заједничко коришћење кода
10) преко *ADO.NET* класа приступ подацима је постао доста ефикаснији

.NET платформа је настала на стандардима технологија *XML* и *SOAP* и омогућава да више *WEB* сервиса који потичу из различитих извора и имају различите начине имплементације раде заједно. Језици у којима се може написати апликација развијена уз помоћ *.NET-a* су *C#, F#,* или *Visual Basic*. Код из програма ће бити покренут на било ком компатибилном оперативном систему. [15]

Microsoft компанија нуди три различите платформе намењене за развој апликација:

1) *.NET Framework*
2) *.NET Core*
3) *Xamarin*



Слика 10 Приказ .NET платформе
https://vtsnis.edu.rs/wp-content/plugins/vts-predmeti/uploads/1_UVOD_u_DOT_NET_2020_21.pdf

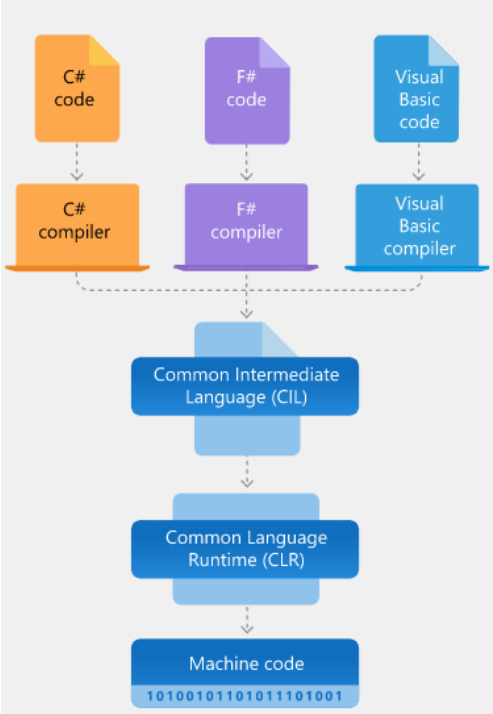
4.1 .NET Framework

Као основа за развој *Windows* десктоп и веб апликација, крајем прошлог века створен је *.NET Framework* под називом „*Windows* сервиси следеће генерације“ (енг. *Next Generation Windows Services*). Он обезбеђује скуп алата који су потребни за изградњу апликација и *XML* веб сервиса. Базира се на објектно-оријентисаном програмском моделу и укључује велики број готових библиотека класа[3] које, између осталог, служе за управљање подацима и рад са базама података. [15]

Две главне компоненте *.NET Framework-a* су управо библиотеке класа (енг. *Class Libraries*) и извршно окружење *CLR* (енг. *Common Language Runtime*), које је заједничко за све *.NET* језике. Компајлирање кода се извршава путем *CIL-a* (енг. *Common Intermediate Language*). Када се апликација покрене, извршно окружење користи поменути компајлер како би написани код био претворен у машински и могао да се извршава на локалном рачунару. [3]

На наредним сликама приказан је процес извршавања апликације и архитектура *.NET Framework-a*.

[3] Библиотека класа представља предефинисан скуп класа које омогућавају извршавање великом броју апликација. [15]



Слика 11 . Компатибилност .NET језика
https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework

Слика 12 .NET Framework - Базна архитектура
https://vtsnis.edu.rs/wp-content/plugins/vts-predmeti/uploads/1_UVOD_u_DOT_NET_2020_21.pdf

.NET Framework управља свим аспектима у раду програма (покреће апликацију и управља њеним извршавањем, додељује јој дозволу за рад и управља меморијом) и стога представља и развојно и извршно и управљиво програмско окружење. [15] Додатни пакети у оквиру ове платформе омогућавају ефикаснији развој десктоп и веб апликација. То су *Windows Presentation Foundation (WPF), Windows Forms* и *ASP.NET*.

У сврху израде апликације за овај дипломски рад коришћен је *ASP.NET MVC* шаблон и *.NET Core* платформа, чији ће кратак преглед бити дат у наредном одељку, а потом ће бити објашњене главне разлике између *.NET Framework* и *.NET Core* платформе.

4.2 .NET Core

.NET Core платформа настаје као нови производ *Microsoft-a* 2016.године, прерадом *.NET-a*, задржавањем основних делова, а изостављањем онога што је уско повезано са *Windows* оперативним системом. Она је међуплатформски наследник *.NET Framework-a*.

Као и *.NET Framework* и *Xamarin*, ова платформа користи *CLR* извршно окружење и процес компајлирања функционише на исти начин као што је то описано за *.NET Framework*. При изградњи апликација омогућава флексибилан рад кроз отворени код. Једна од већих предности је та што укључује међуплатформску имплементацију *CLR-a* познату под именом *CoreCLR* и савремену библиотеку класа *CoreFX*. Ову платформу треба преферирати ако је потребно развити скалабилан софтверски систем високих перформанси, подржан на више платформи. [15]

Слика 13 Упоредни приказ .NET платформи
https://devblogs.microsoft.com/cesardelatorre/net-core-1-0-net-framework-xamarin-the-whatand-when-to-use-it/

Табела 2 илуструје главне разлике између *.NET Framework* и *.NET Core* платформе [2]:

Критеријум	**.NET Framework**	**.NET Core**
Платформа или оквир	Пружа све основне захтеве за развој апликација – кориснички интерфејс, конекцију са базом, сервисе, *API* итд.	Платформа на којој се налазе оквири као што је *ASP.NET Core* и *Universal Windows Platform* који утичу на *.NET Core* и проширују га

Отвореност кода	Није отвореног кода	Јесте отвореног кода
Оперативни систем	Извршава се само на *Windows-y*	Подржава *Windows, MacOS* и *Linux* платформе. Апликација се може узрадити на једној платформи, а покренути на другој.
Врсте апликација	Десктоп и веб апликације *Windows Forms* і *WPF* апликације су веома добро подржане.	Фокус на веб апликацијама, *Windows Mobile* и *Windows Store* апликацијама.
Пакети	Све библиотеке су упаковане заједно и тако се испоручују. Чак и ако програмер не захтева експлитицно одређене библиотеке, оне долазе као део пакета.	Испоручује се као скуп *NuGet* пакета. Програмер има могућност да бира које библиотеке жели и да их инсталира по потреби.
Подршка за WCF/REST услуге	Подржава *WCF* и *REST* услуге.	Нема подршку за *WCF* услуге, мора се креирати *REST API*.
CLI алати	Не подржава *Command Line Interface*.	Подржава *Command Line Interface*, као и рад са корисничким интерфејсом.

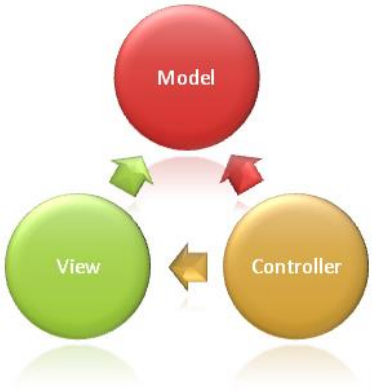
Табела 2 Разлике између .NET Core и .NET Framework платформе

4.2.1 ASP.NET Core

ASP.NET (енгл. *Active Server Pages .NET*) је *Microsoft-ова* технологија која омогућава креирање динамичких веб апликација, веб страница и веб сервиса. Представља главни, извршни алат у *.NET Framework-u*. Брз је, бесплатан и веома популаран. Омогућио је брзу еволуцију програмских језика, перформансе, подршку и стабилност за одржавање апликација. Ради на *Windows, Linux* и *MAC* оперативном систему. [5]

ASP. NET базира своју платрформу на објектно оријентисаном програмском моделу. Сав код се извршава на серверу и враћа се кориснику у облику *HTML* језика. Наследник је класичног *ASP-a*, део *.NET Framework-a* који упрошћава задатке развоја, уклањање грешака и имплементацију веб апликација и потпуно је интегрисан са њим.

Једна од архитектура које *Microsoft* нуди за креирање комплекснијих веб апликација је *ASP.NET MVC. MVC* (енг. *Model-View-Controller*) је патерн који компоненте веб апликације дели у три велике групе: моделе, погледе и контролере.



Слика 14 MVC архитектура
https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-5.0

Ова архитектура је посебно значајна због поделе одговорности када су у питању комплексне апликације. Корисник шаље захтев до контролера, који обрађује тај захтев и враћа одређени поглед (енг. *View*) кориснику. Сваки поглед се заснива на одређеном моделу чије податке приказује.

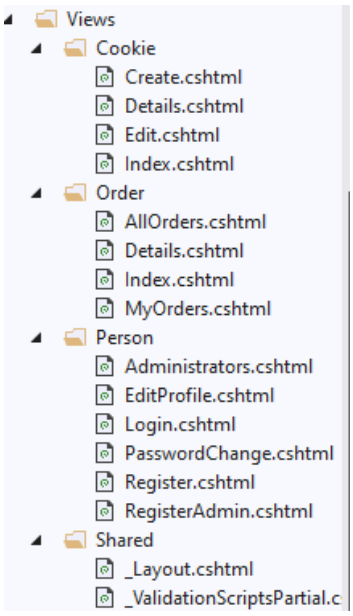
Модел треба да обухвата пословну логику и садржи објекте и њихове атрибуте које је потребно приказивати корисницима. [14] Модел може бити сама доменска класа са својим атрибутима, али је много боља пракса да се формирају комбиновани модели дизајнирани тако да садрже све потребне податке, без обзира на које доменске класе се они односе.

Поглед се односи на кориснички интерфејс и у овом патерну користи се такозвани *Razor View*. За приказивање података на њему користи се HTML језик. На самом корисничком интерфејсу не треба да се налази никаква сложенија логика – она треба да је имплементирана у контролеру који као резултат враћа жељени поглед. [14]

Контролер је задужен за обраду корисничких захтева, представља логичку компоненту корисничког интерфејса. [14] О контролерима ће бити речи у посебном одељку.

4.2.2 Razor View и модели

RazorView је врста погледа која је коришћена за приказ корисничког интерфејса. У овим погледима користи се *HTML* са *Razor* синтаксом и њихова комбинација резултује веб страницом која се шаље клијенту. Када се користи *ASP.NET MVC* патерн и *C#* програмски језик, *RazorView* има екстензију *.cshtml*. Сви погледи су организовани у фолдере и груписани су у зависности од тога за који контролер се везују.



Слика 15 Razor Views

Основни разлог коришћења *RazorView-a* је тај што омогућава да се крене од статичког *HTML* језика, који се може учинити динамичким додавањем серверског кода. На пример, када се у *View* постави форма за регистрацију корисника која треба да се прикаже када он одабере ту опцију, то је статички део кода и може се реализовати једноставном употребом *HTML-a*. Када се тој форми дода падајућа листа из које корисник треба да одабере град, додаје јој се динамика, јер је листа градова променљива и повлачи се из базе.

Сваки поглед почиње знаком @ након ког се специфицира модел који се користи. Модели се налазе у фолдеру *Models* и садрже објекте и/или њихове атрибуте који се приказују на корисничкој страни. За сваки код који се пише на корисничкој страни, а у који је имплементирана нека логика, мора се користити знак @. На пример, ако се у *RazorView-y* дефинише променљива index, која се потом приказује па увећа без позивања сервера, код би изгледао овако:

```
@{int index = 0;} //иницијализација
@index //приказ
@{index++;} //повећавање за 1
```

Код који следи у наставку приказује модел који се користи за регистрацију корисника, начин на који се он користи у погледу и акцију у контролеру којом се моделу додељује листа градова повучена из базе података и враћа одговарајући *Register View*.

```
public class RegisterViewModel
    {
        public int PersonID { get; set; }
        public string JMBG { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
```

```
        public string Username { get; set; }
        public string Password { get; set; }
        public City City { get; set; }
        public int CityID { get; set; }
        public string Email { get; set; }
        public List<SelectListItem> Cities { get; set; }
        public String NewPassword { get; set; }
}

@model Cookies.WebApp.Models.RegisterViewModel
<head>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.7.0/css/font-awesome.min.css">
</head>
<div class="containerLog">
    <i class="fa fa-user-plus"></i>
    <h1 class="h1">Registracija</h1>
    <hr />
    <div class="form">
        <form asp-action="Register">
            <div asp-validation-summary="ModelOnly" class="text-danger" hidden></div>
            <input asp-for="JMBG" class="input" placeholder="JMBG" />
            <span asp-validation-for="JMBG" class="text-danger"></span>
            <input asp-for="FirstName" class="input" placeholder="Ime" />
            <span asp-validation-for="FirstName" class="text-danger"></span>
            <input asp-for="LastName" class="input" placeholder="Prezime" />
            <span asp-validation-for="LastName" class="text-danger"></span>
            <input asp-for="Username" class="input" placeholder="Korisničko ime" />
            <span asp-validation-for="Username" class="text-danger"></span>
          <input asp-for="Password" class="input" placeholder="Lozinka" type="password"
/>
            <span asp-validation-for="Password" class="text-danger"></span>
            <label class="clabel"> Grad: </label>
            @Html.DropDownList("CityID", Model.Cities, "Odaberite grad...", new { @class
= "input" })
            <input type="submit" value="Registrujte se" class="button" />
        </form>
    </div>
</div>
public ActionResult Register()
           {
              RegisterViewModel model = new RegisterViewModel();
              model.Cities = _personService.GetAllCities().Select(c => new
              SelectListItem {Text = c.CityName, Value = c.CityID.ToString()}).ToList();
              return View("Register", model);
           }
```

Предности коришћења *RazorView-a* су:

1. Апликација је лакша за управљање због боље организације
2. Могуће је мењати кориснички интерфејс независно од пословне логике и слоја приступа подацима
3. Лакше је тестирати одређене делове корисничког интерфејса уколико су међусобно независни

4.2.3 Контролери

Контролери су компоненте које управљају захтевима корисника, раде са моделима и врше избор погледа ком ће прослеђивати податке. Представљају се класама чије су јавне методе одговорне за обраду *HTTP* захтева. Те методе се називају акцијама или акционим методама и не могу бити статичке нити се могу преписивати. Позивају се када се у претраживач унесе одређени *URL*. На пример, ако је унети *URL* http://localhost/Order/Details/3, позваће се метода *Details* написана у *OrderController-y* и вратиће податке о наруџбеници чији је идентификатор три. По конвенцији, без обзира на то који је назив контролера, он у себи треба да се завршава речју *Controller*. Ове класе наслеђују базну класу *Microsoft.AspNetCore.Mvc.Controller*, што се види у коду апликације:

```
public class OrderController : Controller
    {
        private readonly IOrderService _orderService;
        public OrderController(IOrderService orderService)
        {
            _orderService = orderService;

        }
              ....
    }
```

Акције могу да имају различите повратне типове чија је базна класа *ActionResult*. [6] То су:

1) **ViewResult** - представља HTML синтаксу, резултат акције је *View*
2) **EmptyResult** – означава да нема резултата
3) **RedirectResult** – преусмерава на нови *URL*
4) **JsonResult** - представља *JavaScript* нотацију која се може користити у *AJAX* апликацијама
5) **JavaScriptResult** – означава *JavaScript* скрипту
6) **ContentResult** – текстуални резултат
7) **FileContentResult** – враћа бинарни фајл који се може преузети
8) **FilePathResult** – враћа бинарни фајл (са путањом)
9) **FileStreamResult** – враћа фајл који се може преузети (који има ток)

У пракси се углавном не враћају ови повратни типови директно, већ се као резултат извршавања методе позива метода базне класе *Controller* која има неку од наведених повратних вредности. Наведене су те методе и типови које враћају [6]:

1) **View** - *ViewResult*
2) **Redirect** - *RedirectResult*
3) **RedirectToAction** - *RedirectToRouteResult*
4) **RedirectToRoute** - *RedirectToRouteResult*
5) **Json** - *JsonResult*
6) **JavaScriptResult** - *JavaScriptResult*.
7) **Content** - *ContentResult*

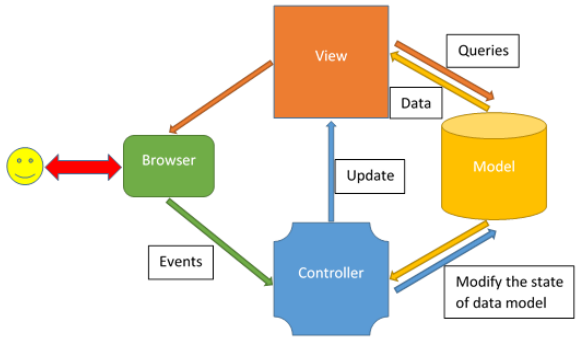
8) **File** - *FileContentResult*, *FilePathResult* или *FileStreamResult* зависно од прослеђених параметара

Генерално, постоје две врсте метода унутар сваког контролера: *GET* и *POST* методе. *GET* методе врше припрему података које треба проследити одређеном погледу. Уколико треба приказати податке о одређеној наруџбеници, позива се *GET* метода која узима те податке из базе, смешта их у модел који се користи и тај модел прослеђује одговарајућем погледу. Дакле, оне су задужене за прикупљање постојећих података. *POST* методе се користе за прикупљање и обраду информација са страница које шаљу или мењају одређене податке, те су оне задужене за измене и додавање података.

Још једна значајна карактеристика контролера је могућност рутирања. Рута описује начин на који се *URL* пресликава у акције контролера. У методи *Configure*, класа *Startup.cs*, позива се *UseEndpoints* метода која се користи за дефинисање рута.

```
app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllerRoute(
                    name: "default",
                    pattern: "{controller=Person}/{action=Index}/{id?}");
            });
```

Ово значи да ће се при покретању апликације позвати метода Index која је дефинисана у *PersonController-y* и враћа форму за пријаву корисника/администратора, дакле враћа *View*. Следећа слика приказује *MVC* архитектуру и међусобне везе између контролера, погледа и модела:



Слика 16 Повезаност модела, погледа и контролера

4.3 Xamarin

Xamarin је трећа врста *.NET* платформе која је 2016. године постала доступна за куповину, а основана је 2011. године у Калифорнији, као производ истоимене компаније. Њени запослени су се бавили међуплатформским имплементацијама CLI-а и као резултат тога настао је *Mono Project*, који је укључивао *MonoTouch* и *Mono for Android*, који су обезбеђивали развој *iOS* и *Android* апликација коришћењем *C#-a*, респективно. Саставни је

део *Visual Studio* окружења и користи се за развој мобилних апликација и развој *cloud* сервиса који су подршка тим апликацијама. *Xamarin* апликације раде унутар међуплатформског извршног *Mono* окружења, које је базирано на отвореном коду за *iOS, OS X* и *Android* уређаје. [20]

4.4 Entity Framework Core

Entity Framework Core представља један од *ORM* (енг. *Object-relational mapping*) алата и како време одмиче, он постаје главна *.NET* технологија за рад са базама података. Развијен је у *Microsoft-y* 2008. године у оквиру *.NET Framework 3.5* платформе. Предности коришћења *Entity Framework Core-a* се огледају у могућности рада са релационим базама података кроз коришћење *.NET* објеката од стране програмера, као и у елиминисању потребе за писањем кода који служи за приступ бази и рад са њеним подацима. [9] Развојем ове технологије, апликације је могуће развијати на много вишем нивоу апстракције када је у питању рад са подацима, а манипулацију подацима је могуће остварити на једноставнији начин, уз много мање кода него што је био случај.

За рад са базом података, ова технологија користи одређени доменски модел. Када је у питању генерисање модела, или генерисање базе података, постоје два приступа [1]:

1) ***Database-First*** приступ подразумева генерисање објектних класа на основу постојеће базе података. То је омогућено помоћу алата који је садржан у *Entity Framework Core-y* и назива се *The Entity Framework Power Tools*.

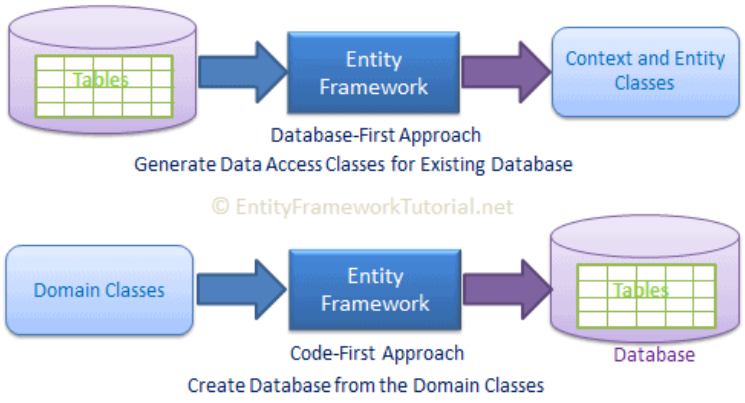
2) ***Code-First*** приступ подразумева усредсређивање на доменске класе, које су дефинисане на основу постојећег концептуалног модела. Сам назив овог приступа указује на то да програмер прво креира доменске класе, тј. пише код за њих, а потом *Entity Framework API* креира табеле у бази података које одговарају доменском моделу. Поред класа које одговарају доменским објектима, неопходно је креирати класу *Context* (назив је произвољан), која наслеђује *DbContext*. Ова класа је уграђена у *Entity Framework Core* и њена инстанца представља једну сесију са базом података. Како би се модел успешно пресликао у базу података и како би се она ажурирала по потреби, користе се миграције (енг. *Migrations*). Класа *Context* је дата кроз следећи код:

```
namespace Cookies.Domain
{
    public class Context : DbContext
    {
        public DbSet<Person> Persons { get; set; }
        public DbSet<User> Users { get; set; }
        public DbSet<Administrator> Administrators { get; set; }
        public DbSet<Cookie> Cookies { get; set; }
        public DbSet<CookieType> CookieTypes { get; set; }
        public DbSet<Order> Orders { get; set; }
        public DbSet<City> Cities { get; set; }
        public DbSet<OrderItem> OrderItem { get; set; }
```

```
        public static readonly ILoggerFactory loggerFactory =
LoggerFactory.Create(builder =>
        {
            builder.AddConsole();
        });
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
optionsBuilder.UseLoggerFactory(loggerFactory).EnableSensitiveDataLogging().UseSqlServer(
@"Server=(localdb)\mssqllocaldb;
            Database=Cookies;");
        }
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
            modelBuilder.Entity<Cookie>().HasOne(c => c.CookieType);
            modelBuilder.Entity<CookieType>().HasMany(ct => ct.Cookies);
            modelBuilder.Entity<Order>().OwnsMany(o=>o.OrderItems,a=> { a.WithOwner(oi =>
oi.Order).HasForeignKey(oi => oi.OrderID); a.HasKey(oi => new { oi.OrderItemID,
oi.OrderID }); });
            modelBuilder.Entity<City>().HasMany(c => c.Persons);
            modelBuilder.Entity<Order>().HasOne(o => o.User);
        }
    }
       }
```

DbSet се користи да би се омогућило креирање и приступање табелама у бази података. Назив овог својства је заправо назив саме табеле. *OnModelCreating* метода служи за дефинисање релација између објеката. На пример, последња линија кода указује на то да се свака наруџбеница (енг. *Order*) односи на једног корисника (енг. *User)* који је извршио поруџбину. За дефинисање наведених веза користи се тзв. *Fluent API*, чију улогу обавља инстанца *modelBuilder*.

На наредној слици је дат приказ ова два приступа:



Слика 17 Упоредни приказ Database-First и Code-First приступа
https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx

Коришћење *Entity Framework Core-a* има бројне предности:

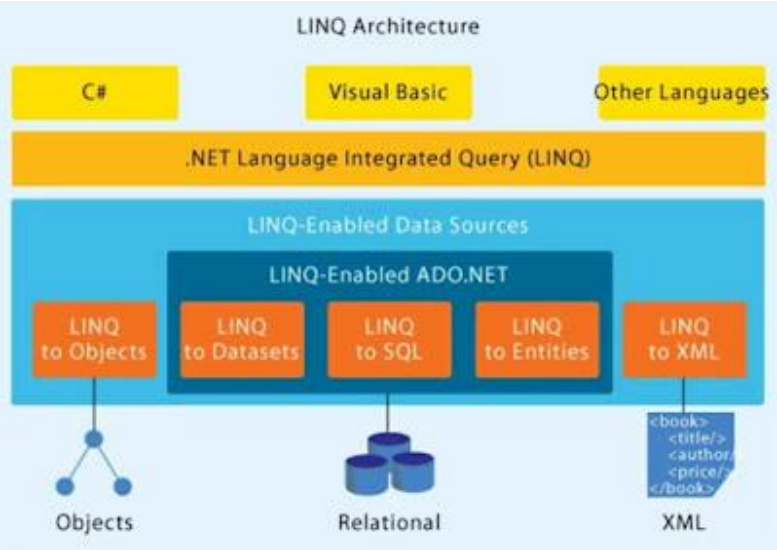
1) Представља вишеплатформски оквир који може да ради на *Windows, Linux* и *MAC* оперативним системима
2) Омогућава коришћење *LINQ* упита
3) Омогућава коришћење *Fluent API-a* за конфигурисање модела
4) Велики део кода се генерише аутоматски
5) Пресликавање између доменског модела и базе података је могуће променити без промене кода саме апликације

4.4.1 LINQ to Entities

LINQ (енг. *Language-Integrated Query*) представља језик структурираних упита за претраживање локалних колекција објеката, као и удаљених извора података, али на начин који не нарушава њихову безбедност. Он подразумева скраћено писање кода у раду са било каквим колекцијама података и динамичким састављањем упита. Ради искључиво са секвенцама и елементима. Сваки објекат који имплементира интерфејс *IEnumerable<T>* се назива секвенца, а ставке секвенце представљају елементе. *LINQ to Entities* је један од *LINQ* провајдера, а оно што се такође много користи када је у питању програмирање у *C#-y* јесу ламбда изрази, који представљају засебан део *LINQ-a*. [27]

Разлози за његово коришћење су, између осталог, то што је много флексибилнији од *LINQ to SQL-a*. Он није ограничен на рад само са базама података у *SQL Server-y*, већ омогућава извршавање упита и у другим провајдерима као што су *Oracle* и *MySQL*. Синтакса је иста као у SQL-у, чак се користе и исте клаузуле: *Join, Order By, Group, Select* и друге. [12]

На слици је приказана LINQ архитектура:



Слика 18 LINQ архитектура
https://www.manuelradovanovic.com/2018/04/uvod-u-linq.html

LINQ to Entities функционише тако што конвертује *LINQ* упите, извршава их над доменским моделом, а резултати које враћа могу да се користе у даљем раду од стране *Entity Framework-a*. Упити се не пишу директно над подацима из базе података, већ над постојећим објектом контекста и враћају колекцију података. *ObjectQuery* је генеричка класа која је задужена за креирање упита и која имплементира генерички интерфејс *IQueryable,* задужен за извршавање упита над одговарајућим извором података, где је тип података познат. Резултат LINQ упита се смешта у променљиву, која служи за чување информација самог упита. [25]

Упит је израз који трансформише секвенце помоћу оператора за упите. Најједноставнији упит се састоји од једног оператора и једне улазне секвенце. У коду је често коришћен оператор *Where,* који омогућава једноставно филтрирање колекције података, без сувишног писања кода.

```
model.ListOfCookies = _orderService.GetAll().Where(s =>
s.CookieName.Contains(searchString)).ToList();
```

Уместо да се прође кроз читаву листу *foreach* петљом, користи се метода *Where* и на тај начин се променљивој додељују сви објекти чији назив садржи унети параметар *searchString*. Овај ламбда израз се користи када се одређује да ли елемент треба укључити у излазну секвенцу или не. За објекте који задовољавају услов филтрирања, метода враћа *true*, а за оне који га не задовољавају враћа *false*. Још један од оператора који се користи за филтрирање јесте *OfType*. Он се може користити у случају негенеричких колекција, као што је на пример *ArrayList*. Пошто *ArrayList* не имплементира *IEnumerable<T>, OfType* је једини LINQ оператор који се може применити на листу. Корисно је употребљавати овај оператор и када постоји више наслеђених класа па треба селектовати само објекте одређеног типа.
У наставку су наведени LINQ изрази који су коришћени при изради апликације:

1. **Where –** служи за филтрирање колекције података, враћа елементе који испуњавају задати услов
2. **Select –** враћа *IEnumerable* колекцију која садржи елементе засноване на функцији трансформације.
3. **Count –** враћа број елемената у колекцији који испуњавају задати услов
4. **SingleOrDefault –** враћа једини елемент колекције који задовољава задати услов; ако их нађе више или не нађе ни један враћа *default* вредност тог типа података
5. **First –** враћа први елемент колекције који задовољава задати услов

Пример коришћења *Select* упита:

```
model.Cities = _personService.GetAllCities().Select(c => new SelectListItem { Text =
c.CityName, Value = c.CityID.ToString() }).ToList();
```

Пример коришћења *SingleOrDefault* упита:

```
User u = (User)context.Persons.SingleOrDefault(p => p.PersonID == id);
```

5. Студијски пример

У овом поглављу биће детаљно објашњен студијски пример израде веб апликације. У те сврхе се користи Ларманова метода. Она је заснована на чињеници да се развој софтверског система састоји из пет фаза [16]:

1. Прикупљања захтева од корисника
2. Анализе
3. Пројектовања
4. Имплементације
5. Тестирања

5.1 Прикупљање корисничких захтева

Захтеви представљају услове које систем мора да задовољи, или својства која треба да поседује. Основни захтеви које сваки софтверски систем треба да задовољи су свакако функционалност, подрживост, употребљивост, поузданост и перформансе. Прва два представљају функционалне захтеве, а остатак су нефункционални захтеви. [16]

Веома је важно да прва фаза пројектовања софтвера буде прецизно одрађена, јер од ње зависи остатак процеса развоја. Како би се произвела апликација која задовољава исказане потребе корисника, неопходно је првенствено идентификовати и препознати те потребе, а затим пројектовати решење које их задовољава.

5.1.1 Вербални опис

Посластичарница „Кућа од медењака“ се бави производњом и продајом различитих врста слаткиша. Како би пословање било унапређено и испратило данашње трендове, створена је могућност поручивања производа онлајн путем и обезбеђивање њихове испоруке.

У посластичарници се праве различите врсте слатких посластица: чоколадне торте, воћне торте, чајно пециво, пуслице, француски макарони, чоколадни колачи, воћни колачи, посне посластице. Свака од посластица које се продају припада некој од поменутих врста. На пример, пуслице са кокосом спадају у тип пуслице, руске капе и сникерс су чоколадни колачи и тако даље.

У систему постоје две улоге – корисник и администратор. Особа која поручује производе је корисник и да би могла да приступи апликацији потребно је да буде улогована под својим корисничким именом и шифром, а такође се може и регистровати уколико већ нема отворен налог. Корисник наручује тако што одабере посластице које жели, количину коју жели и упише адресу на коју жели да добије испоруку. Напомену може да упише, а и не мора. Када потврди своју наруџбеницу, она се чува у складишту података и администратор је може видети. Администратор такође има свој кориснички налог и он може да додаје друге особе

да поред њега имају улогу администратора. Он може да додаје нове колаче, мења постојеће, брише их, а такође може и да рукује наруџбеницама. Уколико је статус наруџбенице Креирано, он га може променити у Одобрено/Одбијено/Испоручено/Отказано у зависности од могућности посластичарнице и потреба корисника. Оног момента када се статус промени из стања Креирано, више се не може мењати. Дакле, уколико се корисник предомисли и жели да поручи више или мање одређених колача, или неке не жели уопште, то може само док наруџбеница не уђе у процес обраде. И корисник и администратор могу да претражују колаче, а пошто администратор има могућност управљања наруџбеницама, он може да претражује и њих по датуму када су настале, корисничком имену особе која је поручила колач, цени и статусу.

5.1.2 Модел случајева коришћења

У овој апликацији, идентификовано је једанаест случајева коришћења:

1. Пријава на систем
2. Креирање корисничког налога
3. Измена корисничког налога
4. Претраживање колача
5. Унос наруџбенице (сложен случај)
6. Унос нових колача
7. Брисање колача
8. Измена колача
9. Унос новог администратора
10. Претрага наруџбеница
11. Измена наруџбенице (сложен случај)

Слика 19 Дијаграм случаја коришћења за корисника

Слика 20 Дијаграм случаја коришћења за администратора

5.1.3 Спецификација случајева коришћења

СК1: Случај коришћења – Пријава на систем

Назив СК
Пријава на систем
Актори СК
Корисник/Администратор
Учесници СК
Корисник/Администратор и систем (програм)
Предуслов: Систем је укључен и приказана је форма за пријаву.

Основни сценарио СК

1. Корисник/Администратор **уноси** своје податке за пријављивање. (АПУСО)
2. Корисник/Администратор **проверава** да ли је коректно унео податке. (АНСО)
3. Корисник/Администратор **позива** систем да нађе пријављеног корисника/администратора. (АПСО)
4. Систем **тражи** пријављеног корисника/администратора. (СО)

5. Систем **приказује** кориснику/администратору поруку: „Успешна пријава”. (ИА)

Алтернативна сценарија

5.1 Уколико систем није успео да нађе пријављеног корисника/администратора, он **приказује** поруку: „Нисте унели одговарајуће податке“. (ИА)

СК2: Случај коришћења – Креирање корисничког налога

Назив СК
Креирање корисничког налога
Актори СК
Корисник
Учесници СК
Корисник и систем (програм)
Предуслов: Систем је укључен и приказана је форма за регистрацију. Учитана је листа свих градова.

Основни сценарио СК
1. Корисник **уноси** основне податке за креирање корисничког налога. (АПУСО)
2. Корисник **проверава** да ли је коректно унео своје податке. (АНСО)
3. Корисник **позива** систем да креира кориснички налог. (АПСО)
4. Систем **креира** кориснички налог. (СО)
5. Систем **приказује** кориснику поруку: „Ваш налог је успешно креиран”. (ИА)

Алтернативна сценарија
5.1 Уколико регистрација није могућа, систем **приказује** кориснику поруку: „Жао нам је, регистрација није успела“. (ИА)

СК3: Случај коришћења – Измена корисничког налога

Назив СК
Измена корисничког налога
Актори СК
Корисник
Учесници СК
Корисник и систем (програм)
Предуслов: Систем је укључен и корисник је пријављен под својом шифром. Систем приказује податке о пријављеном кориснику. Учитани су подаци о корисничком налогу и листа свих градова.

Основни сценарио СК
1. Корисник **бира** податке који жели да измени. (АПУСО)

2. Корисник **мења** податке. (АПУСО)
3. Корисник **проверава** да ли је коректно унео нове податке. (АНСО)
4. Корисник **позива** систем да промени податке. (АПСО)
5. Систем **мења** податке о кориснику. (СО)
6. Систем **приказује** кориснику поруку: „Успешно сте променили податке“. (ИА)

Алтернативна сценарија

6.1 Уколико систем није у могућности да промени податке, **приказује** поруку: „Није могуће променити податке.“ (ИА)

СК4: Случај коришћења – Претраживање колача

Назив СК
Претраживање колача
Актори СК
Корисник
Учесници СК
Корисник и систем (програм)
Предуслов: Систем је укључен и корисник је пријављен под својом шифром. Систем приказује интерфејс за пријављене кориснике. Учитана је листа свих колача.

Основни сценарио СК

1. Корисник **уноси** вредност по којој претражује колаче. (АПУСО)
2. Корисник **проверава** да ли је коректно унео вредност по којој претражује. (АНСО)
3. Корисник **позива** систем да нађе колаче на основу унете вредности. (АПСО)
4. Систем **тражи** колаче по наведеном критеријуму. (СО)
5. Систем **приказује** листу тражених колача. (ИА)

Алтернативна сценарија

5.1 Уколико систем не може да нађе колаче по унетом критеријуму он **приказује** кориснику поруку: “Систем не може да нађе колаче по задатој вредности”. (ИА)

СК5: Случај коришћења – Унос наруџбенице

Назив СК
Унос наруџбенице
Актори СК
Корисник
Учесници СК
Корисник и систем (програм)

Предуслов: Систем је укључен и кориснику, који је улогован под својом шифром, приказана је листа одабраних колача. Систем приказује форму за наручивање. Учитани су подаци о колачима.

Основни сценарио СК

1. Корисник **уноси** податке о наруџбеници. (АПУСО)
2. Корисник **проверава** да ли је правилно унео податке о наруџбеници. (АНСО)
3. Корисник **позива** систем да сачува податке о наруџбеници. (АПСО)
4. Систем **чува** податке о наруџбеници. (СО)
5. Систем **приказује** кориснику поруку: "Наруџбеница је прихваћена". (ИА)

Алтернативна сценарија

5.1 Уколико систем није у могућности да изврши унос наруџбенице, **приказује** поруку: „Наруџбеница је одбијена". (ИА)

СК6: Случај коришћења – Унос нових колача

Назив СК
Унос нових колача
Актори СК
Администратор
Учесници СК
Администратор и систем (програм)
Предуслов: Систем је укључен и администратор је улогован под својом шифром. Систем приказује интерфејс за унос нових колача. Учитана је листа типова колача.

Основни сценарио СК

1. Администратор **уноси** податке о новим колачима. (АПУСО)
2. Администратор **контролише** да ли је коректно унео податке о колачима. (АНСО)
3. Администратор **позива** систем да запамти податке о колачима. (АПСО)
4. Систем **памти** податке о колачима. (СО)
5. Систем **приказује** администратору запамћене податке о колачима и поруку: „Успешно сте унели нове колаче". (ИА)

Алтернативна сценарија

5.1 Уколико систем не може да прихвати податке о новим колачима, он **приказује** администратору поруку „Није могуће унети нове колаче". (ИА)

СК7: Случај коришћења – Брисање колача

Назив СК
Брисање колача
Актори СК
Администратор
Учесници СК
Администратор и систем (програм)
Предуслов: Систем је укључен и администратор је пријављен под својом шифром. Систем приказује интерфејс за брисање колача. Учитана је листа свих колача.

Основни сценарио СК

1. Администратор **уноси** вредност по којој претражује колаче. (АПУСО)
2. Администратор **проверава** да ли је коректно унео вредност по којој претражује. (АНСО)
3. Администратор **позива** систем да нађе колаче на основу унете вредности. (АПСО)
4. Систем **тражи** колаче по наведеном критеријуму. (СО)
5. Систем **приказује** листу тражених колача. (ИА)
6. Администратор **бира** колаче који жели да избрише. (АПУСО)
7. Администратор **позива** систем да избрише податке о одабраним колачима. (АПСО)
8. Систем **брише** податке о колачима. (СО)
9. Систем **приказује** администратору листу постојећих производа и поруку: „Успешно избрисани колачи“. (ИА)

Алтернативна сценарија

5.1 Уколико систем не може да нађе колаче по унетом критеријуму он **приказује** кориснику поруку: “Систем не може да нађе колаче по задатој вредности.”. Прекида се извршење сценарија. (ИА)
9.1 Уколико систем није успешно избрисао тражене колаче, он **приказује** поруку кориснику: „Систем не може да избрише тражене колаче“. (ИА)

СК8: Случај коришћења – Измена колача

Назив СК
Измена колача
Актори СК
Администратор
Учесници СК
Администратор и систем (програм)
Предуслов: : Систем је укључен и администратор је пријављен под својом шифром. Систем приказује интерфејс за измену колача. Учитани су подаци о колачу.

Основни сценарио СК

1. Администратор **бира** податке који жели да измени. (АПУСО)
2. Администратор **мења** податке. (АПУСО)
3. Администратор **проверава** да ли је коректно унео нове податке. (АНСО)
4. Администратор **позива** систем да промени податке. (АПСО)
5. Систем **мења** податке о колачима. (СО)
6. Систем **приказује** администратору поруку: „Успешно измењени подаци о колачима!”. (ИА)

Алтернативна сценарија

6.1 Уколико систем није у могућности да промени податке, **приказује** поруку: „Није могуће променити податке о колачима.“ (ИА)

СК9: Случај коришћења – Унос новог администратора

Назив СК
Унос новог администратора
Актори СК
Администратор
Учесници СК
Администратор и систем (програм)
Предуслов: Систем је укључен и приказана је форма за унос новог администратора. Учитана је листа свих градова.

Основни сценарио СК

1. Администратор **уноси** основне податке за новог администратора. (АПУСО)
2. Администратор **проверава** да ли је коректно унео податке за новог администратора. (АНСО)
3. Администратор **позива** систем да запамти новог администратора. (АПСО)
4. Систем **памти** новог администратора. (СО)
5. Систем **приказује** администратору поруку: „Нови администратор је унет у базу!”. (ИА)

Алтернативна сценарија

5.1 Уколико унос новог администратора није могућ, систем **приказује** администратору поруку: „Унос новог администратора није успео!“. (ИА)

СК10: Случај коришћења – Претрага наруџбеница

Назив СК
Претрага наруџбеница
Актори СК
Администратор
Учесници СК
Администратор и систем (програм)
Предуслов: Систем је укључен и администратор је пријављен под својом шифром. Систем приказује интерфејс за претрагу наруџбеница. Учитана је листа свих наруџбеница.

Основни сценарио СК

1. Администратор **уноси** критеријум по ком претражује наруџбенице. (АПУСО)
2. Администратор **проверава** да ли је коректно унео критеријум по ком претражује. (АНСО)
3. Администратор **позива** систем да нађе наруџбенице на основу унетог критеријума. (АПСО)
4. Систем **тражи** наруџбенице по наведеном критеријуму. (СО)
5. Систем **приказује** листу одговарајућих наруџбеница. (ИА)
6. Администратор **бира** наруџбеницу. (АПУСО)
7. Администратор **позива** систем да прикаже све податке о изабраној наруџбеници. (АПСО)
8. Систем **тражи** податке о наруџбеници. (СО)
9. Систем **приказује** кориснику све податке о наруџбеници уз поруку: “Ово су подаци о наруџбеници коју сте изабрали”. (ИА)

Алтернативна сценарија

5.1 Уколико систем не може да нађе наруџбенице по унетом критеријуму он **приказује** адмнистратору поруку: “Не постоје наруџбенице које одговарају задатом критеријуму”. Прекида се извршење сценарија. (ИА)
9.1 Уколико систем не може да прикаже податке о одабраној наруџбеници, он **приказује** администратору поруку: „Систем не може да прикаже податке о наруџбеници“.

СК11: Случај коришћења – Измена наруџбенице

Назив СК
Измена наруџбенице
Актори СК
Администратор
Учесници СК
Администратор и систем (програм)
Предуслов: Систем је укључен и администратор је пријављен под својом шифром. Систем приказује интерфејс за измену наруџбеница. Учитана је листа свих наруџбеница.

Основни сценарио СК

1. Администратор **уноси** критеријум по ком претражује наруџбенице. (АПУСО)
2. Администратор **проверава** да ли је коректно унео критеријум по ком претражује. (АНСО)
3. Администратор **позива** систем да нађе наруџбенице на основу унетог критеријума. (АПСО)
4. Систем **тражи** наруџбенице по наведеном критеријуму. (СО)
5. Систем **приказује** листу одговарајућих наруџбеница. (ИА)
6. Администратор **бира** наруџбеницу. (АПУСО)
7. Администратор **позива** систем да прикаже све податке о изабраној наруџбеници. (АПСО)
8. Систем **тражи** податке о наруџбеници. (СО)
9. Систем **приказује** кориснику све податке о наруџбеници уз поруку: “Ово су подаци о наруџбеници коју сте изабрали”. (ИА)
10. Администратор **мења** наруџбеницу. (АПУСО)
11. Администратор **проверава** да ли је коректно унео нове податке о наруџбеници. (АНСО)
12. Администратор **позива** систем да промени податке. (АПСО)
13. Систем **мења** податке о наруџбеници. (СО)
14. Систем **приказује** администратору поруку: „Успешно сте променили податке о наруџбеници!”. (ИА)

Алтернативна сценарија

5.1 Уколико систем не може да нађе наруџбенице по унетом критеријуму он **приказује** адмнистратору поруку: “Не постоје наруџбенице које одговарају задатом критеријуму”. Прекида се извршење сценарија. (ИА)
9.1 Уколико систем не може да прикаже податке о одабраној наруџбеници, он **приказује** администратору поруку: „Није могуће приказати податке о наруџбеници“. Прекида се извршење сценарија. (ИА)
14.1. Уколико систем није у могућности да промени податке о одабраној наруџбеници, он **приказује** администратору поруку: “Није могуће променити податке о наруџбеници”. (ИА)

5.2 Анализа

У другој фази развоја софтверског система описује се његова пословна логика, односно структура и понашање тог система. Понашање је у даљем раду приказано кроз системске дијаграме секвенци и системске операције, а структура помоћу концептуалног и релационог модела. [16]

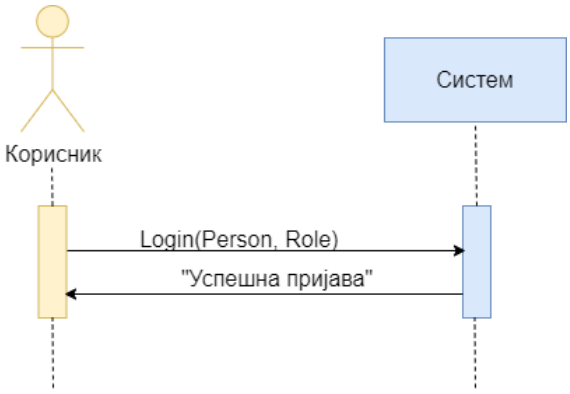
5.2.1 Понашање софтверског система - Системски дијаграми секвенци

Системски дијаграми секвенци приказују двосмерну комуникацију између актора и система. Актори праве догађаје, а након извршења одговарајуће системске операције на страни система, њима се враћа одговор на тај догађај. Догађаји се одвијају у одређеном редоследу и тај редослед је приказан на дијаграмима секвенци. [16]

ДС1 Дијаграм секвенци случаја коришћења – Пријава на систем

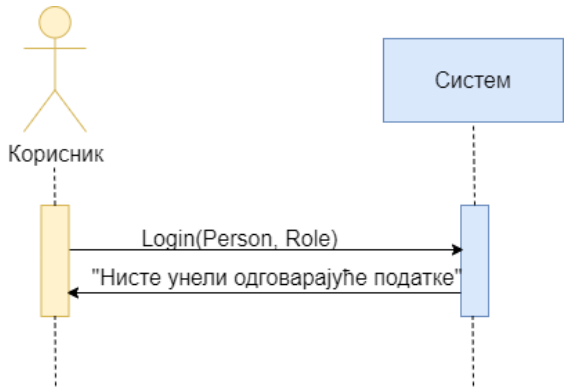
Основни сценарио СК

1.Корисник/Администратор **позива** систем да нађе пријављеног корисника/администратора. (АПСО)
2. Систем **приказује** кориснику/администратору поруку: „Успешна пријава”. (ИА)



Слика 21 ДС Пријава на систем – основни сценарио

Алтернативна сценарија
2.1 Уколико систем није успео да нађе пријављеног корисника/администратора, он **приказује** поруку: Нисте унели одговарајуће податке“. (ИА)



Слика 22 ДС Пријава на систем - алтернативни сценарио

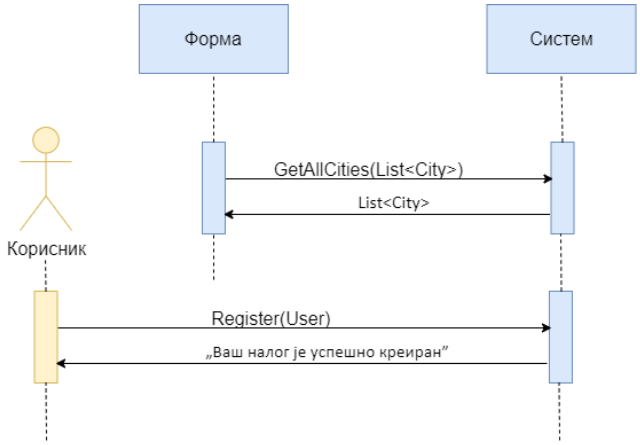
Са наведених секвенцних дијаграма уочава се једна системска операција коју треба пројектовати:

1. *signal* **Login (Person, Role);**

ДС2 Дијаграм секвенци случаја коришћења – Креирање корисничког налога

Основни сценарио СК

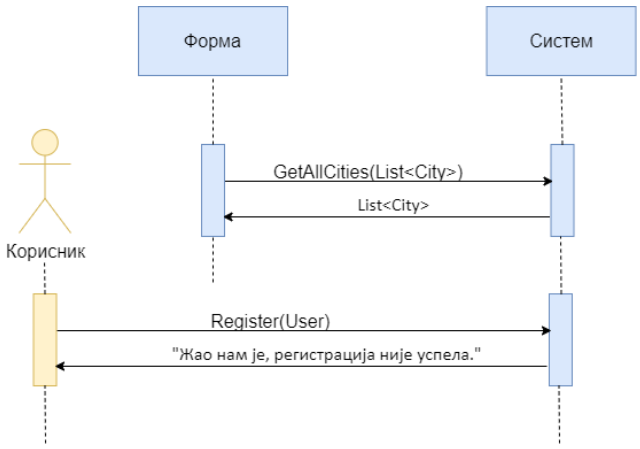
1. Форма **позива** систем да учита листу градова. (АПСО)
2. Систем **враћа** форми листу градова. (ИА)
3. Корисник **позива** систем да креира кориснички налог. (АПСО)
4. Систем **приказује** кориснику поруку: „Ваш налог је успешно креиран”. (ИА)



Слика 23 ДС Креирање корисничког налога - основни сценарио

Алтернативна сценарија

4.1 Уколико регистрација није могућа, систем **приказује** кориснику поруку: „Жао нам је, регистрација није успела“. (ИА)



Слика 24 ДС Креирање корисничког налога - алтернативни сценарио

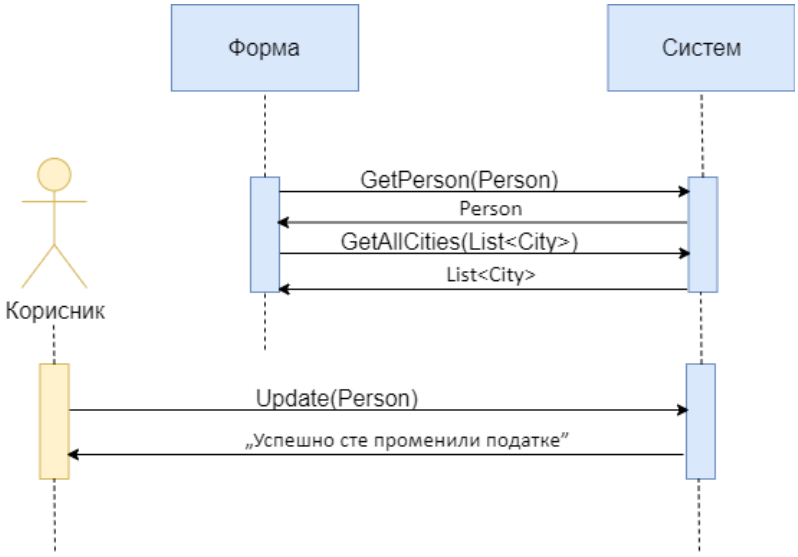
Са наведених секвенцних дијаграма уочавају се две системске операције које треба пројектовати:

1. *signal* **GetAllCities(List<City>)**;
2. *signal* **Register (User)**;

ДС3 Дијаграм секвенци случаја коришћења – Измена корисничког налога

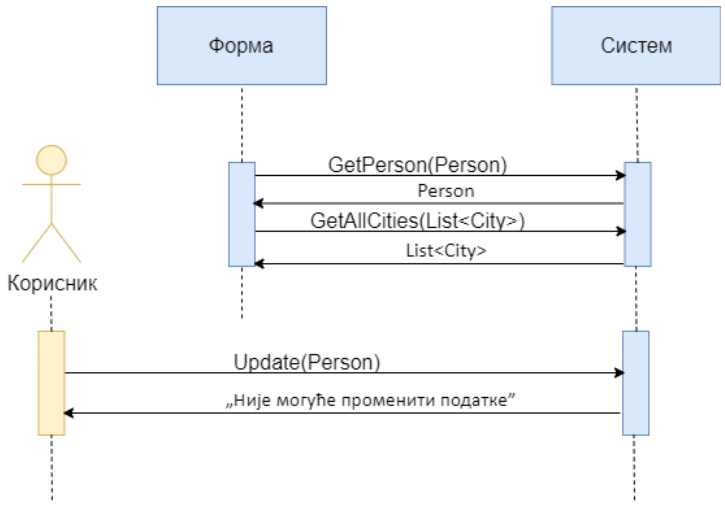
Основни сценарио СК

1. Корисник **позива** систем да промени податке. (АПСО)
2. Систем **приказује** кориснику поруку: „Успешно сте променили податке”. (ИА)



Слика 25 ДС Измена корисничког налога - основни сценарио

Алтернативна сценарија
2.1 Уколико систем није у могућности да промени податке, **приказује** поруку: „Није могуће променити податке.“ (ИА)



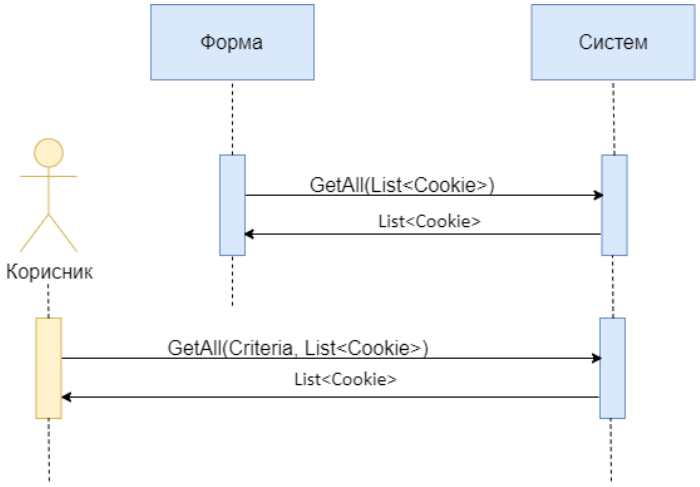
Слика 26 ДС Измена корисничког налога - алтернативни сценарио

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

1. *signal* **GetPerson (Person);**
2. *signal* **GetAllCities (List<City>);**
3. *signal* **Update (Person);**

ДС4 Дијаграм секвенци случаја коришћења – Претраживање колача

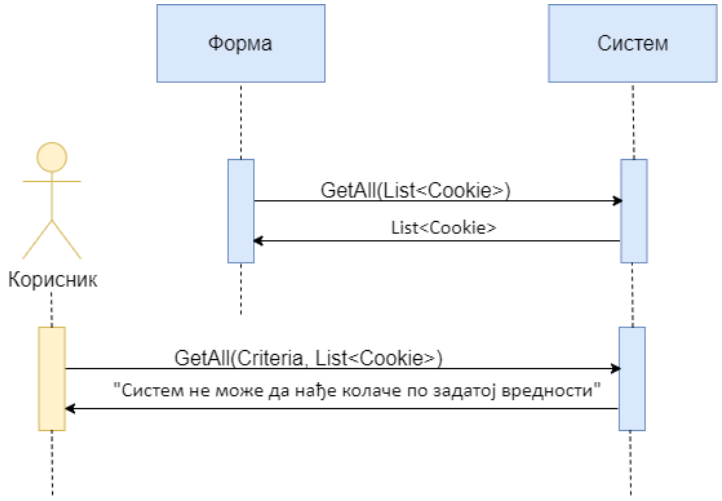
Основни сценарио СК
1. Форма **позива** систем да учита листу колача. (АПСО)
2. Систем **враћа** форми листу колача. (ИА)
3. Корисник **позива** систем да нађе колаче на основу унете вредности. (АПСО)
4. Систем **приказује** листу тражених колача. (ИА)



Слика 27 ДС Претраживање колача - основни сценарио

Алтернативна сценарија

4.1 Уколико систем не може да нађе колаче по унетом критеријуму он **приказује** кориснику поруку: „Систем не може да нађе колаче по задатој вредности”.



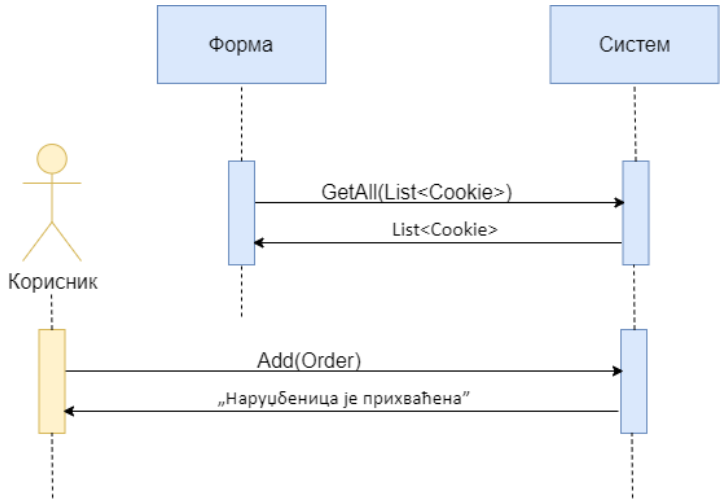
Слика 28 ДС Претраживање колача - алтернативни сценарио

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

1. *signal* **GetAll(List<Cookie>)**;
2. *signal* **GetAll(Criteria, List<Cookie>)**;

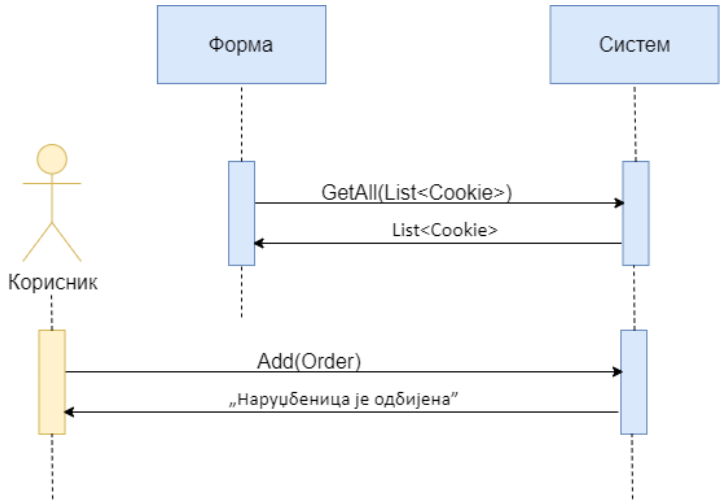
ДС5 Дијаграм секвенци случаја коришћења – Унос наруџбенице

Основни сценарио СК
1. Форма **позива** систем да учита листу колача. (АПСО)
2. Систем **враћа** форми листу колача. (ИА)
3. Корисник **позива** систем да сачува податке о наруџбеници. (АПСО)
4. Систем **приказује** кориснику поруку: „Наруџбеница је прихваћена”. (ИА)



Слика 29 ДС Унос наруџбенице - основни сценарио

Алтернативна сценарија
4. 1 Уколико систем није у могућности да изврши унос наруџбенице, **приказује** поруку: „Наруџбеница је одбијена“. (ИА)



Слика 30 ДС Унос наруџбенице - алтернативни сценарио

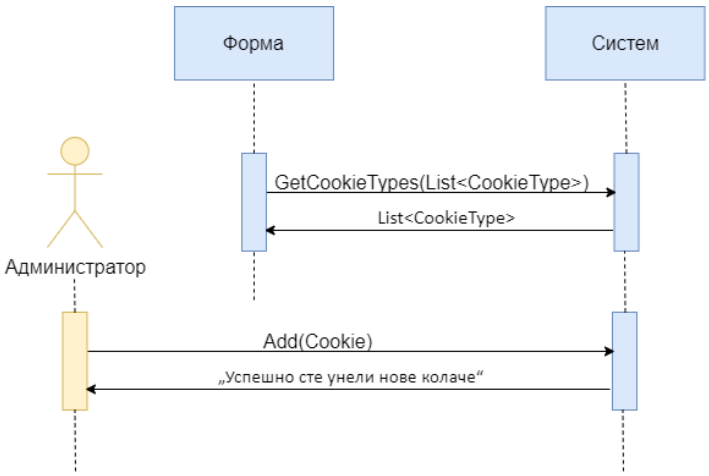
Са наведених секвенцних дијаграма уочавају се две системске операције које треба пројектовати:

1. *signal* **GetAll(List<Cookie>)**;
2. *signal* **Add(Order)**;

ДС6 Дијаграм секвенци случаја коришћења – Унос нових колача

Основни сценарио СК

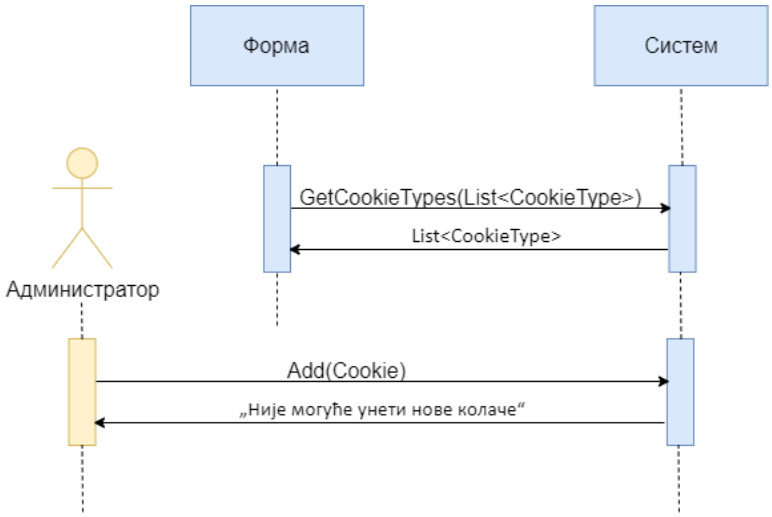
1. Форма **позива** систем да учита листу типова колача. (АПСО)
2. Систем **враћа** форми листу типова колача. (ИА)
3. Администратор **позива** систем да запамти податке о колачима. (АПСО)
4. Систем **приказује** администратору запамћене податке о колачима и поруку: „Успешно сте унели нове колаче“. (ИА)



Слика 31 ДС Унос нових колача - основни сценарио

Алтернативна сценарија

4.1 Уколико систем не може да прихвати податке о новим колачима, он **приказује** администратору поруку „Није могуће унети нове колаче”. (ИА)



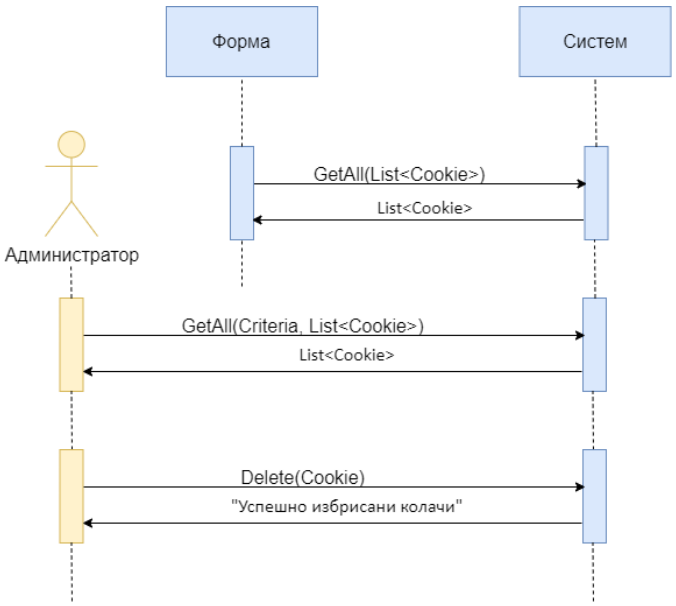
Слика 32 ДС Унос нових колача - алтернативни сценарио

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

1. *signal* **GetCookieTypes(List<CookieType>)**;
2. *signal* **Add(Cookie)**;

ДС7 Дијаграм секвенци случаја коришћења – Брисање колача

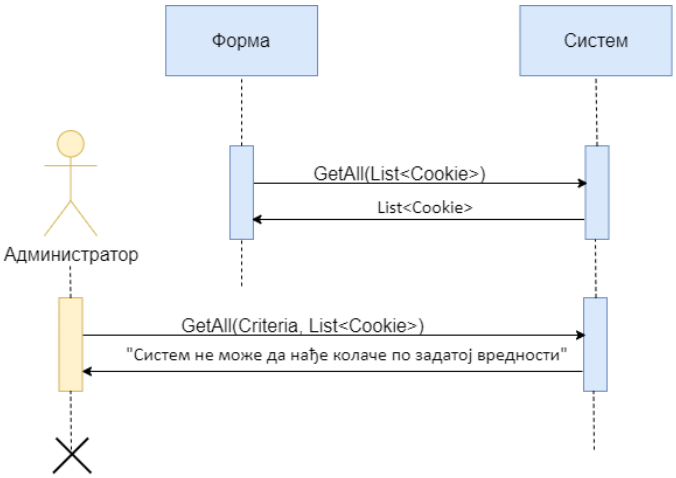
1. Форма **позива** систем да учита листу колача. (АПСО)
2. Систем **враћа** форми листу колача. (ИА)
3. Администратор **позива** систем да нађе колаче на основу унете вредности. (АПСО)
4. Систем **приказује** листу тражених колача. (ИА)
5. Администратор **позива** систем да избрише податке о одабраним колачима. (АПСО)
6. Систем **приказује** администратору листу постојећих производа и поруку: „Успешно избрисани колачи“. (ИА)



Слика 33 ДС Брисање колача - основни сценарио

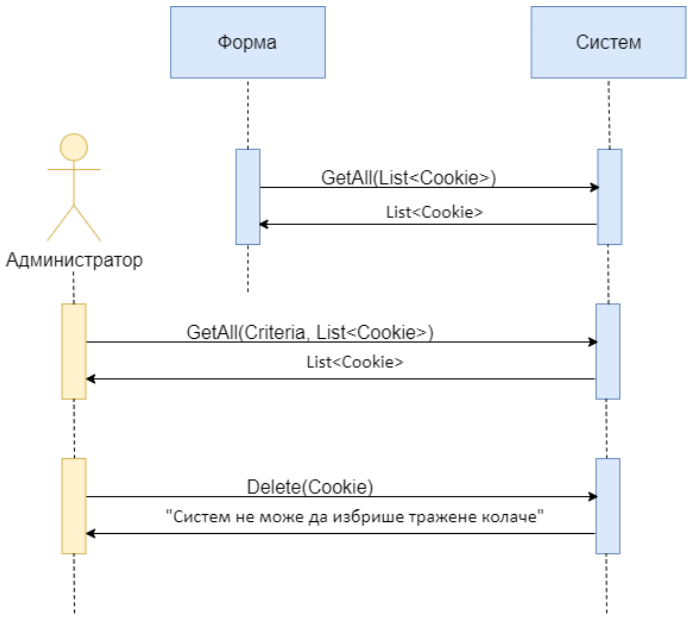
Алтернативна сценарија

4.1 Уколико систем не може да нађе колаче по унетом критеријуму он **приказује** кориснику поруку: “Систем не може да нађе колаче по задатој вредности.”. Прекида се извршење сценарија. (ИА)



Слика 34 Брисање колача - алтернативни сценарио 1

6.1 Уколико систем није успешно избрисао тражене колаче, он **приказује** поруку кориснику: „Систем не може да избрише тражене колаче“. (ИА)



Слика 35 ДС Брисање колача - алтернативни сценарио 2

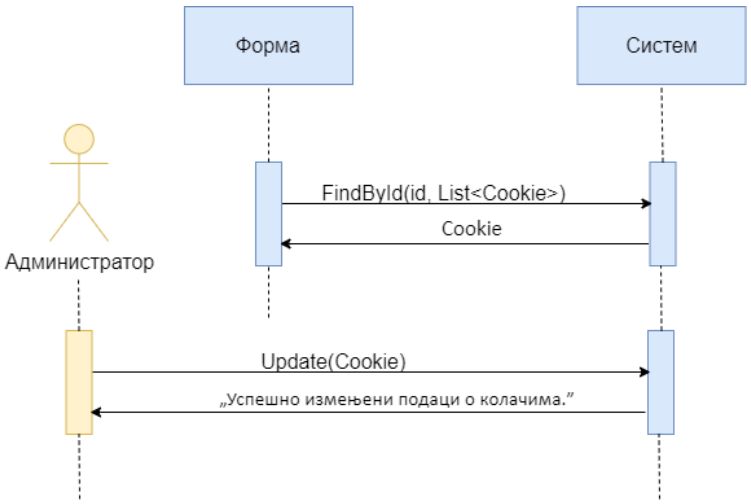
Са наведених секвенцних дијаграма уочавају се три системске операције које треба пројектовати:

1. *signal* **GetAll(List<Cookie>)**;
2. *signal* **GetAll(Criteria, List<Cookie>)**;
3. *signal* **Delete(Cookie);**

ДС8 Дијаграм секвенци случаја коришћења – Измена колача

Основни сценарио СК

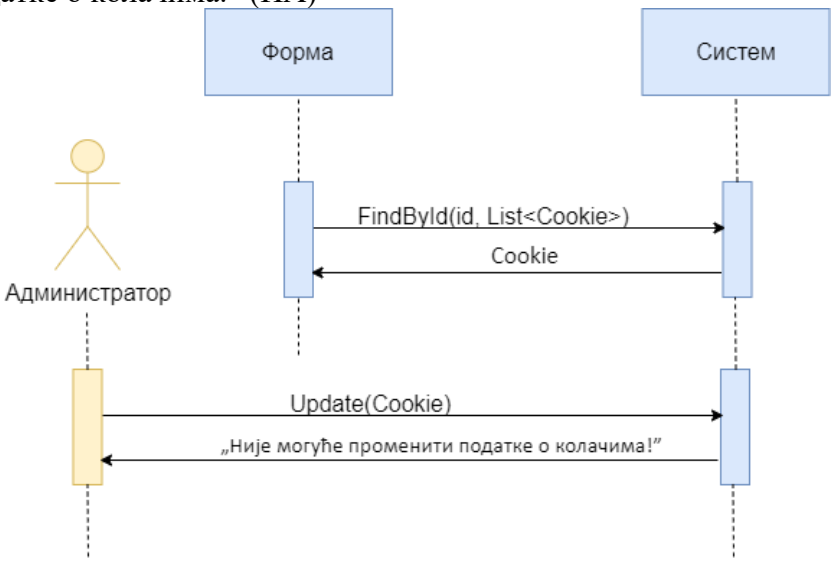
1. Форма **позива** систем да учита податке о колачима. (АПСО)
2. Систем **враћа** форми податке о колачима. (ИА)
3. Администратор **позива** систем да промени податке. (АПСО)
4. Систем **приказује** администратору поруку: „Успешно измењени подаци о колачима!”. (ИА)



Слика 36 ДС Измена колача - основни сценарио

Алтернативна сценарија

8.1 Уколико систем није у могућности да промени податке, **приказује** поруку: „Није могуће променити податке о колачима.“ (ИА)



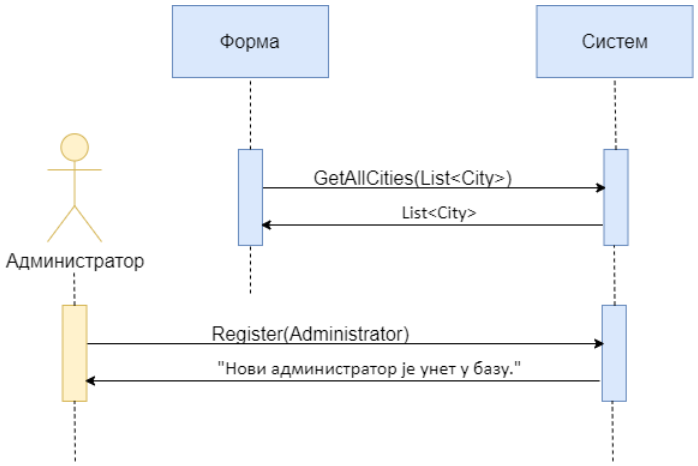
Слика 37 ДС Измена колача - алтернативни сценарио

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

1. *signal* **FindById(id, List<Cookie>)**;
2. *signal* **Update (Cookie)**;

ДС9 Дијаграм секвенци случаја коришћења – Унос новог администратора

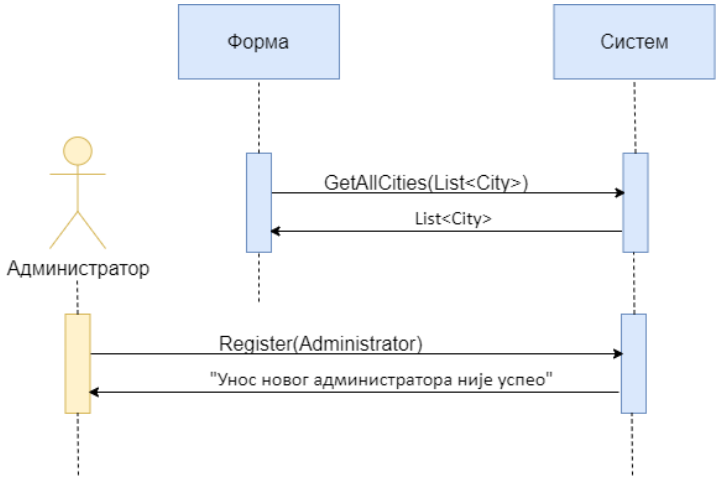
1. Форма **позива** систем да учита листу градова. (АПСО)
2. Систем **враћа** форми листу градова. (ИА)
3. Администратор **позива** систем да запамти новог администратора. (АПСО)
5. Систем **приказује** администратору поруку: „Нови администратор је унет у базу!”. (ИА)



Слика 38 ДС Унос новог администратора - основни сценарио

Алтернативна сценарија

5.1 Уколико унос новог администратора није могућ, систем **приказује** администратору поруку: „Унос новог администратора није успео!“. (ИА)



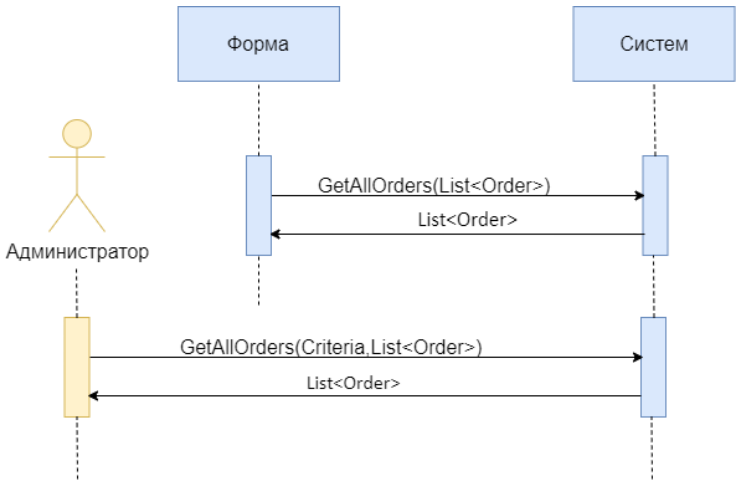
Слика 39 ДС Унос новог администратора - алтернативни сценарио

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

1. *signal* **GetAllCities(List<City>)**;
2. *signal* **Register(Administrator)**;

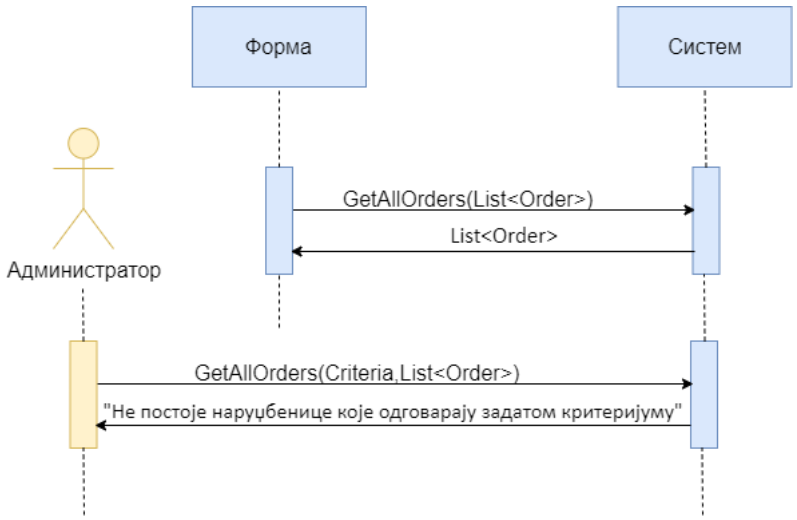
ДС10 Дијаграм секвенци случаја коришћења – Претрага наруџбеница

Основни сценарио СК
1. Форма **позива** систем да учита листу наруџбеница. (АПСО)
2. Систем **враћа** форми листу наруџбеница. (ИА)
3. Администратор **позива** систем да нађе наруџбенице на основу унетог критеријума. (АПСО)
4. Систем **приказује** листу одговарајућих наруџбеница. (ИА)



Слика 40 ДС Претрага наруџбеница - основни сценарио

Алтернативна сценарија
5.1 Уколико систем не може да нађе наруџбенице по унетом критеријуму он **приказује** адмнистратору поруку: „Не постоје наруџбенице које одговарају задатом критеријуму”. (ИА)

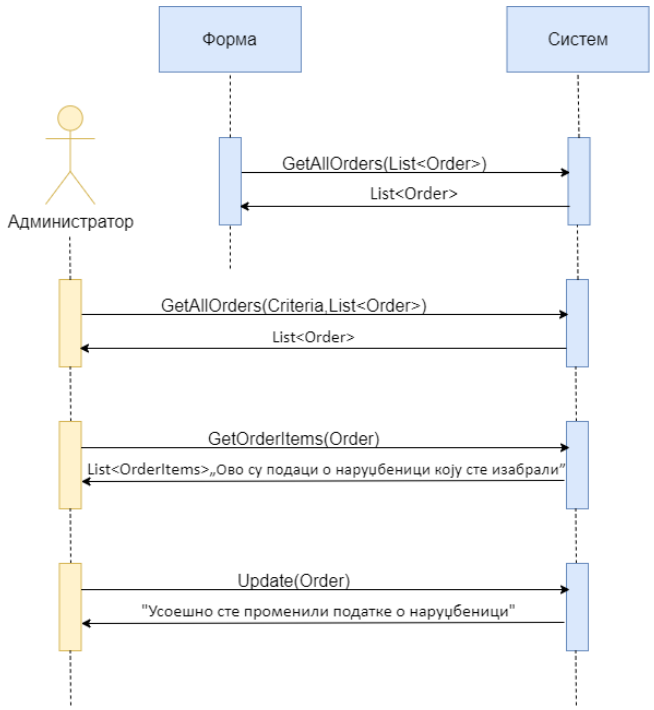


Слика 41 ДС Претрага наруџбеница - алтернативни сценарио

Са наведених секвенцних дијаграма уочавају се две системске операције које треба пројектовати:
1. *signal* **GetAllOrders(List<Order>)**
2. *signal* **GetAllOrders(Criteria, List<Order>)**

ДС11 Дијаграм секвенци случаја коришћења – Измена наруџбенице

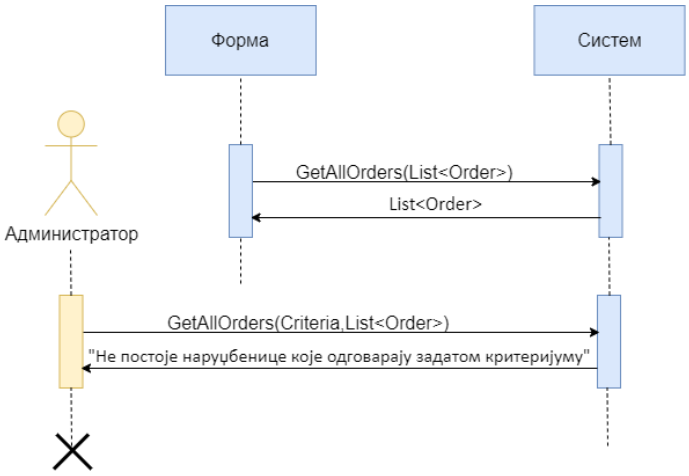
Основни сценарио СК
1. Форма **позива** систем да учита листу наруџбеница. (АПСО)
2. Систем **враћа** форми листу наруџбеница. (ИА)
3. Администратор **позива** систем да нађе наруџбенице на основу унетог критеријума. (АПСО)
4. Систем **приказује** листу одговарајућих наруџбеница (ИА)
5. Администратор **позива** систем да прикаже све податке о изабраној наруџбеници. (АПСО)
6. Систем **приказује** кориснику све податке о наруџбеници уз поруку: „Ово су подаци о наруџбеници коју сте изабрали”. (ИА)
7. Администратор **позива** систем да промени податке. (АПСО)
8. Систем **приказује** администратору поруку: „Успешно сте променили податке о наруџбеници!”. (ИА)



Слика 42 ДС Измена наруџбенице - основни сценарио

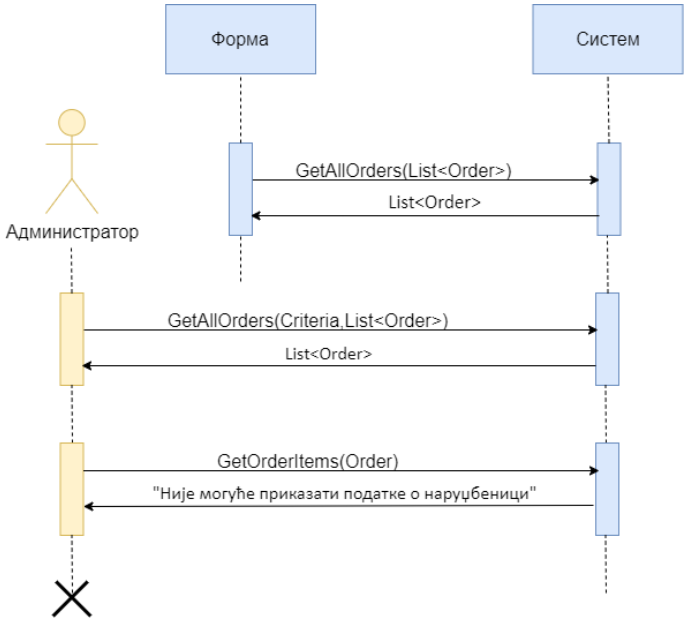
Алтернативна сценарија

4.1 Уколико систем не може да нађе наруџбенице по унетом критеријуму он **приказује** адмнистратору поруку: „Не постоје наруџбенице које одговарају задатом критеријуму". Прекида се извршење сценарија. (ИА)



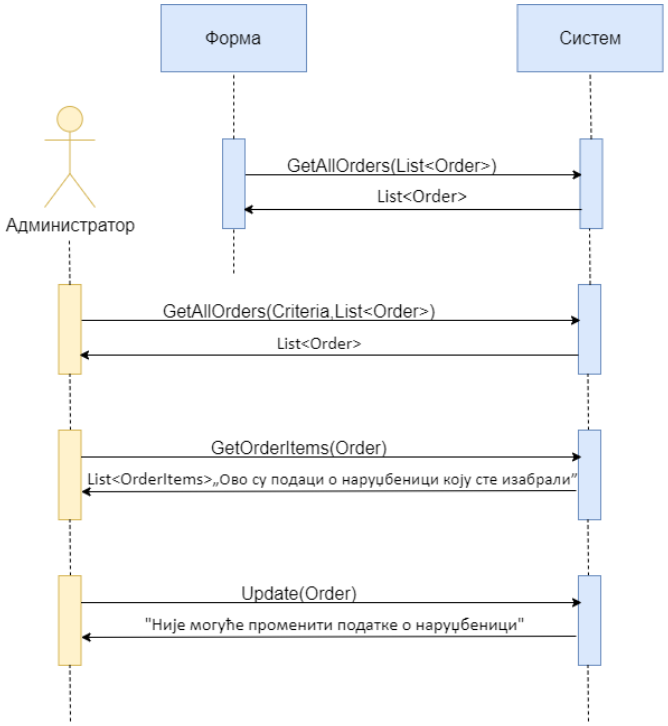
Слика 43 ДС Измена наруџбеница - алтернативни сценарио 1

6.1 Уколико систем не може да прикаже податке о одабраној наруџбеници, он **приказује** администратору поруку: „Није могуће приказати податке о наруџбеници“. Прекида се извршење сценарија. (ИА)



Слика 44 ДС Измена наруџбеница - алтернативни сценарио 2

8.1. Уколико систем није у могућности да промени податке о одабраној наруџбеници, он **приказује** администратору поруку: “Није могуће променити податке о наруџбеници”. (ИА)



Слика 45 ДС Измена наруџбеница - алтернативни сценарио 3

Са наведених секвенцних дијаграма уочавају се четири системске операције које треба пројектовати:
1. *signal* **GetAllOrders(List<Order>)**
2. *signal* **GetAllOrders(Criteria, List(Order>)**
3. *signal* **GetOrderItems(Order)**
4. *signal* **Update(Order)**

Као резултат фазе анализе добијено је укупно 18 системских операција које треба пројектовати:

1. *signal* **Login (Person, Role);**
2. *signal* **GetAllCities(List<City>);**
3. *signal* **Register (User);**
4. *signal* **Update (Person);**
5. *signal* **GetAll(List<Cookie>);**
6. *signal* **GetAll(Criteria,List<Cookie>);**
7. *signal* **Add(Order);**
8. *signal* **GetCookieTypes(List<CookieType>);**
9. *signal* **Add(Cookie);**
10. *signal* **Delete(Cookie);**
11. *signal* **FindById(id, List<Cookie>);**
12. *signal* **Update (Cookie);**
13. *signal* **Register(Administrator);**
14. *signal* **GetAllOrders(List<Order>)**

15. *signal* **GetAllOrders(Criteria, List<Order>)**
16. *signal* **GetOrderItems(Order)**
17. *signal* **Update(Order)**
18. *signal* ***GetPerson(Person)***

5.2.2 Понашање софтверског система - Дефинисање уговора о системским операцијама

За сваку системску операцију прави се уговор који описује њено понашање. Они описују шта операција треба да ради, а не начин на који ће нешто да изврши. Један уговор је везан искључиво за једну системску операцију. [16]

Уговор УГ1: Login

Операција: Login(Person, Role): сигнал
Веза са СК: СК1
Предуслови: Вредносна и структурна ограничења над објектом **Person** морају бити задовољена.
Постуслови: Корисник/Администратор је пријављен.

Уговор УГ2: GetAllCities

Операција: GetAllCities(List<City>): сигнал
Веза са СК: СК1, СК2, СК9
Предуслови:
Постуслови:

Уговор УГ3: Register

Операција: Register(User): сигнал
Веза са СК: СК2
Предуслови: Вредносна и структурна ограничења над објектом **User** морају бити задовољена.
Постуслови: Регистрован је нови корисник.

Уговор УГ4: Update

Операција: Update(Person): сигнал
Веза са СК: СК3
Предуслови: Вредносна и структурна ограничења над објектом **Person** морају бити задовољена.
Постуслови: Корисник је сачуван.

Уговор УГ5: GetAll

Операција: GetAll(List<Cookie>): сигнал
Веза са СК: СК4, СК5, СК7
Предуслови:
Постуслови:

Уговор УГ6: GetAll

Операција: GetAll(Criteria, List<Cookie>): сигнал
Веза са СК: СК4, СК7
Предуслови:
Постуслови:

Уговор УГ7: Add

Операција: Add(Order): сигнал
Веза са СК: СК5
Предуслови: Вредносна и структурна ограничења над објектом **Order** морају бити задовољена.
Постуслови: Наруџбеница је сачувана.

Уговор УГ8: GetCookieTypes

Операција: GetCookieTypes(List<CookieType>): сигнал
Веза са СК: СК6
Предуслови:
Постуслови:

Уговор УГ9: Add

Операција: Add(Cookie): сигнал
Веза са СК: СК6
Предуслови: Вредносна и структурна ограничења над објектом **Cookie** морају бити задовољена.
Постуслови: Колач је сачуван.

Уговор УГ10: Delete

Операција: Delete(Cookie): сигнал
Веза са СК: СК7
Предуслови: Вредносна и структурна ограничења над објектом **Cookie** морају бити задовољена.
Постуслови: Колач је избрисан.

Уговор УГ11: FindById

Операција: FindById(Id, List<Cookie>): сигнал
Веза са СК: СК8
Предуслови:
Постуслови:

Уговор УГ12: Update

Операција: Update(Cookie): сигнал
Веза са СК: СК8
Предуслови: Вредносна и структурна ограничења над објектом **Cookie** морају бити задовољена.
Постуслови: Колач је сачуван.

Уговор УГ13: Register

Операција: Register(Administrtor): сигнал
Веза са СК: СК9
Предуслови: Вредносна и структурна ограничења над објектом **Administrator** морају бити задовољена.
Постуслови: Регистрован је нови администратор.

Уговор УГ14: GetAllOrders

Операција: GetAllOrders(List<Order>): сигнал
Веза са СК: СК10, СК11
Предуслови:
Постуслови:

Уговор УГ15: GetAllOrders

Операција: GetAllOrders(Criteria, List<Order>): сигнал
Веза са СК: СК10, СК11
Предуслови:
Постуслови:

Уговор УГ16: GetOrderItems

Операција: GetOrderItems(Order): сигнал
Веза са СК: СК10, СК11
Предуслови:
Постуслови:

Уговор УГ17: Update

Операција: Update(Order): сигнал

Веза са СК: СК11

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

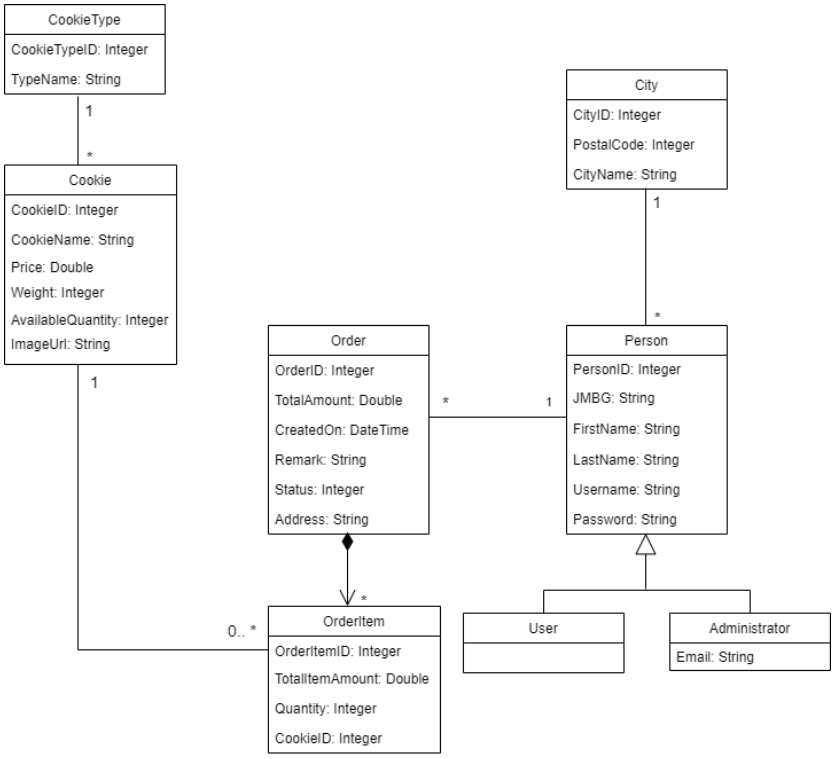
Постуслови: Наруџбеница је сачувана.

Уговор УГ18: GetPerson

Операција: GetPerson(Person): сигнал
Веза са СК: СК3
Предуслови:
Постуслови:

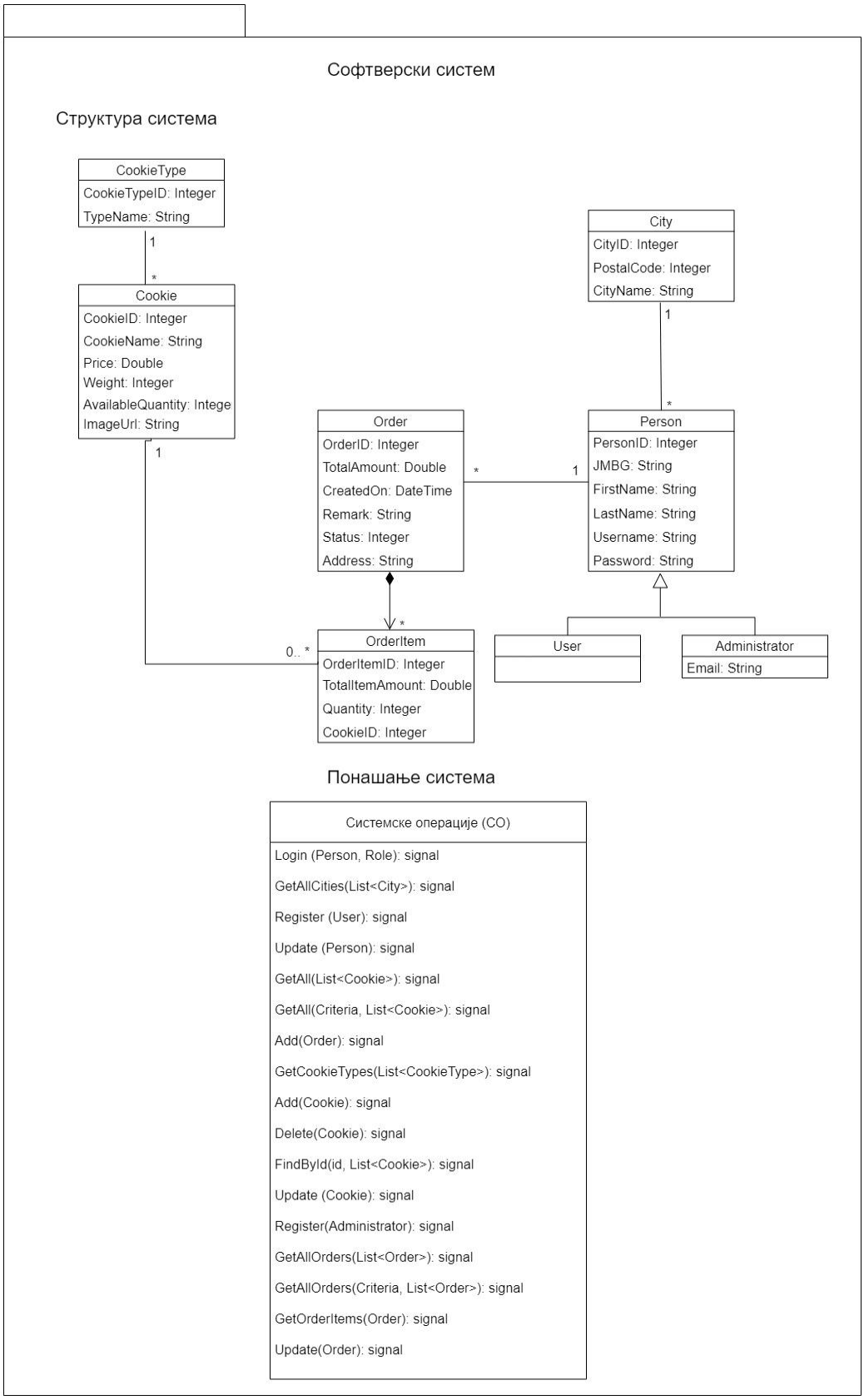
5.2.3 Структура софтверског система – Концептуални (доменски) модел

Концептуални модел садржи доменске објекте и везе између њих. Свака концептуална класа има своје атрибуте који је описују. Сваки од њих се везује за одређени тип података. За конкретно појављивање концептуалне класе сваки атрибут има конкретну вредност. [16]



Слика 46 Концептуални модел

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



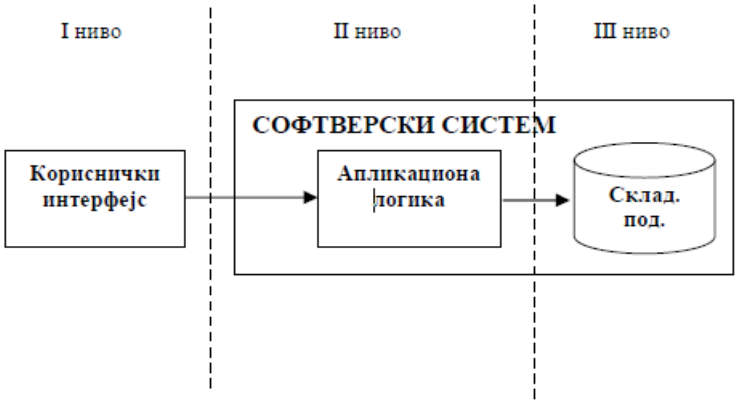
Слика 47 Логичка структура и понашање софтверског система

5.3 Фаза пројектовања

Фаза пројектовања описује физичку структуру и понашање софтверског система (архитектуру софтверског система). Пројектовање архитектуре софтверског система обухвата пројектовање корисничког интерфејса, апликационе логике и складишта података. Пројектовање корисничког интерфејса обухвата пројектовање екранских форми и контролера корисничког интерфејса. Пројектовање апликационе логике се односи на пројектовање контролера апликационе логике, пословне логике и брокера базе података. Пословна логика обухвата пројектовање логичке структуре и понашања софтверског система. [16]

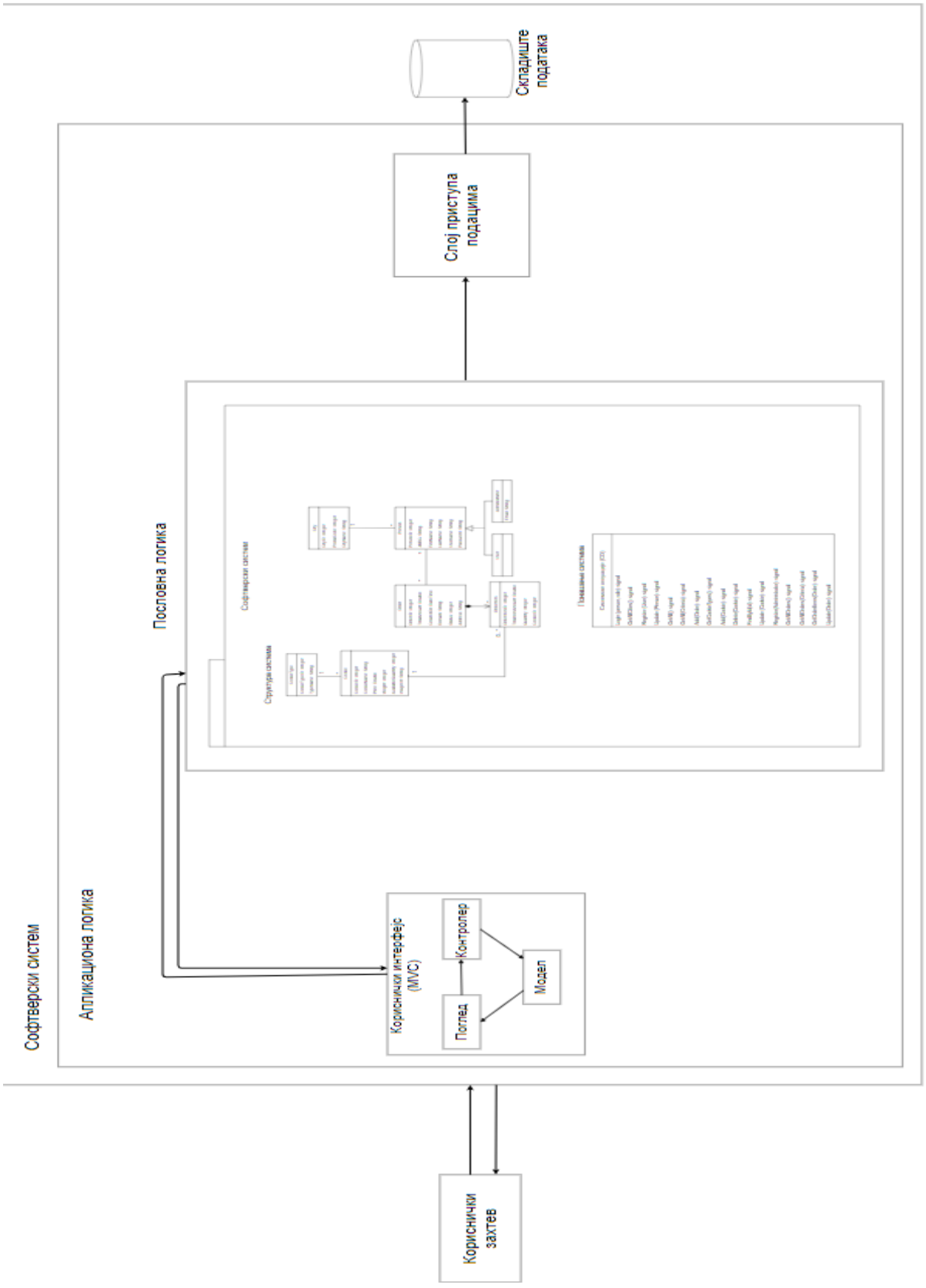
5.3.1 Архитектура софтверског система

Архитектура софтверског система је тронивојска. Њена три нивоа су приказана на следећој слици:



Слика 48 Тронивојска архитектура софтверског система [16]

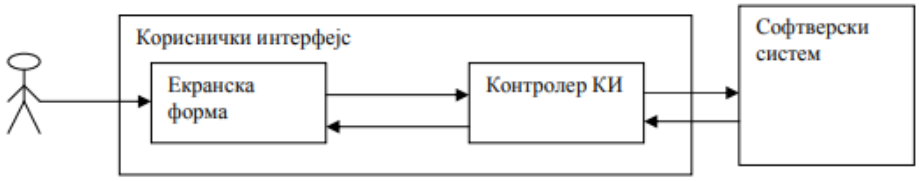
Кориснички интерфејс представља део презентационог слоја. Апликациона логика се састоји од пословне логике и слоја приступа подацима који је пројектован помоћу Entity Framework Core-а, а складиште података чини релациона база података са својим табелама.



Слика 49 Архитектура софтверског система

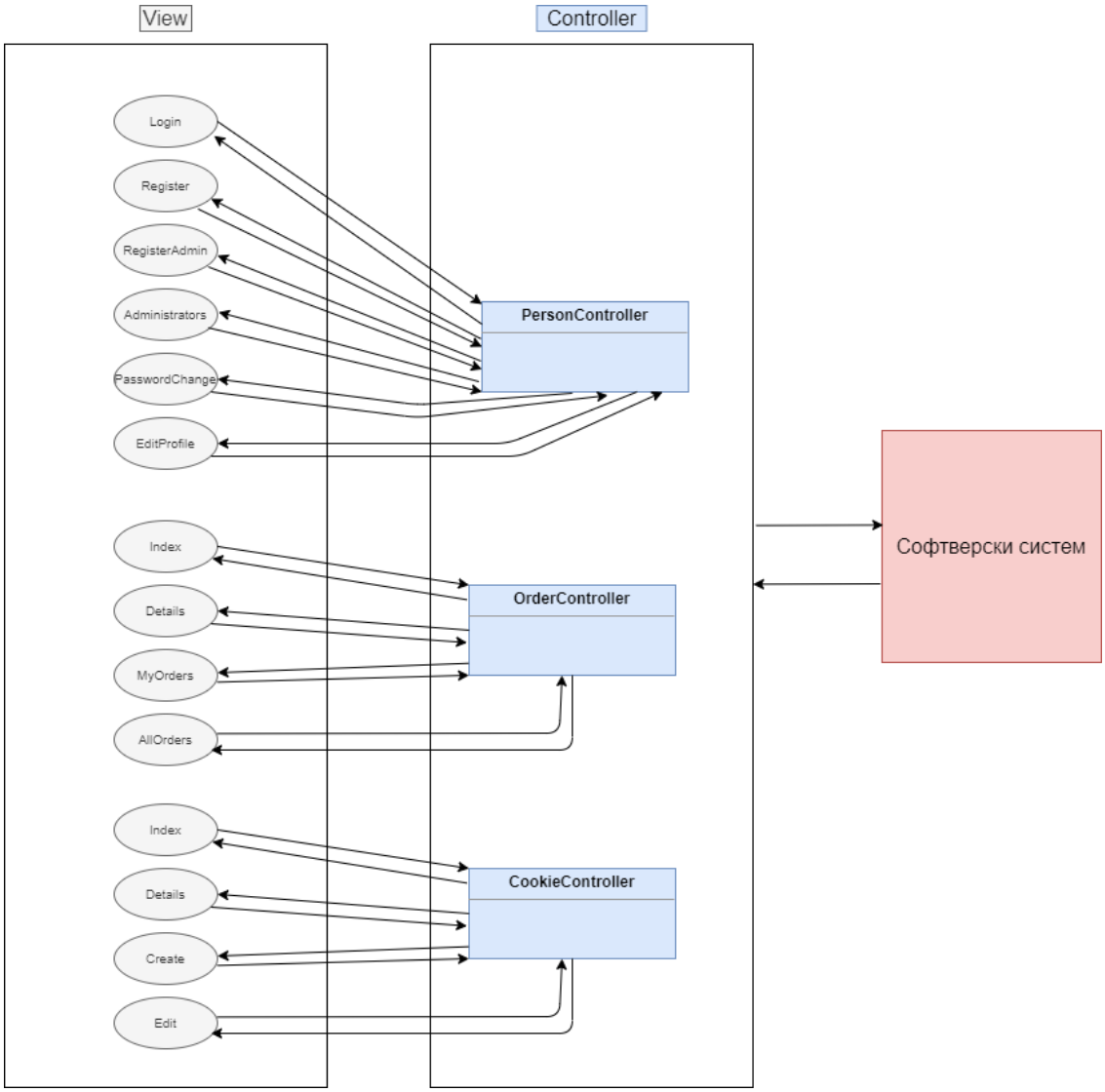
5.3.2 Пројектовање екранских форми

Кориснички интерфејс је начин на који програм комуницира са корисником. Представља реализацију улаза и/или излаза софтверског система. Састоји се од екранских форми и контролера корисничког интерфејса. [16]



Слика 50 Структура корисничког интерфејса [16]

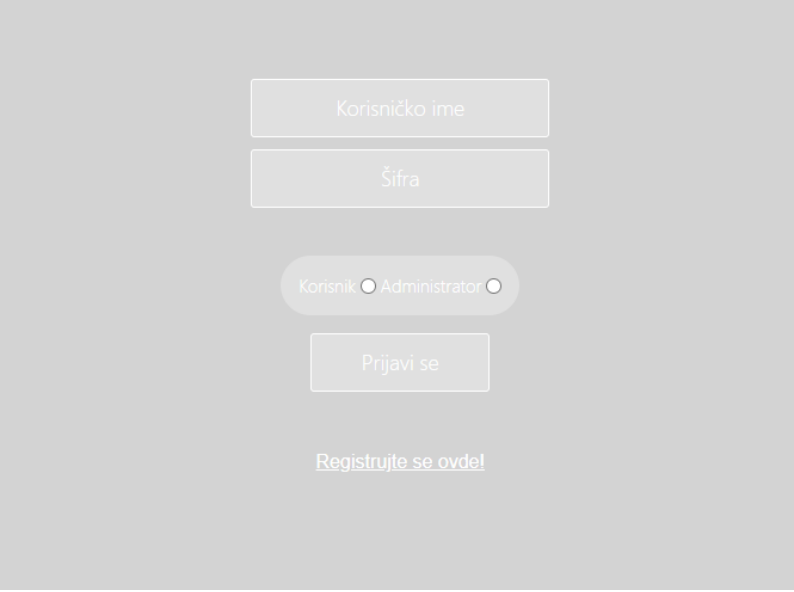
На слици 51 је приказан начин комуникације између контролера и корисничког интерфејса:



Слика 51 Комуникација између контролера и корисничког интерфејса

СК1: Случај коришћења – Пријава на систем

Назив СК
Пријава на систем
Актори СК
Корисник/Администратор
Учесници СК
Корисник/Администратор и систем (програм)
Предуслов: Систем је укључен и приказана је форма за пријаву.



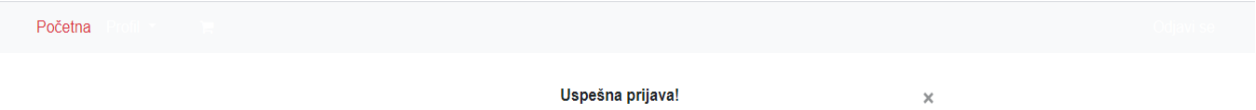
Слика 52 Пројектовање форме за пријаву на систем

Основни сценарио СК

1. Корисник/Администратор **уноси** своје податке за пријављивање. (АПУСО)
2. Корисник/Администратор **проверава** да ли је коректно унео податке. (АНСО)
3.Корисник/Администратор **позива** систем да нађе пријављеног корисника/администратора. (АПСО)

Опис акције: Кликом на дугме „*Prijavi se*“ корисник/администратор позива системску операцију Login(Person, Role)

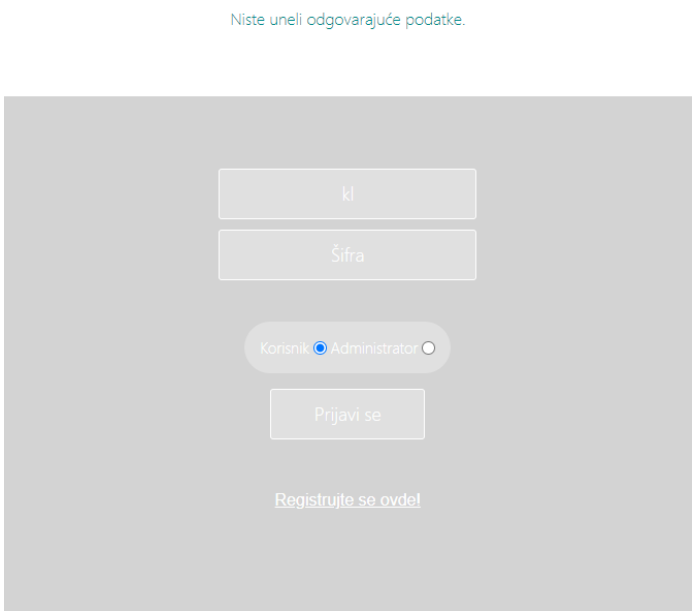
4. Систем **тражи** пријављеног корисника/администратора. (СО)
5. Систем **приказује** кориснику/администратору поруку: „Успешна пријава”. (ИА)



Слика 53 Успешно пријављивање – фаза пројектовања

Алтернативна сценарија

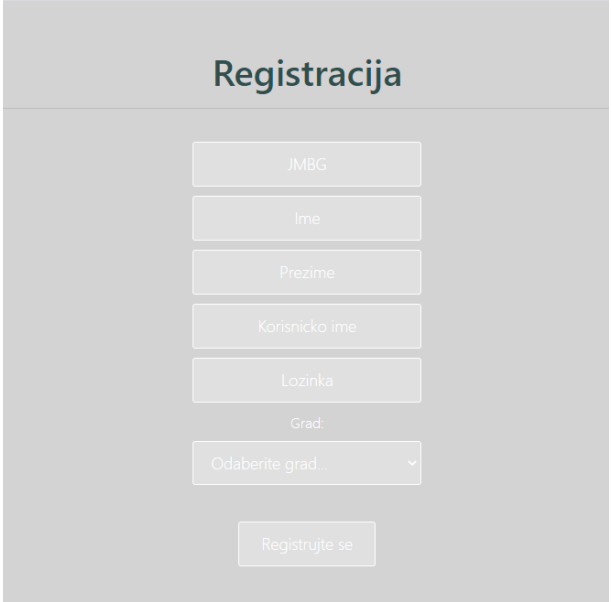
5.1 Уколико систем није успео да нађе пријављеног корисника/администратора, он **приказује** поруку: „Нисте унели одговарајуће податке“. (ИА)



Слика 54 Неуспешно пријављивање на систем – фаза пројектовања

СК2: Случај коришћења – Креирање корисничког налога

Назив СК
Креирање корисничког налога
Актори СК
Корисник
Учесници СК
Корисник и систем (програм)
Предуслов: Систем је укључен и приказана је форма за регистрацију. Учитана је листа свих градова.



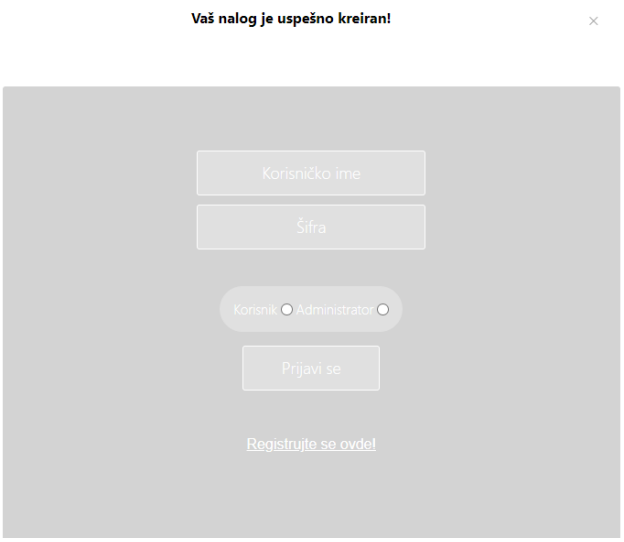
Слика 55 Пројектовање форме за регистрацију

Основни сценарио СК

1. Корисник **уноси** основне податке за креирање корисничког налога. (АПУСО)
2. Корисник **проверава** да ли је коректно унео своје податке. (АНСО)
3. Корисник **позива** систем да креира кориснички налог. (АПСО)

Опис акције: Кликом на дугме „*Registrujte se*“ корисник позива системску операцију Register(User).

4. Систем **креира** кориснички налог. (СО)
5. Систем **приказује** кориснику поруку: „Ваш налог је успешно креиран”. (ИА)



Слика 56 Успешно креирање налога – фаза пројектовања

Алтернативна сценарија

5.4 Уколико регистрација није могућа, систем **приказује** кориснику поруку: „Жао нам је, регистрација није успела“. (ИА)

Слика 57 Неуспешна регистрација – фаза пројектовања

СК3: Случај коришћења – Измена корисничког налога

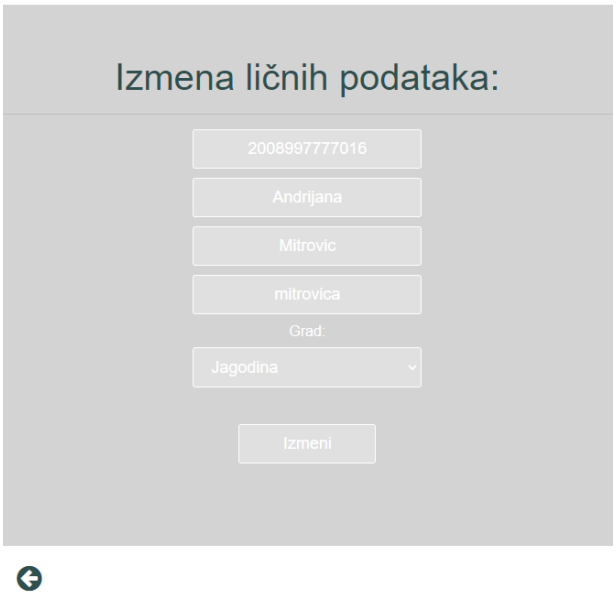
Назив СК

Измена корисничког налога

Актори СК

Корисник

Учесници СК
Корисник и систем (програм)
Предуслов: Систем је укључен и корисник је пријављен под својом шифром. Систем приказује податке о пријављеном кориснику. Учитани су подаци о корисничком налогу и листа свих градова.



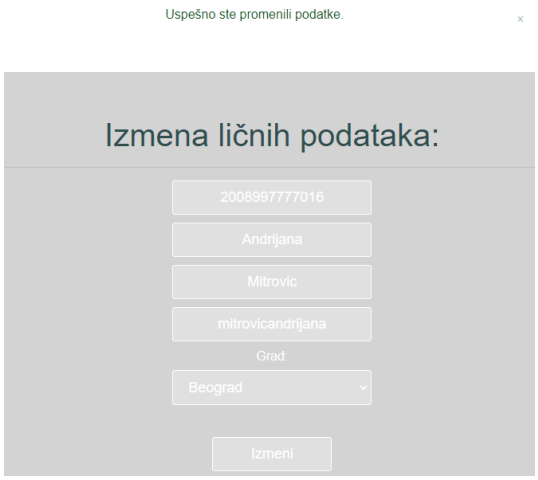
Слика 58 Пројектовање форме за измену корисничког налога

Основни сценарио СК

1. Систем **тражи** податке о кориснику. (СО)
2. Систем **приказује** кориснику пронађене податке. (ИА)
3. Корисник **бира** податке који жели да измени. (АПУСО)
4. Корисник **мења** податке. (АПУСО)
5. Корисник **проверава** да ли је коректно унео нове податке. (АНСО)
6. Корисник **позива** систем да промени податке. (АПСО)

Опис акције: Кликом на дугме „*Izmeni*“ корисник позива системску операцију Update(Person).

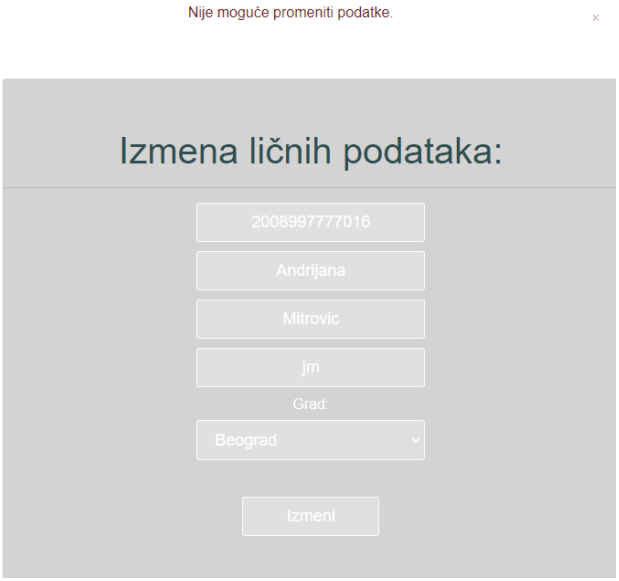
7. Систем **мења** податке о кориснику. (СО)
8. Систем **приказује** кориснику поруку: „Успешно измењени подаци!”. (ИА)



Слика 59 Успешна измена корисничког профила – фаза пројектовања

Алтернативна сценарија

8.1 Уколико систем није у могућности да промени податке, **приказује** поруку: „Није могуће променити податке.“ (ИА)

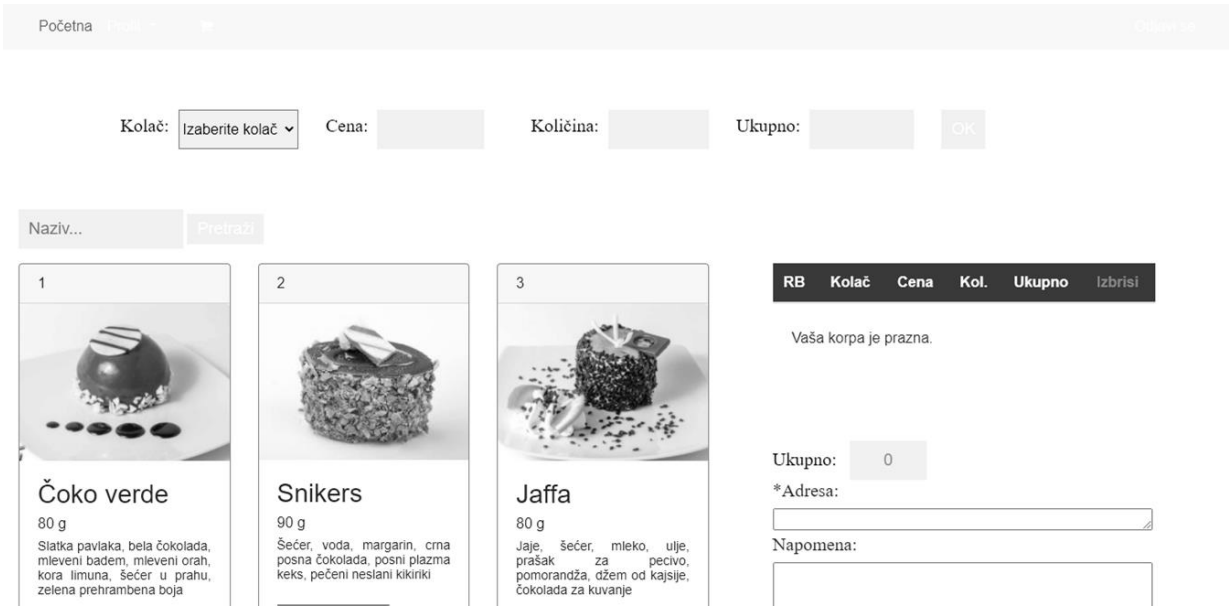


Слика 60 Неуспешна измена корисничког профила – фаза пројектовања

СК4: Случај коришћења – Претраживање колача

Назив СК
Претраживање колача
Актори СК
Корисник
Учесници СК

Корисник и систем (програм)
Предуслов: Систем је укључен и корисник је пријављен под својом шифром. Систем приказује интерфејс за пријављене кориснике. Учитана је листа свих колача.



Слика 61 Пројектовање форме за пријављене кориснике

Основни сценарио СК

1. Корисник **уноси** вредност по којој претражује колаче. (АПУСО)
2. Корисник **проверава** да ли је коректно унео вредност по којој претражује. (АНСО)
3. Корисник **позива** систем да нађе колаче на основу унете вредности. (АПСО)

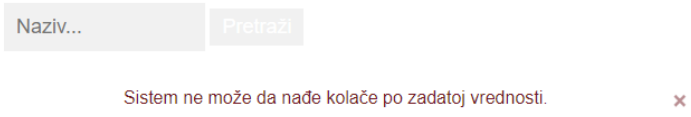
Опис акције: Кликом на дугме „*Pretraži*“ корисник позива системску операцију GetAll(Criteria, List<Cookie>).

4. Систем **тражи** колаче по наведеном критеријуму. (СО)
5. Систем **приказује** листу тражених колача. (ИА)

Слика 62 Листа тражених колача – фаза пројектовања

Алтернативна сценарија

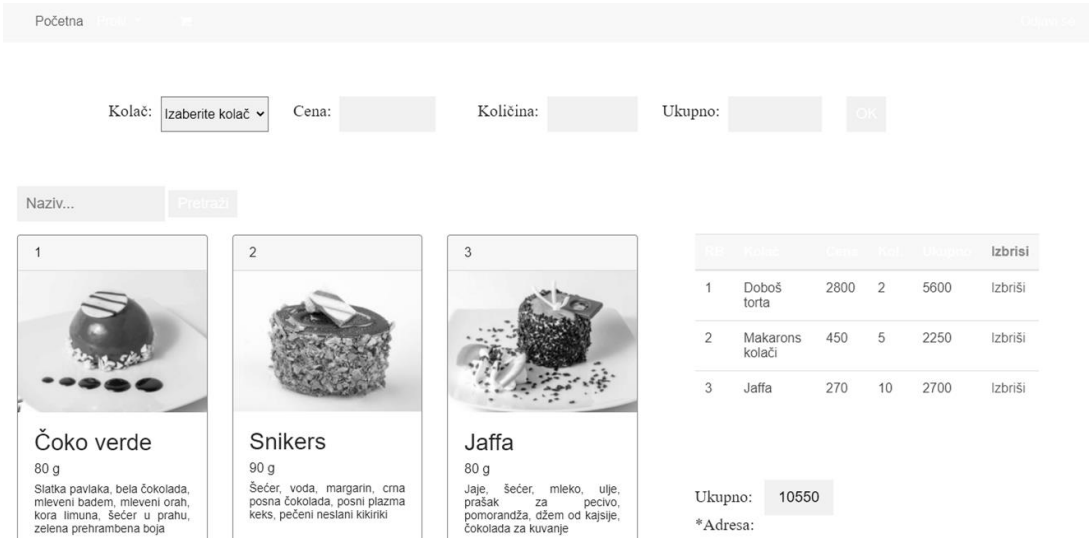
5.1 Уколико систем не може да нађе колаче по унетом критеријуму он **приказује** кориснику поруку: “Систем не може да нађе колаче по задатој вредности”. (ИА)



Слика 63 Неуспешна претрага колача – фаза пројектовања

СК5: Случај коришћења – Унос наруџбенице

Назив СК
Унос наруџбенице
Актори СК
Корисник
Учесници СК
Корисник и систем (програм)
Предуслов: Систем је укључен и кориснику, који је улогован под својом шифром, приказана је листа одабраних колача. Систем приказује форму за наручивање. Учитани су подаци о колачима.



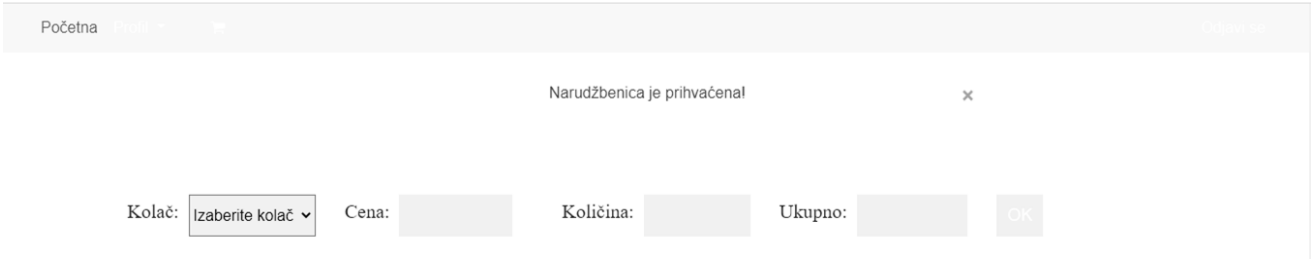
Слика 64 Пројектовање форме за наручивање

Основни сценарио СК

1. Корисник **уноси** податке о наруџбеници. (АПУСО)
2. Корисник **проверава** да ли је правилно унео податке о наруџбеници. (АНСО)
3. Корисник **позива** систем да сачува податке о наруџбеници. (АПСО)

Опис акције: Кликом на дугме *"Kreiraj narudžbenicu"* корисник позива системску операцију Add(Order).

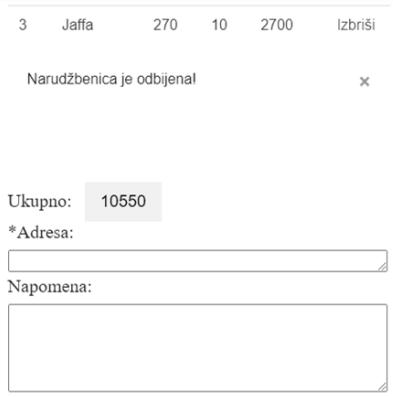
4. Систем **чува** податке о наруџбеници. (СО)
5. Систем **приказује** кориснику поруку: "Наруџбеница је прихваћена". (ИА)



Слика 65 Прихваћена наруџбеница – фаза пројектовања

Алтернативна сценарија

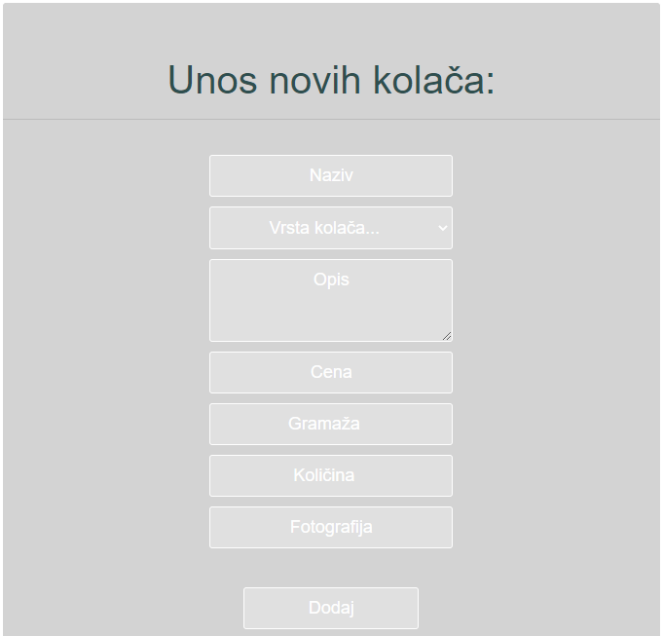
5.1 Уколико систем није у могућности да изврши унос наруџбенице, **приказује** поруку: „Наруџбеница је одбијена". (ИА)



Слика 66 Одбијена наруџбеница – фаза пројектовања

СК6: Случај коришћења – Унос нових колача

Назив СК
Унос нових колача
Актори СК
Администратор
Учесници СК
Администратор и систем (програм)
Предуслов: Систем је укључен и администратор је улогован под својом шифром. Систем приказује интерфејс за унос нових колача. Учитана је листа типова колача.



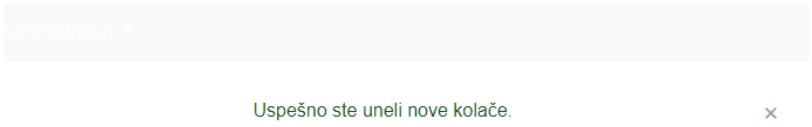
Слика 67 Пројектовање форме за унос нових колача

Основни сценарио СК

1. Администратор **уноси** податке о новим колачима. (АПУСО)
2. Администратор **контролише** да ли је коректно унео податке о колачима. (АНСО)
3. Администратор **позива** систем да запамти податке о колачима. (АПСО)

Опис акције: Кликом на дугме *"Dodaj"* администратор позива системску операцију Add(Cookie).

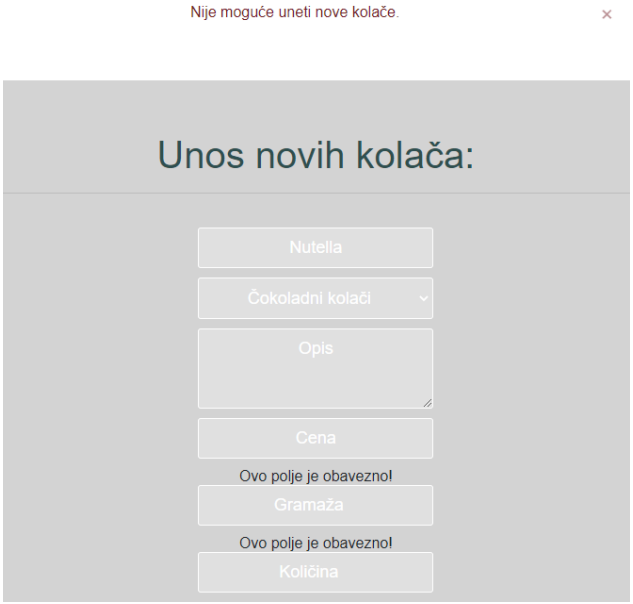
4. Систем **памти** податке о колачима. (СО)
5. Систем **приказује** администратору запамћене податке о колачима и поруку: „Успешно сте унели нове колаче". (ИА)



Слика 68 Успешан унос нових колача – фаза пројектовања

Алтернативна сценарија

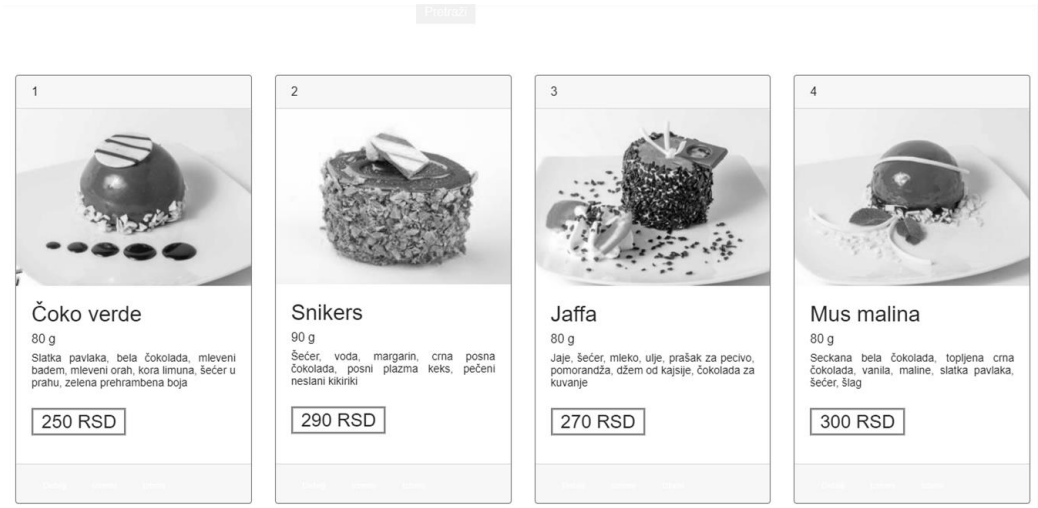
5.1 Уколико систем не може да прихвати податке о новим колачима, он **приказује** администратору поруку „Није могуће унети нове колаче". (ИА)



Слика 69 Неуспешан унос нових колача – фаза пројектовања

СК7: Случај коришћења – Брисање колача

Назив СК
Брисање колача
Актори СК
Администратор
Учесници СК
Администратор и систем (програм)
Предуслов: Систем је укључен и администратор је пријављен под својом шифром. Систем приказује интерфејс за брисање колача. Учитана је листа свих колача.



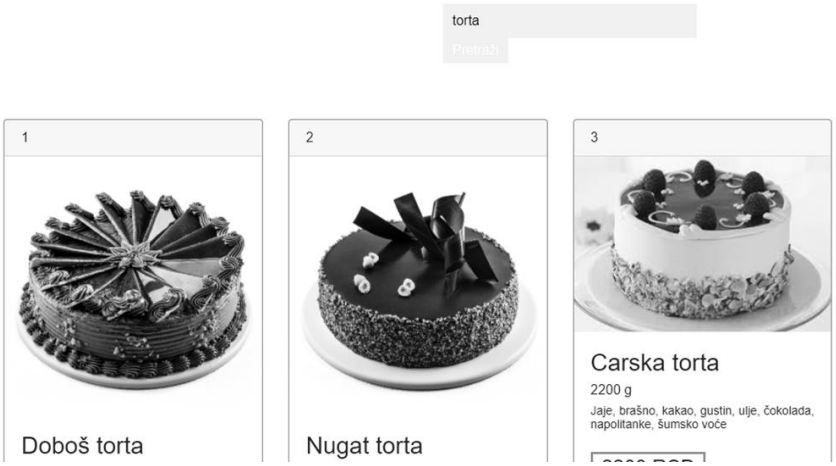
Слика 70 Пројектовање интерфејса за брисање колача

Основни сценарио СК

1. Администратор **уноси** вредност по којој претражује колаче. (АПУСО)
2. Администратор **проверава** да ли је коректно унео вредност по којој претражује. (АНСО)
3. Администратор **позива** систем да нађе колаче на основу унете вредности. (АПСО)

Опис акције: Кликом на дугме *"Pretraži"* администратор позива системску операцију GetAll(Criteria, List<Cookie>).

4. Систем **тражи** колаче по наведеном критеријуму. (СО)
5. Систем **приказује** листу тражених колача. (ИА)



Слика 71 Приказ листе тражених колача – фаза пројектовања

6. Администратор **бира** колаче који жели да избрише. (АПУСО)
7. Администратор **позива** систем да избрише податке о одабраним колачима. (АПСО)

Опис акције: Кликом на дугме „*Izbriši*“ администратор позива системску операцију Delete(Cookie).

8. Систем **брише** податке о колачима. (СО)
9. Систем **приказује** администратору листу постојећих производа и поруку: „Успешно избрисани колачи“. (ИА)

Слика 72 Успешно брисање колача – фаза пројектовања

Алтернативна сценарија

5.1 Уколико систем не може да нађе колаче по унетом критеријуму он **приказује** кориснику поруку: “Систем не може да нађе колаче по задатој вредности.”. Прекида се извршење сценарија. (ИА)

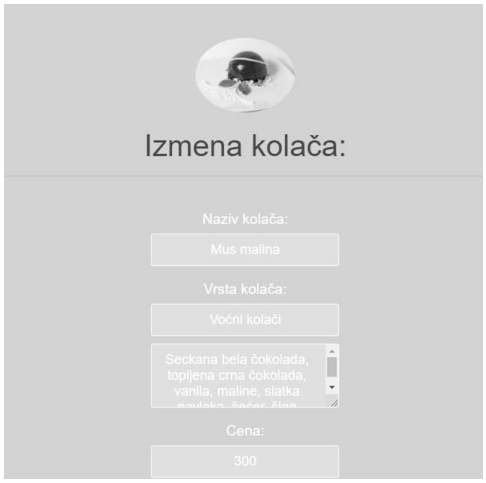
Слика 73 Неуспешно претраживање колача – фаза пројектовања

9.1 Уколико систем није успешно избрисао тражене колаче, он **приказује** поруку кориснику: „Систем не може да избрише тражене колаче“. (ИА)

Слика 74 Неуспешно брисање колача – фаза пројектовања

СК8: Случај коришћења – Измена колача

Назив СК
Измена колача
Актори СК
Администратор
Учесници СК
Администратор и систем (програм)
Предуслов: Систем је укључен и администратор је пријављен под својом шифром. Систем приказује интерфејс за измену колача.



Слика 75 Пројектовање форме за измену података о колачима

Основни сценарио СК

1. Систем **тражи** податке о колачима. (СО)
2. Систем **приказује** администратору пронађене податке. (ИА)
3. Администратор **бира** податке који жели да измени. (АПУСО)
4. Администратор **мења** податке. (АПУСО)
5. Администратор **проверава** да ли је коректно унео нове податке. (АНСО)
6. Администратор **позива** систем да промени податке. (АПСО)

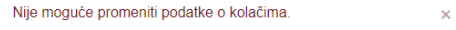
Опис акције: Кликом на дугме „*Izmeni podatke*“ администратор позива системску операцију Update(Cookie).

7. Систем **мења** податке о колачима. (СО)
8. Систем **приказује** администратору поруку: „Успешно измењени подаци о колачима!”. (ИА)

Слика 76 Успешна измена колача – фаза пројектовања

Алтернативна сценарија

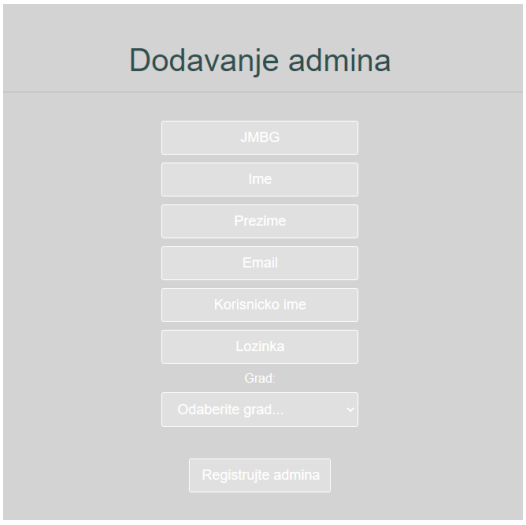
8.1 Уколико систем није у могућности да промени податке, **приказује** поруку: „Није могуће променити податке о колачима.“ (ИА)



Слика 77 Неуспешна промена података о колачима – фаза пројектовања

СК9: Случај коришћења – Унос новог администратора

Назив СК
Унос новог администратора
Актори СК
Администратор
Учесници СК
Администратор и систем (програм)
Предуслов: Систем је укључен и приказана је форма за унос новог администратора. Учитана је листа свих градова.



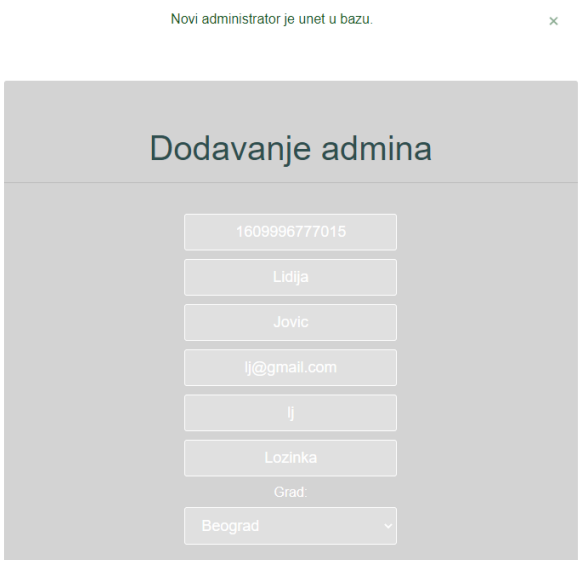
Слика 78 Пројектовање форме за унос новог администратора

Основни сценарио СК

1. Администратор **уноси** основне податке за новог администратора. (АПУСО)
2. Администратор **проверава** да ли је коректно унео податке за новог администратора. (АНСО)
3. Администратор **позива** систем да запамти новог администратора. (АПСО)

Опис акције: Кликом на дугме „*Registrujte admina*“ администратор позива системску операцију Register(Administrator).

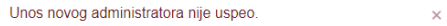
4. Систем **памти** новог администратора. (СО)
5. Систем **приказује** администратору поруку: „Нови администратор је унет у базу!”. (ИА)



Слика 79 Успешан унос новог администартора – фаза пројектовања

Алтернативна сценарија

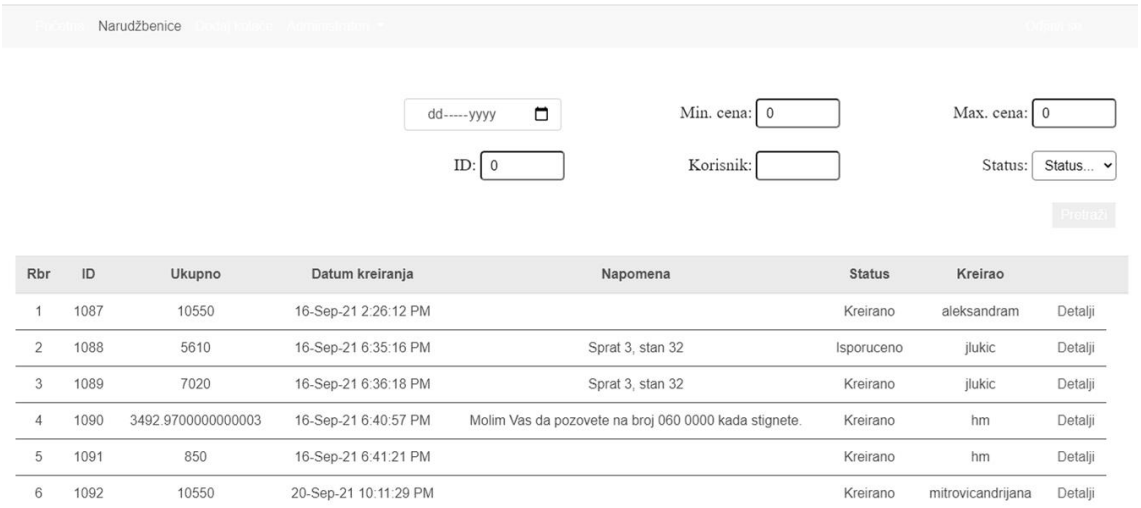
5.1 Уколико унос новог администратора није могућ, систем **приказује** администратору поруку: „Унос новог администратора није успео!“. (ИА)



Слика 80 Неуспешан унос новог администратора – фаза пројектовања

СК10: Случај коришћења – Претрага наруџбеница

Назив СК
Претрага наруџбеница
Актори СК
Администратор
Учесници СК
Администратор и систем (програм)
Предуслов: Систем је укључен и администратор је пријављен под својом шифром. Систем приказује интерфејс за претрагу наруџбеница. Учитана је листа свих наруџбеница.



Rbr	ID	Ukupno	Datum kreiranja	Napomena	Status	Kreirao	
1	1087	10550	16-Sep-21 2:26:12 PM		Kreirano	aleksandram	Detalji
2	1088	5610	16-Sep-21 6:35:16 PM	Sprat 3, stan 32	Isporuceno	jlukic	Detalji
3	1089	7020	16-Sep-21 6:36:18 PM	Sprat 3, stan 32	Kreirano	jlukic	Detalji
4	1090	3492.9700000000003	16-Sep-21 6:40:57 PM	Molim Vas da pozovete na broj 060 0000 kada stignete.	Kreirano	hm	Detalji
5	1091	850	16-Sep-21 6:41:21 PM		Kreirano	hm	Detalji
6	1092	10550	20-Sep-21 10:11:29 PM		Kreirano	mitrovicandrijana	Detalji

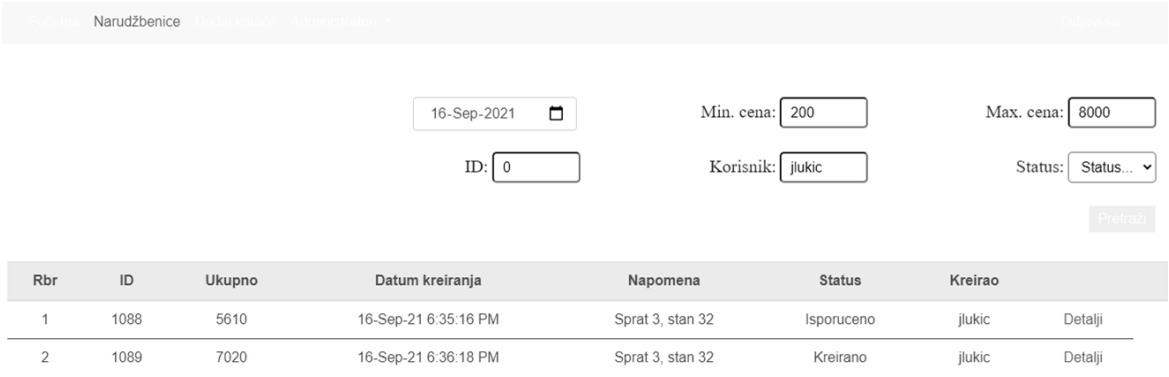
Слика 81 Пројектовање форме за претрагу наруџбеница

Основни сценарио СК

1. Администратор **уноси** критеријум по ком претражује наруџбенице. (АПУСО)
2. Администратор **проверава** да ли је коректно унео критеријум по ком претражује. (АНСО)
3. Администратор **позива** систем да нађе наруџбенице на основу унетог критеријума. (АПСО)

Опис акције: Кликом на дугме „*Pretraži*“ администратор позива системску операцију GetAll(Criteria, List<Order>).

4. Систем **тражи** наруџбенице по наведеном критеријуму. (СО)
5. Систем **приказује** листу одговарајућих наруџбеница. (ИА)



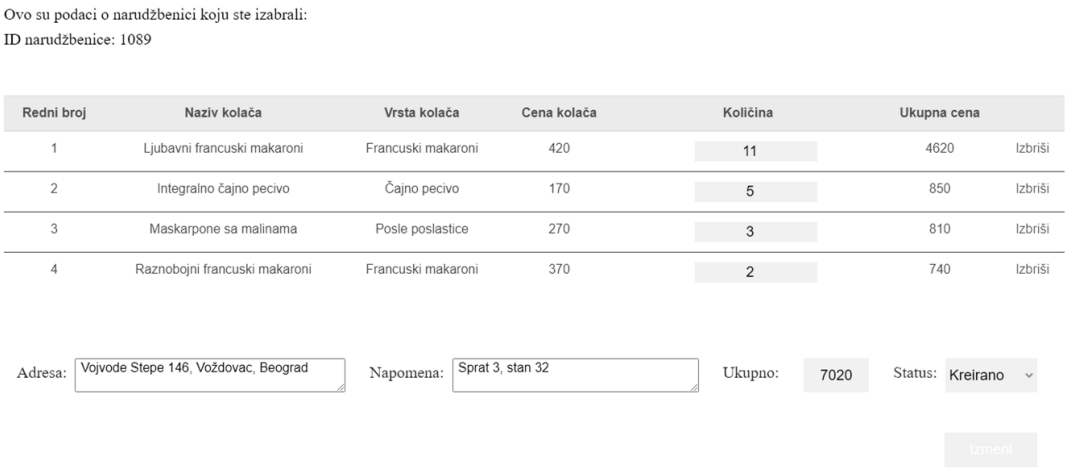
Rbr	ID	Ukupno	Datum kreiranja	Napomena	Status	Kreirao	
1	1088	5610	16-Sep-21 6:35:16 PM	Sprat 3, stan 32	Isporuceno	jlukic	Detalji
2	1089	7020	16-Sep-21 6:36:18 PM	Sprat 3, stan 32	Kreirano	jlukic	Detalji

Слика 82 Листа тражених наруџбеница

6. Администратор **бира** наруџбеницу. (АПУСО)
7. Администратор **позива** систем да прикаже све податке о изабраној наруџбеници. (АПСО)

Опис акције: Кликом на дугме „*Detalji*“ администратор позива системску операцију GetOrderItems(Order).

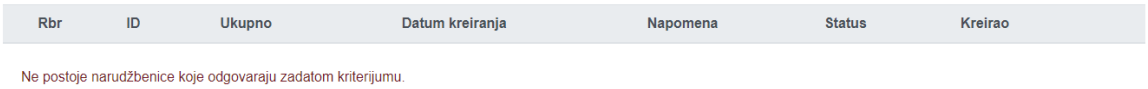
8. Систем **тражи** податке о наруџбеници. (СО)
9. Систем **приказује** кориснику све податке о наруџбеници уз поруку: “Ово су подаци о наруџбеници коју сте изабрали”. (ИА)



Слика 83 Успешно приказани подаци о наруџбеници – фаза пројектовања

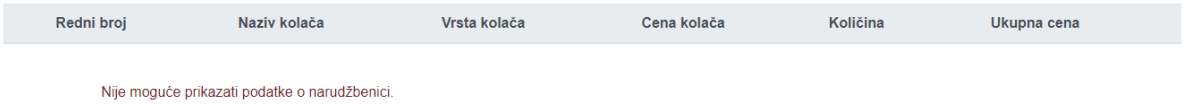
Алтернативна сценарија

5.1 Уколико систем не може да нађе наруџбенице по унетом критеријуму он **приказује** адмнистратору поруку: “Не постоје наруџбенице које одговарају задатом критеријуму.“ Прекида се извршење сценарија. (ИА)



Слика 84 Неуспешна претрага наруџбеница – фаза пројектовања

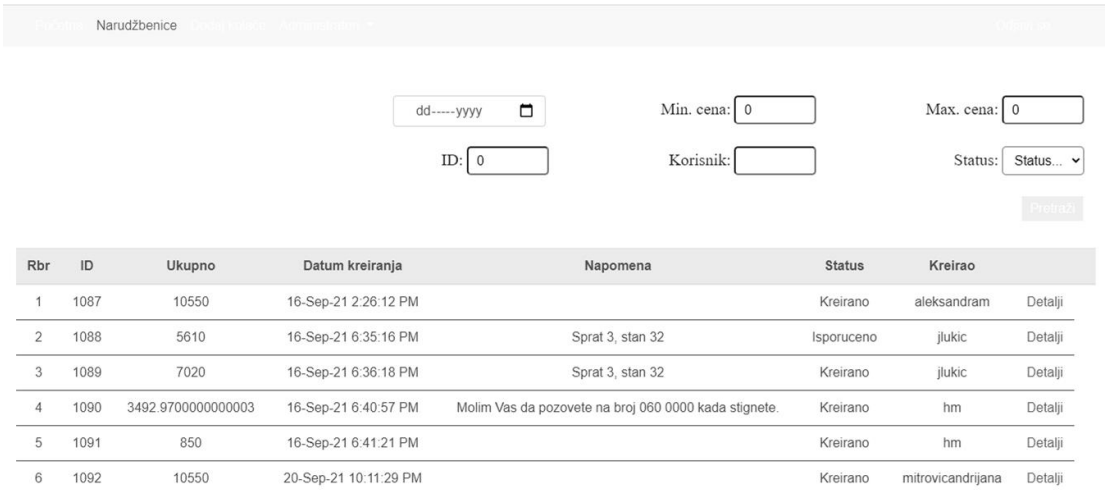
9.1 Уколико систем не може да прикаже податке о одабраној наруџбеници, он **приказује** администратору поруку: „Није могуће приказати податке о наруџбеници.“.



Слика 85 Неуспешно приказивање података о наруџбеници – фаза пројектовања

СК11: Случај коришћења – Измена наруџбенице

Назив СК
Измена наруџбенице
Актори СК
Администратор
Учесници СК
Администратор и систем (програм)
Предуслов: Систем је укључен и администратор је пријављен под својом шифром. Систем приказује интерфејс за измену наруџбеница. Учитана је листа свих наруџбеница.



Rbr	ID	Ukupno	Datum kreiranja	Napomena	Status	Kreirao	
1	1087	10550	16-Sep-21 2:26:12 PM		Kreirano	aleksandram	Detalji
2	1088	5610	16-Sep-21 6:35:16 PM	Sprat 3, stan 32	Isporuceno	jlukic	Detalji
3	1089	7020	16-Sep-21 6:36:18 PM	Sprat 3, stan 32	Kreirano	jlukic	Detalji
4	1090	3492.9700000000003	16-Sep-21 6:40:57 PM	Molim Vas da pozovete na broj 060 0000 kada stignete.	Kreirano	hm	Detalji
5	1091	850	16-Sep-21 6:41:21 PM		Kreirano	hm	Detalji
6	1092	10550	20-Sep-21 10:11:29 PM		Kreirano	mitrovicandrijana	Detalji

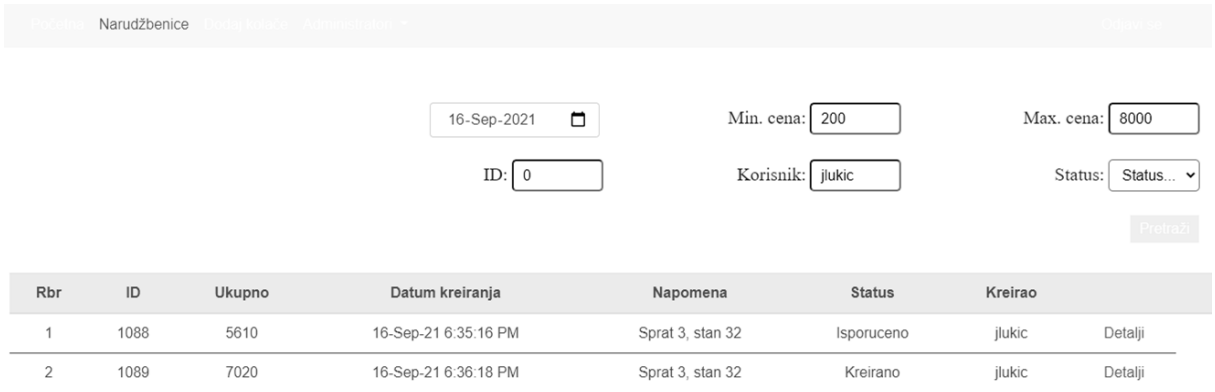
Слика 86 Листа наруџбеница – фаза пројектовања

Основни сценарио СК

1. Администратор **уноси** критеријум по ком претражује наруџбенице. (АПУСО)
2. Администратор **проверава** да ли је коректно унео критеријум по ком претражује. (АНСО)
3. Администратор **позива** систем да нађе наруџбенице на основу унетог критеријума. (АПСО)

Опис акције: Кликом на дугме „*Pretraži*“ администратор позива системску операцију GetAll(Criteria, List<Order>).

4. Систем **тражи** наруџбенице по наведеном критеријуму. (СО)
5. Систем **приказује** листу одговарајућих наруџбеница. (ИА)

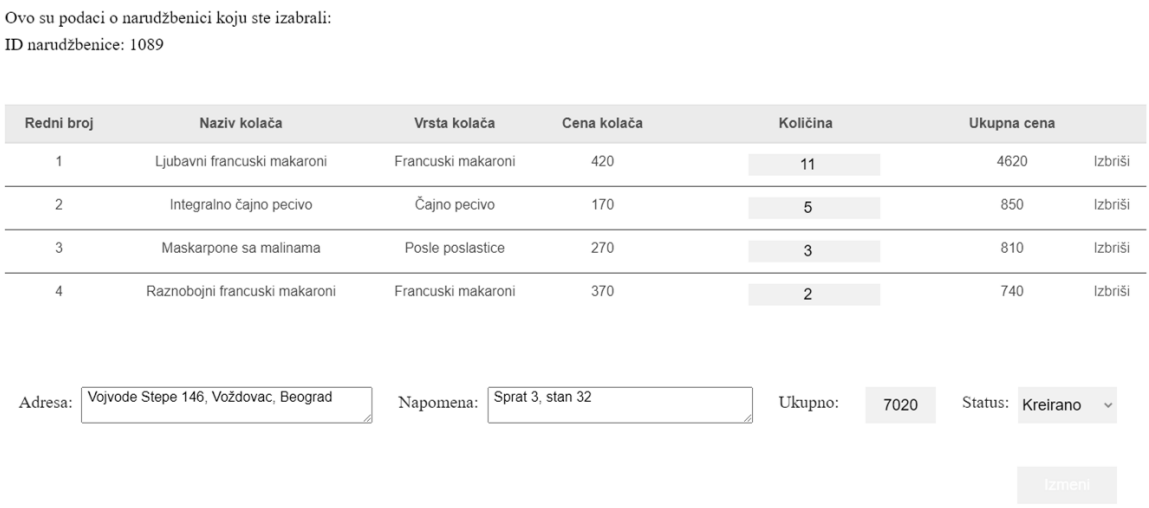


Слика 87 Листа тражених наруџбеница – фаза пројектовања

6. Администратор **бира** наруџбеницу. (АПУСО)
7. Администратор **позива** систем да прикаже све податке о изабраној наруџбеници. (АПСО)

Опис акције: Кликом на дугме „*Detalji*“ администратор позива системску операцију GetOrderItems(Order).

8. Систем **тражи** податке о наруџбеници. (СО)
9. Систем **приказује** кориснику све податке о наруџбеници уз поруку: “Ово су подаци о наруџбеници коју сте изабрали”. (ИА)

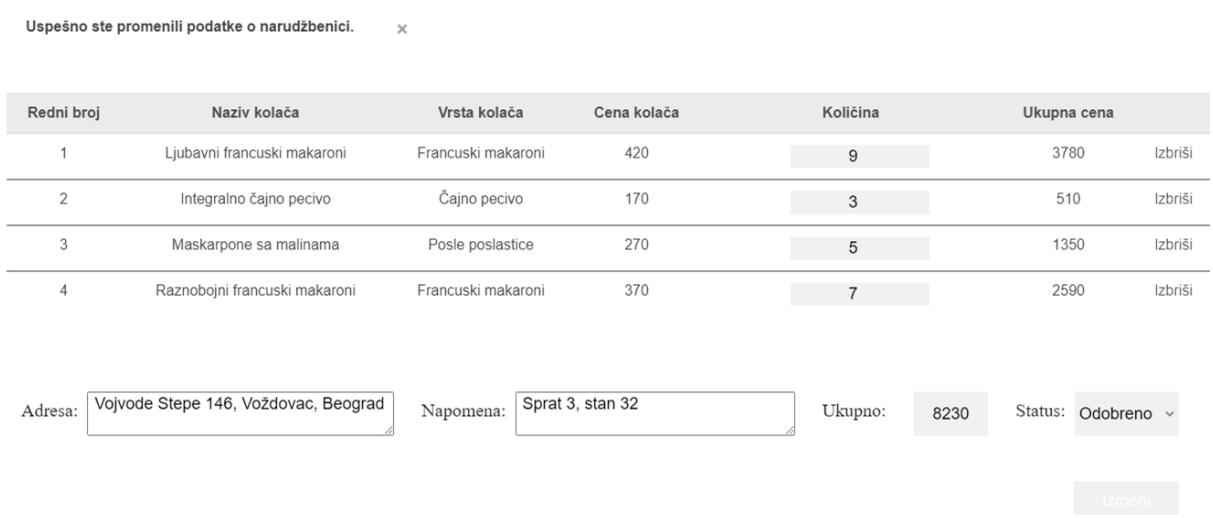


Слика 88 Успешно приказани подаци о наруџбеници – фаза пројектовања

10. Администратор **мења** наруџбеницу. (АПУСО)
11. Администратор **проверава** да ли је коректно унео нове податке о наруџбеници. (АНСО)
12. Администратор **позива** систем да промени податке. (АПСО)

Опис акције: Кликом на дугме „*Izmeni*“ администратор позива системску операцију Update(Order).

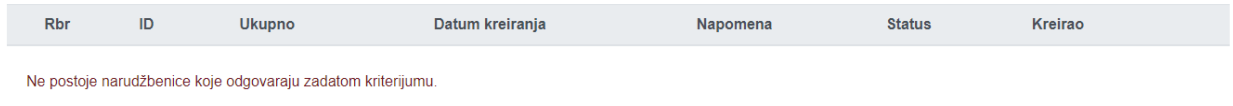
13. Систем **мења** податке о наруџбеници. (СО)
14. Систем **приказује** администратору поруку: „Успешно сте променили податке о наруџбеници!”. (ИА)



Слика 89 Успешно измењена наруџбеница – фаза пројектовања

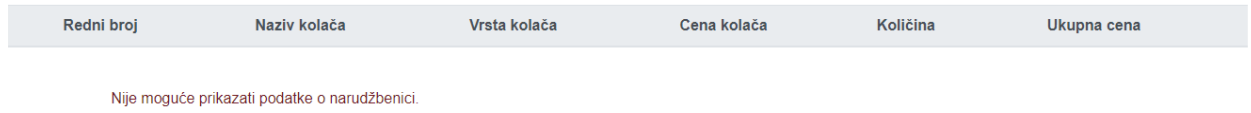
Алтернативна сценарија

5.1 Уколико систем не може да нађе наруџбенице по унетом критеријуму он **приказује** адмнистратору поруку: “Не постоје наруџбенице које одговарају задатом критеријуму”. Прекида се извршење сценарија. (ИА)



Слика 90 Неуспешно претраживање наруџбеница – фаза пројектовања

9.1 Уколико систем не може да прикаже податке о одабраној наруџбеници, он **приказује** администратору поруку: „Није могуће приказати податке о наруџбеници“. Прекида се извршење сценарија. (ИА)



Слика 91 Неуспешно приказивање података о наруџбеници – фаза пројектовања

14.1. Уколико систем није у могућности да промени податке о одабраној наруџбеници, он **приказује** администратору поруку: “Није могуће променити податке о наруџбеници”. (ИА)

Слика 92 Неуспешна измена података о наруџбеници – фаза пројектовања

5.3.3 Пројектовање апликационе логике

Апликациона логика се налази одмах испод слоја корисничког интерфејса. Она је задужена за операције над подацима и сам ток апликације. На овом слоју се контролишу права приступа и све пословне функције.

Апликациона логика садржи све класе које су потребне како би се успешно имплементирала пословна логика, а то су:

1) Контролери – представљају један од три слоја *MVC* (енг. *Model-View-Controller*) програмског модела. Њихова улога је да читају податке са форми и шаљу обрађене податке моделу. Дакле, они представљају везу између модела и корисничког интерфејса.
2) Сервиси – с обзиром на то да је апликација израђена коришћењем *Repository Design Pattern-a*, сервисни слој је задужен за комуникацију између контролера и *UnitOfWork-a*. У сервисима су садржане системске операције и они су задужени да трансформишу податке у онај облик који је погодан за комуникацију са базом.
3) UnitOfWork – ова класа је такође значајна за имплементацију *Repository Design Pattern-a* и првенствено треба да осигура да сви репозиторијуми који се користе у апликацији приступају јединственом *DbContext-y*, како би се избегла могућност настајања конфликата.
4) *Repository* – за сваки објекат креира се посебна *Repository* класа која је задужена за комуникацију са складиштем података. У њима се налазе *CRUD* операције које се извршавају над постојећим објектима. Сваки репозиторијум имплементира по један интерфејс, а сви ти интерфејси наслеђују генерички *IRepository* интерфејс у ком је дата спецификација потребних функција, које су заједничке за све.
5) *Entity* објекти – представљају табеле у бази података, које су настале кришћењем *Code-First*[4] приступа програмирању.

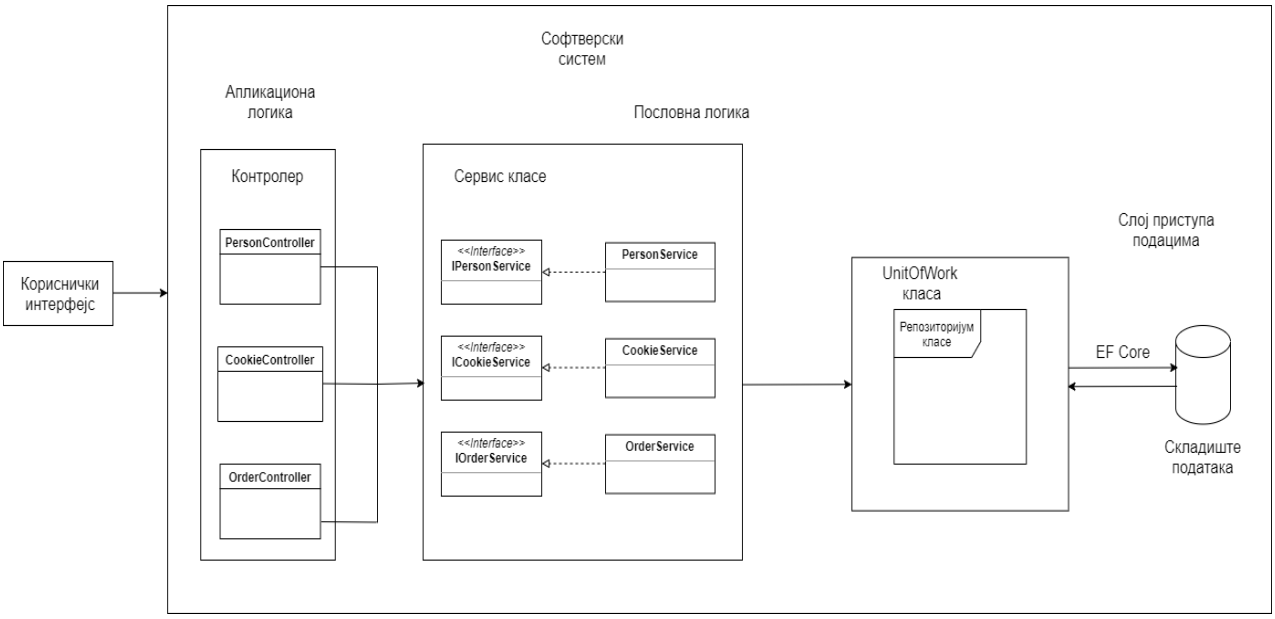
5.3.3.1 Пројектовање контролера апликационе логике

Контролер апликационе логике прихвата захтев за извршење системске операције од клијента и даље га преусмерава до класа које су одговорне за извшење систмских операција, у овом случају до сервиса. Након извршења системских операција, контролер апликационе логике прихвата резултат и прослеђује га оном ко га позива. [16]

У овом раду су пројектована три контролера (*OrderController*, *CookieController*, *PersonController*) и сваки је задужен за рад са одређеним објектима.

[4] Code-First приступ подразумева прво дефинисање доменских класа и објектно-релационо мапирање, на основу чега се коришћењем *Entity Framework-a* креирају одговарајуће табеле у бази података.

На следећој слици приказана је архитектура софтверског система након пројектовања контролера и класа које чине апликациону логику:

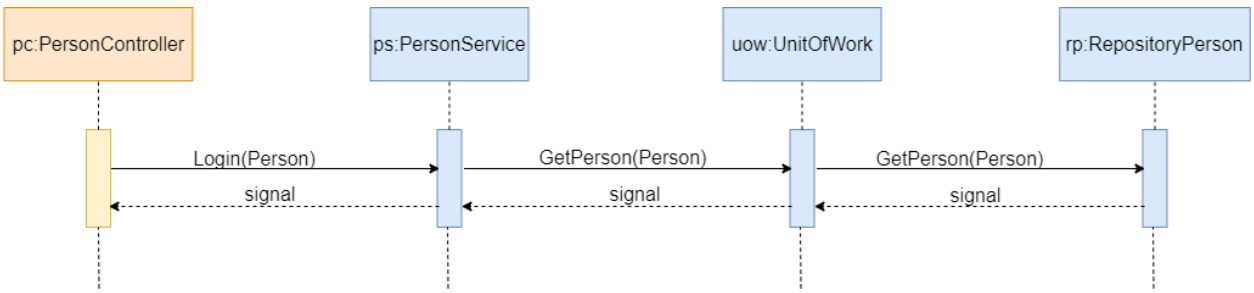


Слика 93 Архитектура софтверског система након пројектовања контролера и апликационе логике

5.3.3.2 Пословна логика

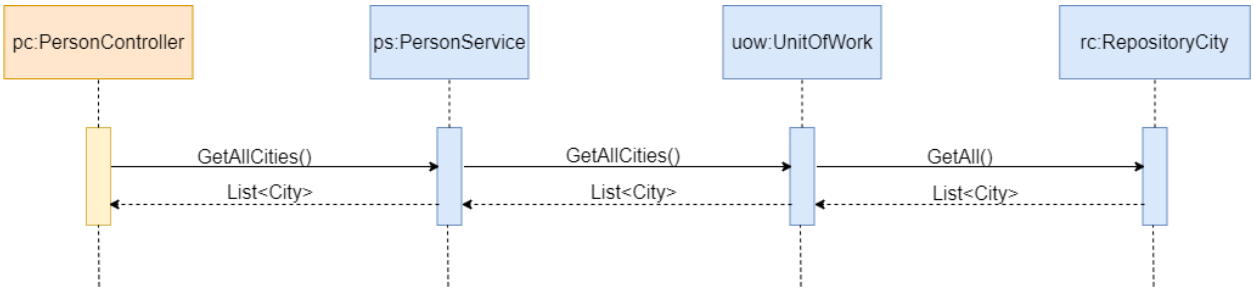
Пословна логика је описана структуром (доменским класама) и понашањем(системским операцијама). [16] У наставку је дата концептуална реализација уговора обрађених у фази анализе и то помоћу секвенцних дијаграма.

Уговор УГ1: Login
Операција: Login(Person, Role): сигнал
Веза са СК: СК1
Предуслови: Вредносна и структурна ограничења над објектом **Person** морају бити задовољена.
Постуслови: Корисник/Администратор је пријављен.



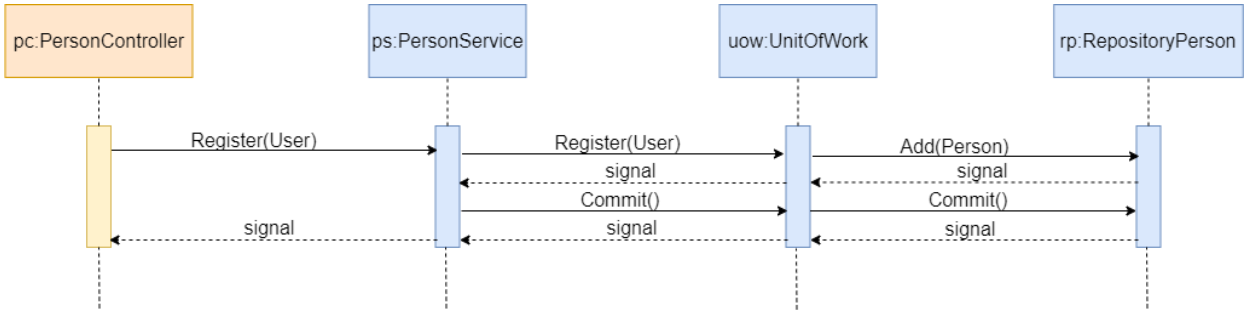
Слика 94 Дијаграм секвенци - УГ1

Уговор УГ2: GetAllCities
Операција: GetAllCities(List<City>): сигнал
Веза са СК: СК1, СК2, СК9
Предуслови:
Постуслови:



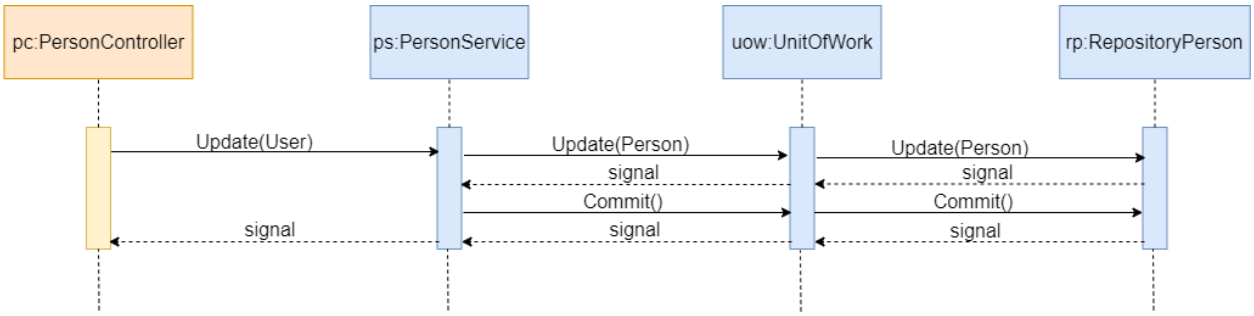
Слика 95 Дијаграм секвенци - УГ2

Уговор УГ3: Register
Операција: Register(User): сигнал
Веза са СК: СК2
Предуслови: Вредносна и структурна ограничења над објектом **User** морају бити задовољена.
Постуслови: Регистрован је нови корисник.



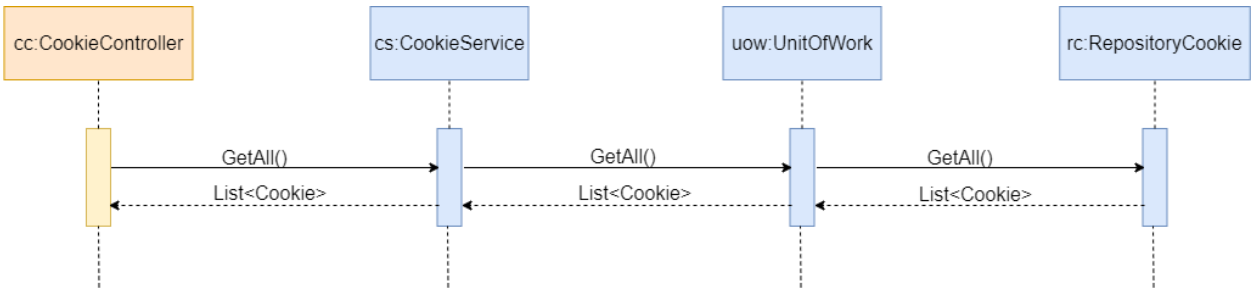
Слика 96 Дијаграм секвенци - УГ3

Уговор УГ4: Update
Операција: Update(Person): сигнал
Веза са СК: СК3
Предуслови: Вредносна и структурна ограничења над објектом **Person** морају бити задовољена.
Постуслови: Корисник је сачуван.



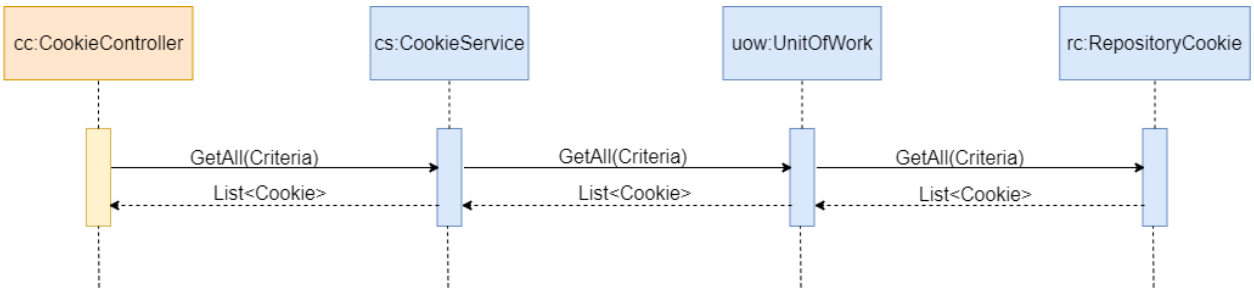
Слика 97 Дијаграм секвенци - УГ4

Уговор УГ5: GetAll
Операција: GetAll(List<Cookie>): сигнал
Веза са СК: СК4, СК5, СК7
Предуслови:
Постуслови:



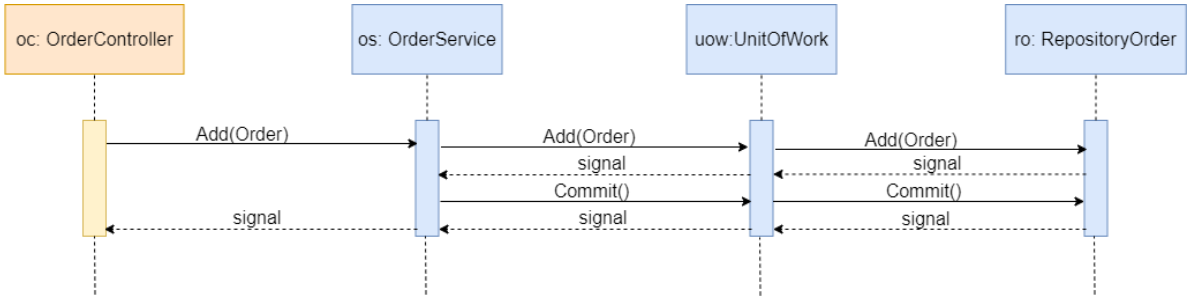
Слика 98 Дијаграм секвенци - УГ5

Уговор УГ6: GetAll
Операција: GetAll(Criteria, List<Cookie>): сигнал
Веза са СК: СК4, СК7
Предуслови:
Постуслови:



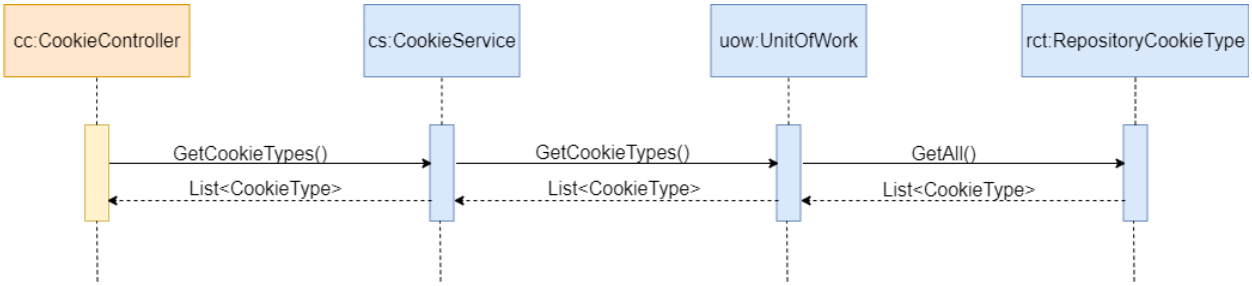
Слика 99 Дијаграм секвенци - УГ6

Уговор УГ7: Add
Операција: Add(Order): сигнал
Веза са СК: СК5
Предуслови: Вредносна и структурна ограничења над објектом **Order** морају бити задовољена.
Постуслови: Наруџбеница је сачувана.



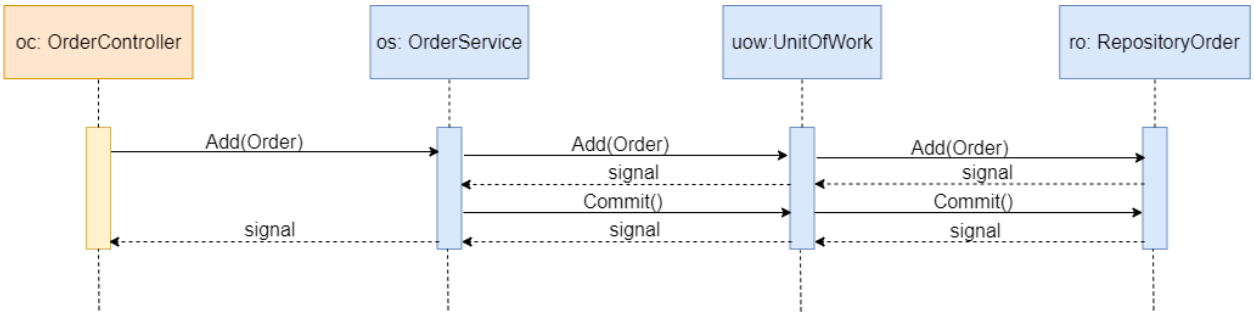
Слика 100 Дијаграм секвенци - УГ7

Уговор УГ8: GetCookieTypes
Операција: GetCookieTypes(List<CookieType): сигнал
Веза са СК: СК6
Предуслови:
Постуслови:



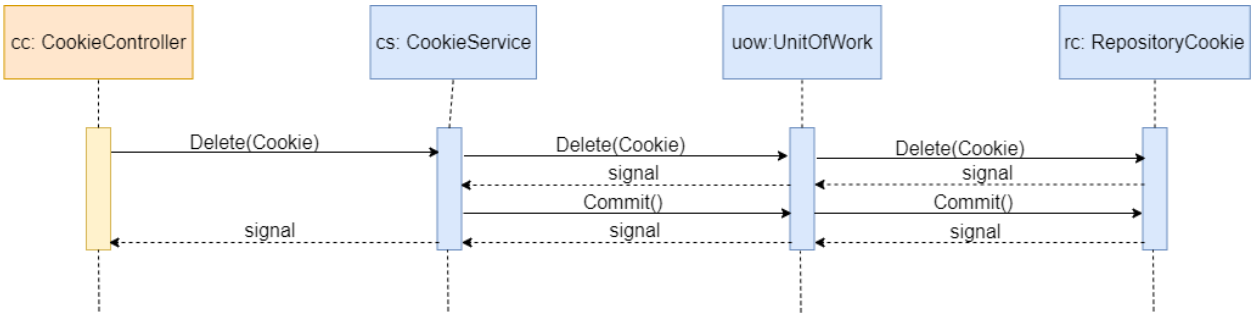
Слика 101 Дијаграм секвенци - УГ8

Уговор УГ9: Add
Операција: Add(Cookie): сигнал
Веза са СК: СК6
Предуслови: Вредносна и структурна ограничења над објектом **Cookie** морају бити задовољена.
Постуслови: Колач је сачуван.



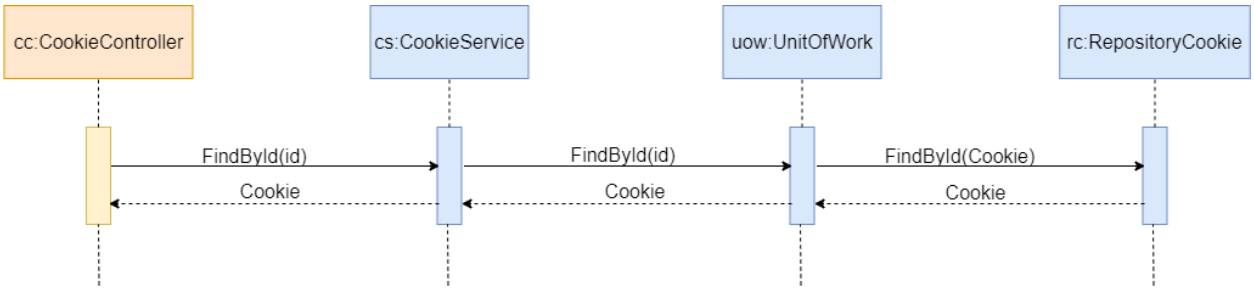
Слика 102 Дијаграм секвенци - УГ9

Уговор УГ10: Delete
Операција: Delete(Cookie): сигнал
Веза са СК: СК7
Предуслови: Вредносна и структурна ограничења над објектом **Cookie** морају бити задовољена.
Постуслови: Колач је избрисан.



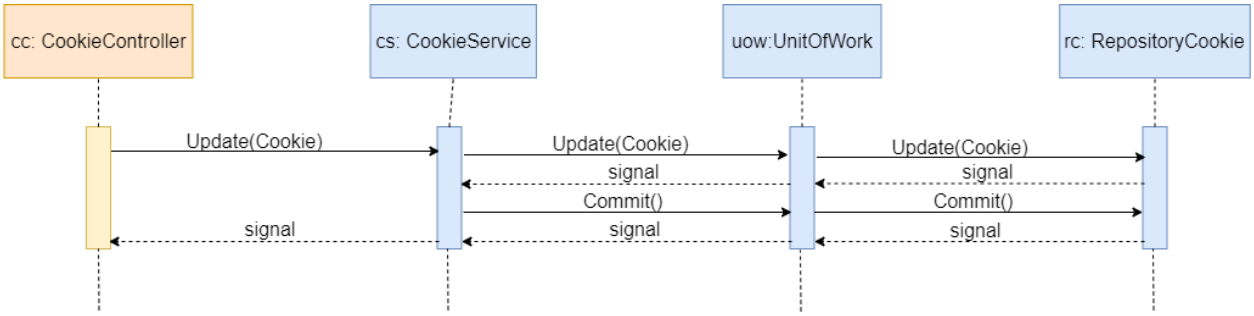
Слика 103 Дијаграм секвенци УГ10

Уговор УГ11: FindById
Операција: FindById(Id, List<Cookie>): сигнал
Веза са СК: СК8
Предуслови:
Постуслови:



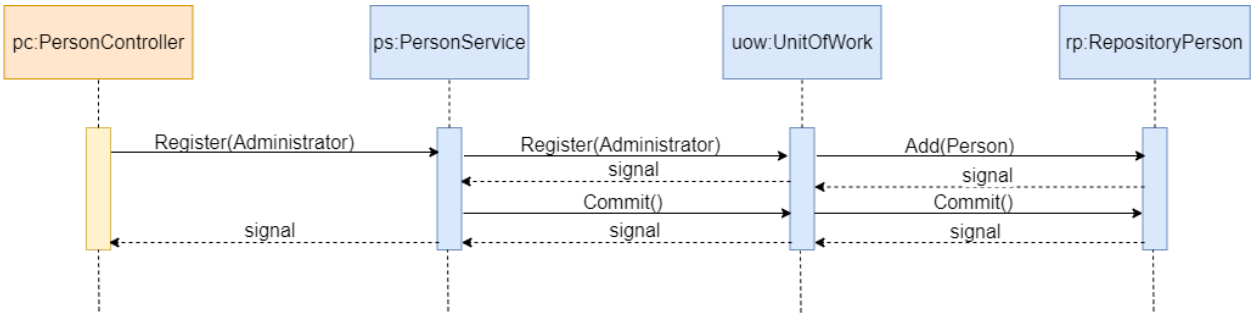
Слика 104 Дијаграм секвенци УГ11

Уговор УГ12: Update
Операција: Update(Cookie): сигнал
Веза са СК: СК8
Предуслови: Вредносна и структурна ограничења над објектом **Cookie** морају бити задовољена.
Постуслови: Колач је сачуван.



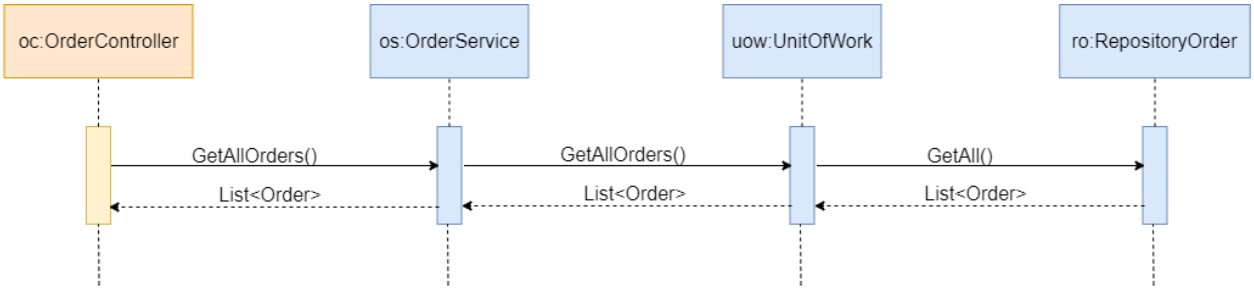
Слика 105 Дијаграм секвенци - УГ12

Уговор УГ13: Register
Операција: Register(Administrtor): сигнал
Веза са СК: СК9
Предуслови: Вредносна и структурна ограничења над објектом **Administrator** морају бити задовољена.



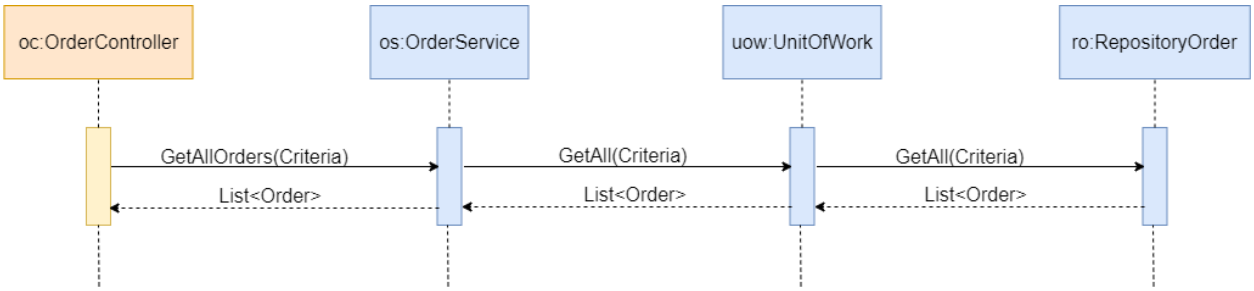
Слика 106 Дијаграм секвенци - УГ13

Уговор УГ14: GetAllOrders
Операција: GetAllOrders(List<Order>): сигнал
Веза са СК: СК10, СК11
Предуслови:
Постуслови:



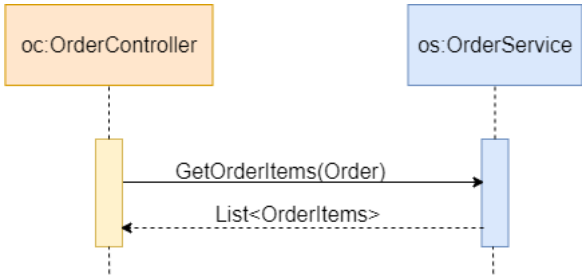
Слика 107 Дијаграм секвенци - УГ14

Уговор УГ15: GetAllOrders
Операција: GetAllOrders(Criteria, List<Order>): сигнал
Веза са СК: СК10, СК11
Предуслови:
Постуслови:



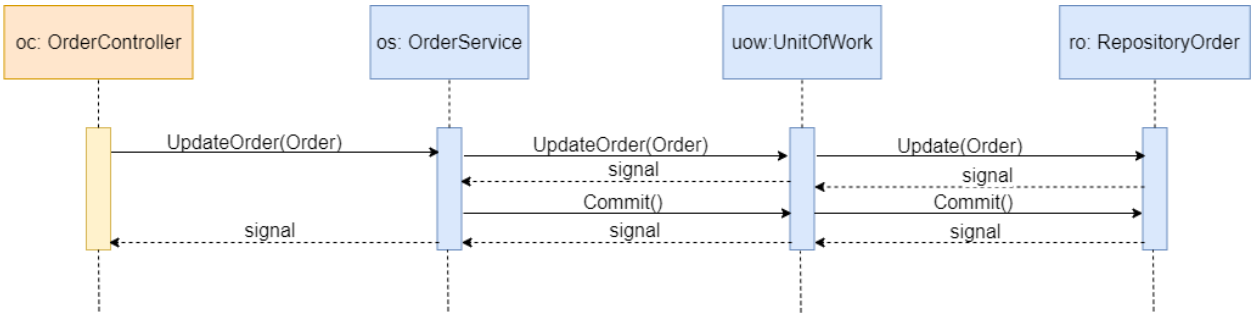
Слика 108 Дијаграм секвенци - УГ15

Уговор УГ16: GetOrderItems
Операција: GetOrderItems(Order): сигнал
Веза са СК: СК10, СК11
Предуслови:
Постуслови:



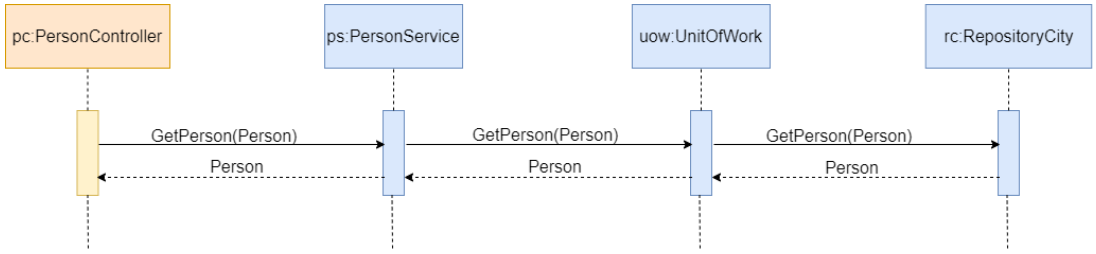
Слика 109 Дијаграм секвенци - УГ16

Уговор УГ17: Update
Операција: Update(Order): сигнал
Веза са СК: СК11
Предуслови: Вредносна и структурна ограничења над објектом **Order** морају бити задовољена.
Постуслови: Наруџбеница је сачувана.



Слика 110 Дијаграм секвенци - УГ17

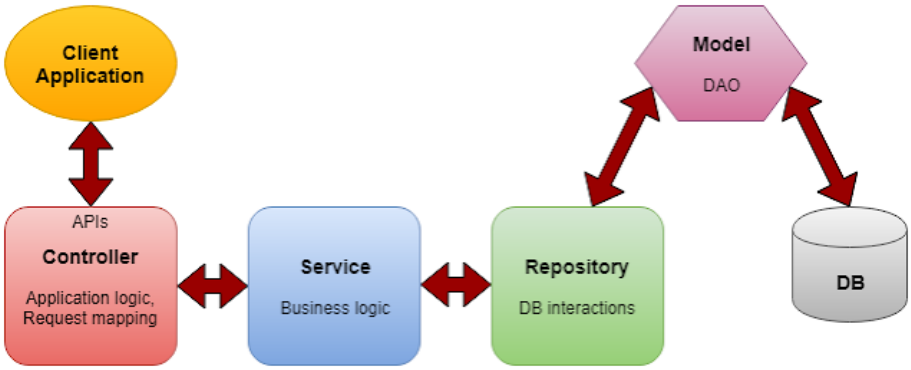
Уговор УГ18: GetPerson
Операција: GetPerson(Person): сигнал
Веза са СК: СК3
Предуслови:
Постуслови:



Слика 111 Дијаграм секвенци - УГ18

5.3.4 Пројектовање слоја приступа подацима

Комуникација између пословне логике и складишта података се врши путем *Repository* класа, поменутих у претходним одељцима. На следећој слици приказан је ток ове комуникације.

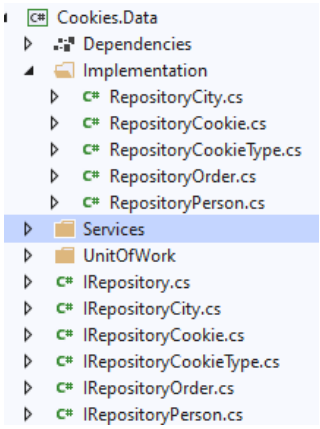


Слика 112 Комуникација између пословне логике и складишта података
http://randikatech.blogspot.com/2019/09/get-your-hands-dirty-with-micro-services.html

Репозиторијум је, у овом контексту, суштина коришћеног *Repository патерна*. Ова класа представља апстрактни слој, чије коришћење за комуникацију између слоја података и слоја пословне логике има бројне предности [19]:

1) пословна логика може бити тестирана независно од логике за приступање подацима
2) код написан за приступ складишту података се може вишекратно употребљавати
3) тај код је централно управљан
4) доменску логику је лако имплементирати

Постоји више специфичних репозиторијума који се односе на одређени објекат и сви они имплементирају свој специфични интерфејс, а сви ти интерфејси наслеђују један генерички који је назван *IRepository*. Та структура је приказана на слици 113.

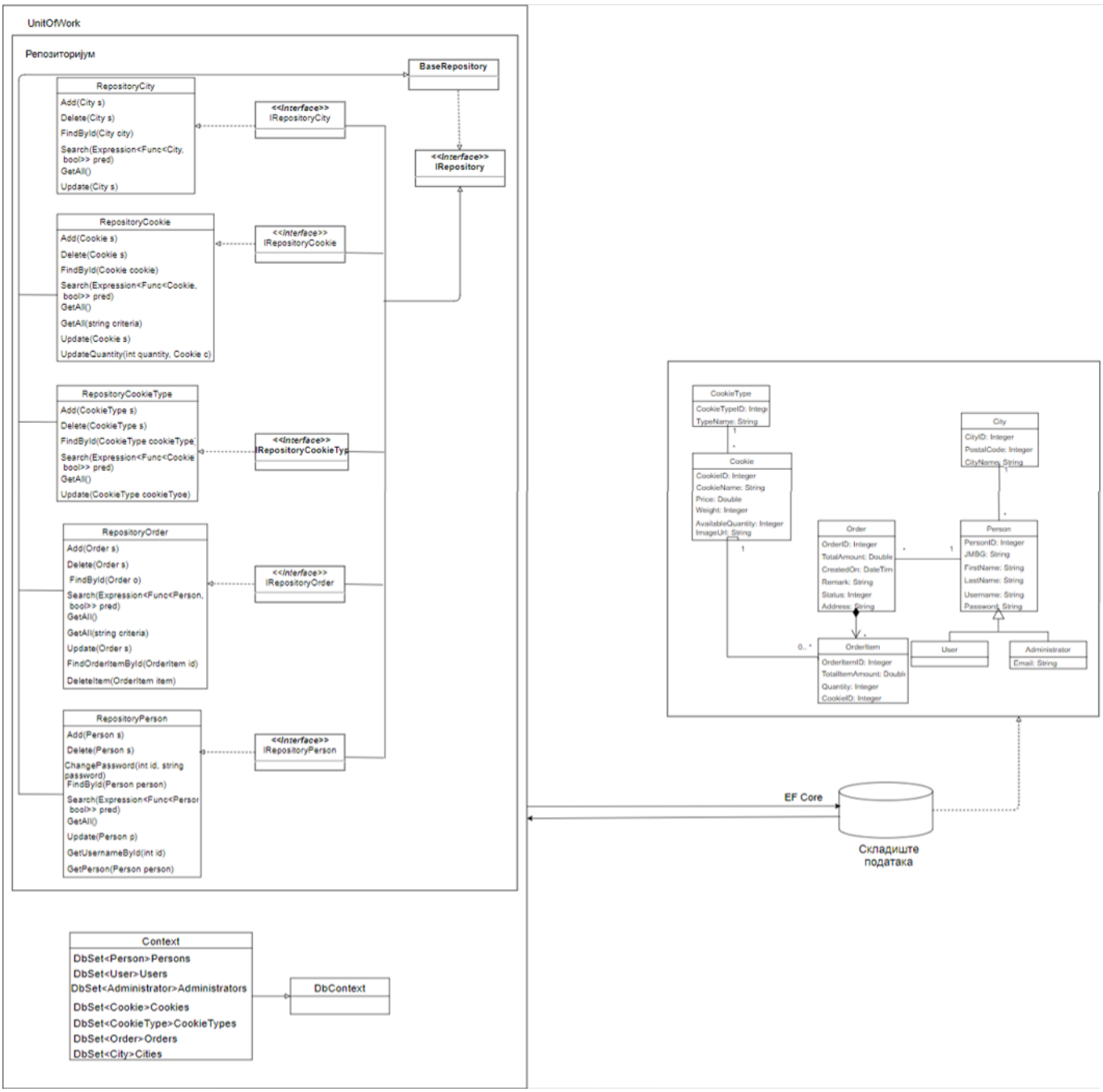


Слика 113 Структура апстрактног слоја

Једна од *Repository* класа изгледа овако:

```
public class RepositoryCity : IRepositoryCity
    {
        private Context context;
        public RepositoryCity(Context context)
        {
            this.context = context;
        }
        public void Add(City s)
        {
            context.Cities.Add(s);
        }
        public void Delete(City s)
        {
            context.Cities.Remove(s);
        }
        public City FindById(City city)
        {
            return context.Cities.SingleOrDefault(k => k.CityID == city.CityID);
        }
        public List<City> GetAll()
        {
            return context.Cities.ToList();
        }
        public List<City> Search(Expression<Func<City, bool>> pred)
        {
            return context.Cities.Where(pred).ToList();
        }
        public void Update(City s)
        {
            City city = context.Cities.SingleOrDefault(c => c.CityID == s.CityID);
            context.Entry(city).CurrentValues.SetValues(s);
        }
    }
```

У претходном примеру се примећује да свака *Repository* класа има свој конструктор, кроз који инстанцира објекат класе *Context* и имплементира специфични интерфејс (у овом примеру класа *RepositoryCity* имплементира *IRepositoryCity*), који је наследио генерички интерфејс *IRepository*. Ова класа мора да имплементира све методе чија је спецификација дата у оба интерфејса.



Слика 114 Пројектовање апстрактног слоја између пословне логике и складишта података

5.3.5 Пројектовање складишта података

На основу концептуалног модела се формира релациони модел који представља основу за пројектовање релационе базе података. [16]

5.3.5.2 Пројектовање табела релационог модела

У наставку је приказан релациони модел који је креиран на основу доменског модела.

CookieType (CookieTypeID, TypeName)
Cookie (CookieID, CookieName, Price, Weight, Description, AvailableQuantity, ImageUrl, *CookieTypeID*)
City (CityID, CityName, PostalCode)
Person (PersonID, JMBG, FirstName, LastName, Username, Password, Email, Discriminator, *CityID*)
Order (OrderID, TotalAmount, CreatedOn, Remark, Status, Address, *UserPersonID*)
OrderItem (OrderItemID, *OrderID*, TotalItemAmount, Quantity, *CookieID*)

Табела CookieType		**Просто вредносно ограничење**		**Сложено вредносно ограничење**		**Структурно ограничење**
Атрибути	**Име**	**Тип атрибута**	**Вредност атрибута**	**Међузависност атрибута једне табеле**	**Међузависност атрибута више табела**	**INSERT /**
	CookieTypeID	Integer	Not null AND >0			**UPDATE CASCADES Cookie**
	TypeName	String	Not null			**DELETE RESTRICTED Cookie**

Табела 3 Табела CookieType

Табела Cookie		**Просто вредносно ограничење**		**Сложено вредносно ограничење**		**Структурно ограничење**
Атрибути	**Име**	**Тип атрибута**	**Вредност атрибута**	**Међузависност атрибута једне табеле**	**Међузависност атрибута више табела**	**INSERT RESTRICTED CookieType**
	CookieID	Integer	Not null AND > 0			
	CookieName	String	Not null			**UPDATE RESTRICTED CookieType, CASCADES OrderItem**
	Price	Double	Not null AND > 0			
	Weight	Integer	Not null AND > 0			
	Description	String				
	AvailableQuantity	Integer	Not null			**DELETE RESTRICTED OrderItem**
	ImageUrl	String				
	CookieTypeID	Integer	Not null AND > 0			

Табела 4 Табела Cookie

Табела City		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT / UPDATE CASCADES Person DELETE RESTRICTED Person
	CityID	Integer	Not null AND > 0			
	CityName	String	Not null			
	PostalCode	Integer	Not null AND > 0			

Табела 5 Табела City

Табела Person		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT RESTRICTED City UPDATE CASCADES Order DELETE RESTRICTED Order
	PersonID	Integer	Not null AND > 0			
	JMBG	String	Not null			
	FirstName	String	Not null			
	LastName	String	Not null			
	Username	String	Not null and unique			
	Password	String	Not null			
	Email	String	Unique			
	Discriminator	String	Not null			
	CityID	Integer	Not null AND > 0			

Табела 6 Табела Person

Табела Order		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT RESTRICTED Person UPDATE RESTRICTED Person, CASCADES OrderItem
	OrderID	Integer	Not null AND > 0			
	TotalAmount	Double	Not null			
	CreatedOn	DateTime	Not null			
	Remark	String				

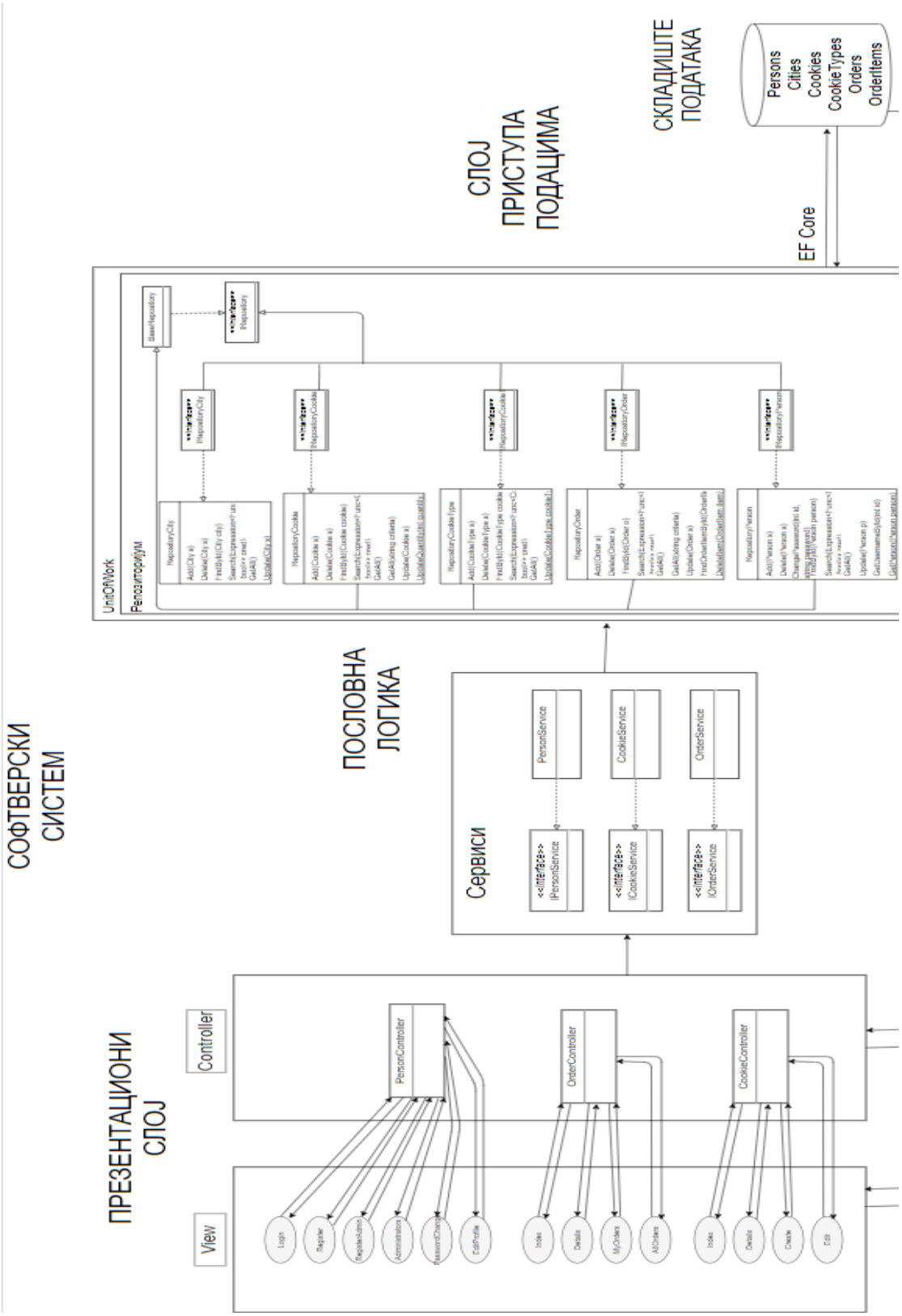
	Status	Integer	Not null			**DELETE CASCADES OrderItem**
	Address	String	Not null			
	UserPersonID	Integer	Not null AND > 0			

Табела 7 Табела Order

Табела OrderItem		**Просто вредносно ограничење**		**Сложено вредносно ограничење**		**Структурно ограничење**
Атрибути	**Име**	**Тип атрибута**	**Вредност атрибута**	**Међузависност атрибута једне табеле**	**Међузависност атрибута више табела**	**INSERT RESTRICTED Order, Cookie** **UPDATE RESTRICTED Order, Cookie** **DELETE /**
	OrderItemID	Integer	Not null AND > 0			
	OrderID	Integer	Not null AND > 0			
	TotalItemAmount	Double	Not null			
	Quantity	Integer	Not null			
	CookieID	Integer	Not null AND > 0			

Табела 8 Табела OrderItem

5.3.6 Коначан изглед архитектуре софтверског система



Слика 115 Коначна архитектура софтверског система I

Слика 116 Коначна архитектура софтверског система II

5.4 Фаза имплементације

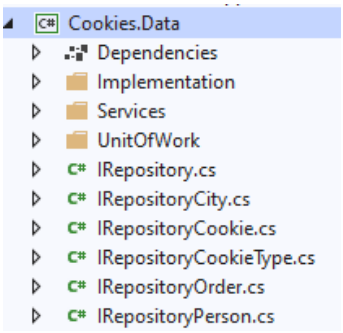
Пројекат је реализован у програмском језику *C#*, коришћењем *MVC* патерна и *.NET Core* платформе. Као развојно окружење коришћен је *Visual Studio 2019* и апликација се извршава на *IIS* веб серверу. За управљање базом података се користи *SqlServer*.

5.4.1 Структура софтверског решења

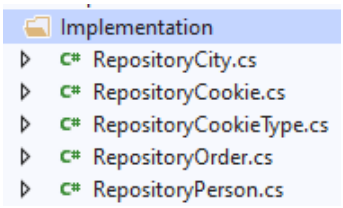
Систем је састављен од следећих пројеката:

1. Cookies.Data
2. Cookies.Domain
3. Cookies.WebApp

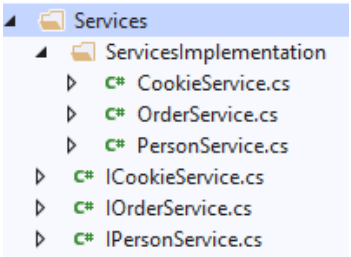
Структура пројекта Cookies.Data је следећа:



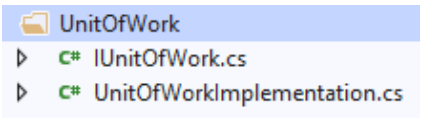
Слика 117 Пројекат Cookies.Data



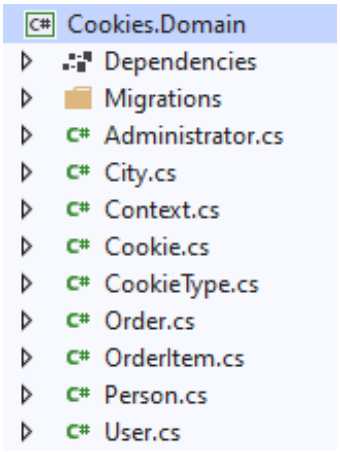
Слика 118 Фолдер Implementation



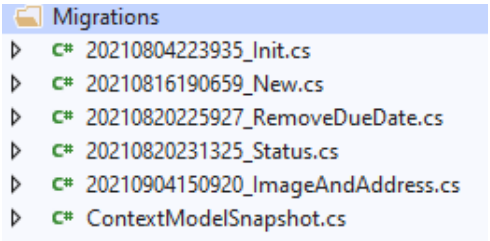
Слика 119 Фолдер Services



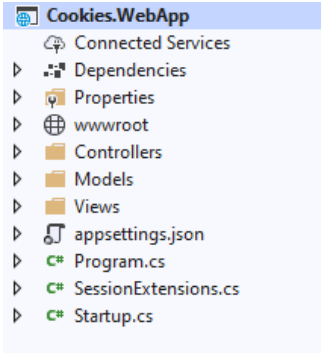
Слика 120 Фолдер UnitOfWork



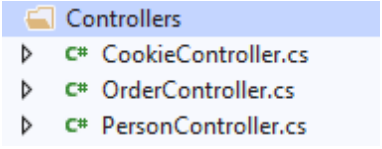
Слика 121 Пројекат Cookies.Domain



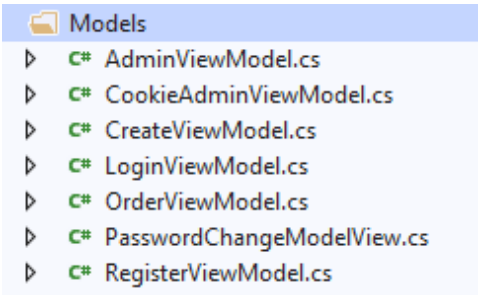
Слика 122 Фолдер Migrations



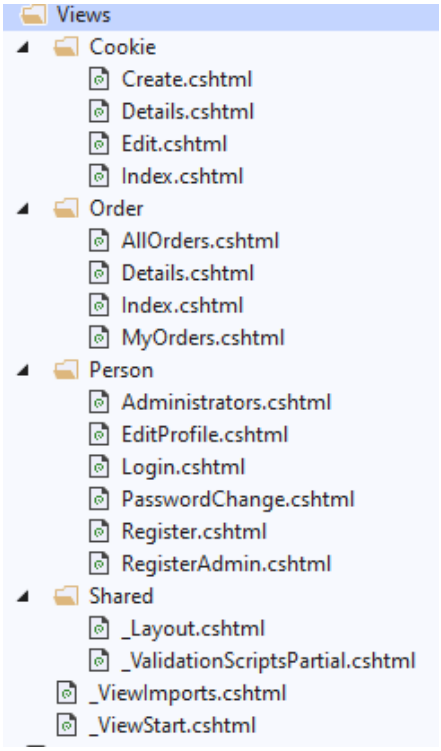
Слика 123 Пројекат Cookies.WebApp



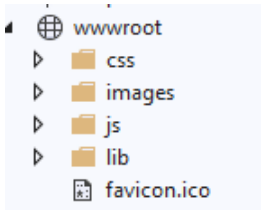
Слика 124 Фолдер Controllers



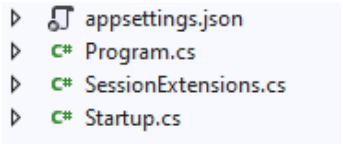
Слика 125 Фолдер Models



Слика 126 Фолдер Views



Слика 127 Фолдер Root



Слика 128 Остале класе

5.4.2 Имплементација апликационе логике

У овом поглављу биће приказана имплементација апликационе логике која обухвата комуникацију са клијентом, пословну логику, имплементацију слоја приступа подацима, имплементацију презентационог слоја.

5.4.2.1 Комуникација са клијентом

Полазну тачку у комуникацији са клијентом представља *Main* метода, која се налази у оквиру класе *Program.cs*.

```
public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }
        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
    }
```

Комуникација са клијентима се остварује слањем *HTTP* захтева и примањем одговора. Кориснички захтев се шаље првенствено до контролера, који су одговорни за његову обраду. Контролери их шаљу до одговарајућих сервиса, у којима су имплементиране системске операције. Помоћу инстанце *UnitOfWork-a* се приступа складишту података и врше одговарајуће операције над њиме.

У класи *Startup.cs* у методи *ConfigureServices* која се прва позива, дефинишу се и додају функционалности неопходне за функционисање софтверског система.

```
public void ConfigureServices(IServiceCollection services)
        {
            services.AddSession(options =>
            {
                options.IdleTimeout = TimeSpan.FromMinutes(60);
            });
            services.AddDistributedMemoryCache();
            services.AddControllersWithViews();
            services.AddControllers().AddNewtonsoftJson(x =>
x.SerializerSettings.ReferenceLoopHandling =
Newtonsoft.Json.ReferenceLoopHandling.Ignore);
            services.AddScoped<IUnitOfWork, UnitOfWorkImplementation>();
            services.AddScoped<IPersonService, PersonService>();
            services.AddScoped<ICookieService, CookieService>();
            services.AddScoped<IOrderService, OrderService>();
            services.AddDbContext<Context>();
            services.AddRazorPages().AddMvcOptions(options =>
```

```
            {
                options.ModelBindingMessageProvider.SetValueMustNotBeNullAccessor(
_ => "Ovo polje je obavezno!");
            });
        }
```

У истој класи се налази и метода *Configure*, приказана у наставку.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Home/Error");
                app.UseHsts();
            }
            app.UseHttpsRedirection();
            app.UseStaticFiles();
            app.UseRouting();
            app.UseSession();
            app.UseAuthorization();
            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllerRoute(
                    name: "default",
                    pattern: "{controller=Person}/{action=Index}/{id?}");
            });
        }
```

У делу *pattern* дефинисано је који контролер се позива када се апликација покрене и коју акцију он извршава. У овом случају позива се *PersonController* да изврши акцију *Index* која враћа *Login* страницу. Особа која приступа систему, било да је у питању корисник или администратор, мора бити пријављена под својим корисничким именом и шифром како би могла да користи апликацију.

5.4.2.2 Пословна логика

Пословна логика је описана структуром и понашањем софтверског система. Структура се односи на доменске класе, а понашање на системске операције. [16]

Доменске класе су дефинисане у пројекту *Cookies.Domain*. Дат је пример доменске класе *Cookie,* а остале класе су дефинисане налик на ову, са себи својственим атрибутима.

```
public class Cookie
    {
        public int CookieID { get; set; }
        public string CookieName { get; set; }
        public double Price { get; set; }
        public int Weight { get; set; }
        public int AvailableQuantity { get; set; }
```

```
        public string Description { get; set; }
        public int CookieTypeID { get; set; }
        public CookieType CookieType { get; set; }
        public String ImageUrl { get; set; }
    }
```

Имплементација системских операција

Уговор УГ1: Login
Операција: Login(Person, Role): сигнал
Веза са СК: СК1
Предуслови: Вредносна и структурна ограничења над објектом **Person** морају бити задовољена.
Постуслови: Корисник/Администратор је пријављен.

```
public Person Login(Person p, String role)
        {
            try
            {
                if (role == "User")
                {
                    User user = (User)uow.Person.GetPerson(new Person { Username =
p.Username, Password = p.Password });
                    return user;
                }
                if (role == "Administrator")
                {
                    Administrator administrator = (Administrator)uow.Person.GetPerson(new
Person { Username = p.Username, Password = p.Password });
                    return administrator;
                }
                return null;
            }
            catch (Exception)
            {
                return null;
            };
        }
```

Уговор УГ2: GetAllCities
Операција: GetAllCities(List<City>): сигнал
Веза са СК: СК1, СК2, СК9
Предуслови:
Постуслови:

```
public List<City> GetAllCities()
        {
            return uow.City.GetAll();
        }
```

Уговор УГ3: Register
Операција: Register(User): сигнал
Веза са СК: СК2

Предуслови: Вредносна и структурна ограничења над објектом **User** морају бити задовољена.
Постуслови: Регистрован је нови корисник.

```
public void Register(User p)
        {
            uow.Person.Add(p);
            uow.Commit();
        }
```

Уговор УГ4: Update
Операција: Update(Person): сигнал
Веза са СК: СК3
Предуслови: Вредносна и структурна ограничења над објектом **Person** морају бити задовољена.
Постуслови: Корисник је сачуван.

```
public void Update(Person p)
        {
            uow.Person.Update(p);
            uow.Commit();
        }
```

Уговор УГ5: GetAll
Операција: GetAll(List<Cookie>): сигнал
Веза са СК: СК4, СК5, СК7
Предуслови:
Постуслови:

```
public List<Cookie> GetAll()
        {
            return uow.Cookie.GetAll();
        }
```

Уговор УГ6: GetAll
Операција: GetAll(Criteria, List<Cookie>): сигнал
Веза са СК: СК4, СК7
Предуслови:
Постуслови:

```
public List<Cookie> GetAll(string criteria)
        {
            return uow.Cookie.GetAll(criteria);
        }
```

Уговор УГ7: Add
Операција: Add(Order): сигнал
Веза са СК: СК5
Предуслови: Вредносна и структурна ограничења над објектом **Order** морају бити задовољена.

Постуслови: Наруџбеница је сачувана.

```
  public void Add(Order s)
        {
            s.OrderItems.ForEach(c => c.Cookie = null);
            s.OrderItems.ForEach(c => c.Order = null);
            uow.Order.Add(s);
            uow.Commit();
        }
```

Уговор УГ8: GetCookieTypes
Операција: GetCookieTypes(List<CookieType>): сигнал
Веза са СК: СК6
Предуслови:
Постуслови:

```
  public List<CookieType> GetCookieTypes()
        {
            return uow.CookieType.GetAll();
        }
```

Уговор УГ9: Add
Операција: Add(Cookie): сигнал
Веза са СК: СК6
Предуслови: Вредносна и структурна ограничења над објектом **Cookie** морају бити задовољена.
Постуслови: Колач је сачуван.

```
public void Add(Cookie c)
        {
            uow.Cookie.Add(c);
            uow.Commit();
        }
```

Уговор УГ10: Delete
Операција: Delete(Cookie): сигнал
Веза са СК: СК7
Предуслови: Вредносна и структурна ограничења над објектом **Cookie** морају бити задовољена.
Постуслови: Колач је избрисан.

```
public void Delete(Cookie c)
        {
            uow.Cookie.Delete(c);
            uow.Commit();
        }
```

Уговор УГ11: FindById
Операција: FindById(Id, List<Cookie>): сигнал
Веза са СК: СК8
Предуслови:

Постуслови:

```
public Cookie FindById(int id)
        {
            return uow.Cookie.FindById(new Cookie { CookieID = id });
        }
```

Уговор УГ12: Update
Операција: Update(Cookie): сигнал
Веза са СК: СК8
Предуслови: Вредносна и структурна ограничења над објектом **Cookie** морају бити задовољена.
Постуслови: Колач је сачуван.

```
public void Update(Cookie c)
        {
            uow.Cookie.Update(c);
            uow.Commit();
        }
```

Уговор УГ13: Register
Операција: Register(Administrtor): сигнал
Веза са СК: СК9
Предуслови: Вредносна и структурна ограничења над објектом **Administrator** морају бити задовољена.
Постуслови: Регистрован је нови администратор.

```
public void Register(Administrator p)
        {
            uow.Person.Add(p);
            uow.Commit();
        }
```

Уговор УГ14: GetAllOrders
Операција: GetAllOrders(List<Order>): сигнал
Веза са СК: СК10, СК11
Предуслови:
Постуслови:

```
public List<Order> GetAllOrders()
        {
            return uow.Order.GetAll();
        }
```

Уговор УГ15: GetAllOrders
Операција: GetAllOrders(Criteria): сигнал
Веза са СК: СК10, СК11
Предуслови:
Постуслови:

```
public List<Order> GetAll(string criteria)
        {
            return uow.Order.GetAll(criteria);
        }
```

Уговор УГ16: GetOrderItems
Операција: GetOrderItems(): сигнал
Веза са СК: СК10, СК11
Предуслови:
Постуслови:

```
public List<OrderItem> GetOrderItems(Order o)
        {
            return o.OrderItems;
        }
```

Уговор УГ17: Update
Операција: Update(Order): сигнал
Веза са СК: СК11
Предуслови: Вредносна и структурна ограничења над објектом **Order** морају бити задовољена.
Постуслови: Наруџбеница је сачувана.

```
public void UpdateOrder(Order o)
        {
            uow.Order.Update(o);
            uow.Commit();
        }
```

Уговор УГ18: GetPerson
Операција: GetPerson(Person): сигнал
Веза са СК: СК3
Предуслови:
Постуслови:

```
public Person GetPerson(Person p)
        {
            return uow.Person.GetPerson(new Person { Username = p.Username, Password =
p.Password });
        }
```

5.4.2.3 Имплементација сервиса

Приказан је код класе *CookieService* и одговарајућег интерфејса. Остали сервиси су креирани налик на приказани пример.

```
using Cookies.Data.UnitOfWork;
using Cookies.Domain;
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Text;

namespace Cookies.Data.Services.ServicesImplementation
{
    public class CookieService : ICookieService
    {
        private readonly IUnitOfWork uow;
        public CookieService(IUnitOfWork uow)
        {
            this.uow = uow;
        }
        public void Add(Cookie c)
        {
            uow.Cookie.Add(c);
            uow.Commit();
        }
        public void Delete(Cookie c)
        {
            uow.Cookie.Delete(c);
            uow.Commit();
        }
        public Cookie FindById(int id)
        {
            return uow.Cookie.FindById(new Cookie { CookieID = id });
        }
        public List<Cookie> GetAll()
        {
            return uow.Cookie.GetAll();
        }
        public List<Cookie> GetAll(string criteria)
        {
            return uow.Cookie.GetAll(criteria);
        }
        public List<CookieType> GetCookieTypes()
        {
            return uow.CookieType.GetAll();
        }
        public CookieType GetTypeByID(int id)
        {
            return uow.CookieType.FindById(new CookieType { CookieTypeID = id });
        }
        public List<String> StartsWith(string prefix)
        {
            var names = uow.Cookie.GetAll().Where(c =>
c.CookieName.Contains(prefix)).Select(c => c.CookieName).ToList();
            return names;
        }
        public void Update(Cookie c)
        {
            uow.Cookie.Update(c);
            uow.Commit();
        }
        public void UpdateQuantity(int quantity, Cookie c)
        {
            Cookie k = uow.Cookie.FindById(new Cookie { CookieID = c.CookieID });
```

```
            uow.Cookie.UpdateQuantity(quantity, c);
            uow.Commit();
        }
    }
}

namespace Cookies.Data.Services
{
    public interface ICookieService
    {
        List<Cookie> GetAll();
        List<Cookie> GetAll(string criteria);
        List<CookieType> GetCookieTypes();
        CookieType GetTypeByID(int id);
        void Delete(Cookie c);
        Cookie FindById(int id);
        void Add(Cookie c);
        void Update(Cookie c);
        void UpdateQuantity(int quantity, Cookie c);
        public List<String> StartsWith(String prefix);
    }
}
```

5.4.2.4 Имплементација слоја приступа подацима

Коришћење патерна при развоју софтвера се уводи ради лакшег одржавања и измена програмског кода. Њихов главни циљ је да омогуће поновну употребу дизајнерских решења и спрече непотребно понављање кода. Изграђени су на основу искуства програмера и уочених шаблона који се могу користити у решавању различитих типова проблема.

Repository патерн

Овај патерн је заснован на постојању *Repository* класа, које обезбеђују комуникацију са базом података. Свака класа се односи на одређену табелу у бази података и садржи имплементирану логику за *CRUD* операције и све остале потребне операције над објектом. Захваљујући њима, апликациони слој не мора да води рачуна о томе како је ова логика имплементирана, већ је само употребљава. [10]

Да би се имплементирале *CRUD* операције, свака креирана *Repository* класа имплементира један специфични интерфејс. Тај интерфејс садржи спецификацију свих метода, односно операција које се могу извршити над објектом. Под претпоставком да над сваком класом треба обезбедити основне операције уноса, брисања, измене и читања података, потребно је креирати један генерички интерфејс у ком ће бити дата спецификација тих заједничких метода. Сваки специфични интерфејс наслеђује тај генерички и додаје своје методе, које су специфичне само за објекат на који се он односи. На крају, класа *Repository* имплементира специфични интерфејс и самим тим пружа имплементацију свих метода. Једна од многих предности коришћења овог патерна је то што се у контролеру пише много мање кода када

се он користи, што програм чини прегледнијим и лакшим за одржавање. Пример из апликације је дат у наставку.

```
public interface IRepository<T>
    {
        void Add(T s);
        List<T> GetAll();
        T FindById(T id);
        void Delete(T s);
        void Update(T s);
        List<T> Search(Expression<Func<T, bool>> pred);
    }
}

public interface IRepositoryCity: IRepository<City>
    {
    }

public class RepositoryCity : IRepositoryCity
    {
        private Context context;

        public RepositoryCity(Context context)
        {
            this.context = context;
        }
        public void Add(City s)
        {
            context.Cities.Add(s);
        }
        public void Delete(City s)
        {
            context.Cities.Remove(s);
        }
        public City FindById(City city)
        {
            return context.Cities.SingleOrDefault(k => k.CityID == city.CityID);
        }
        public List<City> GetAll()
        {
            return context.Cities.ToList();
        }
        public List<City> Search(Expression<Func<City, bool>> pred)
        {
            return context.Cities.Where(pred).ToList();
        }
        public void Update(City s)
        {
            City city = context.Cities.SingleOrDefault(c => c.CityID == s.CityID);
            context.Entry(city).CurrentValues.SetValues(s);
        }
  }
```

UnitOfWork патерн

UnitOfWork патерн чини да све трансакције репозиторијума функционишу као једна. Промене се чувају само једном, а биће поништене уколико само једна трансакција не буде извршена како треба због нарушавања интегритета података. То значи да он омогућава да све класе репозиторијума деле један исти *DbContext*. На тај начин се смањује број приступа бази података при извршавању трансакција. Када се у овој апликацији не би користио *UnitOfWork* патерн, када корисник формира наруџбеницу све њене ставке би морале да се памте посебно, у оквиру посебних трансакција. У том случају, уколико само једна ставка наруџбенице не би задовољавала услове за извршавање трансакције, само она не би била сачувана, док би остале биле. [13]

UnitOfWork заправо представља обичну класу, у којој је дефинисана метода *Commit()*, као и својства свих репозиторијума који се користе у раду. Сви они добијају исту референцу на објекат класе *Context*. [13] *UnitOfWork* имплементира интерфејс *IUnitOfWork* који имплементира интерфејс *IDisposable*, а он омогућава ослобађање ресурса чим се трансакција изврши.

```
public interface IUnitOfWork: IDisposable
    {
        public IRepositoryCity City { get; set; }
        public IRepositoryCookie Cookie { get; set; }
        public IRepositoryCookieType CookieType { get; set; }
        public IRepositoryOrder Order { get; set; }
        public IRepositoryPerson Person { get; set; }
        void Commit();
    }

public class UnitOfWorkImplementation: IUnitOfWork
    {
        private Context context;
        public UnitOfWorkImplementation(Context context)
        {
            this.context = context;
            Cookie = new RepositoryCookie(context);
            CookieType = new RepositoryCookieType(context);
            Person = new RepositoryPerson(context);
            City = new RepositoryCity(context);
            Order = new RepositoryOrder(context);
        }
        public IRepositoryCity City { get; set; }
        public IRepositoryCookie Cookie {get; set; }
        public IRepositoryCookieType CookieType { get; set; }
        public IRepositoryOrder Order { get; set; }
        public IRepositoryPerson Person { get; set; }
        public void Commit()
        {
            context.SaveChanges();
        }
        public void Dispose()
        {
            context.Dispose();
        }
```

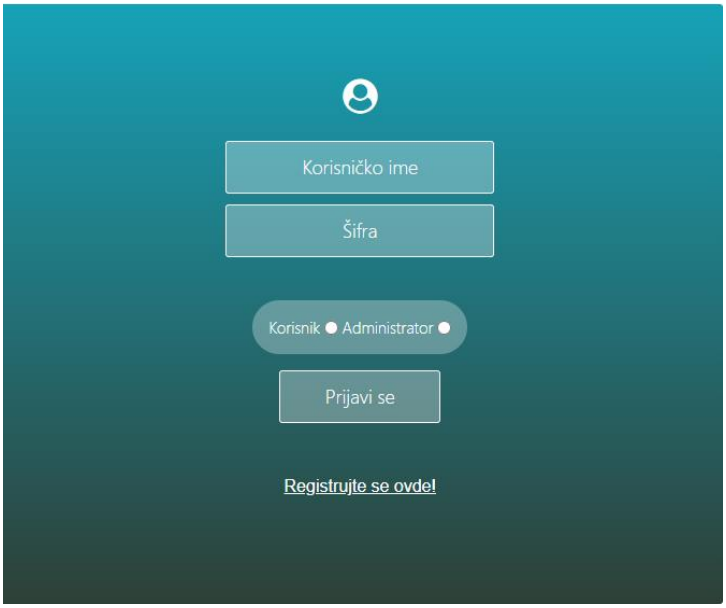
```
}
```

5.4.3 Имплементација презентационог слоја

Презентациони слој представља везу са корисником. Чине га пословна логика (енг. *Business logic*) и кориснички интерфејс (енг. *User Interface, UI*). Помоћу корисничког интерфејса, корисник или администратор приступају апликацији и користе њене функционалности. Он треба да буде једноставан за коришћење и да обезбеђује функције за које је намењен. У *MVC* патерну, кориснички интерфејс је имплементиран кроз погледе (енг. *Views*) у којима треба да је садржано што мање логике и који треба да буду заменљиви, тако да њихове промене не утичу на логику која је имплементирана у контролерима.

СК1: Случај коришћења – Пријава на систем

Назив СК
Пријава на систем
Актори СК
Корисник/Администратор
Учесници СК
Корисник/Администратор и систем (програм)
Предуслов: Систем је укључен и приказана је форма за пријаву.



Слика 129 Форма за пријаву на систем

Основни сценарио СК

1. Корисник/Администратор **уноси** своје податке за пријављивање. (АПУСО)

2. Корисник/Администратор **проверава** да ли је коректно унео податке. (АНСО)
3.Корисник/Администратор **позива** систем да нађе пријављеног корисника/администратора. (АПСО)

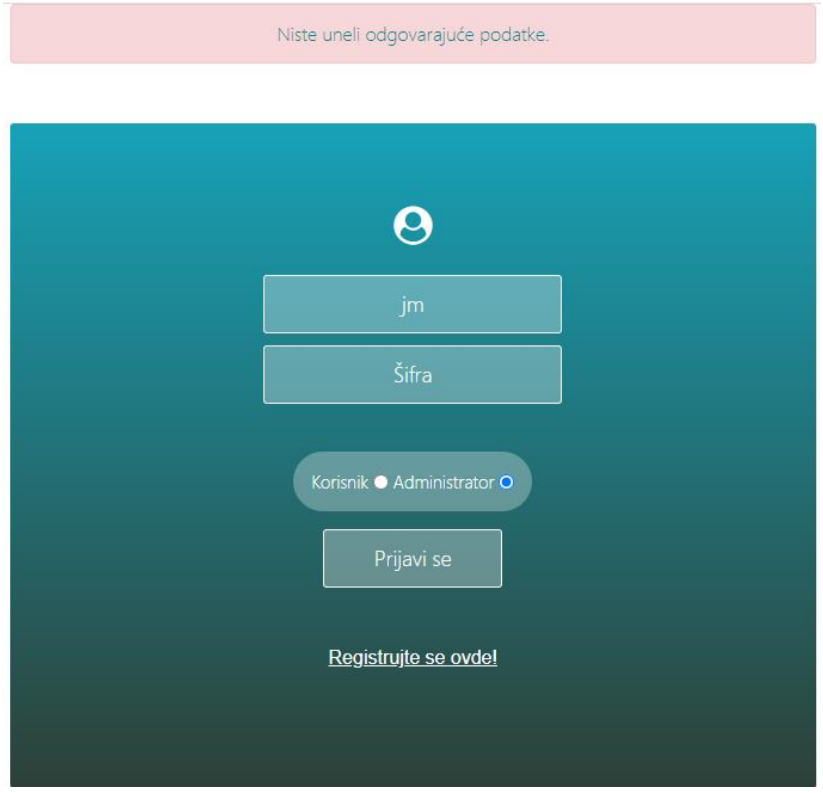
Опис акције: Кликом на дугме „*Prijavi se*“ корисник/администратор позива системску операцију Login(Person, Role)

4. Систем **тражи** пријављеног корисника/администратора. (СО)
5. Систем **приказује** кориснику/администратору поруку: „Успешна пријава”. (ИА)

Слика 130 Успешно пријављивање

Алтернативна сценарија

5.1 Уколико систем није успео да нађе пријављеног корисника/администратора, он **приказује** поруку: „Нисте унели одговарајуће податке“. (ИА)



Слика 131 Неуспешно пријављивање на систем

СК2: Случај коришћења – Креирање корисничког налога

Назив СК
Креирање корисничког налога
Актори СК
Корисник
Учесници СК
Корисник и систем (програм)
Предуслов: Систем је укључен и приказана је форма за регистрацију. Учитана је листа свих градова.

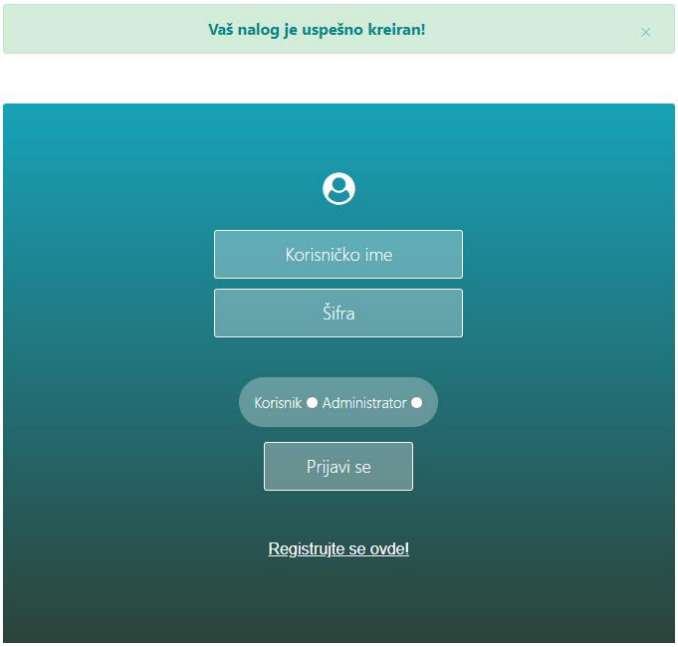
Слика 132 Форма за регистрацију

Основни сценарио СК

1. Корисник **уноси** основне податке за креирање корисничког налога. (АПУСО)
2. Корисник **проверава** да ли је коректно унео своје податке. (АНСО)
3. Корисник **позива** систем да креира кориснички налог. (АПСО)

Опис акције: Кликом на дугме „*Registrujte se*“ корисник позива системску операцију Register(User).

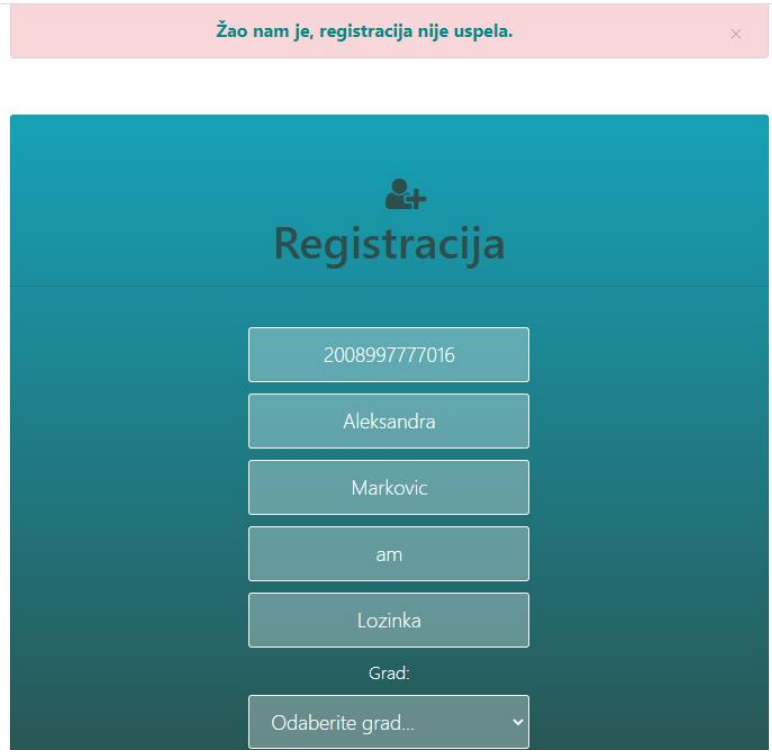
4. Систем **креира** кориснички налог. (СО)
5. Систем **приказује** кориснику поруку: „Ваш налог је успешно креиран”. (ИА)



Слика 133 Успешно креирање налога

Алтернативна сценарија

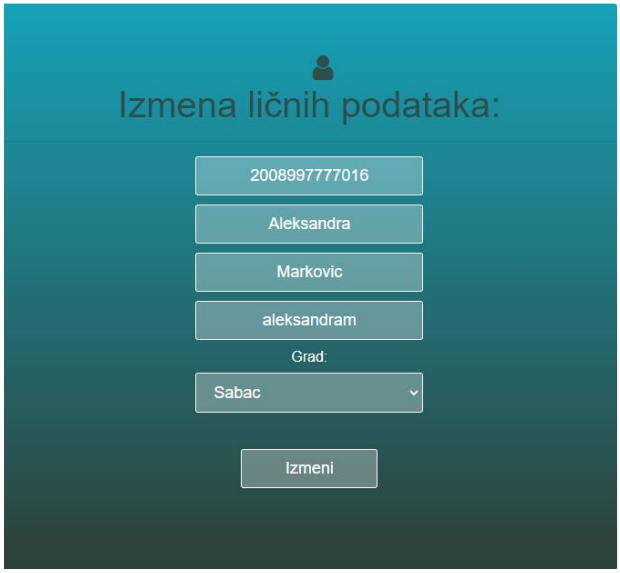
5.5 Уколико регистрација није могућа, систем **приказује** кориснику поруку: „Жао нам је, регистрација није успела“. (ИА)



Слика 134 Неуспешна регистрација

СК3: Случај коришћења – Измена корисничког налога

Назив СК
Измена корисничког налога
Актори СК
Корисник
Учесници СК
Корисник и систем (програм)
Предуслов: Систем је укључен и корисник је пријављен под својом шифром. Систем приказује податке о пријављеном кориснику. Учитани су подаци о корисничком налогу и листа свих градова.



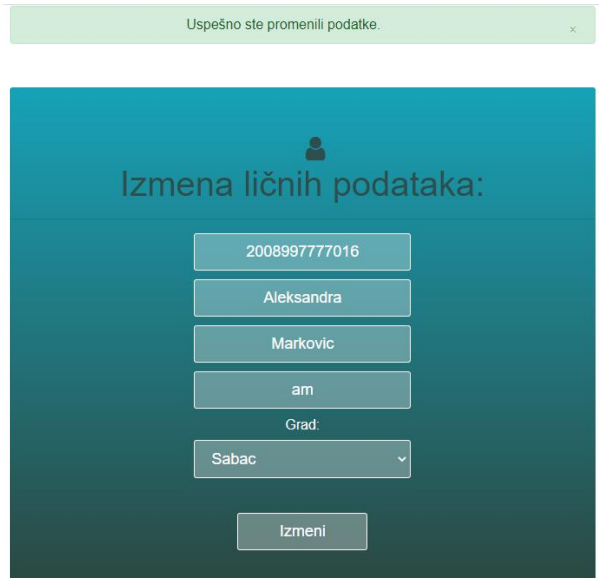
Слика 135 Форма за измену корисничког налога

Основни сценарио СК

1. Систем **тражи** податке о кориснику. (СО)
2. Систем **приказује** кориснику пронађене податке. (ИА)
3. Корисник **бира** податке који жели да измени. (АПУСО)
4. Корисник **мења** податке. (АПУСО)
5. Корисник **проверава** да ли је коректно унео нове податке. (АНСО)
6. Корисник **позива** систем да промени податке. (АПСО)

Опис акције: Кликом на дугме „*Izmeni*“ корисник позива системску операцију Update(Person).

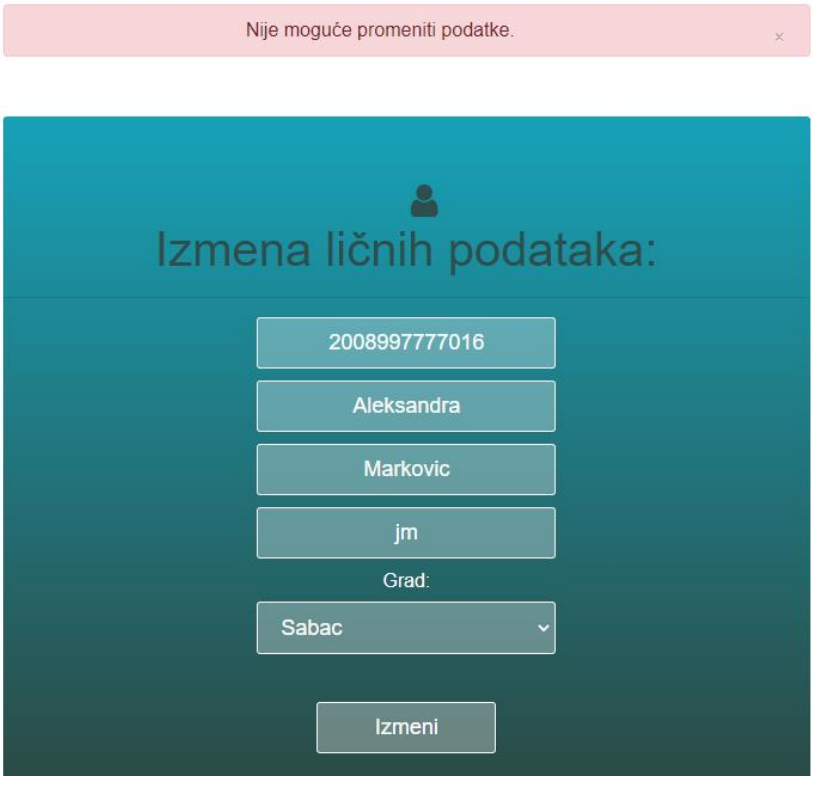
7. Систем **мења** податке о кориснику. (СО)
8. Систем **приказује** кориснику поруку: „Успешно измењени подаци!”. (ИА)



Слика 136 Успешна измена корисничког профила

Алтернативна сценарија

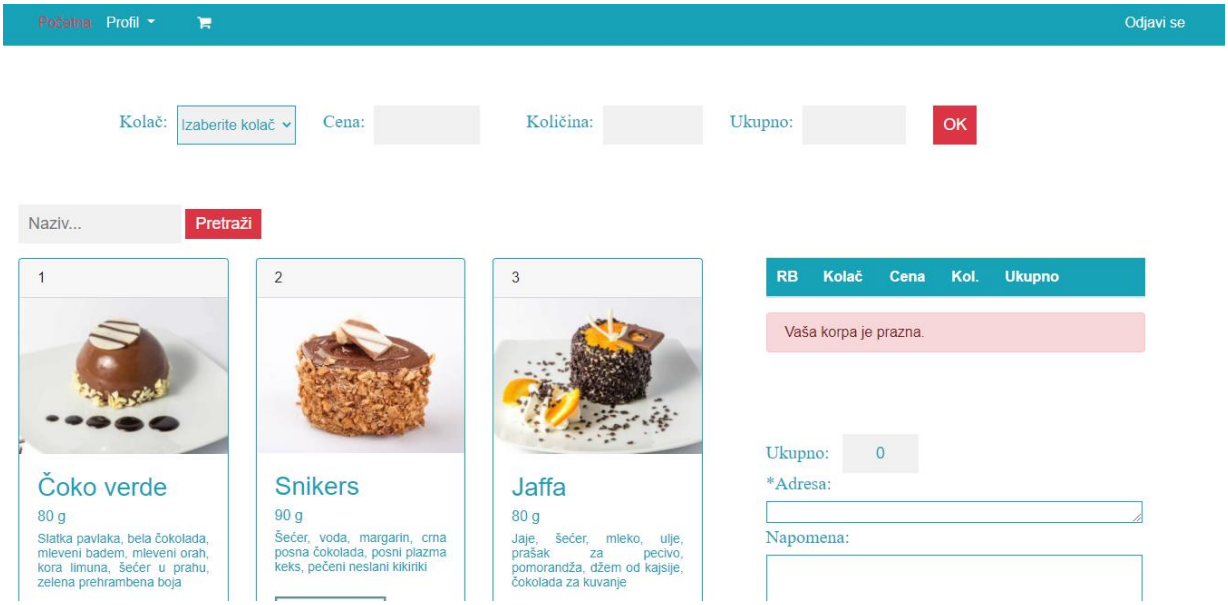
8.1 Уколико систем није у могућности да промени податке, **приказује** поруку: „Није могуће променити податке.“ (ИА)



Слика 137 Неуспешна измена корисничког профила

СК4: Случај коришћења – Претраживање колача

Назив СК
Претраживање колача
Актори СК
Корисник
Учесници СК
Корисник и систем (програм)
Предуслов: Систем је укључен и корисник је пријављен под својом шифром. Систем приказује интерфејс за пријављене кориснике. Учитана је листа свих колача.



Слика 138 Форма за пријављене кориснике

Основни сценарио СК

1. Корисник **уноси** вредност по којој претражује колаче. (АПУСО)
2. Корисник **проверава** да ли је коректно унео вредност по којој претражује. (АНСО)
3. Корисник **позива** систем да нађе колаче на основу унете вредности. (АПСО)

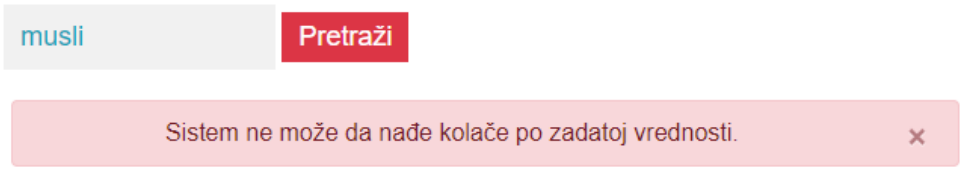
Опис акције: Кликом на дугме „*Pretraži*“ корисник позива системску операцију GetAll(Criteria, List<Cookie>).

4. Систем **тражи** колаче по наведеном критеријуму. (СО)
5. Систем **приказује** листу тражених колача. (ИА)

Слика 139 Листа тражених колача

Алтернативна сценарија

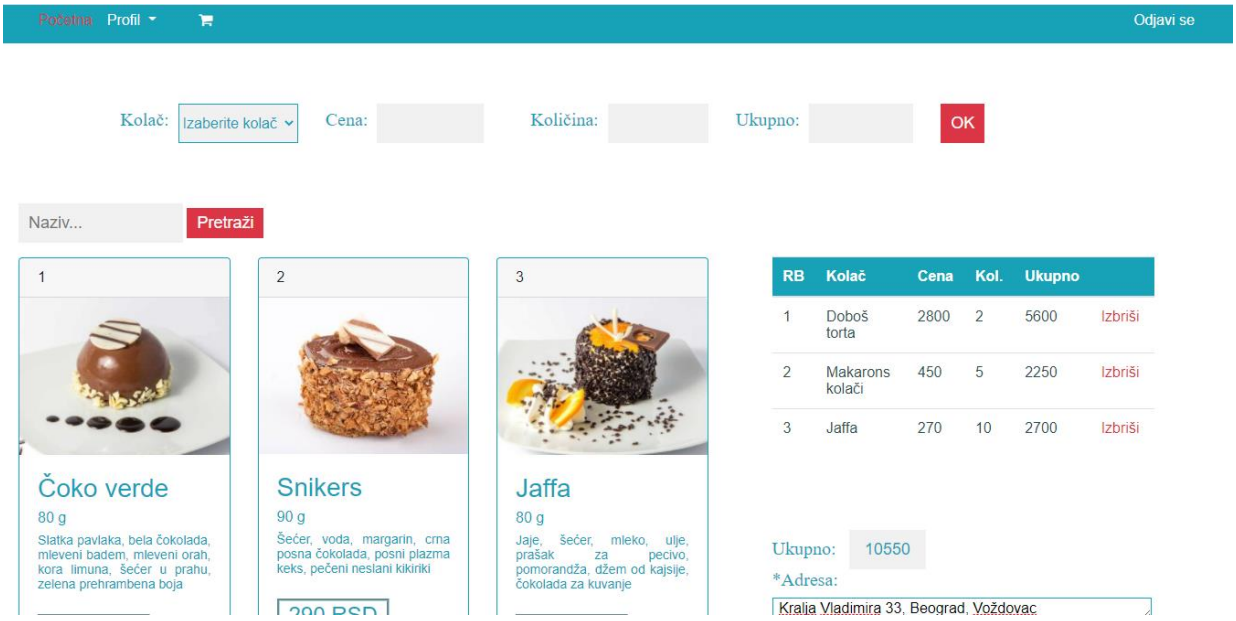
5.1 Уколико систем не може да нађе колаче по унетом критеријуму он **приказује** кориснику поруку: “Систем не може да нађе колаче по задатој вредности”. (ИА)



Слика 140 Неуспешна претрага колача

СК5: Случај коришћења – Унос наруџбенице

Назив СК
Унос наруџбенице
Актори СК
Корисник
Учесници СК
Корисник и систем (програм)
Предуслов: Систем је укључен и кориснику, који је улогован под својом шифром, приказана је листа одабраних колача. Систем приказује форму за наручивање. Учитани су подаци о колачима.



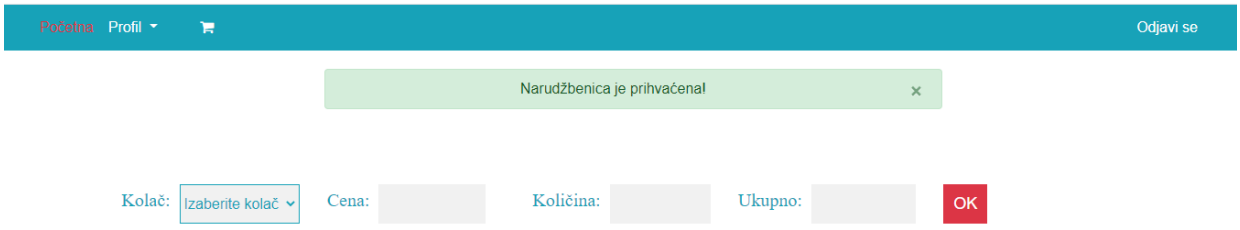
Слика 141 Форма за наручивање

Основни сценарио СК

1. Корисник **уноси** податке о наруџбеници. (АПУСО)
2. Корисник **проверава** да ли је правилно унео податке о наруџбеници. (АНСО)
3. Корисник **позива** систем да сачува податке о наруџбеници. (АПСО)

Опис акције: Кликом на дугме *"Kreiraj narudžbenicu"* корисник позива системску операцију Add(Order).

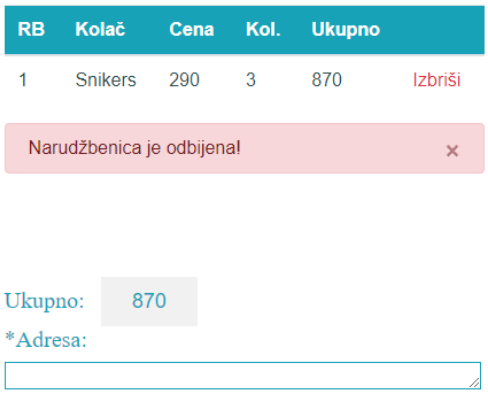
4. Систем **чува** податке о наруџбеници. (СО)
5. Систем **приказује** кориснику поруку: "Наруџбеница је прихваћена". (ИА)



Слика 142 Прихваћена наруџбеница

Алтернативна сценарија

5. 1 Уколико систем није у могућности да изврши унос наруџбенице, **приказује** поруку: „Наруџбеница је одбијена“. (ИА)



Слика 143 Одбијена наруџбеница

СК6: Случај коришћења – Унос нових колача

Назив СК
Унос нових колача
Актори СК
Администратор
Учесници СК
Администратор и систем (програм)
Предуслов: Систем је укључен и администратор је улогован под својом шифром. Систем приказује интерфејс за унос нових колача. Учитана је листа типова колача.

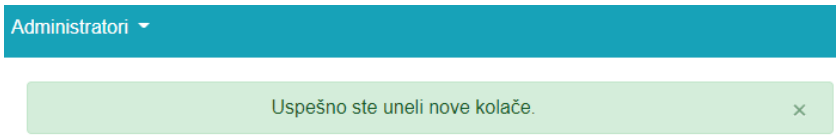
Слика 144 Форма за унос нових колача

Основни сценарио СК

1. Администратор **уноси** податке о новим колачима. (АПУСО)
2. Администратор **контролише** да ли је коректно унео податке о колачима. (АНСО)
3. Администратор **позива** систем да запамти податке о колачима. (АПСО)

Опис акције: Кликом на дугме *"Dodaj"* администратор позива системску операцију Add(Cookie).

4. Систем **памти** податке о колачима. (СО)
5. Систем **приказује** администратору запамћене податке о колачима и поруку: „Успешно сте унели нове колаче". (ИА)



Слика 145 Успешан унос нових колача

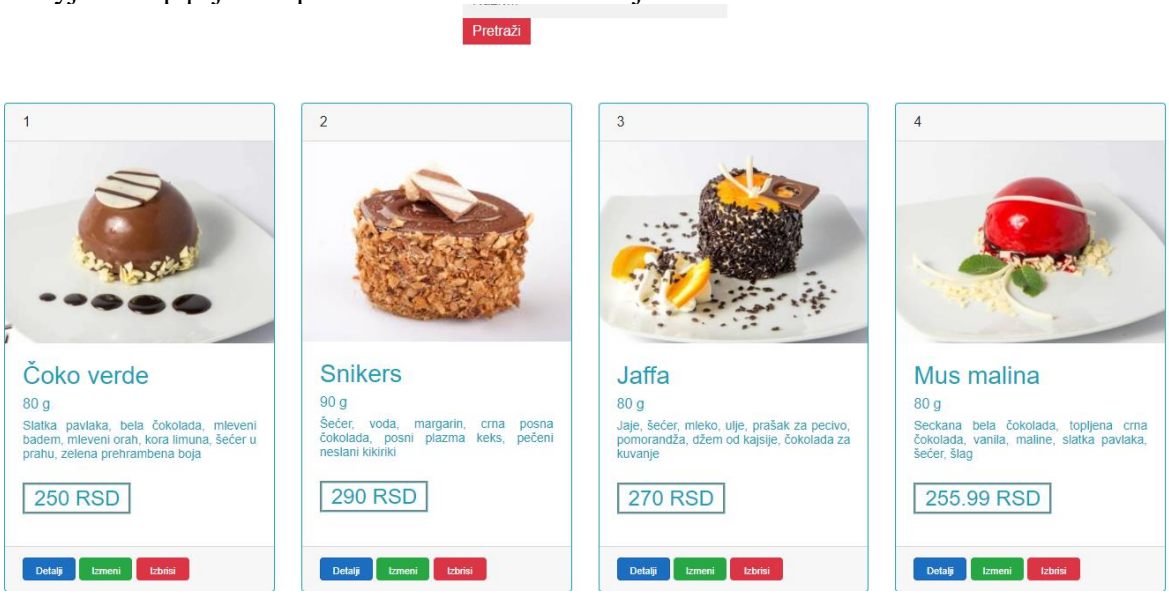
Алтернативна сценарија

5.1 Уколико систем не може да прихвати податке о новим колачима, он **приказује** администратору поруку „Није могуће унети нове колаче". (ИА)

Слика 146 Неуспешан унос нових колача

СК7: Случај коришћења – Брисање колача

Назив СК
Брисање колача
Актори СК
Администратор
Учесници СК
Администратор и систем (програм)
Предуслов: Систем је укључен и администратор је пријављен под својом шифром. Систем приказује интерфејс за брисање колача. Учитана је листа свих колача.



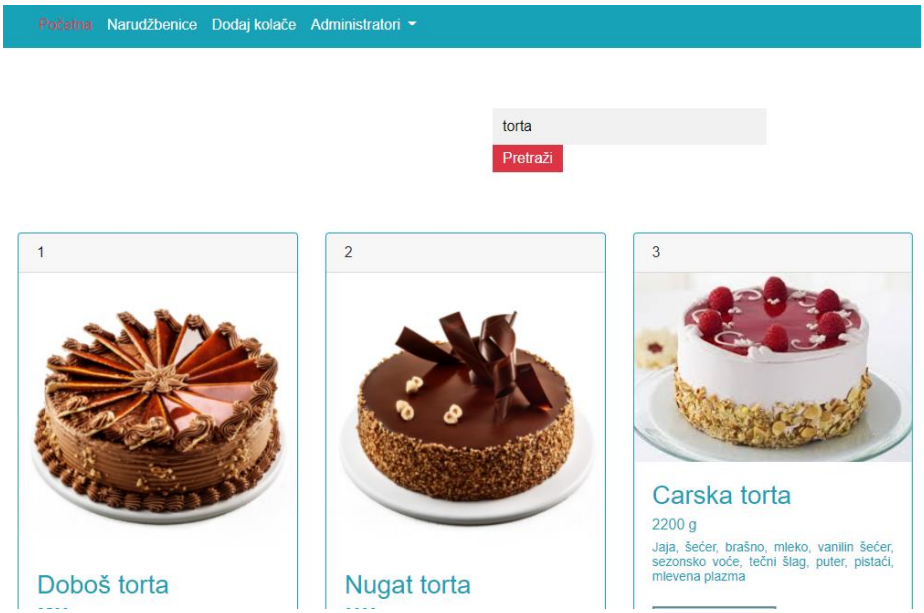
Слика 147 Интерфејс за брисање колача

Основни сценарио СК

1. Администратор **уноси** вредност по којој претражује колаче. (АПУСО)
2. Администратор **проверава** да ли је коректно унео вредност по којој претражује. (АНСО)
3. Администратор **позива** систем да нађе колаче на основу унете вредности. (АПСО)

Опис акције: Кликом на дугме „*Pretraži*“ администратор позива системску операцију GetAll(Criteria, List<Cookie>).

4. Систем **тражи** колаче по наведеном критеријуму. (СО)
5. Систем **приказује** листу тражених колача. (ИА)

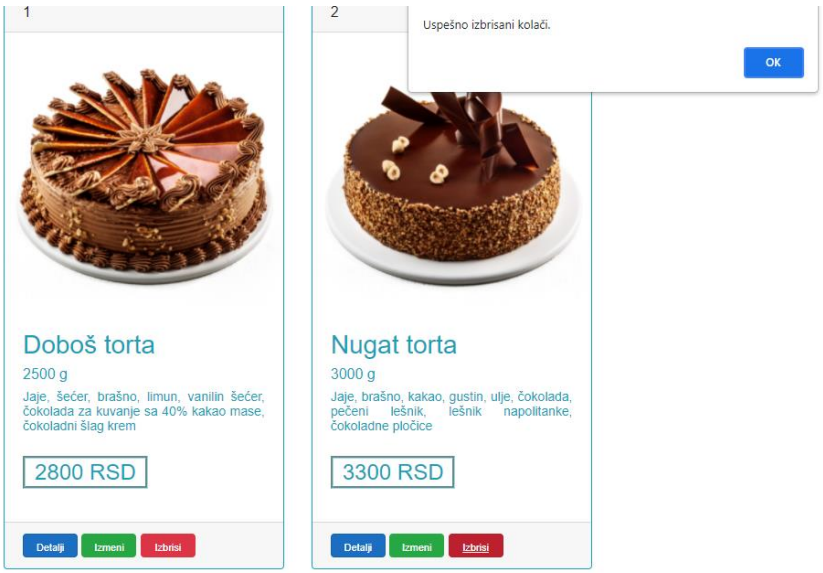


Слика 148 Приказ листе тражених колача

6. Администратор **бира** колаче који жели да избрише. (АПУСО)
7. Администратор **позива** систем да избрише податке о одабраним колачима. (АПСО)

Опис акције: Кликом на дугме „*Izbriši“* администратор позива системску операцију Delete(Cookie).

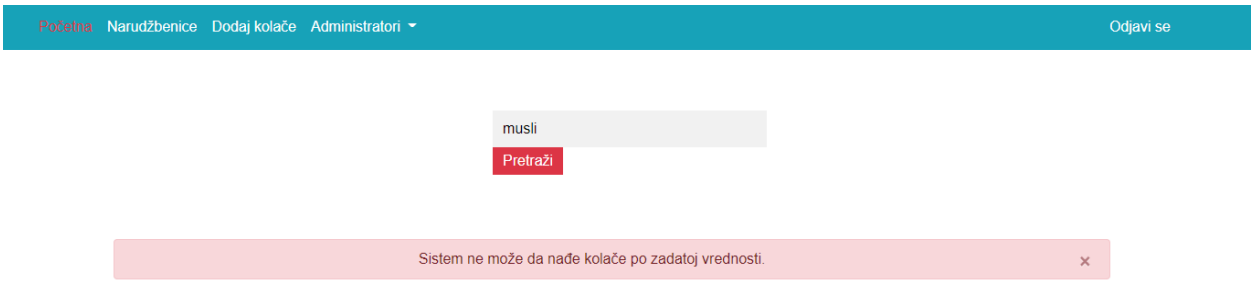
8. Систем **брише** податке о колачима. (СО)
9. Систем **приказује** администратору листу постојећих производа и поруку: „Успешно избрисани колачи“. (ИА)



Слика 149 Успешно брисање колача

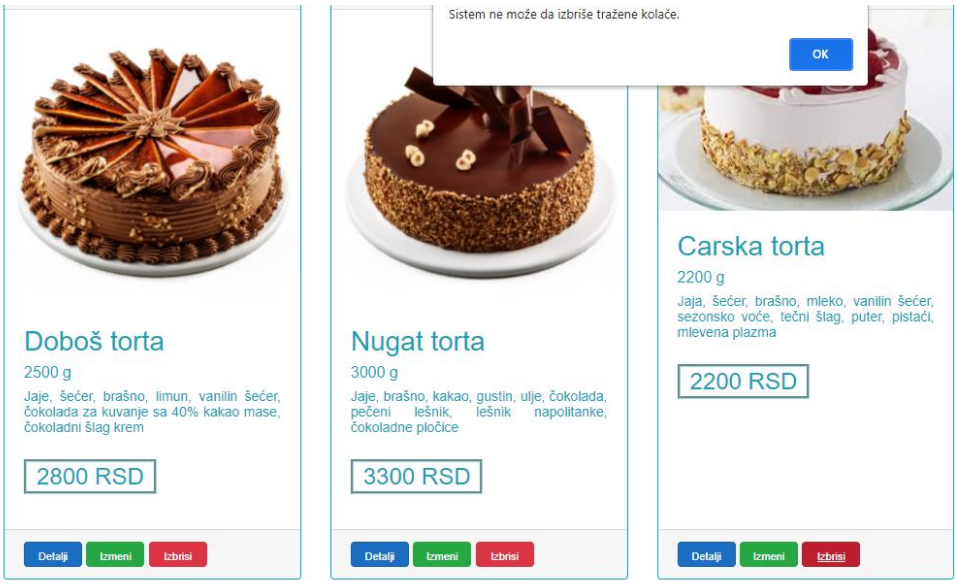
Алтернативна сценарија

5.1 Уколико систем не може да нађе колаче по унетом критеријуму он **приказује** кориснику поруку: “Систем не може да нађе колаче по задатој вредности.”. Прекида се извршење сценарија. (ИА)



Слика 150 Неуспешно претраживање колача

9.1 Уколико систем није успешно избрисао тражене колаче, он **приказује** поруку кориснику: „Систем не може да избрише тражене колаче“. (ИА)



Слика 151 Неуспешно брисање колача

СК8: Случај коришћења – Измена колача

Назив СК
Измена колача
Актори СК
Администратор
Учесници СК
Администратор и систем (програм)
Предуслов: : Систем је укључен и администратор је пријављен под својом шифром. Систем приказује интерфејс за измену колача.

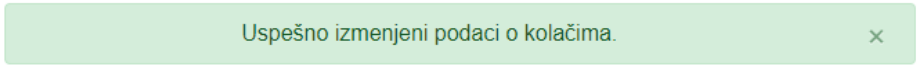
Слика 152 Форма за измену података о колачима

Основни сценарио СК

1. Систем **тражи** податке о колачима. (СО)
2. Систем **приказује** администратору пронађене податке. (ИА)
3. Администратор **бира** податке који жели да измени. (АПУСО)
4. Администратор **мења** податке. (АПУСО)
5. Администратор **проверава** да ли је коректно унео нове податке. (АНСО)
6. Администратор **позива** систем да промени податке. (АПСО)

Опис акције: Кликом на дугме „*Izmeni podatke"* администратор позива системску операцију Update(Cookie).

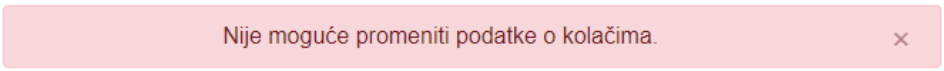
7. Систем **мења** податке о колачима. (СО)
8. Систем **приказује** администратору поруку: „Успешно измењени подаци о колачима!". (ИА)



Слика 153 Успешна измена колача

Алтернативна сценарија

8.1 Уколико систем није у могућности да промени податке, **приказује** поруку: „Није могуће променити податке о колачима.“ (ИА)



Слика 154 Неуспешна промена података о колачима

СК9: Случај коришћења – Унос новог администратора

Назив СК
Унос новог администратора
Актори СК
Администратор
Учесници СК
Администратор и систем (програм)
Предуслов: Систем је укључен и приказана је форма за унос новог администратора. Учитана је листа свих градова.

Слика 155 Форма за унос новог администратора

Основни сценарио СК

1. Администратор **уноси** основне податке за новог администратора. (АПУСО)
2. Администратор **проверава** да ли је коректно унео податке за новог администратора. (АНСО)
3. Администратор **позива** систем да запамти новог администратора. (АПСО)

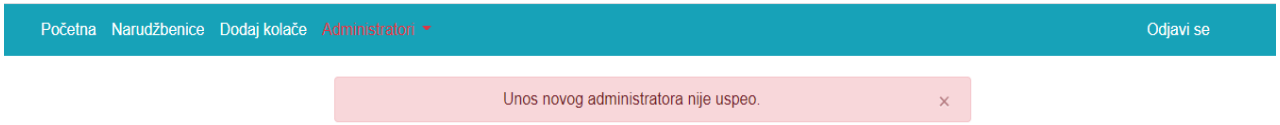
Опис акције: Кликом на дугме „*Registrujte admina*“ администратор позива системску операцију Register(Administrator).

4. Систем **памти** новог администратора. (СО)
5. Систем **приказује** администратору поруку: „Нови администратор је унет у базу!”. (ИА)

Слика 156 Успешан унос новог администартора

Алтернативна сценарија

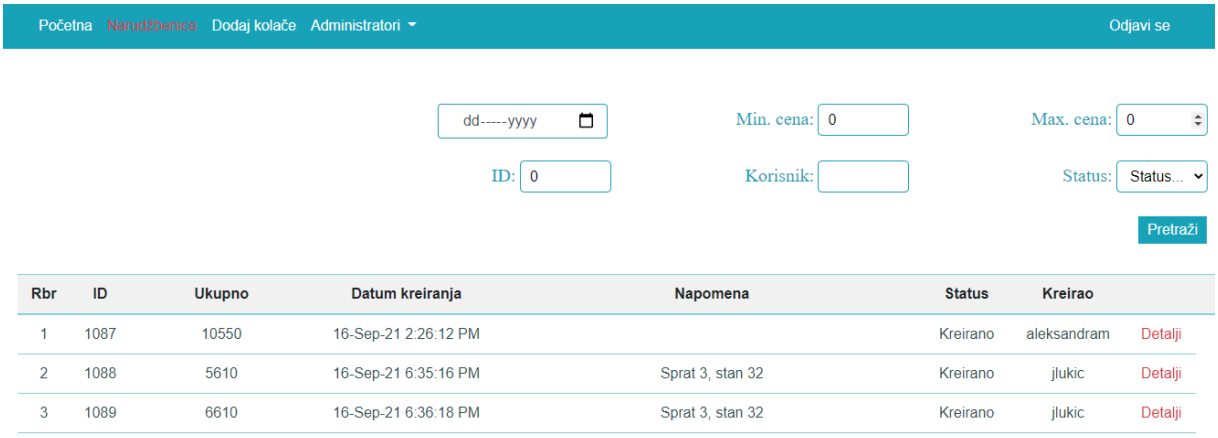
5.1 Уколико унос новог администратора није могућ, систем **приказује** администратору поруку: „Унос новог администратора није успео!“. (ИА)



Слика 157 Неуспешан унос новог администратора

СК10: Случај коришћења – Претрага наруџбеница

Назив СК
Претрага наруџбеница
Актори СК
Администратор
Учесници СК
Администратор и систем (програм)
Предуслов: Систем је укључен и администратор је пријављен под својом шифром. Систем приказује интерфејс за претрагу наруџбеница. Учитана је листа свих наруџбеница.



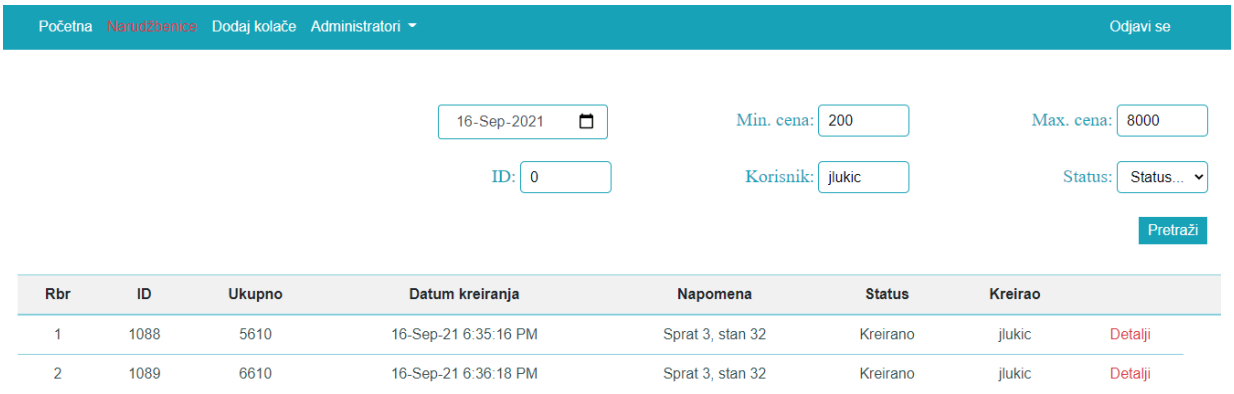
Слика 158 Форма за претрагу наруџбеница

Основни сценарио СК

1. Администратор **уноси** критеријум по ком претражује наруџбенице. (АПУСО)
2. Администратор **проверава** да ли је коректно унео критеријум по ком претражује. (АНСО)
3. Администратор **позива** систем да нађе наруџбенице на основу унетог критеријума. (АПСО)

Опис акције: Кликом на дугме „*Pretraži*“ администратор позива системску операцију GetAll(Criteria, List<Order>).

4. Систем **тражи** наруџбенице по наведеном критеријуму. (СО)
5. Систем **приказује** листу одговарајућих наруџбеница. (ИА)

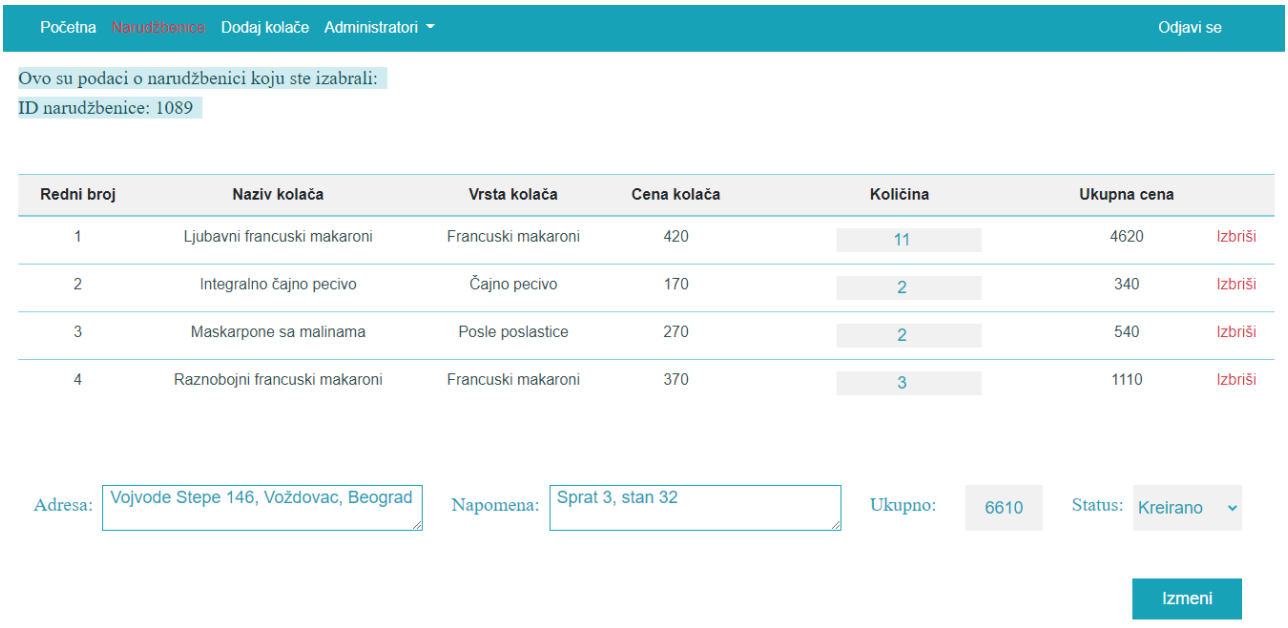


Слика 159 Листа тражених наруџбеница

6. Администратор **бира** наруџбеницу. (АПУСО)
7. Администратор **позива** систем да прикаже све податке о изабраној наруџбеници. (АПСО)

Опис акције: Кликом на дугме „*Detalji*“ администратор позива системску операцију GetOrderItems(Order).

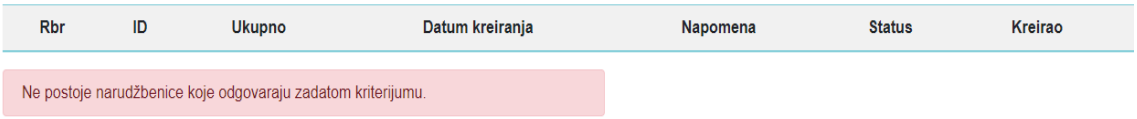
8. Систем **тражи** податке о наруџбеници. (СО)
9. Систем **приказује** кориснику све податке о наруџбеници уз поруку: “Ово су подаци о наруџбеници коју сте изабрали”. (ИА)



Слика 160 Успешно приказани подаци о наруџбеници

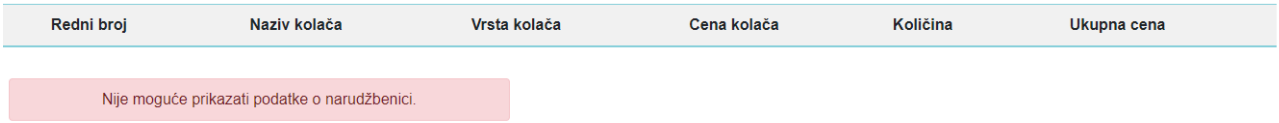
Алтернативна сценарија

5.1 Уколико систем не може да нађе наруџбенице по унетом критеријуму он **приказује** адмнистратору поруку: „Не постоје наруџбенице које одговарају задатом критеријуму.“ Прекида се извршење сценарија. (ИА)



Слика 161 Неуспешна претрага наруџбеница

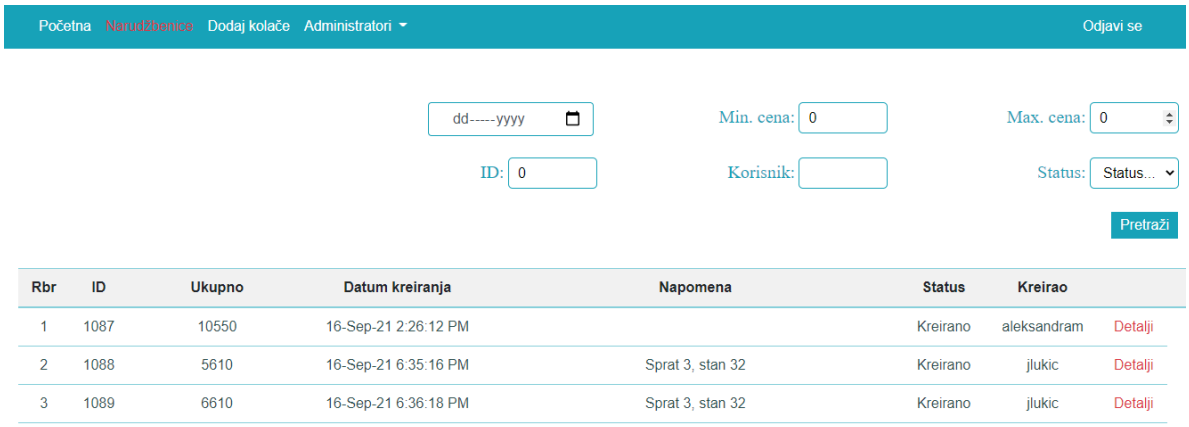
9.1 Уколико систем не може да прикаже податке о одабраној наруџбеници, он **приказује** администратору поруку: „Није могуће приказати податке о наруџбеници.“.



Слика 162 Неуспешно приказивање података онаруџбеници

СК11: Случај коришћења – Измена наруџбенице

Назив СК
Измена наруџбенице
Актори СК
Администратор
Учесници СК
Администратор и систем (програм)
Предуслов: Систем је укључен и администратор је пријављен под својом шифром. Систем приказује интерфејс за измену наруџбеница. Учитана је листа свих наруџбеница.



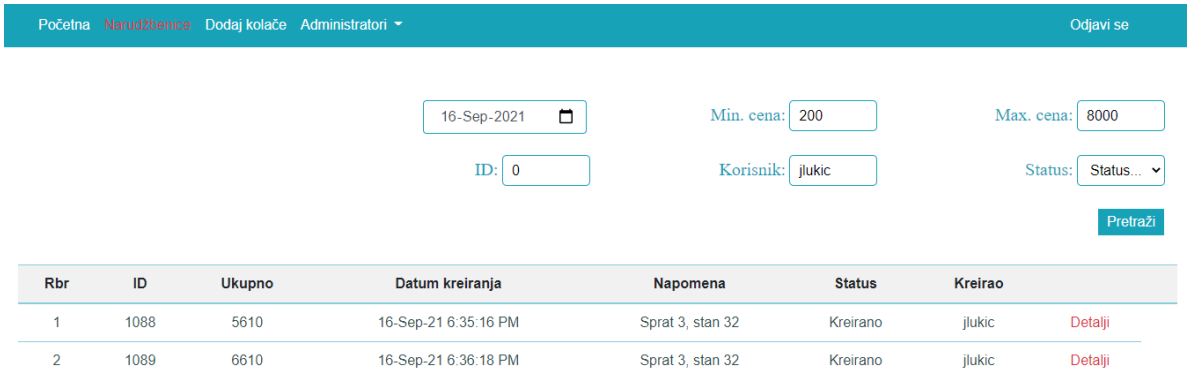
Слика 163 Листа наруџбеница

Основни сценарио СК

1. Администратор **уноси** критеријум по ком претражује наруџбенице. (АПУСО)
2. Администратор **проверава** да ли је коректно унео критеријум по ком претражује. (АНСО)
3. Администратор **позива** систем да нађе наруџбенице на основу унетог критеријума. (АПСО)

Опис акције: Кликом на дугме „*Pretraži*“ администратор позива системску операцију GetAll(Criteria, List<Order>).

4. Систем **тражи** наруџбенице по наведеном критеријуму. (СО)
5. Систем **приказује** листу одговарајућих наруџбеница. (ИА)

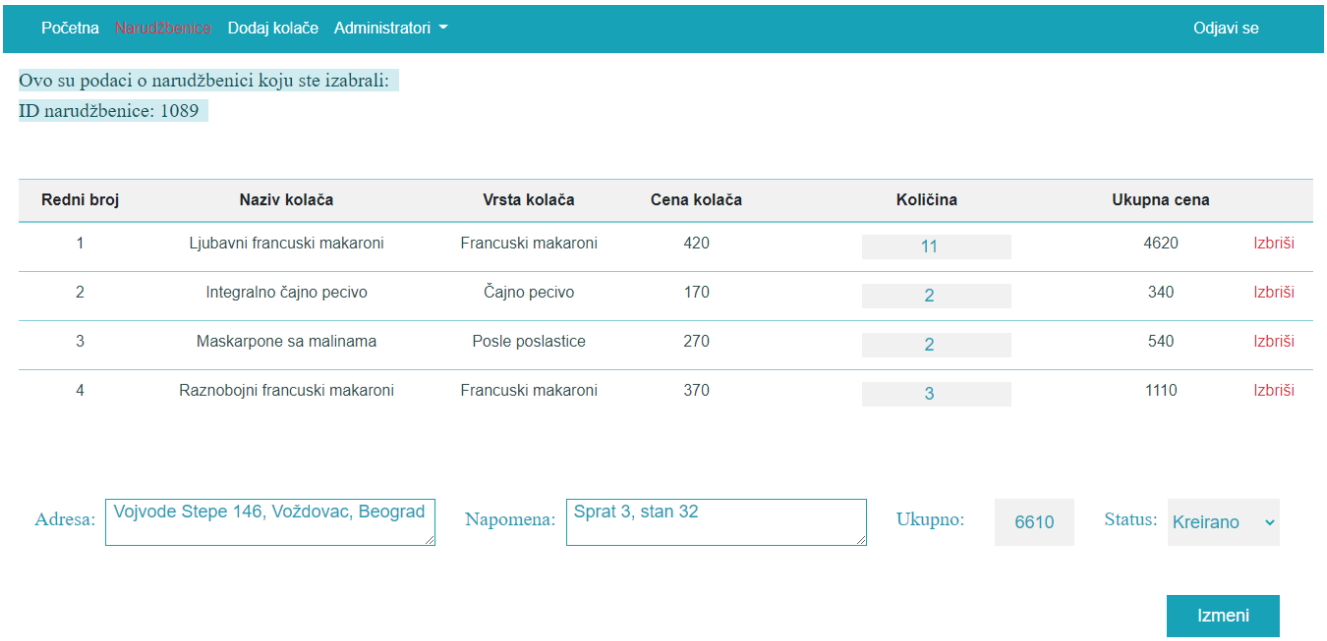


Слика 164 Листа тражених наруџбеница

6. Администратор **бира** наруџбеницу. (АПУСО)
7. Администратор **позива** систем да прикаже све податке о изабраној наруџбеници. (АПСО)

Опис акције: Кликом на дугме „*Detalji*“ администратор позива системску операцију GetOrderItems(Order).

8. Систем **тражи** податке о наруџбеници. (СО)
9. Систем **приказује** кориснику све податке о наруџбеници уз поруку: “Ово су подаци о наруџбеници коју сте изабрали”. (ИА)



Слика 165 Успешно приказани подаци о наруџбеници

10. Администратор **мења** наруџбеницу. (АПУСО)
11. Администратор **проверава** да ли је коректно унео нове податке о наруџбеници. (АНСО)
12. Администратор **позива** систем да промени податке. (АПСО)

Опис акције: Кликом на дугме „*Izmeni*“ администратор позива системску операцију Update(Order).

13. Систем **мења** податке о наруџбеници. (СО)
14. Систем **приказује** администратору поруку: „Успешно сте променили податке о наруџбеници!”. (ИА)

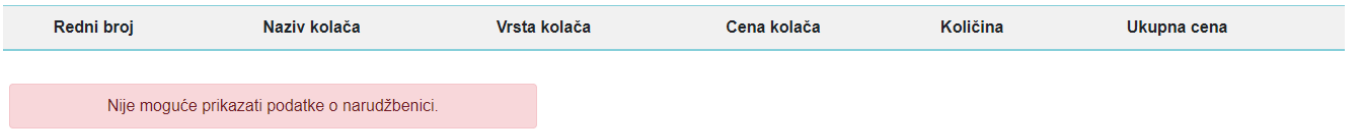
Слика 166 Успешно измењена наруџбеница

Алтернативна сценарија

5.1 Уколико систем не може да нађе наруџбенице по унетом критеријуму он **приказује** адмнистратору поруку: “Не постоје наруџбенице које одговарају задатом критеријуму”. Прекида се извршење сценарија. (ИА)

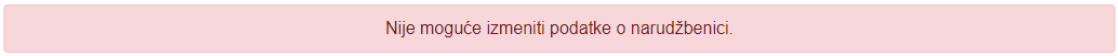
Слика 167 Неуспешно претраживање наруџбеница

9.1 Уколико систем не може да прикаже податке о одабраној наруџбеници, он **приказује** администратору поруку: „Није могуће приказати податке о наруџбеници“. Прекида се извршење сценарија. (ИА)



Слика 168 Неуспешно приказивање података о наруџбеници

14.1. Уколико систем није у могућности да промени податке о одабраној наруџбеници, он **приказује** администратору поруку: “Није могуће променити податке о наруџбеници”. (ИА)



Слика 169 Неуспешна измена података о наруџбеници

5.4.4 Имплементација складишта података

На основу доменских класа софтвера пројектоване су табеле (складишта података) релационог модела за управљање базом података. Систем за управљање базом података који је коришћен у завршном раду је *SqlServer*.

	Name	Data Type	Allow Nulls
🔑	CityID	int	☐
	PostalCode	int	☐
	CityName	nvarchar(MAX)	☐

Слика 170 Табела Cities

	Name	Data Type	Allow Nulls
🔑	CookieID	int	☐
	CookieName	nvarchar(MAX)	☐
	Price	float	☐
	Weight	int	☐
	AvailableQuantity	int	☐
	Description	nvarchar(MAX)	☑
	CookieTypeID	int	☐
	ImageUrl	nvarchar(MAX)	☑

Слика 171 Табела Cookies

	Name	Data Type	Allow Nulls
🔑	CookieTypeID	int	☐
	TypeName	nvarchar(MAX)	☐

Слика 172 Табела CookieTypes

	Name	Data Type	Allow Nulls
🔑	OrderItemID	int	☐
🔑	OrderID	int	☐
	TotalItemAmount	float	☐
	Quantity	int	☐
	CookiePrice	float	☐
	CookieID	int	☐

Слика 173 Табела OrderItem

	Name	Data Type	Allow Nulls
🔑	OrderID	int	☐
	TotalAmount	float	☐
	CreatedOn	datetime2(7)	☐
	Status	int	☐
	UserPersonID	int	☐
	Remark	nvarchar(MAX)	☑
	Address	nvarchar(MAX)	☐

Слика 174 Табела Orders

	Name	Data Type	Allow Nulls
🔑	PersonID	int	☐
	JMBG	nvarchar(MAX)	☐
	FirstName	nvarchar(MAX)	☐
	LastName	nvarchar(MAX)	☐
	Username	nvarchar(MAX)	☐
	Password	nvarchar(MAX)	☐
	CityID	int	☐
	Discriminator	nvarchar(MAX)	☐
	Email	nvarchar(MAX)	☑

Слика 175 Табела Persons

5.5 Фаза тестирања

Фаза тестирања представља пету и последњу фазу Ларманове методе. У њој се тестирају имплементационе компоненте тако што се за сваку од њих праве тест компоненте, тест случајеви и тест процедуре. Коришћењем патерна у развоју неког софтвера, сложеност система се повећава. Самим тим, брзина извршавања програма се знатно смањује, а тестирање се отежава. [16]

Апликација је тестирана покретањем и уношењем неисправних података, као и оних које због одређених услова систем не би требало да прихвати. Подразумева се да су унесени и правилни подаци, како би се тестирали и презентовали сви већ наведени случајеви коришћења. Тестирањем су примећене и отклоњене одређене грешке и сваки случај коришћења је у потпуности имплементиран. Након одређеног броја тестова и уношења различитих података, закључак је да апликација правилно функционише и испуњава све задате захтеве.

6. Закључак

Кроз рад „Развој софтверског система за праћење рада посластичарнице применом *ASP.NET Core* оквира“ описан је целокупан поступак креирања веб апликације.

Објашњене су технологије које су коришћене за развој софтвера. Дат је преглед *.NET* платформи, *ASP.NET Core* технологије, *Entity Framework Core-a*, *MVC* архитектуре. Објашњене су предности објектно-оријентисаних програмских језика, C# као програмски језик који је коришћен, *LINQ* упити који представљају незаобилазни део сваке веб апликације. Теоријски је објашњена имплементација апстрактног слоја и приказани су примери програмског кода. Ради прегледности кода и постизања независности логике приступа подацима од апликационог слоја, коришћени су патерни – *Repository* и *UnitOfWork*. Приказане су основне функционалности апликације, прости и сложени случајеви коришћења, пројектовање и имплементација корисничког интерфејса, изглед складишта података и друге потребне ставке у Ларамановој методи развоја софтвера.

У фази тестирања утврђено је да апликација у потпуности имплементира сваки случај коришћења и задовољава корисничке захтеве. Ипак, увек има простора за побољшање апликације. Оно би могло бити остварено кроз слање нотификација корисницима уколико се уведе нова врста посластица, увођење каталога у ком се налазе производи који су на попусту, омогућавање корисницима да остављају коментаре у којима изражавају своје задовољство/незадовољство услугом и многе друге.

За израду саме апликације коришћене су *.NET* технологије, са којима сам се први пут сусрела током завршне године студија на предмету „Напредне *.NET* технологије“. Приликом рада на пројекту за тај предмет, први пут сам заволела неки програмски језик и пожелела да посветим доста времена учењу и усавршавању у тој области.

7. Литература

[1] *Entity Framweork Core.* (2020, Септембар 20). Преузето са docs.microsoft.com: https://docs.microsoft.com/en-us/ef/core/

[2] *.Net Core vs. Net Framework*. (.). Преузето са https://hr.education-wiki.com: https://hr.education-wiki.com/2427394-.net-core-vs-.net-framework

[3] *Architecture of .NET Framework*. (.). Преузето са https://dotnet.microsoft.com/: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework

[4] Aroraa, G., & Chilberto, J. (2019). *C# и .NET Core пројектни обрасци.* Београд: Компјутер библиотека.

[5] *ASP.NET*. (2021, Фебруар 13). Преузето са sr.wikipedia.org: https://sr.wikipedia.org/sr-ec/ASP.NET

[6] *ASP.NET MVC Controller Overview (C#)*. (2008, Фебруар 16). Преузето са docs.microsoft.com: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/controllers-and-routing/aspnet-mvc-controllers-overview-cs

[7] *C#*. (2021, Јул 12). Преузето са sr.wikipedia.org: https://sr.wikipedia.org/sr-ec/C_Sharp

[8] *C# курс програмирања*. (.). Преузето са www.it-akademija.com: https://www.it-akademija.com/introduction-to-programming-in-c-sharp-kurs-programiranja

[9] *Entity Framework Core*. (2020). Преузето са www.entityframeworktutorial.net: https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx

[10] *How to implement Repository & Unit of Work design patterns in .NET Core with practical examples [Part 1]*. (2021, Јануар 7). Преузето са https://enlabsoftware.com/: https://enlabsoftware.com/development/how-to-implement-repository-unit-of-work-design-patterns-in-dot-net-core-practical-examples-part-one.html

[11] *http://www.ekof.bg.ac.rs*. (2015). Преузето са http://www.ekof.bg.ac.rs/wp-content/uploads/2014/05/PI2015-CAS06-Programski-jezici.pdf

[12] *LINQ - Entities*. (.). Преузето са www.tutorialspoint.com: https://www.tutorialspoint.com/linq/linq_entities.htm

[13] *Understanding and Implementing UnitOfWork pattern in ASP.NET Core*. (2020, Maj 28). Преузето са referbruv.com: https://referbruv.com/blog/posts/understanding-and-implementing-unitofwork-pattern-in-aspnet-core

[14] *What is the MVC pattern?* (2020, Децембар 2). Преузето са docs.microsoft.com: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-5.0

[15] Величковић, З. (2021). *.NET технологије*. Преузето са https://vtsnis.edu.rs/: https://vtsnis.edu.rs/wp-content/plugins/vts-predmeti/uploads/1_UVOD_u_DOT_NET_2020_21.pdf

[16]Влајић, С. (2015). *Пројектовање софтвера (скрипта).* Београд: Факултет организационих наука.

[17]Вукићевић, Н. (2020, Март 4). *Увод у објектно оријентисано програмирање*. Преузето са codeblog.rs: https://www.codeblog.rs/clanci.php?p=uvod_u_objektno_orijentisano_programiranje

[18]Вучковић, М., Турајлић, Н., & Петровић, М. (2010). *Објектно-оријентисани приступ.* Преузето са is.fon.bg.ac.rs: http://is.fon.bg.ac.rs/wp-content/uploads/2020/04/Objektno-orijentisani-pristup.pdf

[19]Думанчић, И. (2018). *Веб апликација за преглед и уношење интерних евиденцијских листи.* Загреб: Високо училиште Алгебра.

[20]Илић, Н. (2018, Октобар). *Имплементација друштвене мреже Birdtouch коришћењем Xamarin технологије и REST сервиса.* Преузето са www.pmf.ni.ac.rs: https://www.pmf.ni.ac.rs/download/master/master_radovi_ra%C4%8Dunarske_nauke/ra%C4%8Dunarske_nauke_master_radovi/2018/2018-10-31-in.pdf

[21]*Клијентске и серверске технологије Web-а*. (.). Преузето са casoviracunarstva.wordpress.com: https://casoviracunarstva.wordpress.com/2013/03/03/klijentske-i-serverske-tehnologije-web-a/

[22]Митковић, Н. (2019, Јун 15). *Објектно оријентисано програмирање*. Преузето са https://cubes.edu.rs/: https://cubes.edu.rs/sr/30/obuke-i-kursevi/sta-je-objektno-orijentisano-programiranje

[23]*Објектно оријентисано програмирање*. (2019, Фебруар 20). Преузето са www.ucim-programiranje.com: https://www.ucim-programiranje.com/2019/02/objektno-orijentisano-programiranje/

[24]*Основе .NET платформе*. (.). Преузето са http://weblibrary.apeiron-uni.eu:8080/: http://weblibrary.apeiron-uni.eu:8080/WebDokumenti/4794-ip.pdf

[25]Пајевић, И. (2015). *Обезбеђивање комуникације веб апликације коришћењем SQL упита и развој апликације за електронско пословање.* Преузето са www.racunarstvo.matf.bg.ac.rs: http://www.racunarstvo.matf.bg.ac.rs/MasterRadovi/2013_03_21_Irena_Pajevic/rad.pdf

[26]Пековић, С. (2014, Децембар 3). *Развој програмских језика*. Преузето са https://stefanpeka121.wordpress.com/: https://stefanpeka121.wordpress.com/razvoj-programskih-jezika/

[27]Радовановић, М. (2018, Април 17). *Увод у LINQ*. Преузето са www.manuelradovanovic.com: https://www.manuelradovanovic.com/2018/04/uvod-u-linq.html