УНИВЕРЗИТЕТ У БЕОГРАДУ
ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

ЗАВРШНИ РАД

Тема: Развој софтверског система за студентски центар применом ASP.NET Core и Ionic оквира

Ментор:
проф. др Саша Лазаревић

Студент:
Николина Пашић 264/17

Београд, септембар 2021. године

УНИВЕРЗИТЕТ У БЕОГРАДУ
ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

ЗАВРШНИ РАД

Тема: Развој софтверског система за студентски центар применом ASP.NET Core и Ionic оквира

Ментор:
проф. др Саша Лазаревић

Студент:
Николина Пашић 264/17

Београд, септембар 2021. године

Развој софтверског система за студентски центар применом ASP.NET Core и Ionic оквира

Сведоци смо доба у којем живимо и чињенице да су мобилне апликације и сервиси све више укључени у нашу свакодневницу, испуњавајући наше потребе за информацијама, комуникацијом, забавом и разонодом. Посебно у последњих неколико година, употреба мобилних телефона еволвирала је од чисте радозналости до апсолутне потребе у данашњем, дигитално повезаном, свету. Утицај и коришћење мобилних телефона, посебно паметних телефона изразито су доминантни међу студентском популацијом. Због ове чињенице, интегрисање мобилних технологија у комуникационе планове и стратегије постало је императив за образовне институције данашњице.

Рад представља развој софтверског система за студентски центар уз коришћење ASP.NET Core развојног оквира и EntityFrameworkCore објектно - релационог мапера на серверској страни и комбинације Angular и Ionic оквира на клијентској страни.

Описан је процес развоја софтверског система помоћу упрошћене Ларманове методе, што је омогућило да развој апликације буде испраћен одговарајућом документацијом кроз све фазе развоја: прикупљање захтева, анализа, пројектовање, имплементација и тестирање.

Кључне речи: софтверски систем, мобилна апликација, Ларманова метода, .NET Core, ASP.NET Core, EntityFramework Core, Angular, Ionic

Садржај

Списак слика*:*

Списак табела:

1. Увод

Мобилне апликације су присутне на тржишту већ дужи временски период. На њиховом почетку, крајем прошлог века, њихова најчешћа сврха била је обављање одређених једноставних задатака. Касније, произвођачи мобилних телефона развијали су мобилне апликације како би се њихови уређаји истакли у мору осталих на тржишту мобилних телефона. Данас, када се у конкуренцији налази прегршт различитих апликација, јако је битно обратити пажњу да апликација која се креира има смисла или је пожељна онда када је њен циљ активно ангажовање корисника или када је њена функционалност сроднија раду рачунара него што је то случај са веб страницама.

У овом раду мобилна апликација имплементирана је уз помоћ ASP.NET Core-а, комбинације Angular и Ionic оквира и заснована је на *REST* архитектури. ASP.NET је популаран оквир за веб развој који служи за израду веб апликација на .NET платформи. ASP.NET Core је верзија ASP.NET-а отвореног кода која ради на *macOS*-у, *Linux*-у, и *Windows*-у. ASP.NET Core је први пут објављен 2016. године и представља редизајн ранијих верзија програма ASP.NET-а осмишљених за рад само на *Windows* оперативном систему. Angular као оквир пружа бројне значајне предности. Омогућава програмерима да развијају велике апликације на одржив начин. Потпуно је надоградив и добро функционише са другим библиотекама. Ionic оквир је моћна алатка за креирање хибридних мобилних апликација[1]. Наравно да није једини у трци када је у питању програмирање оваквих апликација али је онај који привлачи јако велику пажњу и препоручен је као први избор од стране бројних програмера.

Циљ овог рада је развој софтверског система за студентски центар, развијен коришћењем Ларманове методе развоја софтвера.

Рад је организован у десет поглавља и подељен је на теоријски и практични део.

У другом поглављу описан је теоријски део који почиње описом коришћених технологија. Биће описане већ поменуте технологије коришћене на серверској страни, карактеристике C# програмског језика, затим поменуте технологије коришћене за потребе имплементације система на клијентској страни.

У трећем поглављу почиње опис практичног дела рада. Описана је Ларманова метода развоја софтвера, након чега је кроз низ поглавља детаљно објашњен процес развоја софтверског система коришћењем исте. У четвртом поглављу описан је развој система кроз фазу прикупљања захтева. Затим је у петом поглављу дат приказ развоја софтверског система кроз фазу анализе, док је у шестом поглављу приказан развој помоћу фазе пројектовања. Седмо поглавље описује процес развоја система кроз фазу имплементације, а у осмом је приказана последња фаза развоја софтверског система - фаза тестирања. У деветом поглављу дат је закључак, док је у последњем, десетом поглављу приказана коришћена литература.

[1] Хибридна апликација је она која је написана истом технологијом која се користи за веб странице и мобилне веб имплементације, а која је хостована или ради унутар изворног контејнера на мобилном уређају.

2. Преглед коришћених технологија

Веб технологије можемо поделити у две групе:

- Клијентске (енг. *Client side*) – односе се на све сегменте апликације који се приказују или одвијају на уређају крајњег корисника. Овде убрајамо све што корисник види као и све акције које се одвијају на страни корисниковог претраживача
- Серверске (енг. *Server side*) – извршавају се на удаљеном, дељеном рачунару – серверу и за разлику од клијентских односе се на рад који се одвија иза кулиса како би се исправно управљало подацима

2.1 Серверска страна

У изради студијског примера на серверској страни коришћена је .NET Core платформа и то ASP.NET Core оквир како би се креирао веб API. ASP .NET Core је креиран тако да буде брз, флексибилан, модеран и једноставан за коришћење. За складиштење података коришћена је SQL релациона база података.

2.1.1 .NET платформа

.NET Framework, .NET Core, Xamarin, и .NET Standard су повезане и међусобно преклапајуће платформе које програмери користе за креирање апликација и сервиса. У овом поглављу биће описан сваки од ових .NET концепата.

Три главне и међусобно повезане теме које омогућавају све предности које употреба .NET-а пружа су CLR, CTS, и CLS. Из угла програмера, .NET се може посматрати као

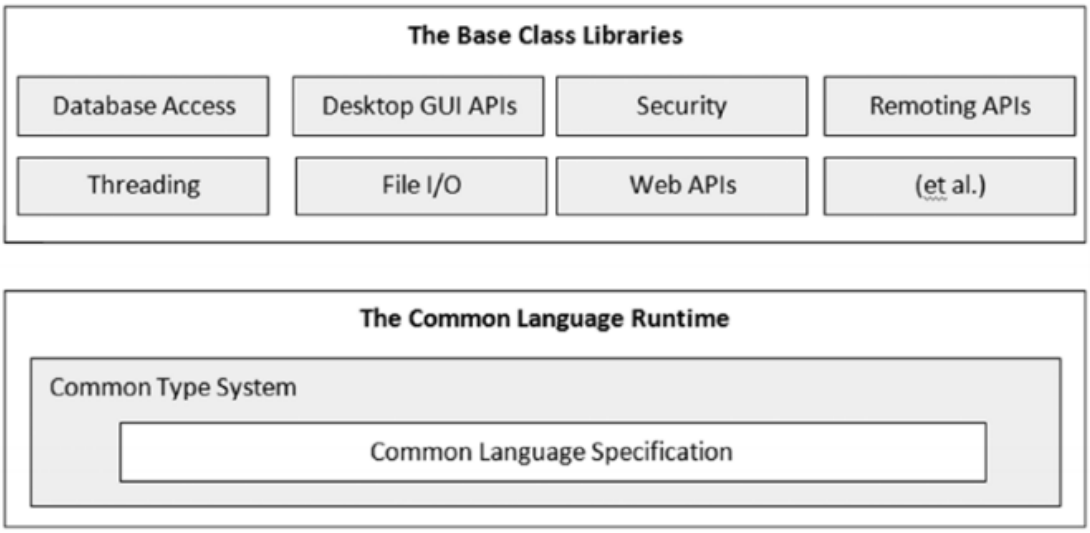
runtime окружење и као разумљива библиотека основних класа. Тзв. *runtime* слој се исправно назива *Common Language Runtime,* или CLR. Његова основна улога је да лоцира, учита и управља .NET објектима уместо програмера. CLR такође води рачуна о бројним детаљима попут управљања меморијом, хостовања апликације, координације нити и основних сигурносних провера.

Други битан сегмент .NET платформе је Common Type System, или CTS. Спецификација CTS-а у потпуности описује све могуће типове података и програмске конструкције подржане од стране runtime-а, специфицира како ентитети могу интераговати једни са другима и бави се детаљима приказивања ентитета у .NET *metadata* формату.

Због чињенице да врло лако може да се деси да неки језик подржан на .NET платформи не подржава свако својство дефинисано креиран је CLS или *Common Language Specification*, који као спецификација дефинише подскуп свих познатих типова и свих програмских конструкција о којима језици подржани .NET платформом морају да се слажу.

2.1.1.1 .NET Framework

.NET Framework је развојна платформа која укључује *Common Language Runtime* [2](CLR), који управља извршавањем кода и *Base Class Library*[3](BCL), која пружа богату библиотеку класа за изградњу апликација. Microsoft је превасходно дизајнирао .NET Framework да има могућност независности од платформе али највећи напор су уложили у компатибилност са Windows оперативним системом. .NET Framework је данас инсталиран на више од милијарду рачунара тако да се промене, чак и најмање могуће обављају под будним оком инжењера. Чак и исправке багова могу проузроковати бројне проблеме тако да се ажурирања врше ретко (Price, 2019).



Слика 1CLR,CTS,CLS, BCL однос

Све апликације на рачунару које су написане у .NET Framework-у деле исту верзију

CLR-а и библиотеке који су складиштени у *Global Assembly Cache*[4]*-y* (GAC), што може довести до проблема ако неки од њих захтевају посебну верзију због компатибилности.

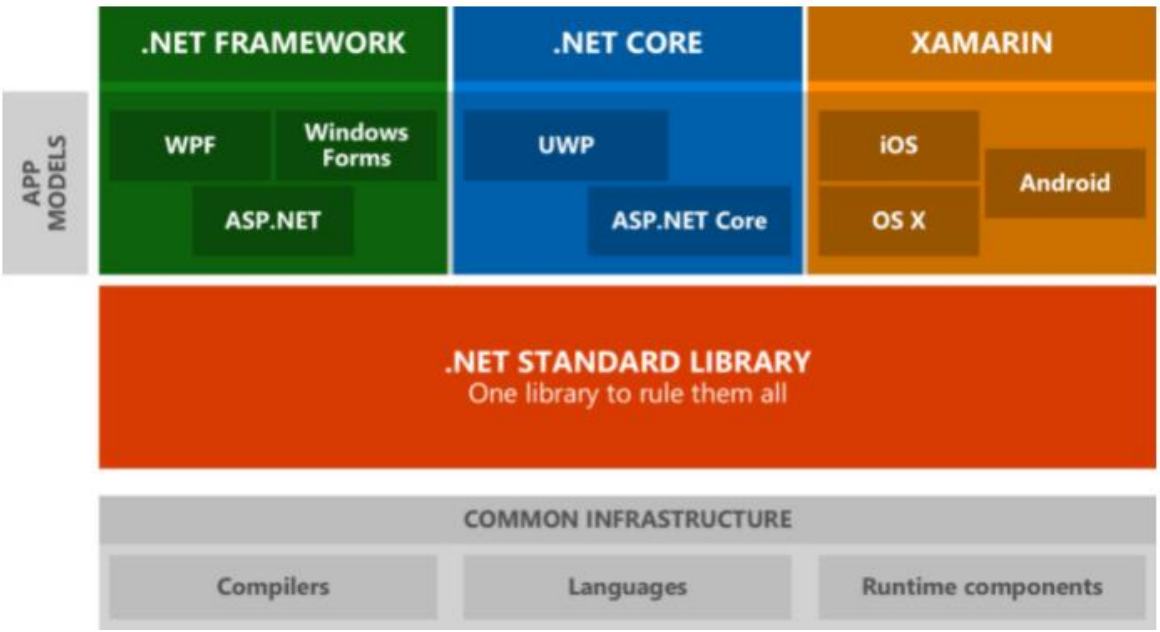
Треће стране су развиле имплементацију .NET Framework-а под називом Mono пројекат. Иако је независна од платформе, Mono изузетно добро функционише иза званичне имплементације .NET Framework-а. Mono је своју нишу нашао као оснивач Xamarin мобилне платформе као и платформи за развој игрица попут Unity-ја.

.NET Standard је базиран на API-ју који је заједнички за све .NET имплементације. Направљен је за успостављање веће униформности у .NET екосистему. Свака имплементација може да има своје API-је који су специфични за оперативни систем на којем требају да се извршавају (Microsoft, 2021).

[2] CLR (Common Language Runtime) је основна компонента Microsoft .NET Framework-а. То је Microsoft-ова имплементација стандарда заједничке језичке инфраструктуре, који дефинише окружење програмског кода.

[3] BCL (Base Class Library) је основна компонента Microsoft .NET Framework-а која укључује *namespace и основне типове у .NET Framework-y*

[4] GAC (Global Assembly Cache) - је целокупни предмеморијски склоп за заједничку језичку инфраструктуру у Мајкрософтовом .NET Framework-у.



Слика 2 Приказ .NET платформе

2.1.1.2 .NET Core

Данас, живимо у свету потупуно независном од платформи у којем су мобилно и клауд програмирање постали приоритетнији од Windows-а као оперативног система. Због тога је *Microsoft* уложио велике напоре да одвоји .NET од његове јаке спреге са Windows-ом. Док су унапређивали .NET, искористили су прилику да преобликују и уклоне велике делове кода који се више нису сматрали кључним.

Овако креиран нови производ брендиран је под називом .NET Core и укључује имплементацију независну од платформе, верзију CLR-а под новим називом CoreCLR и поједностављену библиотеку класа познату као CoreFX. .NET Core се брзо развија јер може да се унапређује упоредо са апликацијом и може да се мења често због чињенице да те промене више неће утицати на друге .NET Core апликације на истој машини. Промене које Microsoft може да уведе за .NET Core не могу да се примене на .NET Framework (Price, 2019).

Рад на додатним оперативним системима осим Windows -а био је довољно висок циљ, али то није био једини циљ креирања нове платформе. Као производни оквир, .NET Core постоји од 2002. године, и много тога се променило од тада. Програмери су постали паметнији, рачунари су постали бржи, захтеви корисника су се повећали, а развој мобилних апликација је постао доминантна сила.

Неки од циљева .NET Core-а:

- Коришћење на различитим оперативним системима
- Високе перформансе
- Портабилне библиотеке класа
- Преносив или самосталан развој
- Пуна подршка командне линије
- Отворени код (*eng. Open source*)

- Интероперабилност са .NET Framework-ом (2.0 верзија .NET Core-а омогућава референцирање библиотека у оквиру .NET Framework-а (Price, 2019)

2.1.1.3 ASP.NET Core

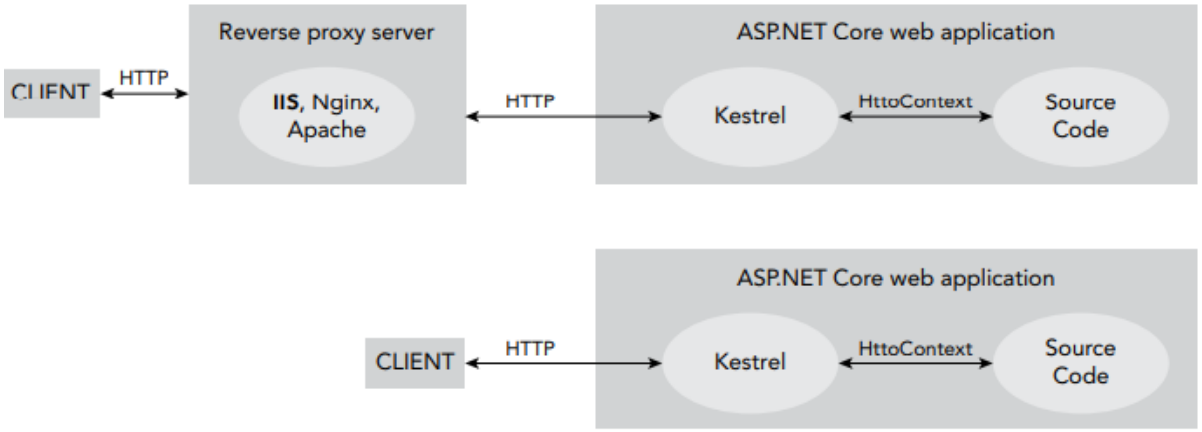
Предности које је .NET Core донео са собом пренесене су и на ASP.NET Core тип апликације такође. Према томе:

- ASP.NET Core је независан од платформе
- ASP.NET Core је независан од IIS-а
- ASP.NET Core не ослања се у потпуности на .NET Framework
- ASP.NET Core је оптимизован за клауд и има боље перформансе

ASP .NET Core омогућава да инсталирамо велики број додатних пакета. На тај начин се побољшавају брзина, перформансе и скалабилност апликације. Омогућава употребу и управљање модерним и популарним оквирима (eng. framework) за кориснички интерфејс као што су AngularJS, Ionic,Umber, Bootstrap и тако даље. Попут .NET Core-а, ASP.NET Core се може користити и на оперативним системима попут MacOS-а и Linux-а.

Историјски, када се причало о било којој апликацији типа ASP.NET без икакве сумње се везало за *Internet Information Services* (IIS). Међутим, ASP.NET Core укључује нови веб сервер под именом Kestrel, тако да се ASP.NET Core може извршавати на IIS-у као обрнути прокси сервер али и у самосталном контејнеру који покреће само *Kestrel*.Самосталан и модуларизован апликациони пакет високих перформанси је оно што се поставља на сервер или клауд платформу за извршавање и употребу. Internet Information Services (IIS) представља веб сервер развијен од стране компаније Microsoft који одговара на HTTP и HTTPS захтеве од клијената. Међутим IIS је немогуће покренути на другим оперативним системима сем Windows-а. Због тога се јавила потреба за развојем већ поменутог Kerstel-а који је веб сервер независан од платформе, укључен у ASP.NET Core пројекте (Benjamin Perkins, 2018).

Као што се може видети на *слици 2*, када је Kerstel конфигурисан за рад са IIS-ом, HTTP захтеви са клијентске стране се просто прослеђују на Kerstel веб сервер. Kerstel интерагује са ASP.NET Core изворним кодом (*енг. Source code*) прослеђујући му HttpContext класу, која садржи специфичне информације о конкретном HTTP захтеву (Benjamin Perkins, 2018).



Слика 3 Рад Kerstela и IIS-a (Benjamin Perkins, 2018)

2.1.2 REST архитектура

Сваки просечан корисник који сурфује интернетом у потрази са подацима о теми која га интересује, усмери свој претраживач на URL адресу сајта који би могао да му пружи одговарајуће податке. На пример: http://www.randomsite.com/. Након тога, корисник добија веб страну, документ у формату HTML-а који његов претраживач графички рендерује. Следећи корак би било визуелно скенирање стране у потрази за пољем за претрагу у које би се унели параметри претраге. Тада претраживач креира други HTTP захтев, до странице која одговара параметрима. На пример: http://www.randomsite.com/s?url=search-alias%3Dstripbooks&field-keywords=web+services.Међутим, поставља се питање, шта се тачно подразумева под HTTP захтевом? HTTP захтев је протокол заснован на документима, у којем клијент ставља документ у „коверту“ и шаље га серверу. Сервер враћа услугу постављањем одговора у „коверту“ и слања га назад клијенту. HTTP има строге стандарде када је изглед „коверте“ у питању.

Основне компоненте сваког HTTP захтева су следеће:

- **Заглавље захтева** (*енг. header*) – обично га чине парови кључ-вредност који се понашају као налепнице које се каче за коверте приликом слања. Постоји листа стандарда HTTP захтева а апликације могу дефинисати и своје сопствене.
- **Тело захтева** (*енг. body*) – често називан документом или репрезентацијом. Чине га подаци који су унутар коверте. Празно тело захтева је карактеристично за GET захтеве у којима су све информације неопходне да се изврши захтев, прослеђене у оквиру заглавља (Leonard Richardson, 2007)

Основне компоненте сваког HTTP одговора су следеће:

- **HTTP статусни код** (*енг. response code)* - нумерички код који говори кориснику да ли је захтев обрађен успешно или неуспешно и упућује га како треба да се односи према коверти и њеном садржају.
- **Заглавље одговора** (*енг. response header*) – функционалност и дефиниција потпуно одговарају заглављу захтева

- **Тело одговора** (*енг. response body*) – као и код тела захтева, односи се на документ који се добија унутар коверте. Тип зависи од тога шта се захтевало приликом слања захтева (Leonard Richardson, 2007)

REST API-ји као одговор морају да врате JSON или XML. Због своје једноставности и лакоће коришћења JavaScript и JSON су постали стандард за модерне API-је.

Статусни кодови нам приказују стање захтева који смо послали са клијентске стране према серверу. Постоје пет категорија статусних кодова који су обавезни у знању код креирања саме веб апликације:

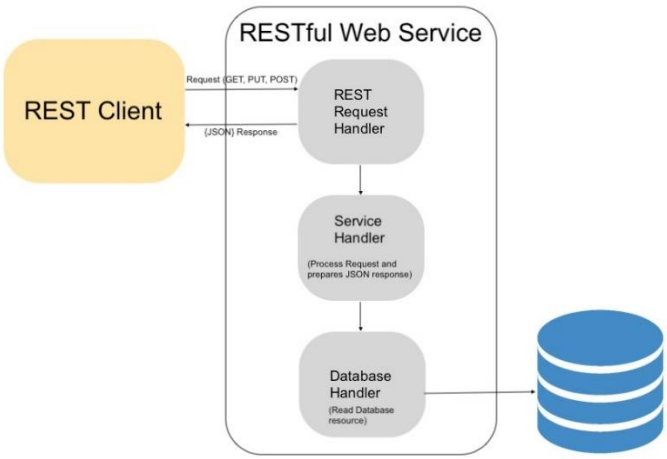
1. 1xx: Информацијски статусни код
2. 2xx: Статусни код успеха
3. 3xx: Статусни код преусмеравања
4. 4xx: Статусни код клијентске грешке
5. 5xx: Статусни код серверске грешке

Када се говори о REST архитектури, мисли се на термин *Representational State Transfer*, који је установио Roy Fielding још 2000. године. REST је тип архитектуре за пројектовање слабо повезаних апликација преко HTTP-а који се изузетно често користи у развоју веб сервиса. REST не форсира никаква правила која се тичу имплементације на нижем нивоу, напротив, поставља путоказе за архитектуру на вишем нивоу а програмеру оставља слободу за саму имплементацију (https://restfulapi.net/, 2017).

REST дефинише шест архитектурних ограничења која чине сваки веб сервис – истинским RESTful API-јем:

- **Клијент-Сервер** - Одвајањем корисничког интерфејса од базе података, побољшава се преносивост корисничког интерфејса на више платформи и побољшава се скалабилност одређених компоненти сервера.
- **Без стања** (*енг. stateless*) - Сваки захтев од клијента ка серверу мора садржати све информације потребне за разумевање захтева и не може се користити било који сачувани контекст на серверу. Стање сесије се због тога у потпуности чува на клијентској страни.
- **Кеширајући** (*енг. cacheable*) - Кеш ограничења захтевају да подаци у одговору на захтев буду имплицитно или експлицитно наведени као кеширајући или некеширајући.
- **Слојни систем** - Слојевити систем омогућава REST архитектури да буде састављена од хијерархијских слојева ограничавањем понашања компоненти тако да свака компонента не може видети изван слоја са којим комуницира. (https://restfulapi.net/, 2017)

RESTful API је API који користи HTTP захтеве уз постојеће HTTP методологије које су дефинисане RFC 2616 протоколом. Користе се захтеви типа GET за проналажење ресурса, PUT I PATCH за промену стања или ажурирање ресурса, POST за креирање новог ресурса, DELETE за уклањање ресурса и слично.



Слика 4 Restful веб сервис

2.1.3 Entity Framework Core

Entity Framework Core, или EF Core, представља библиотеку која омогућава програмерима приступ бази података. Постоји много начина на које је могуће направити такву библиотеку али Entity Framework Core је дизајнирана као објектно-релациони мапер. Објектно-релациони мапери, као што им сам назив каже, мапирају између два света: релационе базе података са сопственим API-јем, и објектно-оријентисаног софтвера за брзи приступ бази података који пишу програмери (Smith, 2018) .Основна предност Entity Framework-а је управо брзо писање приступног кода бази података.

Entity Framework Core, који је први пут објављен 2016. године, је независан од платформе. Може се извршавати на Windows, Linux, али и на MacOS оперативном систему. То чини као део .NET Core иницијативе, отуда и основни део имена “Core”(Али EF Core се може користити и у спрези са .NET Frameworl-ом). EF Core, ASP.NET Core , и .NET Core су такође сви отвореног кода и сваки од њих има страницу са активним проблемима за интеракције са развојним тимовима.

Entity Framework Core није прва верзија Entity Framework-а; постојећа, не-Core, библиотека је позната као EF6.x. Entity Framework Core је заснован на дугогодишњем искуству уграђеном у њега путем повратних информација из свих претходних верзија, 4 до 6.x. Задржао је исту врсту интерфејса као EF6.x, али испод површине има велике промене, попут способности руковања нерелационим базама података за које EF6.x није дизајниран (Smith, 2018).

Следећа слика приказује како EF Core врши поменуто мапирање:

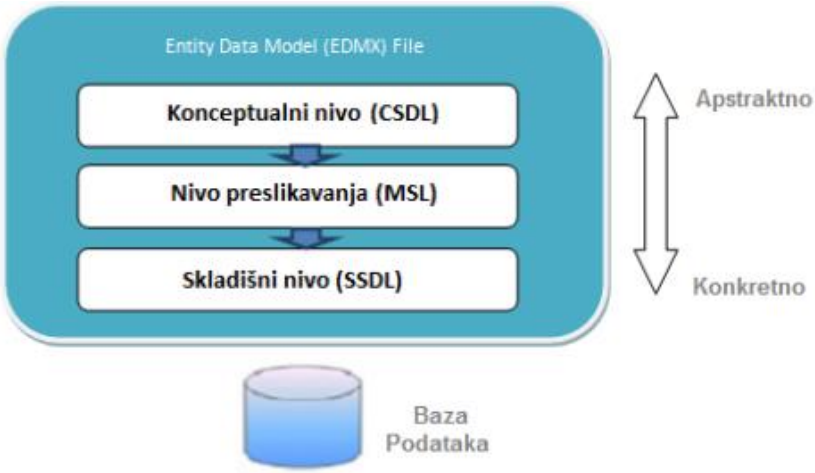
Relational database	.NET software
Table	.NET class
Table columns	Class properties/fields
Rows	Elements in .NET collections—for instance, `List`
Primary keys: unique row	A unique class instance
Foreign keys: define a relationship	Reference to another class
SQL—for instance, `WHERE`	.NET LINQ—for instance, `Where(p => …`

Слика 5 Мапирање између Entity Framework Core-а и .NET софтвера

Језгро Entity Framework- чини модел података (*енг. Entity Data Model -EDM*).Овај модел омогућава прилагођавање пресликавања између објекта класе и конкретне табеле из базе података. Уводи се са намером да се рад са подацима олакша и учини још ефикаснијим.

EDM омогућава програмеру да креира објекат класе који ће што прецизније одговарати домену проблема и састоји се из три одвојена нивоа:

- концептуалног (*енг. conceptual*)
- складишног (*енг. store*)
- пресликавајућег (*енг. mapping*)



Слика 6 Нивои EDM-а

Када се једном имплементира модел, онда се све измене раде над моделом, а не над колонама и редовима из базе података. Да би се у пројекту употребио Entity Framework, потребно је прво конфигурисати тај пројекат у развојном окружењу што најпре подразумева дефинисање EDM-а, дефинисање параметара за повезивање са базом података, као и додавање одређених пројектних референци.

Постоје три начина дефинисања ЕДМ-а:

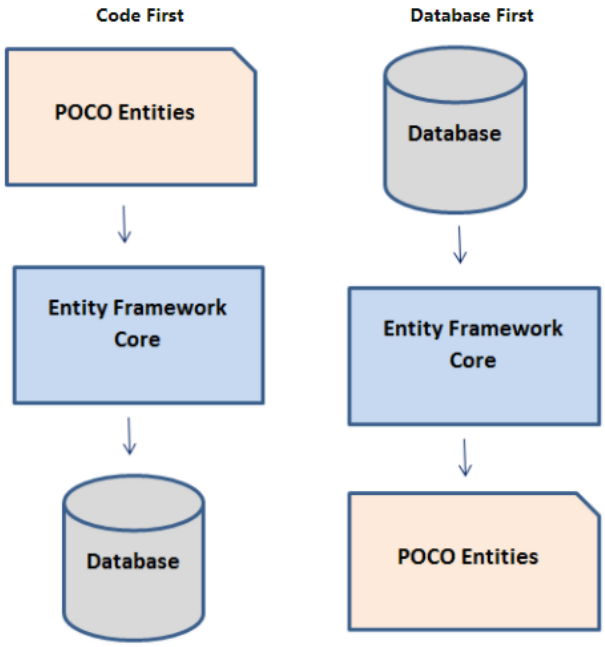
- База-прво (*енг. Database-first*) – подржава Entity Framework Core

- Модел-прво (енг. *Model-first*)

- Кôд -прво (*енг. Code-first*) – подржава Entity Framework Core (Smith, 2018)

База-прво дефинисање EDM-а подразумева генерисање објектних класа на основу постојеће базе података. Модел-прво дефинисање ЕДМ-а омогућава прављење модела података у дизајнеру а затим генерисање базе података на основу направљеног модела.

Поред ова два начина, постоји и трећи начин дефинисања EDM-а, кôд -прво, који омогућава да се направи модел коришћењем POCO класа (*енг. Plain old CLR objects*). Овај начин подразумева да се направе POCO класе које имају исту структуру као и шема базе података са којом се остварује пресликавање.



Слика 7 База-прво и код-прво приступ

Поред ових POCO класа које одговарају табелама из базе података, неопходно је генерисати класу DbContext[5], која ће вршити повезивање POCO класа са ObjectContext-ом. ObjectContext је основна класа за манипулисање упитима и рад са ентитетским подацима као објектима. Entity Framework Core DbContext класа представља једну сесију са базом података и пружа API за комуникацију са базом података уз следеће могућности:

- Конекције са базом (*енг. Database Connections*) – DbContext је одговорна за отварање и управљање конекцијама са базом
- Операције над базом – DbContext пружа методе за додавање, мењање и брисање података директно из базе
- Промене праћења ентитета (*енг. Change Tracking) – Change Tracker* детектује промене које се направе над ентитетом и поставља стање објекта (*Entity state*) на одговарајућу вредност

[5] System.Data.Entity.DbContext

- Изградња модела (*енг. Model building)*-DbContext гради концептуални модел на основу конвенције и конфигурације и мапира га у базу података
- Мапирање података (*енг. Data Mapping)*- DbContext укључује слој мапера података који је одговоран за мапирање резултата SQL упита на инстанце ентитета и других типова дефинисаних у корисничкој апликацији
- Кеширање објекта (*енг. Object caching)*- DbContext омогућава кеширање података добијених извршавањем SQL упита чиме Entity Framework Core смањује отварање беспотребних конекција са базом
- Управљање трансакцијама – Када се позове метода SaveChanges() класе DbContext, креира се трансакција која обавије све промене на чекању као једну јединицу посла (*енг. Unit of work*). Уколико дође до грешке приликом чувања промена, оне се повлаче и база остаје непромењена. (Brind, 2016)

Не би било лоше поменути и пар карактеристика, које бројни програмери сврставају у мане овог мапера. Иако се EF6.x и EF Core чине једноставним за употребу, понекад се деси да „магија“ Entity Framework Core-а проузрокује неочекиване проблеме програмерима. Сервери база података и објектно-релациони софтвер се заснивају на различитим принципима: базе података користе примарне кључеве да дефинишу да је ред јединствен, са друге стране подразумева се да су инстанце .NET класа јединствене по својој референци. Entity Framework Core се успешно избори са већином овог проблема али чист код у .NET класама се „загади“ овим кључевима, чија вредност је пресудна. У већини случајева EF Core ће радити сасвим исправно, али понекад је потребно написати код мало другачије него иначе, како би се задовољила база података. Друга ситуација у којој долази до проблема са објектно-релационим маперима, посебно оним са степеном разумљивости попут Entity Framework Core-а, јесте ситуација у којој мапер сакрије базу толико добро да понекад програмери забораве шта се дешава у позадини. Овај проблем може проузроковати писање кода који ради сјајно у тестној апликацији али су му перформансе у стварној употреби бескорисне, када је база комплексна и постоји много истовремено активних корисника. Поред релационих база података, значајно је истаћи да Entity Framework Core пружа подршку за рад и са нерелационим базама података (*енг. NoSQL databases*).

2.2 Клијентска страна

Програмирање било које нетривијалне апликације данашњице незамисливо је без употребе оквира. За програмирање клијентске стране овог рада, коришћена је комбинација Angular и Ionic оквира. Ionic оквир (*енг. Framework*) је комплет алатки корисничког интерфејса отвореног кода који се користи за изградњу ефикасних, висококвалитетних мобилних и десктоп апликација помоћу веб технологија- HTML, CSS и JavaScript, са интеграцијама за популарне оквире попут Angular-а. Комбинација ова два, изузетно моћна оквира, све чешћи је избор програмера за развијање клијентске стране апликација.

2.2.1 Angular

Angular је нова генерација популарног JavaScript оквира под именом AngularJS 1. Angular је потпуно самосталан оквир за развој веб апликација и подразумевани је везивни оквир за Ionic 4 (Cheng, 2018).

Основни градивни блокови Angular-а су:

• Модули (*енг. Modules*)

• Компоненте (*енг. Components*)

• Шаблони (*енг. Templates*)

• Метаподаци (*енг. Metadata*)

• Везивање података(*енг. Data binding*)

• Директиве (*eng. Directives*)

• Услуге (*енг. Serivices*)

• *Dependency Injection*

2.2.1.1 Модули

Angular модули обезбеђују решење за поделу логике апликације у одвојене секције. Свака апликација мора да има бар један модул, *root module,* којим се *bootstrap*-ује цела апликација.(Bootstraping представља технику производње само-компајлирајућег компајлера).За апликације мањих размера, један модул је углавном довољан, међутим, када су комплексне апликације у питању, уобичајено је да се креира више различитих модула груписаних по њиховим функционалностима. Са тачке имплементације, модул је само класа са *NgModule*[6]-ом. Приликом декларације модула, морају бити обезбеђени неопходни метаподаци које користи NgModule() метода и Angular ће сам направити инстанцу модела. Метаподаци модула се прослеђују као пропертији јединог параметра NgModule() методе. Ово су неке од важних особина NgModule метаподатака (Cheng, 2018):

• Декларације (*енг. Declarations)*- компоненте, директиве и путање које припадају модулу

[6] NgModule (*@NgModule*) – класа која конфигурише инџектор и компајлер и помаже при организацији међусобно повезаних елемената.

• Извози (*енг. Exports*) – сет декларација које се могу користити и у другим модулима

• Увози*(енг. Imports)*- Скуп импортованих модула. Декларације експортованих из импортованих модула се могу користити у конкретном модулу.

• Провајдери *(енг. Providers)* – Скуп сервиса доступних за *dependency injection* у конкретном модулу.

• Улазне компоненте*(енг. entryComponents)* – Скуп компоненти које би требало да се компајлирају када се дефинише модул.

• *bootstrap* – Скуп компоненти које морају да се *bootstrap-yjy* када и модул. Овај проперти се дефинише само за корени модул.

Angular модул:

```
@NgModule({
declarations: [
MyComponent,
],
providers: [
MyService,
],
imports: [
CommonModule,
]
})
export class MyModule {
}
```

2.2.1.2 Компоненте

Компоненте су одговорне за управљање делом корисничког интерфејса апликације. Angular апликација је организована као хијерархијска структура компонената. Компонента има шаблон који се односи на њу. Шаблон служи за дефиницију HTML-а

који ће бити приказан када се компонента рендерује. Рендеровани HTML компоненте обично је зависан од података који се прослеђују на компоненту. Неки делови шаблона могу да користе специјалну синтаксу која омогућава манипулацију подацима који су прослеђени на шаблон. У току рендеровања, ти „плејсхолдери(*енг. placeholder*)“ у шаблону, бивају замењени са вредностима података који долазе са серверске стране или из локалног складишта. Повезивање података (*Data binding*) чини писање динамичног HTML-а елегантним и лаким за одржавање. Директиве се користе да прикаче додатна понашања на стандардне DOM елементе. Структуралне директиве могу да додају, уклоне или замене DOM елементе. На пример, ngFor креира понављајуће елементе на основу низа ставки, док ngIf додаје елементе само ако је испуњен услов. Атрибутске

директиве мењају изглед или понашање елемента. На пример, ngClass додаје додатне CSS класе елементу.

Компонента се креира анотирањем класе декоратором. Декоратор компоненете подржава листу различитих својстава за конфигурисање компоненте, укључујући следеће: (Cheng, 2018)

• template - HTML шаблон у једној линији

• templateUrl- URL ка екстерном шаблону

• selector - CSS селектор за ову компоненту кад се референцира у неким другим шаблонима.

• style - CSS код у једној линији.

• styleUrls – листа URL-ова ка екстерним stylesheet-ovima.

• providers – листа провајдера доступних за конкретну компоненту и њене „*child components*[7]“.

Angular компонента:

```
@Component({
 template: '<p>Hello {{name}}!</p>',
 selector: 'my-component',
})
export class MyComponent {
 name: string = 'Nikolina';
}
```

2.2.1.3 Синтакса шаблона

У Angular-у, могуће је уграђивање различитих типова израза за повезивање података у шаблоне.

• {{expr}} – интерполација стринга

• [name] – везивање вредности својстава, на пример `<img [src]="profileImageUrl">.`

• (event) – везивање догађаја, на пример `<button (click)="save()">Save</button>.`

• [(ngModel)] – двосмерно везивање својстава и догађаја, нпр. `<input [(ngModel)]="username">.`

• [attr.attr-name] – везивање атрибута, на пример `<td [attr.colspan]="colspan"></td>.`

• [class.class-name] – везивање својстава класе, нпр. `<div [class.header]="isHeader"></div>.`

• [style.style-property] – везивање својстава за стилизовање, нпр. `<span [style.color]="textColor"> </span>.`

[7] Child компонента је компонента која се налази унутар друге (Parent) компоненте.

2.2.1.4 Сервиси

За разлику од модула и компоненти, Angular нема дефиницију сервиса. Не постоји ограничење у смислу шта може да буде сервис. Може бити и класа и вредност.Сервиси су дизајнирани да енкапсулирају логику која је невезана за изглед. Компоненте би требале да хендлују само презентацију изгледа и корисничке интеракције. Сва друга логика, попут комуникације са back-end сервером, логовања, валидације инпута и сл. треба хендловати у сервисима (Cheng, 2018).

Следећи сервис показује различите исписе порука. Због декоратора @Injectable, Logger сервис је доступан у dependency injection-у.

```
@Injectable()
export class Logger {
 debug(message: string) {}
 info(message: string) {}
 warn(message: string) {}
 error(message: string) {}
}
```

2.2.1.5 Dependency Injection

Сервиси могу да се користе свугде у модулу. Компоненте могу да декларишу њихове зависности о сервисима користећи параметре конструктора. Инџектор (*енг. Injector*) у Angular-у је одговоран за обезбеђивање правих инстанци сервиса када се компоненте инстанцирају. Он креира инстанцу сервиса онда када је потребно, за шта му је потребно да зна како се креирају нове инстанце. Ово је омогућемо модулима или компонентама који декларишу провајдере. Провајдери су уствари, рецепти за креирање инстанци сервиса и разликујемо три различита типа провајдера: *class*, *value*, и *factory* (Cheng, 2018). Да би се сервиси користили у компонентама потребно је додати их низу провајдера у NgModule декоратору, као што је приказано у следећем сегменту кода:

```
@NgModule({
 providers: [
 Logger,
 ]
})
export class MyModule {
}
```

Потребно је да се сервис дода у конструктору, а Angular *injector inject-yje* инстанцу када се компонента креира, као што се може видети у следећем сегменту кода:

```
export class MyComponent {
 constructor(public logger: Logger) {
 }
```

2.2.1.6 Метаподаци и Bootstrapping

Метаподаци се односе на то како Angular ради интерно. Angular користи декораторе да декларативно дода понашања нормалним класама. Метаподаци су омогућени употребом декоратора. Angular апликације се стартује bootstrap-овањем *root* модула. Као што је већ поменуто, овакве апликације могу да се покрећу на различитим платформама, на пример, претраживачима или серверској страни.

Следећи пример кода приказује bootstrap-овање кореног модула коришћењем динамичке платформе претраживача:

```
import { platformBrowserDynamic } from '@angular/platformbrowser-dynamic';
import { AppModule } from './app.module';
platformBrowserDynamic().bootstrapModule(AppModule);
```

2.2.1.7 RxJS

RxJS је још једна важна библиотека која се користи у Angular апликацијама. RxJS је веома моћна за апликације које користе асинхроно и процесирање података засновано на догађајима. Језгро RxJS-а чине *observable* секвенце које дозвољавају уписивање вишеструких вредности синхроно или асинхроно. Једна RxJS *observable* може произвести вишеструке вредности својим темпом. Обзервер (*енг. observer*)који је заинтересован за вредности *observable* променљиве може да се претплати(*енг. subscribe)* на њу пружајући различите функције повратног позива(*енг. callback functions*) за различита сценарија вредности променљиве (Cheng, 2018).

RxJS пружа два различита приступа креирању *observable* променљивих. Први приступ подразумева коришћење методе Observable.create(). Аргумент методе create() је функција која дефинише како се производе вредности. Функција прима инстанцу обзервера и користи методе next(), complete(), и error() да произведе различита обавештења.

• next – производи нормалну вредност.

• complete – производи обавештење да је *observable* завршила производњу вредности

• error – прозиводи обавештење да је дошло до грешке приликом производње вредности.

```
Observable.create(observer => {
 observer.next('hello');
 observer.next('world');
 observer.complete();
});
```

Други приступ подразумева коришћење уграђених RxJS оператора за креирање *observable* променљивих. На пример:

```
of('hello', 'world');
// -> Observable са вредностима "hello" и "world"
from([1, 2, 3]);
// -> Observable са вредностима 1, 2 или 3
```

Да би примили обавештења од *observable* променљиве, потребно је претплатити се на исту користећи методу subscribe(). Том приликом могуће је дефинисати било који број хендлера а различите вредности обавештења. У коду испод додат је хендлер за обавештење типа next али error и complete обавештења се потпуно игноришу

```
from([1, 2, 3]).subscribe(v => console.log(v));
```

Повратна вредност методе subscribe() је инстанца класе RxJS Subscription class која се користи за управљање претплатом. У моментима након коришћења претплаћивања битно је позвати методу unsubscribe() како би се ослободили ресурси и отказала извршавања.

```
const subscription = from([1, 2, 3]).subscribe(v => console.log(v));
subscription.unsubscribe();
```

2.2.2 Ionic

Ionic је бесплатан програмски оквир који омогућује развој хибридних мобилних апликација, помоћу HTML-а, JavaScript-а, CSS-а и у основи је као и било који други мобилни оквир. Ionic на једном месту пружа сет готових функционалности за брзу израду мобилних апликација које су подржане на више платформи (одатле и назив хибридне). Израдом апликација користећи Ionic оквир, исти код се (после компилирања) може користити за iOS, Android и Windows phone верзију апликације. Управо то га сврстава у један јако користан алат, зато што није потребно развијати посебну апликацију за сваку мобилну платформу. Ionic оквир је отвореног кода *(енг. Open source)*, што значи да му је код јавно доступан те га програмери могу модификовати како желе (Cheng, 2018).

Оно што Ionic чини правим оквиром је јако велика количина унапред доступних ресурса. Тако су доступне разне мобилне компоненте, типографија, интерактивна понашања елемената и лако проширив основни изглед. Тако да је Ionic од почетка опремљен са сетом готових функција које омогућавају контролу тока програма. Ionic Framework развијен је у екосистемs који укључује AngularJS (или само Angular) и *Cordova*-у за развој структуре и логике апликације те омогућава приступање могућностима уређаја на којем се апликација користи. Ionic у наведеном екосистему омогућава кориснику осећај изворне мобилне апликације. Ти додатни алати олакшавају коришћење и интеграцију Cordova-е и њених програмских додатака потребних за развој хибридне мобилне апликације. Оно што Ionic чини јединственим је и подршка за SaSS (Syntactically Awesome Style Sheets) CSS екстензију. Како је помоћу уграђених CSS класа и аутоматског препознавања платформе могуће уредити кориснички интерфејс, тако је уз помоћ JavaScript-а могуће уредити и прилагодити изглед и интеракцију корисничког интерфејса.

Ionic знатно поједностављује *front-end* део апликације јер кроз примену класа платформе, употребом JavaScript-а контролише изглед и кориснички интерфејс мобилне

апликације. Ionic Framework аутоматски додељује класе платформе у елемент странице. Класа се додељује с обзиром на уређај који покреће апликацију. Ionic се тако придржава препоручених упутстава за стилизовање корисничког интерфејса зависно о платформи. Постоје две врсте класа (Wilken, 2018):

- класе платформе уређаја –користесе за добављање информација о уређају и додељују класе елементу странице. Класе ове врсте се користе са намером да се апликација што боље прилагоди специфичном изгледу и интеракцији платформе. У класе платформе уређаја спадају *platform-browser, platform-cordova, platformwebview, platform-ios, platform-android...*
- класе ОС верзије платформе–користе се за прилагођавање апликације за одређену верзију оперативног система. Неке од ових класа су: *platform-ios8, platformios8_4, platform-android4, platform-android4_4*

2.2.2.1 BaaS

Позадински сервиси као услуга или Baas *(енгл.Backend as a Service)*, другим именом MBaaS (*енгл. Mobile Backend as a Service),* су начини спајања веб и мобилних апликација са сервисима заснованима на облаку *(енгл. cloud)*. BaaS уместо кориштења међупрограма *(енгл. middleware)* и самосталног развијања позадинског система, ствара API и SDK, а они омогућавају комуникацију веб и мобилних апликација са складиштем у облаку. Такав приступ развијању апликације програмеру омогућује фокусирање на кориснички интерфејс, без трошења времена на развој самосталног сервиса. Услуге које уобичајени BaaS систем нуди су: обавештења *(енгл. push notifications),* интеграција са социјалним медијима *(енгл. social marketing integration),* локацијске услуге *(енгл. location services)* и управљање корисницима *(енгл. user management)*.

BaaS услуге Ionic платтформе су (Wilken, 2018):

- Ionic Push – омогућава објаву нотификација, односно обавештења, корисницима Ionic апликације. Обавештења кориснику могу дати информације о апликацији, иако корисник у том тренутку можда не користи паметни мобилни уређај.
- Ionic Users – омогућава регистрацију и верификацију корисника. Свака апликација у којој је могуће створити кориснички налог и комуницирати с корисницима, садржи начин регистрације те верификације тих корисника. Наведени сервис, осим API-a, који олакшава управљање регистрацијом и верификацијом корисника, омогућује и веб интерфејс којим програмер приступа информацијама о поједином кориснику.
- Ionic Deploy – омогућава ажурирање апликације на захтев, без дуготрајних процеса кроз Google Play услуге за Android платформу те App Store услуге за iOS платформу. Ажурирање апликације на захтев омогућено је само за промене HTML, CSS, JavaScript и мултимедијских датотека смештених у www директоријуму пројекта.
- Ionic Package – омогућава изградњу апликације за дистрибуцију. Главна намена овог сервиса је слање апликације другима и изградња апликације за платформе које рачунар програмера не подржава.
- Ionic Analytics – омогућава праћење понашања корисника Ionic апликације. Могуће је пратити колико је активних корисника којег дана те је та могућност

уграђена као задана могућност. Такође, могу се додати и властити догађаји *(енгл. event)* праћени од стране сервиса.
- Security Profiles – омогућава груписање Android и iOS акредитива у један профил. Ти подаци се тада могу користити од стране других сервиса.
- API – омогућава стандардне и додатне могућности које у неким сервисима нису подржане. Путем Ionic платформе приступа се токенима на којима је заснована верификација апликације. API такође захтева и кориштење HTTP метода, омогућених AngularJS-ом.

Ionic Framework, осим наведених сервиса, користи неке додатне алате који омогућавају олакшани развој хибридне мобилне апликације. Неки од тих алата су:

- Ionic Lab –десктоп апликација за Mac, Windows и Linux оперативне системе. Помоћу једноставног корисничког интерфејса омогућава лако коришћење неких од наведених Ionic сервиса. Уз то омогућава и велики број CLI функционалности, на пример покретање апликације на емулатору или уређају
- Ionic View – омогућава тестирање Ionic апликација на више уређаја, без потребе за дистрибуирањем апликације. Ionic апликацију могуће је послати било коме са IOS или Android уређајем и инсталираном IonicView апликацијом.
- Ionic Serve – омогућава покретање Ionic апликације локално у претраживачу уређаја. Ionic Serve уз Ionic View може послужити као алтернатива тестирању хибридних мобилних апликација на физичким уређајима.
- Ionic Creator – веб апликација која помоћу *drag and drop* интерфејса омогућава развој мобилних апликација.
- Ionic Native –скуп ES5/ES6/TypeScript омотача за Cordova програмске додатке који омогућавају додавање изворне функционалности, без директног коришћења Cordova програмских додатака.

2.2.2.2 Ionic CLI

Још један врло користан алат из Ionic породице јесте Ionic *CLI (енгл. Command Line Utility)*, односно Ionic командна линија. Ionic CLI је примарни алат при развоју хибридне мобилне апликације помоћу Ionic Framework-a. CLI садржи врло битне наредбе којима се програмер може служити током развоја апликације. Ionic CLI потребно је инсталирати на властити рачунар те се наредбама приступа преко командне линије уређаја (Wilken, 2018).

Неке од битнијих наредби су:

- start –наредба која покреће нови Ionic пројекат са свим потребним интеграцијама Cordova-е и AngularJS-а. Приликом покретања могуће је изабрати готове шаблоне или празан пројекат те жељену верзију Ionic Framework-а.
- build –припрема и преводи Ionic пројекат за одређену платформу
- serve –покреће локални сервер у којем се покреће апликацију у претраживачу рачунара
- platform –додаје ресурсе потребне за развој апликације за одређену платформу

- info –даје информације о верзијама потребних програмских пакета (Cordova CLI-a, Ionic CLI-a, Ionic Framework-a, Node-a)

2.2.2.3 Структура Ionic апликације

С обзиром да су Ionic апликације направљене помоћу Cordova-е, користи се Cordova структура података. Приказ уобичајене структуре Ionic апликације приказана је на следећој слици:

Слика 8 Структура Ionic апликације

Неки од најважнијих директоријума приказаних на слици 8 су:

- plugins – овде се чувају сви програмски додаци које програмер додаје у пројекат
- src – у овом директоријуму се налазе најбитније датотеке за развој апликације, а то укључује: .scss датотеку у којој се налази SaSS апликације, *assets* мапа с кориштеним ресурсима (слике, фонтови итд.), мапа са страницама (екранима) апликације
- www – овде се одвија сам развој мобилне апликације, препоручена структура апликације аутоматски је постављена од стране Ionic-a

2.2.2.4 Apache Cordova

Apache Cordova (или само Cordova) је бесплатан *framework* за изградњу *cross-platform* нативних апликација употребом интернет технологија: HTML5, CSS3 и JavaScript-a. Framework је настао од стране *Apache Software Foundation*-а као резултат једноставније израде *cross-platform* мобилних апликација, заснован на комбинацији нативне и веб апликације. Примарна корист Cordova-е је омогућавање употребе нативних функционалности изван мобилног претраживача. Како би се заобишла ова ограничења, Cordova имплементира пакет додатака који се протежу на нативне могућности уређаја (нпр. камера, акцелерометар, контакти) покрећући веб апликацију унутар нативног контејнера. Развој интернет апликације која има интеракцију са нативним хардвером и нативном функционалношћу је било немогуће, до појављивања Cordova-е (Wilken, 2018).

Cordova апликација састоји се од неколико компоненти.

- веб поглед *(engl. Web View)* – апликацији може пружити целокупни кориснички интерфејс
- веб апликација *(engl. WebApp)* – место где је смештен апликацијски код. Апликација је имплементирана као веб страница која се подразумевано састоји од локалне датотеке index.html која показује на CSS правила, JavaSript код, те остале датотеке потребне за њено покретање
- програмски додаци *(engl. plugins)* – саставни део Cordova екосистема који пружају интерфејс за комуникацију између Cordova-е и изворних компоненти те везе до стандардних API-ја уређаја

3. Студијски пример

У овом поглављу биће представљен студијски пример развоја дела софтверског система реализован коришћењем Ларманове методе. Имплементација студијског примера је извршена коришћењем ASP.NET Core и Ionic оквира.

3.1 Ларманова метода

Ларманова метода за развој софтвера се заснива на итеративно-инкременталном моделу животног циклуса софтвера. Упрошћена Ларманова метода се састоји од следећих фаза: (Влајић, 2019)

1. Прикупљање захтева
2. Фаза анализе
3. Фаза пројектовања
4. Фаза имплементације
5. Фаза тестирања

У фази прикупљања захтева се идентификују и дефинишу кориснички захтеви везани за пројекат. Како би ова фаза била успешно извршена потребно је прикупити што више релевантних информација од корисника. Захтеви се могу разликовати као функционални и нефункционални. Функционални захтеви дефинишу захтеване функције система, док нефункционални захтеви дефинишу све остале захтеве. Захтеви се описују преко UML модела случаја коришћења који се састоји од скупа случаја коришћења, актора и веза између актора и случаја коришћења. Један случај коришћења описује скуп сценарија, односно скуп жељених коришћења система од стране актора. Случај коришћења има више алтернативних и један основни сценарио. Сценарио представља секвенцу акција које описују интеракцију актера и система. Једну акцију сценарија може извршити актер или систем, па их стога можемо поделити на две групе (Влајић, 2019):

Акције које изводи актор су (Влајић, 2019):

1. АПУСО – Актор припрема улаз за системску операцију
2. АПСО – Актор позива системску операцију
3. АНСО – Актро извршава несистемску операцију

Систем изводи две акције једну за другом (Влајић, 2019):

1. СО – Систем извршава системску операцију
2. ИА – Излазни аргументи који представљају резултат системске операције

Фаза анализе описује логичку структуру и понашање софтв. система (пословну логику софт. система). Понашање софтверског система је описано помоћу системских дијаграма секвенци и преко системских операција. Структура софтерског система се описује помоћу концептуалног и релационог модела. Секвенцни дијаграм се прави посебно за сваки случај коришћења и као резултат анализе секвенцних дијаграма добијају се системске операције које треба пројектовати у коду. За сваку системску операцију прави се по један уговор. Уговори се састоје из следећих секција (Влајић, 2019):

- Операција – име операције и њени улазни и излазни аргументи
- Веза са СК – имена СК у којима се позива системска операција
- Предуслов – пре извршења системске операције морају бити задовољени одређени предуслови
- Постуслови – после извршења системске операције у систему морају бити задовољени одређени постуслови.Фаза пројектовања – У овој фази се описује физичка структура и понашање софтверског система (архитектура). Пројектовање архитектуре представља трослојну архитектуру која обухвата: пројектовање корисничког интерфејса, апликационе логике и складишта података.

У фази пројектовања се прави архитектура софтверског система која је тронивојска (кориснички интерфејс, апликациона логика и складиште података). Имплементационе компоненте, из фазе имплементације, треба да реализују компоненте које су добијене у фази пројектовања. Свака од имплементационих компоненти се тестира у фази тестирања (Влајић, 2019).

4. Кориснички захтеви

Прикупљање захтева од корисника је први, можда и најважнији, корак у процесу развоја софтвера Ларамановом методом, јер ако се корисникове потребе не идентификују на прави начин може доћи до проблема у каснијим фазама развоја. Већ у овој фази потребно је уочити до каквих све проблема може доћи у каснијем развоју и одредити прави модел података који би задовољио потребе система. (Влајић, 2019)

4.1 Вербални опис модела

Софтверски систем за студентски центар је софтверски систем осмишљен да студентима Београдског али и свих универзитета у Србији омогући праћење стања и лакше управљање подацима везаним за живот у студентском дому, у случају да је студент станар студентског дома, и коришћење студентске мензе.

Инспирација за креирање оваквог софтверског система јавила се због чињенице да је због начина на који тренутно функционише праћење студентског налога у студентском центру, једина опција коју студенти имају, како би проверили стање на истом и евентуално направили неке измене, да физички оду у једну од установа студентских центара и тамо на апаратима, који врло често нису у функцији, употребе студентску картицу.

Свесна чињенице да је то за данашњи свет, свет мобилних апликација, у којем живимо поприлично застарео начин, а и вођена личним искуством, одлучила сам да креирам мобилну апликацију која ће у потпуности заменити одлазак до апарата и употребу студентске картице. Све неопходне информације и акције које би студент добио на апарату, биле би доступне у пар кликова, у свако доба и на сваком месту. Што мислим да и јесте поента развоја било које мобилне апликације.

У систему постоје улога администратора и улога студента. Фокус је био на функционалностима осмишљеним да покрију потребе студената, па је самим тим тај сегмент доста сложенији од сегмента везаног за администратора. Администратор након пријављивања на систем може да претражује собе у оквиру студентских домова које покрива студентски центар у којем ради и да истим додељује студенте које претражује по називу факултета и броју индекса. Друга функционалност везана је за евидентирање уплата које су пристигле од студената. Наиме, администратор уносом позива на број претражује студенте и евидентира им износ уплате на рачун.

Када је реч о студенту, он након пријаве на систем има доста комплекснији мени него што је то био случај са администратором. Студент може да изабере опције менза, машина, станарина и подешавања. У зависности од избора отвара му се могућност куповине оброка у мензи, употребе истих скенирањем QR кода, резервисање веш машине у блоку чији је станар, опција плаћања станарине за собу чији је станар или евентуално мењање приступне лозинке.

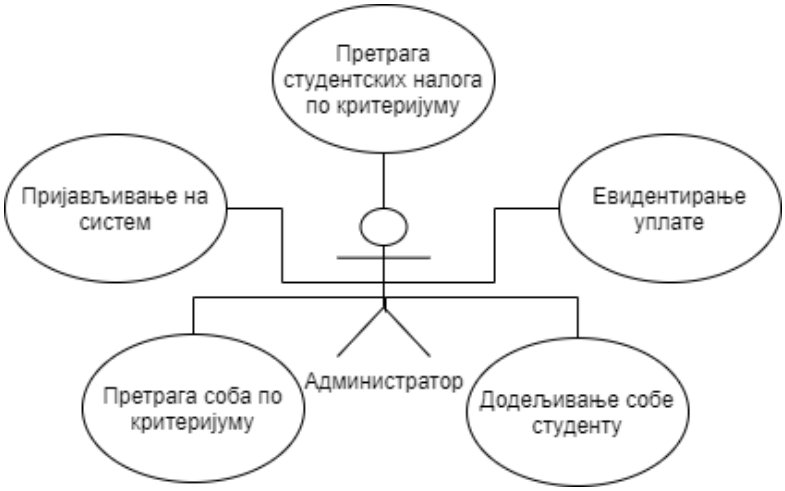
4.2. Спецификација захтева помоћу модела случајева коришћења

Админу, као кориснику система, треба омогућити следеће функционалности:

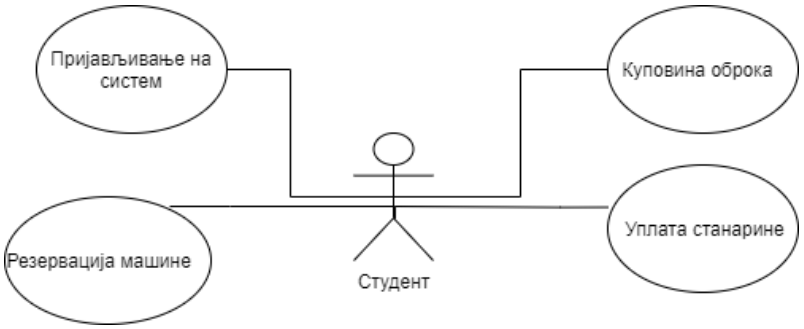
- Пријављивање на систем
- Претрага студентских налога по критеријуму
- Евидентирање уплате
- Претрага соба по критеријуму
- Додељивање собе студенту

Студенту, као кориснику система, треба омогућити следеће функционалности:

- Пријављивање на систем
- Куповина оброка
- Резервација машине
- Уплата станарине



Слика 9 Случајеви коришћења администратор



Слика 10 Случајеви коришћења студент

4.2.1 СК1: Случај коришћења - Пријављивање на систем

Назив СК

Пријављивање на систем

Актори СК

Корисник (администратор или студент)

Учесници СК

Корисник и систем (програм)

Предуслов: Систем је укључен. Систем приказује форму за пријављивање на систем.

Основни сценарио СК

1. Корисник уноси корисничко име и лозинку. (АПУСО)
2. Корисник контролише да ли је коректно унео корисничко име и лозинку. (АНСО)
3. Корисник позива систем да се пријави на систем (провери податке). (АПСО)
4. Систем проверава податке о кориснику. (СО)
5. Систем приказује кориснику почетну страну и поруку:
   “Успешно пријављивање!“. (ИА)

Алтернативна сценарија

5.1. Уколико систем не може да нађе корисника, он приказује кориснику поруку:

“ Неуспешно пријављивање на систем!”. (ИА)

4.2.2 СК2: Случај коришћења - Претрага студентских налога по критеријуму

Назив СК

Претрага студентских налога по критеријуму

Актори СК

Администратор

Учесници СК

Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је пријављен на систем. Систем приказује форму за рад са студентима. Учитана је листа студената.

Основни сценарио СК

1. Администратор уноси вредност по којој претражује студента. (АПУСО)

2. Администратор позива систем да нађе студенте по задатој вредности. (АПСО)

3. Систем тражи студенте по задатој вредности. (СО)

4. Систем приказује администратору нађене студенте. (ИА)

5. Администратор бира студента чије податке жели да види.(АПУСО)

6. Администратор позива систем да прикаже податке о изабраном студенту.(АПСО)

7. Систем проналази податке о изабраном студенту.(СО)

8. Систем приказује администратору податке о изабраном студенту.(ИА)

Алтернативна сценарија

4.1 Уколико систем не може да пронађе студента по задатим вредностима приказује се порука: ”Грешка приликом проналаска студента“. (ИА)

8.1. Уколико систем не може да прикаже податке о изабраном студенту приказује се порука „Систем не може да прикаже податке о студенту“(ИА)

4.2.3 СК3: Случај коришћења - Евидентирање уплате

Назив СК

Евидентирање уплате

Актори СК

Администратор

Учесници СК

Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је пријављен на систем. Систем приказује форму за рад са студентима. Учитана је листа студената.

Основни сценарио СК

1. Администратор уноси вредност по којој претражује студента. (АПУСО)

2. Администратор позива систем да нађе студенте по задатој вредности. (АПСО)

3. Систем тражи студенте по задатој вредности. (СО)

4. Систем приказује администратору нађене студенте. (ИА)

5. Администратор бира студента чије податке жели да измени.(АПУСО)

6. Администратор позива систем да учита податке о студенту. (АПСО)

7. Систем учитава податке о студенту.(СО)

8. Систем приказује администратору задатог студента.(ИА)

9. Администратор мења податке о студенту.(АПУСО)

10. Администратор проверава да ли је исправно унео измењене податке.(АНСО)

11. Администратор позива систем да запамти податке о студенту.(АПСО)

12. Систем памти измењене податке о студенту.(СО)

13. Систем приказује администратору поруку:“Систем је успешно евидентирао уплату.“(ИА)

Алтернативна сценарија

4.1. Уколико систем не може да пронађе задатог студента приказује се порука: ”Грешка приликом проналаска студента”. Прекида се извршење сценарија. (ИА)

8.1. Уколико систем не пронађе информације о студенту, систем приказује администратору поруку:“Систем не може да прикаже податке о студенту“. Прекида се извршење сценарија.(ИА)

13.1. Уколико систем не може да сачува измене о студенту, систем приказује следећу поруку:“Неуспешно евидентирање.“(ИА)

4.2.4 СК4: Случај коришћења - Претрага соба по критеријуму

Назив СК

Претрага соба по критеријуму

Актори СК

Администратор

Учесници СК

Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је пријављен на систем. Систем приказује форму за рад са собама. Учитана је листа соба.

Основни сценарио СК

1. Администратор уноси вредност по којој претражује собу. (АПУСО)

2. Администратор позива систем да нађе собе по задатој вредности. (АПСО)

3. Систем тражи собе по задатој вредности. (СО)

4. Систем приказује администратору нађене собе. (ИА)

5. Администратор бира собу чије податке жели да види.(АПУСО)

6. Администратор позива систем да прикаже податке о изабраној соби.(АПСО)

7. Систем проналази податке о изабраној соби.(СО)

8. Систем приказује администратору податке о изабраној соби.(ИА)

Алтернативна сценарија

4.1 Уколико систем не може да пронађе собу по задатим вредностима приказује се порука: ”Грешка приликом проналаска собе“. Прекида се извршење сценарија.(ИА)

8.1. Уколико систем не може да прикаже податке о изабраној соби приказује се порука „Систем не може да прикаже податке о соби“(ИА)

4.2.5 СК5: Случај коришћења - Додељивање собе студенту

Назив СК

Додељивање собе студенту

Актори СК

Администратор

Учесници СК

Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је пријављен на систем. Систем приказује форму за доделу соба. Учитане су листа студената и листа соба.

Основни сценарио СК

1. Администратор уноси вредност по којој претражује собу. (АПУСО)
2. Администратор позива систем да нађе собе по задатој вредности. (АПСО)
3. Систем тражи собе по задатој вредности. (СО)
4. Систем приказује администратору нађене собе. (ИА)
5. Администратор бира собу чије податке жели да види.(АПУСО)
6. Администратор позива систем да прикаже податке о изабраној соби.(АПСО)
7. Систем проналази податке о изабраној соби.(СО)
8. Систем приказује администратору податке о изабраној соби.(ИА)
9. Администратор уноси вредност по којој претражује студента. (АПУСО)
10. Администратор позива систем да нађе студенте по задатој вредности. (АПСО)
11. Систем тражи студенте по задатој вредности. (СО)
12. Систем приказује администратору нађене студенте. (ИА)
13. Администратор бира студента чије податке жели да измени.(АПУСО)
14. Администратор позива систем да учита податке о студенту. (АПСО)
15. Систем учитава податке о студенту.(СО)
16. Систем приказује администратору задатог студента.(ИА)
17. Администратор мења податке о студенту.(АПУСО)
18. Администратор проверава да ли је исправно унео измењене податке.(АНСО)
19. Администратор позива систем да запамти податке о студенту.(АПСО)
20. Систем памти измењене податке о студенту.(СО)
21. Систем приказује администратору поруку:“Систем је успешно извршио доделу.“(ИА)

Алтернативна сценарија

4.1. Уколико систем не може да пронађе задату собу приказује се порука: ”Грешка приликом проналаска собе”. Прекида се извршење сценарија. (ИА)

8.1. Уколико систем не пронађе информације о соби, систем приказује администратору поруку:“Систем не може да прикаже податке о соби“. Прекида се извршење сценарија.(ИА)

12.1. Уколико систем не може да пронађе задатог студента приказује се порука: ”Грешка приликом проналаска студента”. Прекида се извршење сценарија. (ИА)

16.1. Уколико систем не пронађе информације о студенту, систем приказује администратору поруку:“Систем не може да прикаже податке о студенту“. Прекида се извршење сценарија.(ИА)

21.1. Уколико систем не може да сачува измене о студенту, систем приказује следећу поруку:“Неуспешна додела.“(ИА)

4.2.6 СК6: Случај коришћења - Куповина оброка

Назив СК:

Куповина оброка

Аутори СК:

Студент

Учесници СК:

Студент и систем

Предуслов: Систем је укључен. Систем приказује страну за куповину оброка. Учитани су подаци о студенту.

Основни сценарио СК:

1. Студент уноси количину оброка које жели да купи (АПУСО)

2. Студент контролише да ли је коректно унео количину оброка. (АНСО)

3. Студент позива систем да запамти измењене податке на студентском налогу. (АПСО)

4. Систем памти измењене податке на студентском налогу (СО)

5. Систем приказује кориснику поруку: “Успешно сте купили оброке”. (ИА)

Алтернативни сценарио СК:

5.1 Уколико систем не може да измени податке о корисничком налогу, систем приказује кориснику поруку: “Дошло је до грешке ”. (ИА)

4.2.7 СК7: Случај коришћења - Резервација машине

Назив СК:

Резервација машине

Аутори СК:

Студент

Учесници СК:

Студент и систем

Предуслов: Систем је укључен. Систем приказује страну за резервацију машине. Учитани су подаци о студенту и листа резервација.

Основни сценарио СК:

1. Студент уноси термин у којем жели да резервише машину. (АПУСО)

2. Студент контролише да ли је коректно унео податке о термину. (АНСО)

3. Студент позива систем да изврши резервацију. (АПСО)

4. Систем памти податке о резервацији. (СО)

5. Систем приказује админу поруку: “Резервација успешна!“. (ИА)

Алтернативна сценарија

5.1. Уколико систем не може да запамти податке о резервацији, он приказује студенту поруку: “Дошло је до грешке приликом обрађивања захтева!” .(ИА)

4.2.8 СК8: Случај коришћења - Уплата станарине

Назив СК:

Уплата станарине

Аутори СК:

Студент

Учесници СК:

Студент и систем

Предуслов: Систем је укључен. Систем приказује страну за плаћање станарине. Учитани су подаци о студенту и соби.

Основни сценарио СК:

1. Студент уноси износ цене станарине. (АПУСО)

2. Студент контролише да ли је коректно унео податке о станарини. (АНСО)

3. Студент позива систем да запамти уплату. (АПСО)

4. Систем памти податке о уплати. (СО)

5. Систем приказује админу поруку: “Уплата успешна!“. (ИА)

Алтернативна сценарија:

5.1. Уколико систем не може да запамти податке о уплати, он приказује студенту поруку: “Дошло је до грешке приликом обрађивања захтева!” .(ИА)

5. Фаза анализе

У фази анализе описује се логичка структура и понашање софтверског система (пословна логика софтверског система). Понашање софтверског система описујемо помоћу системских дијаграма секвенци, који се праве за сваки случај коришћења, и помоћу уговора о системским операцијама, које се добијају на основу системских дијаграма секвенци. Структура софтверског система је описана помоћу концептуалног и релационог модела.

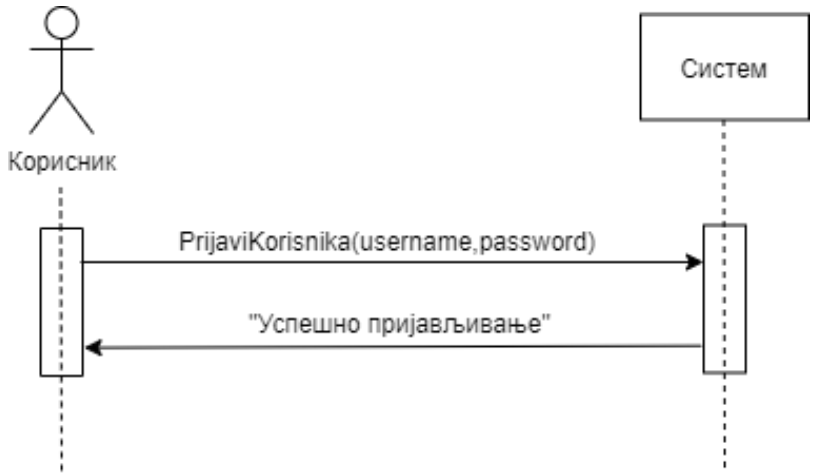
5.1 Понашање софтверског система – Системски дијаграми секвенци

Да би се на најбољи начин представило понашање софтверског система, за сваки случај коришћења уочен у фази прикупљања захтева приказује се одговарајући системски дијаграм секвенци, путем кога се моделује интеракција између актера и система путем активности у одређеном редоследу. (Влајић, 2019)

ДС1. Дијаграм секвенци случаја коришћења - Пријављивање на систем

Основни сценарио СК

1. Корисник позива систем да се пријави на систем. (АПСО)
2. Систем приказује кориснику почетну страну и поруку:“Успешно пријављивање!“ (ИА)

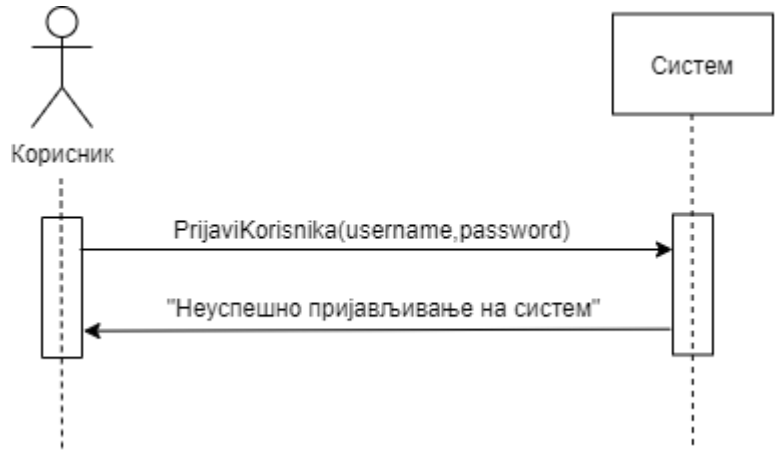


Слика 11Пријављивање на систем

Алтернативна сценарија

2.1. Уколико систем не може да нађе корисника, он приказује кориснику поруку:

“ Неуспешно пријављивање на систем!”. (ИА)



Слика 12Неуспешно пријављивање на систем

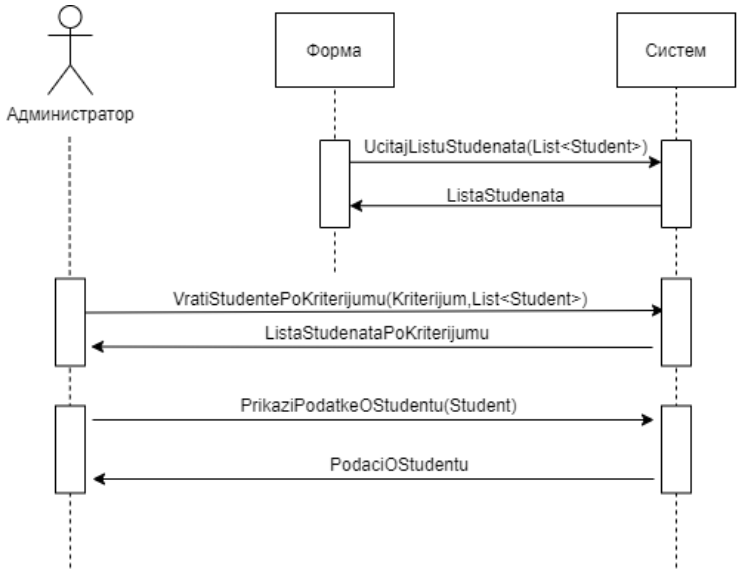
Идентификоване системске операције:

- signal PrijaviKorisnika(username, password)

ДС2: Дијаграм секвенци случаја коришћења - Претрага студентских налога по критеријуму

Основни сценарио СК

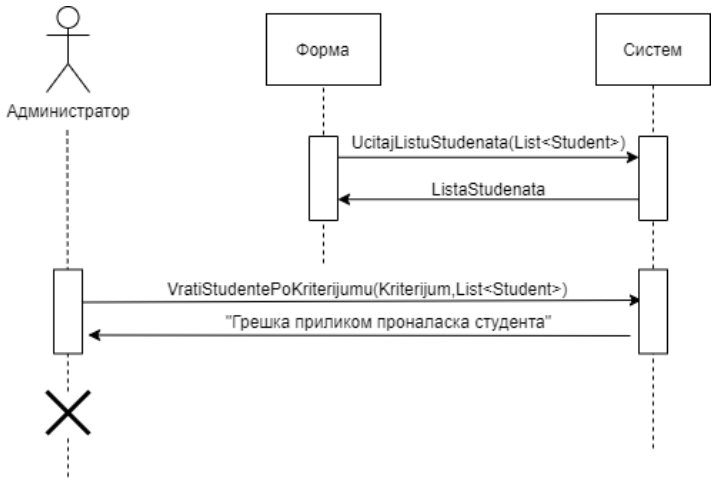
1. Форма позива систем да учита листу студената. (АПСО)
2. Систем враћа форми листу студената. (ИА)
3. Администратор позива систем да нађе студенте по задатој вредности. (АПСО)
4. Систем приказује администратору нађене студенте. (ИА)
5. Администратор позива систем да прикаже податке о изабраном студенту.(АПСО)
6. Систем приказује администратору податке о изабраном студенту.(ИА)



Слика 13 Претрага студената по критеријуму

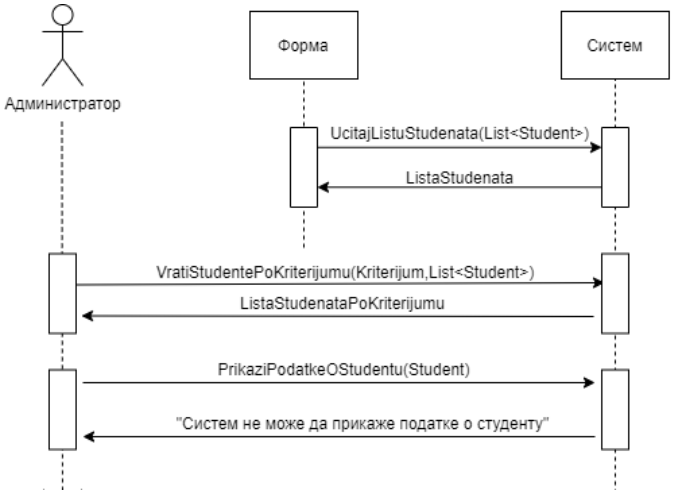
Алтернативна сценарија

4.1 Уколико систем не може да пронађе студента по задатим вредностима приказује се порука: ”Грешка приликом проналаска студента“. (ИА)



Слика 14 Грешка приликом проналаска студента

6.1. Уколико систем не може да прикаже податке о изабраном студенту приказује се порука „Систем не може да прикаже податке о студенту“(ИА)



Слика 15 Неуспешно приказивање студента

Идентификоване системске операције:

- signal UcitajListuStudenata(List<Student>)
- signal VratiStudentePoKriterijumu(Kriterijum,List<Student>)
- signal PrikaziPodatkeOStudentu(Student)

ДС3: Дијаграм секвенци случаја коришћења – Евидентирање уплате

Основни сценарио СК

1. Форма позива систем да учита листу студената. (АПСО)

2. Систем враћа форми листу студената. (ИА)

3. Администратор позива систем да нађе студенте по задатој вредности. (АПСО)

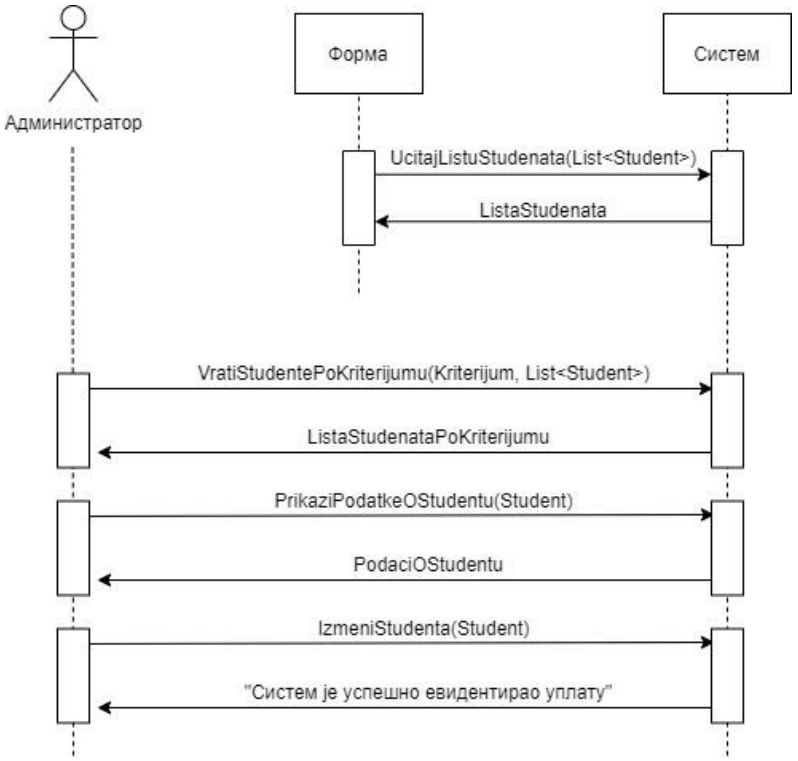
4. Систем приказује администратору нађене студенте. (ИА)

5. Администратор позива систем да учита податке о студенту. (АПСО)

6. Систем приказује администратору задатог студента.(ИА)

7. Администратор позива систем да запамти податке о студенту.(АПСО)

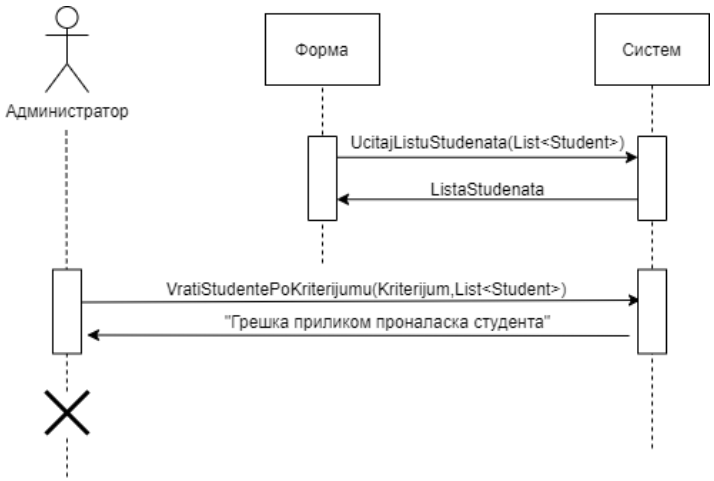
8. Систем приказује администратору поруку:“Систем је успешно евидентирао уплату.“(ИА)



Слика 16 Евидентирање уплате

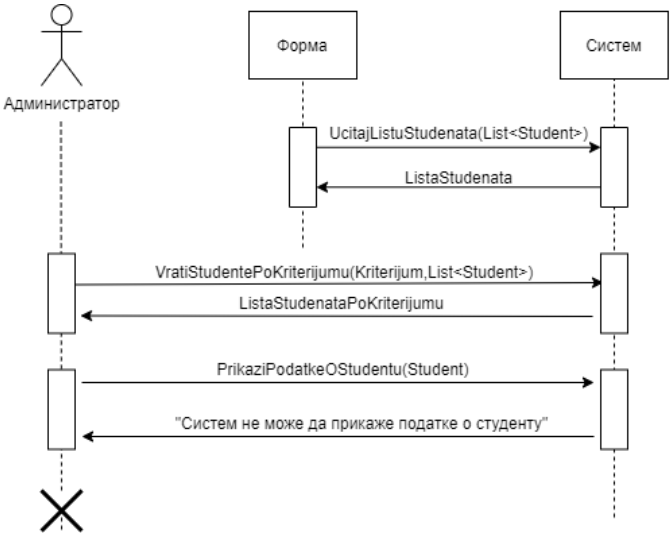
Алтернативна сценарија

4.1. Уколико систем не може да пронађе задатог студента приказује се порука: ”Грешка приликом проналаска студента”. Прекида се извршење сценарија. (ИА)



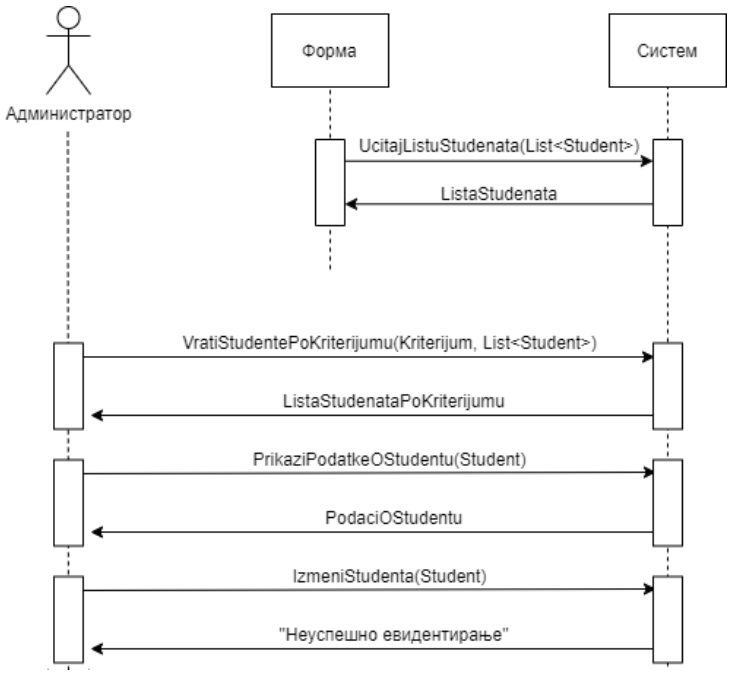
Слика 17 Грешка приликом проналаска студента

6.1. Уколико систем не пронађе информације о студенту, систем приказује администратору поруку:“Систем не може да прикаже податке о студенту“. Прекида се извршење сценарија.(ИА)



Слика 18 Неуспешно приказивање студента

8.1. Уколико систем не може да сачува измене о студенту, систем приказује следећу поруку:“Неуспешно евидентирање.“(ИА)



Слика 19 Неуспешно евидентирање

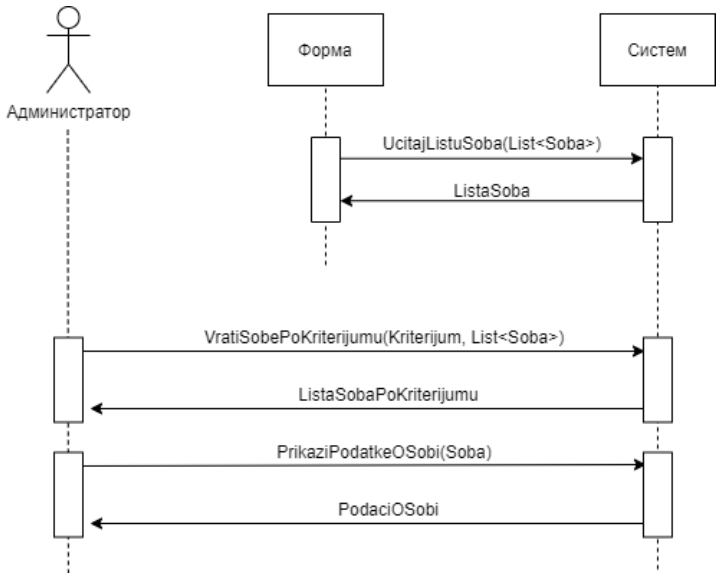
Идентификоване системске операције:

- signal UcitajListuStudenata(List<Student>)
- signal VratiStudentePoKriterijumu(Kriterijum,List<Student>)
- signal PrikaziPodatkeOStudentu(Student)
- signal IzmeniStudenta(Student)

ДС4: Дијаграм секвенци случаја коришћења – Претрага соба по критеријуму

Основни сценарио СК

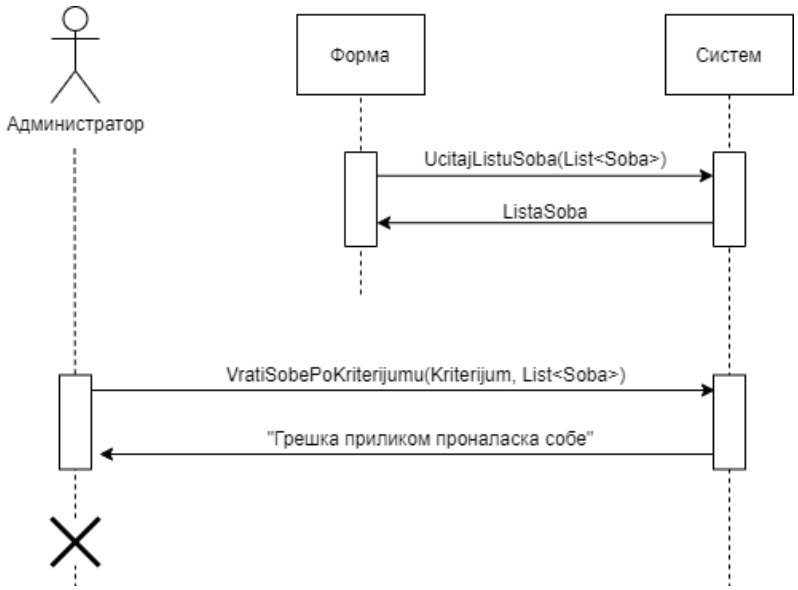
1. Форма позива систем да учита листу соба. (АПСО)
2. Систем враћа форми листу соба. (ИА)
3. Администратор позива систем да нађе собе по задатој вредности. (АПСО)
4. Систем приказује администратору нађене собе. (ИА)
5. Администратор позива систем да прикаже податке о изабраној соби.(АПСО)
6. Систем приказује администратору податке о изабраној соби.(ИА)



Слика 20 Претрага соба по критеријуму

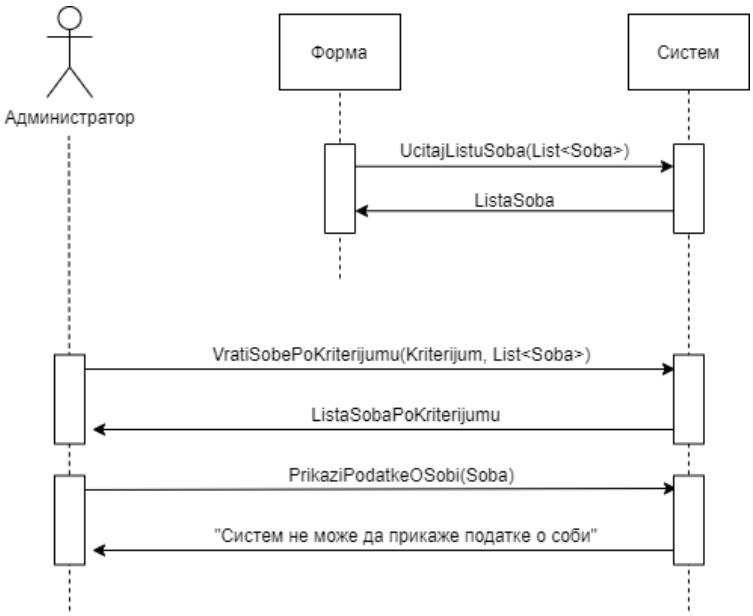
Алтернативна сценарија

4.1 Уколико систем не може да пронађе собу по задатим вредностима приказује се порука: ”Грешка приликом проналаска собе“. Прекида се извршење сценарија.(ИА)



Слика 21 Грешка приликом проналаска собе

6.1. Уколико систем не може да прикаже податке о изабраној соби приказује се порука “Систем не може да прикаже податке о соби”(ИА)



Слика 22 Грешка приликом приказивања собе

Идентификоване системске операције:

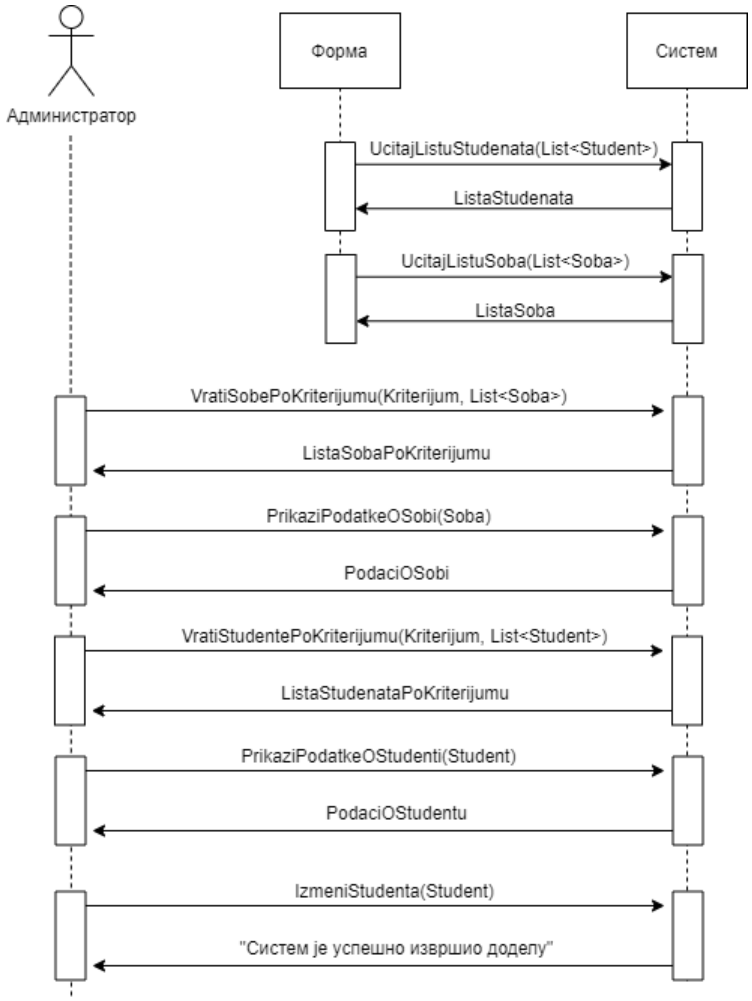
- signal UcitajListuSoba(List<Soba>)
- signal VratiSobePoKriterijumu(Kriterijum,List<Soba>)
- signal PrikaziPodatkeOSobi(Soba)

ДС5: Дијаграм секвенци случаја коришћења – Додељивање собе студенту

Основни сценарио СК

1. Форма позива систем да учита листу студената. (АПСО)
2. Систем враћа форми листу студената. (ИА)
3. Форма позива систем да учита листу соба. (АПСО)
4. Систем враћа форми листу соба. (ИА)
5. Администратор позива систем да нађе собе по задатој вредности. (АПСО)
6. Систем приказује администратору нађене собе. (ИА)
7. Администратор позива систем да прикаже податке о изабраној соби.(АПСО)
8. Систем приказује администратору податке о изабраној соби.(ИА)
9. Администратор позива систем да нађе студенте по задатој вредности. (АПСО)
10. Систем приказује администратору нађене студенте. (ИА)
11. Администратор позива систем да учита податке о студенту. (АПСО)
12. Систем приказује администратору задатог студента.(ИА)
13. Администратор позива систем да запамти податке о студенту.(АПСО)

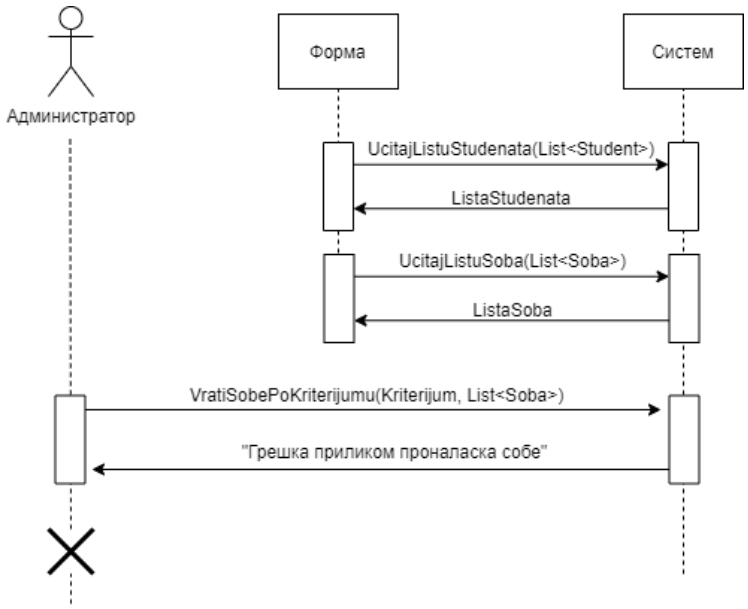
14. Систем приказује администратору поруку:“Систем је успешно извршио доделу.“(ИА)



Слика 23 Додела собе

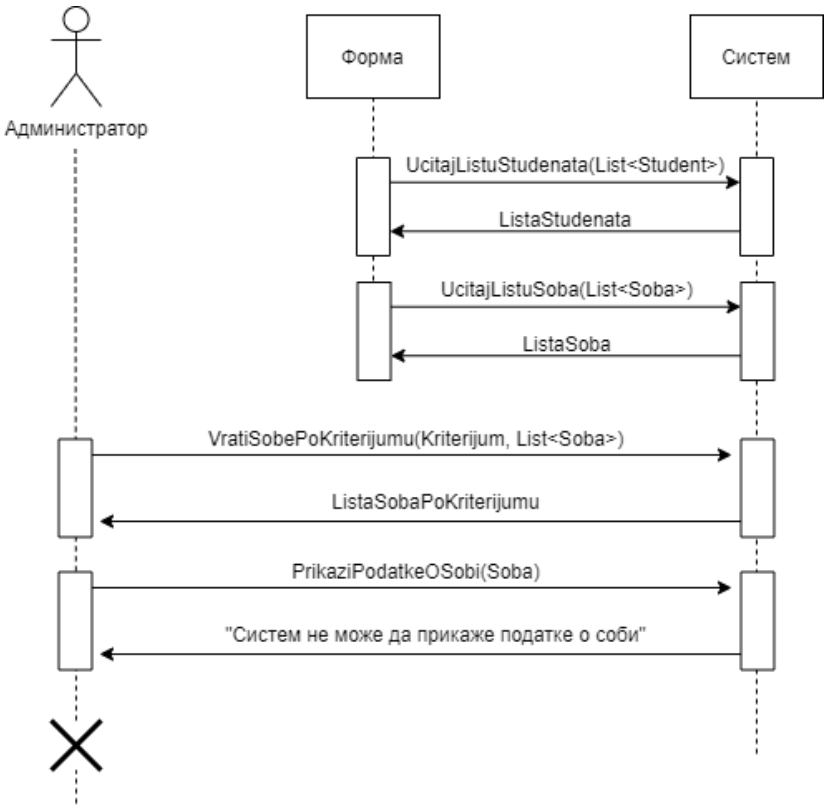
Алтернативна сценарија

6.1. Уколико систем не може да пронађе задату собу приказује се порука: ”Грешка приликом проналаска собе”. Прекида се извршење сценарија. (ИА)



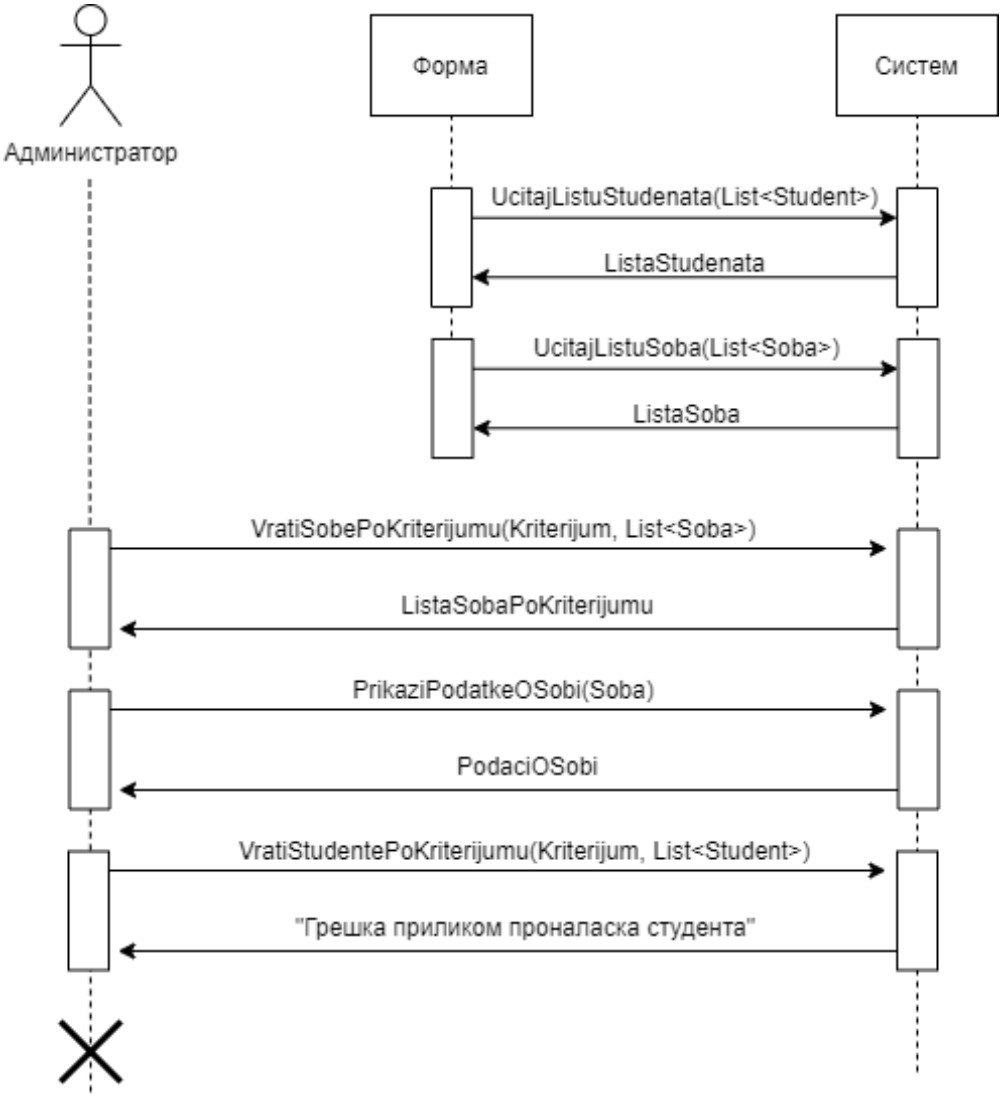
Слика 24 Грешка приликом проналаска собе

8.1. Уколико систем не пронађе информације о соби, систем приказује администратору поруку:“Систем не може да прикаже податке о соби“. Прекида се извршење сценарија.(ИА)



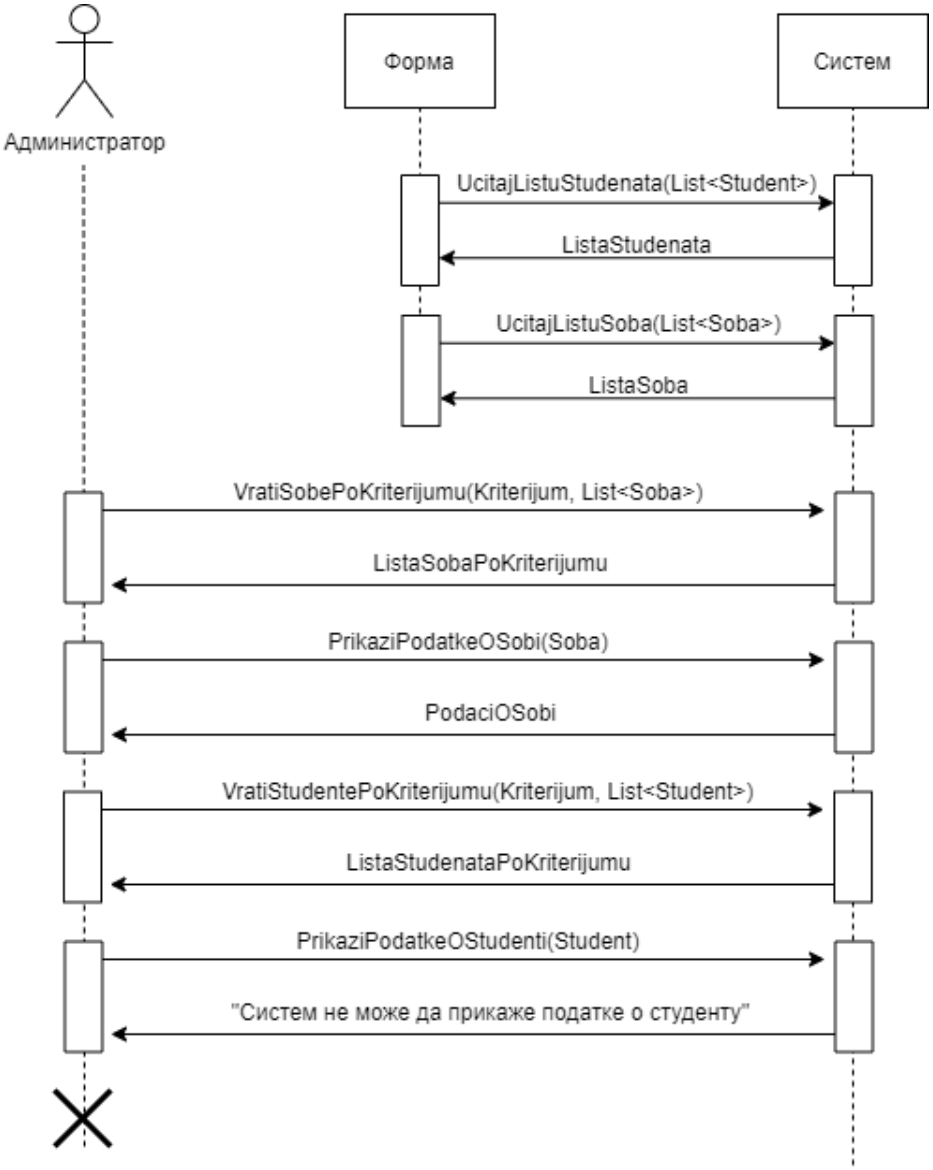
Слика 25 Грешка приликом приказивања собе

10.1. Уколико систем не може да пронађе задатог студента приказује се порука: ”Грешка приликом проналаска студента”. Прекида се извршење сценарија. (ИА)



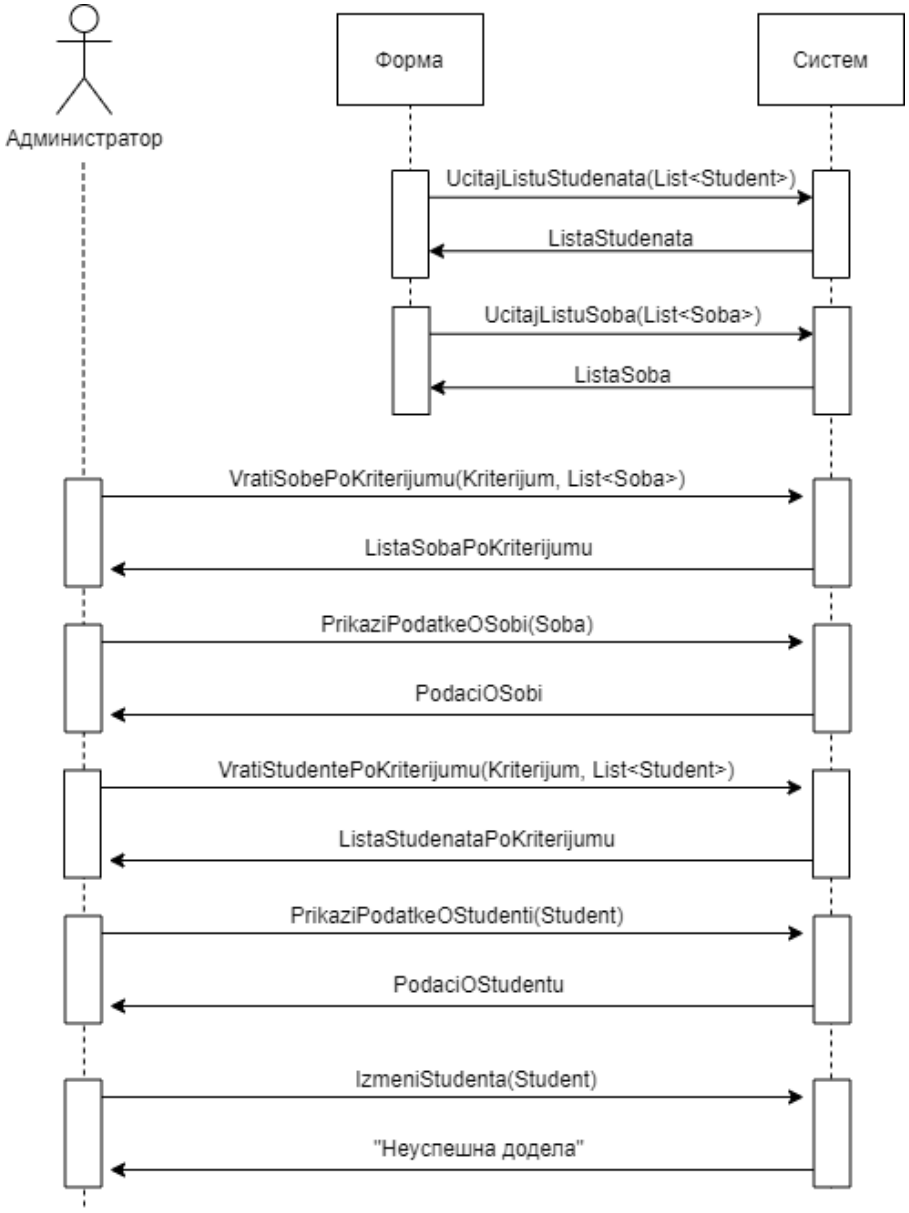
Слика 26 Грешка приликом проналаска студента

12.1. Уколико систем не пронађе информације о студенту, систем приказује администратору поруку:“Систем не може да прикаже податке о студенту“. Прекида се извршење сценарија.(ИА)



Слика 27 Грешка приликом приказивања студента

14.1 Уколико систем не може да сачува измене о студенту, систем приказује следећу поруку:“Неуспешна додела.“(ИА)



Слика 28 Неуспешна додела собе

Идентификоване системске операције:

- signal UcitajListuStudenata(List<Student>)
- signal UcitajListuSoba(List<Soba>)
- signal VratiSobePoKriterijumu(Kriterijum,List<Soba>)
- signal PrikaziPodatkeOSobi(Soba)
- signal VratiStudentePoKriterijumu(Kriterijum,List<Student>)
- signal PrikaziPodatkeOStudentu(Student)
- signal IzmeniStudenta(Student)

ДС6: Дијаграм секвенци случаја коришћења – Куповина оброка

Учитани су подаци о студенту.

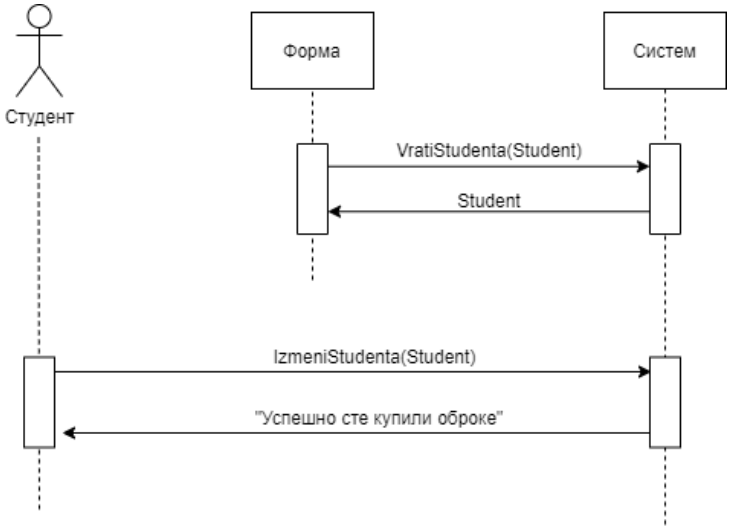
Основни сценарио СК:

1. Форма позива систем да учита студента. (АПСО)

2. Систем враћа форми студента. (ИА)

3. Студент позива систем да запамти измењене податке на студентском налогу. (АПСО)

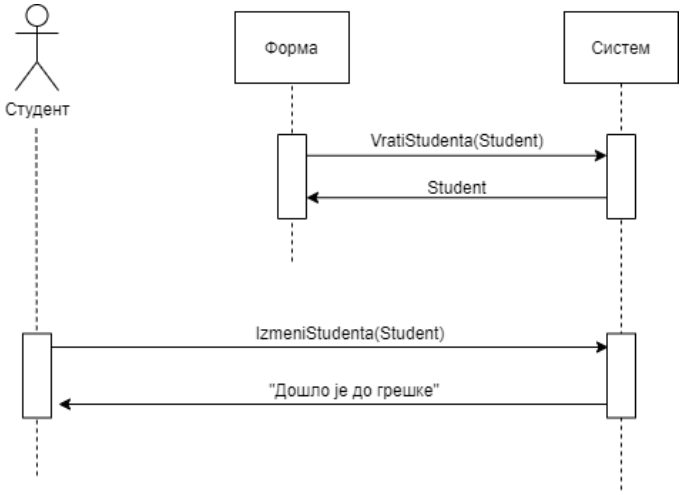
4. Систем приказује кориснику поруку: “Успешно сте купили оброке”. (ИА)



Слика 29 Куповина оброка

Алтернативни сценарио СК:

4.1 Уколико систем не може да измени податке о корисничком налогу, систем приказује кориснику поруку: “Дошло је до грешке ”. (ИА)



Слика 30 Грешка приликом куповине

Идентификоване системске операције:

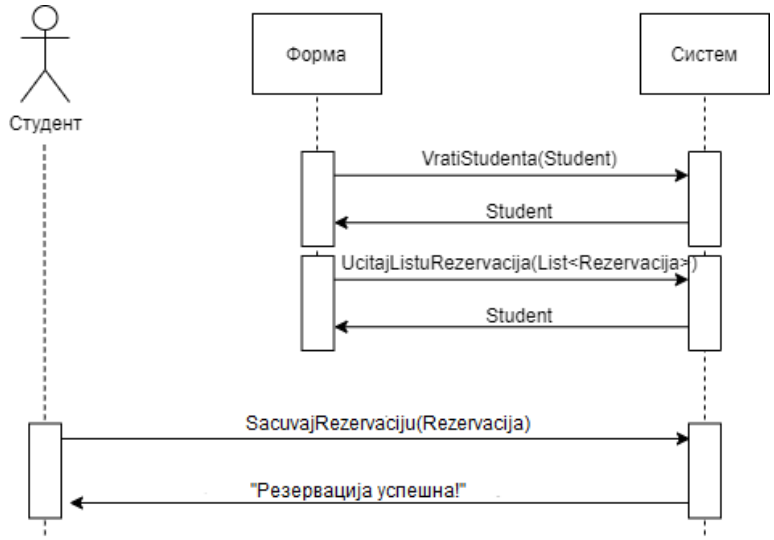
- signal VratiStudenta(Student)

- signal IzmeniStudenta(Student)

ДС7: Дијаграм секвенци случаја коришћења – Резервација машине

Основни сценарио СК:

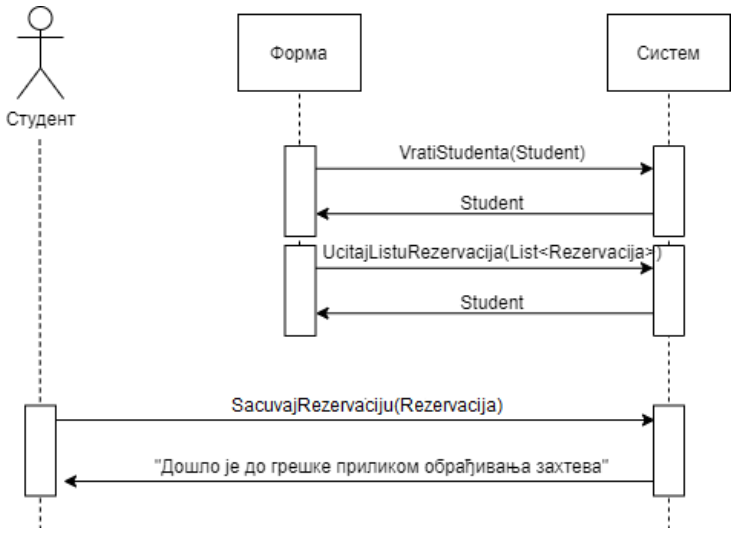
1. Форма позива систем да учита студената. (АПСО)
2. Систем враћа форми студента. (ИА)
3. Форма позива систем да учита листу резервација. (АПСО)
4. Систем враћа форми листу резервација. (ИА)
5. Студент позива систем да изврши резервацију. (АПСО)
6. Систем приказује админу поруку: “Резервација успешна!“. (ИА)



Слика 31 Резервација машине

Алтернативна сценарија

6.1. Уколико систем не може да запамти податке о резервацији, он приказује студенту поруку: “ Дошло је до грешке приликом обрађивања захтева!” .(ИА)



Слика 32Грешка приликом резервације

Идентификоване системске операције:

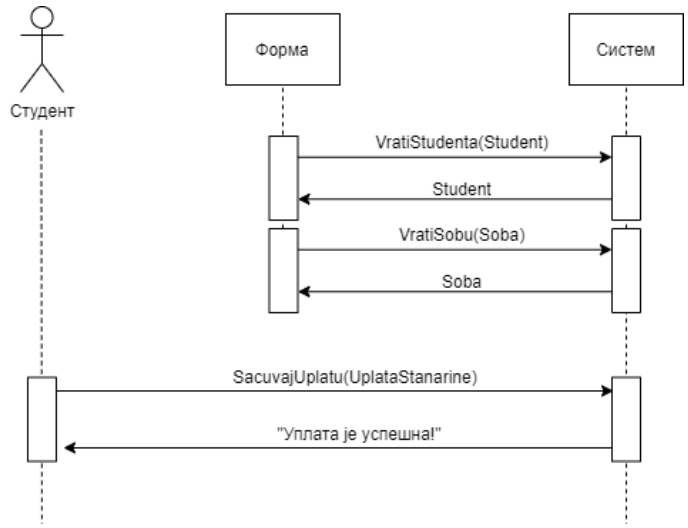
- signal VratiStudenta(Student)
- signal UcitajListuRezervacija(List<Rezervacija>)
- signal SacuvajRezervaciju(Rezervacija)

ДС8: Дијаграм секвенци случаја коришћења – Уплата станарине

Учитани су подаци о студенту и соби.

Основни сценарио СК:

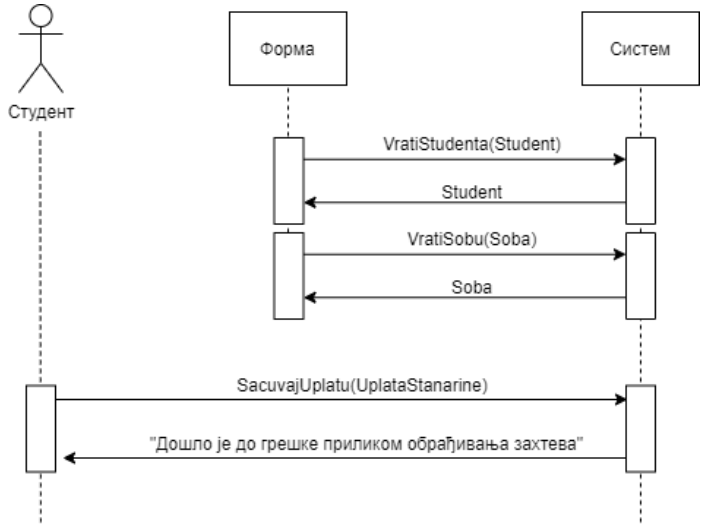
1. Форма позива систем да учита студента. (АПСО)
2. Систем враћа форми студента. (ИА)
3. Форма позива систем да учита собу. (АПСО)
4. Систем враћа форми собу. (ИА)
5. Студент позива систем да запамти уплату. (АПСО)
6. Систем приказује админу поруку: “Уплата је успешна!“. (ИА)



Слика 33 Уплата станарине

Алтернативна сценарија:

6.1. Уколико систем не може да запамти податке о уплати, он приказује студенту поруку: “Дошло је до грешке приликом обрађивања захтева!” .(ИА)



Слика 34 Грешка приликом плаћања станарине

Идентификоване системске операције:

- signal VratiStudenta(Student)
- signal VratiSobu(Soba)
- signal SacuvajUplatu(UplataStanarine)

5.2 Понашање софтверског система - Дефинисање уговора о системским операцијама

Понашање софтверског система дефинисано је системским операцијама за које се праве уговори. Уговори описују понашање саме операције, оно што операција треба да одради, али не описују начин на који ће се то одрадити. Један уговор се везује за једну системску операцију. (Влајић, 2019)

Уговор УГ1: PrijaviKorisnika

Операција: PrijaviKorisnika(username, password)

Веза са СК: СК1

Предуслови: Вредносна и структурна ограничења над објектом Студент морају бити задовољена

Постуслов: Корисник је пријављен на систем

Уговор УГ2: UcitajListuStudenata

Операција: UcitajListuStudenata(List<Student>)

Веза са СК: СК2, СК3, СК5

Предуслови:

Постуслов:

Уговор УГ3: VratiStudentePoKriterijumu

Операција:VratiStudentePoKriterijumu (Kriterijum,List<Student>)

Веза са СК: СК2, СК3, СК5

Предуслови:

Постуслови: Враћена је листа студената по критеријуму

Уговор УГ4: PrikaziPodatkeOStudentu

Операција: PrikaziPodatkeOStudentu(Student)

Веза са СК: СК2, СК3, СК5

Предуслови:

Постуслови: Подаци о студенту су приказани

Уговор УГ5: SacuvajRezervaciju

Операција: SacuvajRezervaciju (Rezervacija)

Веза са СК: СК7

Предуслови: Вредносна и структурна ограничења над објектом Rezervacija морају бити задовољена

Постуслови: Подаци о резервацији су сачувани

Уговор УГ6: IzmeniStudenta

Операција: IzmeniStudenta (Student)

Веза са СК: СК3, СК5, СК6

Предуслови: Вредносна и структурна ограничења над објектом Student морају бити задовољена

Постуслови: Подаци о студенту су измењени

Уговор УГ7: UcitajListuSoba

Операција: UcitajListuSoba(List<Soba>)

Веза са СК: СК4, СК5

Предуслови:

Постуслови:

Уговор УГ8: VratiSobePoKriterijumu

Операција: VratiSobePoKriterijumu(Kriterijum,List<Soba>)

Веза са СК: СК4, СК5

Предуслови:

Постуслови: Враћена је листа соба по критеријуму

Уговор УГ9: PrikaziPodatkeOSobi

Операција: PrikaziPodatkeOSobi (Soba)

Веза са СК: СК4, СК5

Предуслови:

Постуслови: Подаци о соби су приказани

Уговор УГ10: UcitajListuRezervacija

Операција: UcitajListuRezervacija(List<Rezervacija>)

Веза са СК: СК7

Предуслови:

Постуслови:

Уговор УГ11: VratiStudenta

Операција: VratiStudenta(Student)

Веза са СК: СК6, СК7, СК8

Предуслови:

Постуслови:

Уговор УГ12: VratiSobu

Операција: VratiSobu(Soba)

Веза са СК: СК8

Предуслови:

Постуслови:

Уговор УГ13: SacuvajUplatu

Операција: SacuvajUplatu(UplataStanarine)

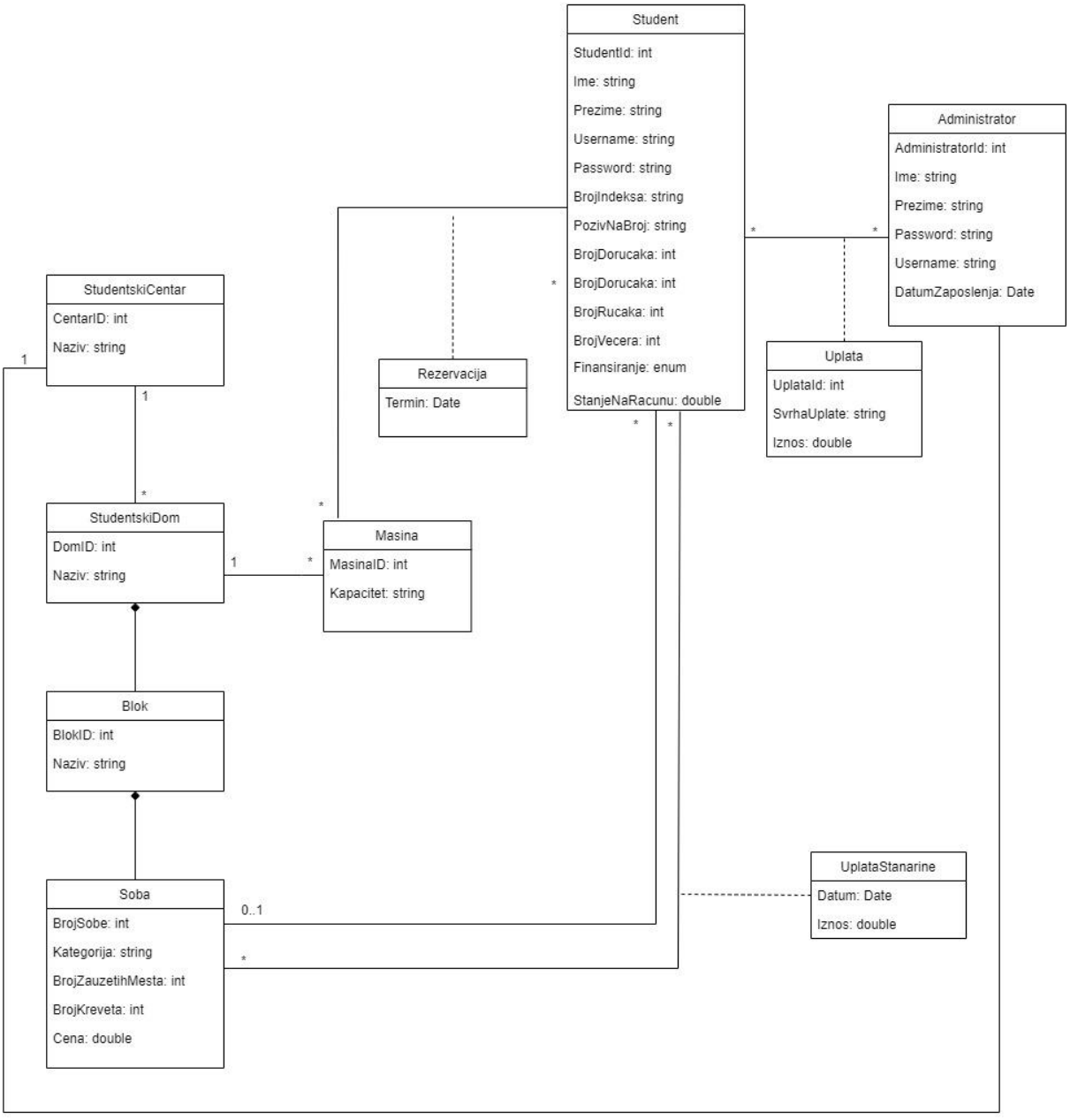
Веза са СК: СК8

Предуслови: Вредносна и структурна ограничења над објектом UplataStanarine морају бити задовољена

Постуслови: Подаци о уплати су сачувани

5.3 Структура софтверског система - Концептуални модел

Структура софтверског система се описује помоћу концептуалног модела. На концептуалном моделу приказане су концептуалне класе доменског модела као и њихове међусобне везе, односно асоцијације између њих. Називају доменским моделима или моделима објектне анализе. (Влајић, 2019)



Слика 35Дијаграм класа

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

6. Фаза пројектовања

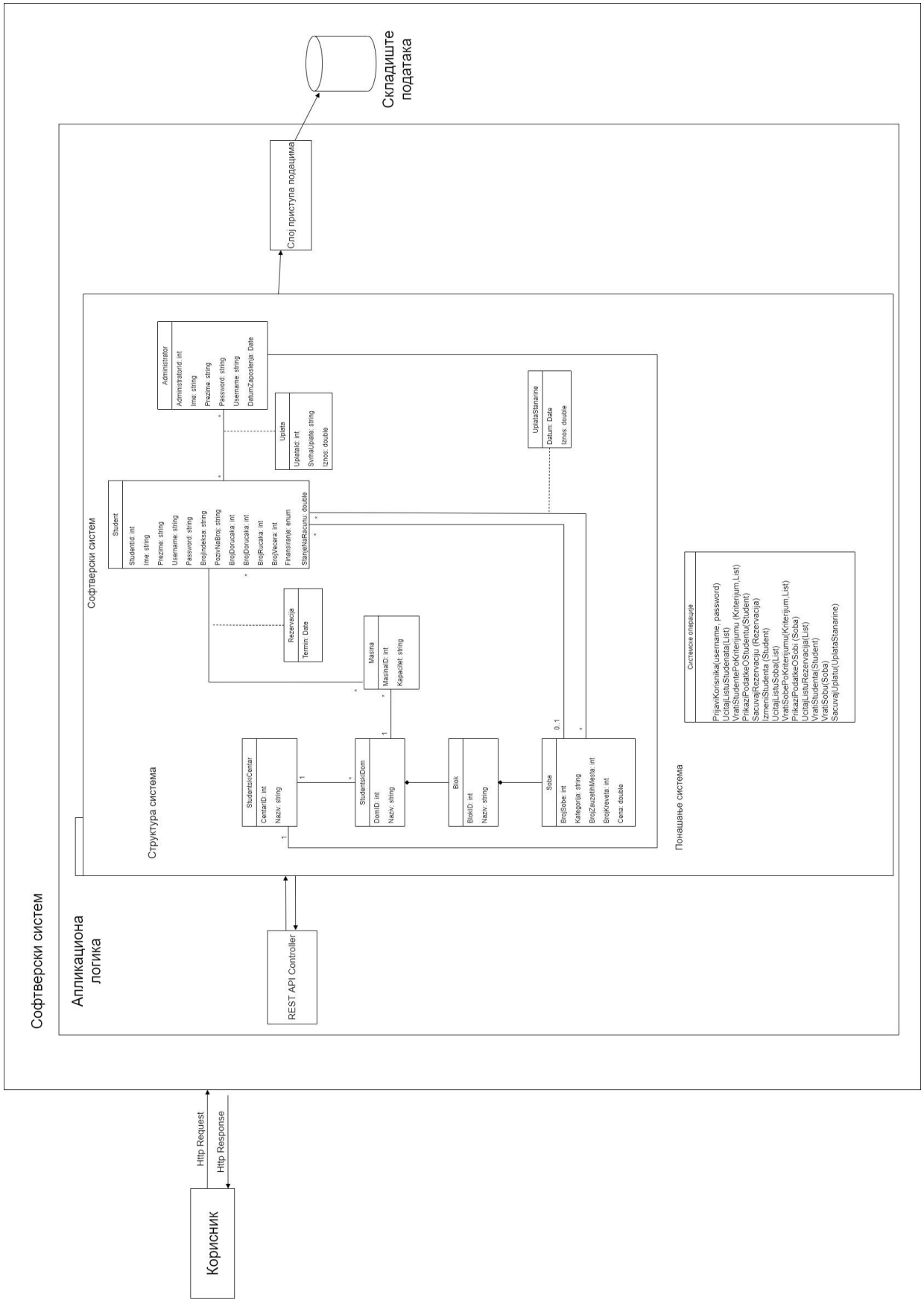
Фаза пројектовања описује физичку структуру и понашање софтверског система (архитектуру софтверскогсистема). Најчешће коришћена архитектура је тронивојска архитектура која се састоји из следећих нивоа (Влајић, 2019):

- корисничког интерфејса односно екранских форми
- апликационе логике
- складишта података

Пројектовање архитектуре софтверског система обухвата пројектовање наведена три слоја. Ниво корисничког интерфејса смештен је на страни клијента, док се апликациона логика и складиште података налазе на страни сервера.

6.1 Архитектура софтверског система

Кориснички интерфејс представља део презентационог слоја, апликациону логику чини пословна логика и слој приступа подацима је остварен преко Entity Framework Core-а. Складиште података чини релациона SQL база података.



Слика 36 Архитектура софтверског система

6.2 Структура софтверског система - Релациони модел

Релациони модел за управљање базама података представља базу података као колекцију релација. Релација је у суштини табела са вредностима. Сваки ред у табели представља колекцију међузависних вредности података и представљају ентитет или везу у реалном свету.

Релациони модел:

- StudentskiCentar (CentarID, Naziv)
- StudentskiDom (DomID, Naziv, *CentarID*)
- Blok (BlokID, *DomI*, Naziv)
- Soba (BrojSobe, *BlokID, DomID,*Kategorija, BrojZauzetihMesta, BrojKreveta, Cena)
- Administrator (AdminstratorID, Ime, Prezime, Password, Username, DatumZaposlenja, *CentarID*)
- Student (StudentID, Ime, Prezime, Username, Password, BrojIndeksa, PozivNaBroj, BrojDorucaka, BrojRucaka, BrojVecera, Finansiranje, StanjeNaRacunu, *BrojSobe*, *BlokID, DomID*)
- Masina (MasinaID, Kapacitet, *DomID*)
- Rezervacija (*MasinaID, StudentID*, Termin)
- Uplata (UplataID ,*StudentID, AdministratorID*, SvrhaUplate, Iznos)
- UplataStanarine (*StudentID, BrojSobe, BlokID, DomID*, Datum, Iznos)

Табела StudentskiCentar		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Назив атрибута	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT / UPDATE CASCADES
	CentarID	Integer	not null			Blok, StudentskiDom, Soba, Administrator, Student, Masina, UplataStanarine
	Naziv	String	not null			UPDATE RESTRICTED / DELETE RESTRICTED Blok, StudentskiDom, Soba, Administrator, Student, Masina, UplataStanarine

						Blok, Dom, Soba, Administrator, Student, Masina, UplataStanarine DELETE CASCADES /

Табела 1Табела студентски центар

Табела Studentski Dom		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Назив атрибута	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT RESTRICTED StudentskiCentar UPDATE CASCADES Blok, Soba, Student, Masina, UplataStanarine UPDATE RESTRICT StudentskiCentar DELETE RESTRICTED Blok, Soba, Student, Masina, UplataStanarine DELETE CASCADES /
	DomID	Integer	not null			
	CentarID	Integer	not null			
	Naziv	String	not null			

Табела 2 Табела студентски дом

Табела Blok		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Назив атрибута	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT RESTRICTED StudentskiDom
	BlokID	Integer	not null			UPDATE CASCADES Soba, Student, UplataStanarine
	DomID	Integer	not null			
	Naziv	String	not null			UPDATE RESTRICT StudentskiDom DELETE RESTRICTED Soba, Student, UplataStanarine

Табела 3 Табела блок

Табела Soba		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Назив атрибута	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT RESTRICTED StudentskiDom, Blok
	BrojSobe	Integer	not null			UPDATE CASCADES Student, UplataStanarine
	BlokID	Integer	not null			
	DomID	Integer	not null			
	Kategorija	String	not null			UPDATE RESTRICT
	BrojZauzetihMesta	int	not null			

	BrojKreveta	int				StudentskiCentar, StudentskiDom, Blok DELETE RESTRICTED
	Cena	double				Student, UplataStanarine

Табела 4 Табела соба

Табела Student		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Назив атрибута	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	
	StudentID	Integer	not null			INSERT RESTRICTED /
	Ime	String	not null			
	Prezime	String	not null			UPDATE RESTRICTED /
	Password	String	not null			
	Username	String	not null			UPDATE CASCADES
	BrojIndeksa	String	not null			Rezervacija, Uplata, UplataStanarine
	PozivNaBroj	String	not null			
	BrojDorucaka	Integer	not null			
	BrojRucaka	Integer	not null			DELETE RESTRICTED
	BrojVecera	Integer	not null			
	Finansiranje	String	not null			Rezervacija, Uplata, UplataStanarine
	StanjeNaRacunu	Double	not null			
	BrojSobe	Integer				
	BlokID	Integer				
	DomID	Integer				

Табела 5 Табела студент

Табела Masina		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Назив атрибута	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT RESTRICTED StudentskiDom
	MasinaID	Integer	not null			UPDATE CASCADES Rezervacija
	Kapacitet	String	not null			
	DomID	Integer	not null			UPDATE RESTRICTED StudentskiDom DLETE CASCADES / DELETE RESTRICTED Rezervacija

Табела 6 Табела машина

Табела Rezervacija		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Назив атрибута	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT RESTRICTED Masina, Student
	MasinaID	Integer	not null			UPDATE CASCADES /
	StudentID	Integer	not null			
	Termin	String	not null			UPDATE RESTRICTED Masina, Student DELETE CASCADES /

						DELETE RESTRICTED /

Табела 7 Табела резервација

Табела Uplata		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Назив атрибута	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT RESTRICTED Administrator, Student
	UplataID	Integer	not null			UPDATE CASCADES /
	StudentID	Integer	not null			
	AdministratorID	Integer	not null			UPDATE RESTRICTED Administrator, Student
	SvrhaUplate	String	not null			
	Iznos	Double	not null			DELETE CASCADES / DELETE RESTRICTED /

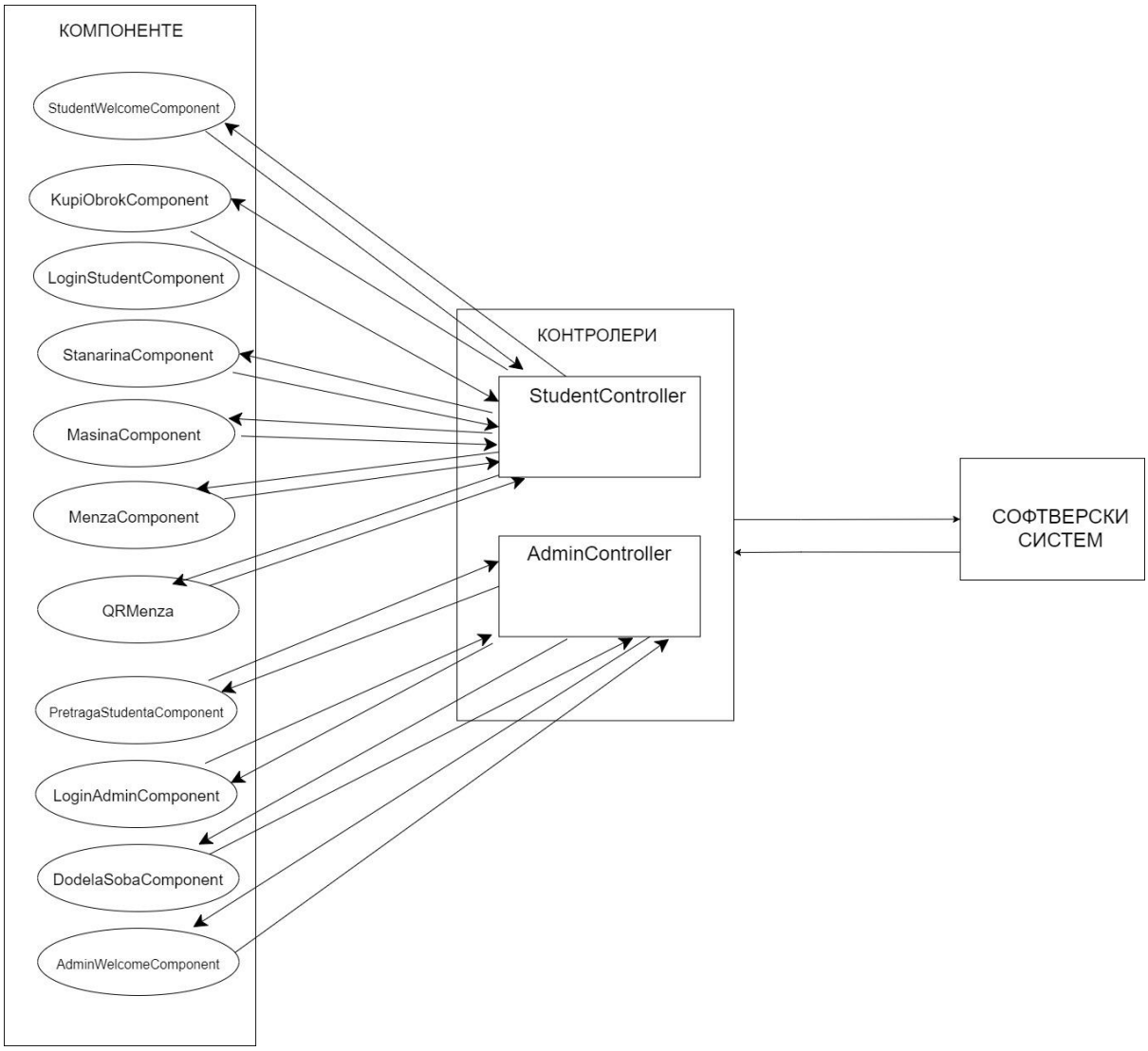
Табела 8 Табела уплата

Табела UplataStanarine		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Назив атрибута	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT RESTRICTED Student, Soba, Blok, StudentskiDom
	BrojSobe	Integer	not null			UPDATE CASCADES /
	StudentID	Integer	not null			
	BlokID	Integer	not null			UPDATE RESTRICTED
	DomID	String	not null			Student, Soba, Blok, StudentskiDom
	Datum					DELETE CASCADES /
	Iznos					DELETE RESTRICTED /

Табела 9 Табела уплата станарине

6.3 Пројектовање екранских форми

Кориснички интерфејс представља реализацију улаза и/или излаза софтверског система (Влајић, 2019), Кориснички интерфејс чине веб странице које прихватају податке које корисник уноси, шаљу HTTP захтев ка серверу и прихватају одговор од сервера на тај захтев, након чега се тај одговор приказује кориснику.



Слика 37Повезаност контролера са корисничким интерфејсом

СК1: Случај коришћења - Пријављивање на систем

Назив СК

Пријављивање на систем

Актори СК

Корисник (администратор или студент)

Учесници СК

Корисник и систем (програм)

Предуслов: Систем је укључен. Систем приказује форму за пријављивање на систем.

Слика 37Пријављивање корисника на систем

Основни сценарио СК

1. Корисник уноси корисничко име и лозинку. (АПУСО)

2. Корисник контролише да ли је коректно унео корисничко име и лозинку. (АНСО)

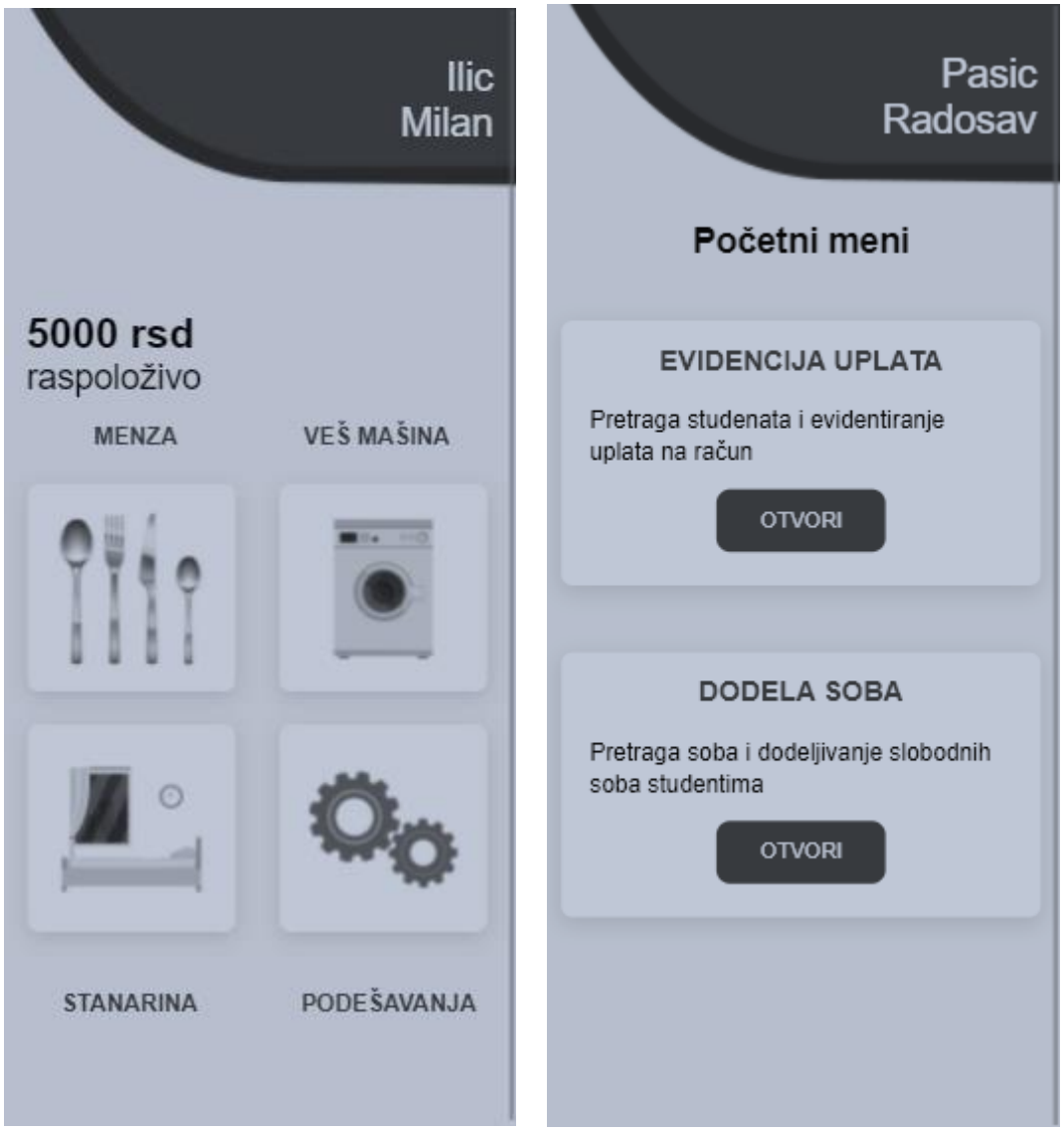
3. Корисник позива систем да се пријави на систем (провери податке). (АПСО)

Опис акције: Корисник кликом на дугме „Prijavi se“ позива системску операцију PrijaviKorisnika(username,password)

4. Систем проверава податке о кориснику. (СО)

5. Систем приказује кориснику почетну страну и поруку:

“Успешно пријављивање!“. (ИА)

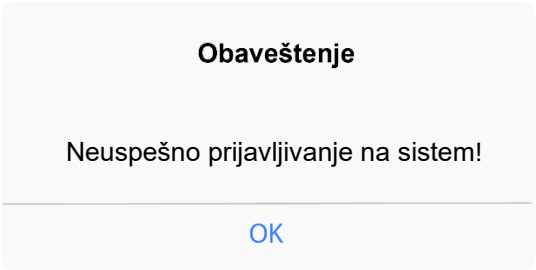


Слика 38 Почетна страна- стидент- администратор

Алтернативна сценарија

5.1. Уколико систем не може да нађе корисника, он приказује кориснику поруку:

“ Неуспешно пријављивање на систем!”. (ИА)



Слика 39Неуспешно пријављивање

СК2: Случај коришћења - Претрага студентских налога по критеријуму

Назив СК

Претрага студентских налога по критеријуму

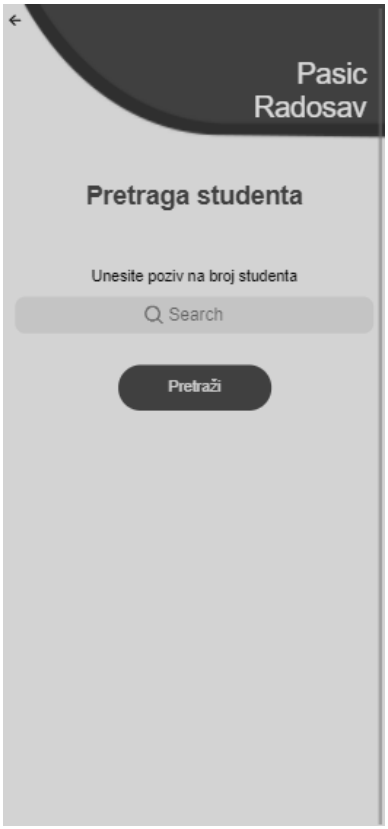
Актори СК

Администратор

Учесници СК

Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је пријављен на систем. Систем приказује форму за рад са студентима. Учитана је листа студената.



Слика 40Форма за рад са студентима

Основни сценарио СК

1. Администратор уноси вредност по којој претражује студента. (АПУСО)

2. Администратор позива систем да нађе студенте по задатој вредности. (АПСО)

Опис акције: Администратор кликом на дугме „Pretraži“ позива системску операцију VratiStudentePoKriterijumu(Kriterijum, List<Student>)

3. Систем тражи студенте по задатој вредности. (СО)

4. Систем приказује администратору нађене студенте. (ИА)

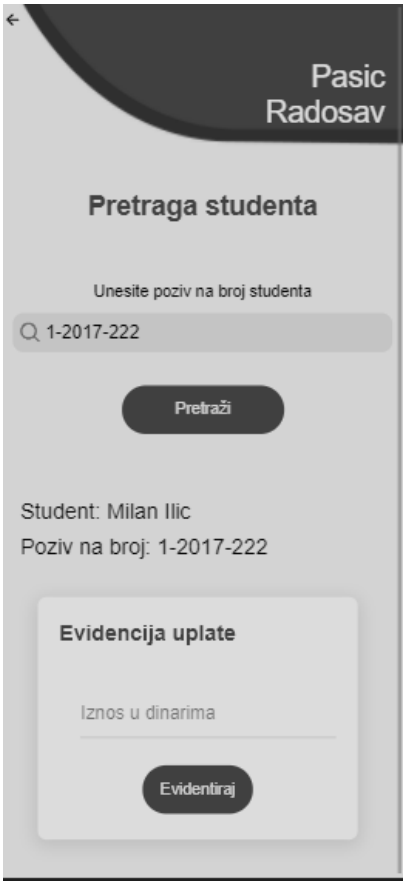
5. Администратор бира студента чије податке жели да види.(АПУСО)

6. Администратор позива систем да прикаже податке о изабраном студенту.(АПСО)

Опис акције: Администратор кликом на картицу студента позива системску операцију PrikaziPodatkeOStudentu(Student)

7. Систем проналази податке о изабраном студенту.(СО)

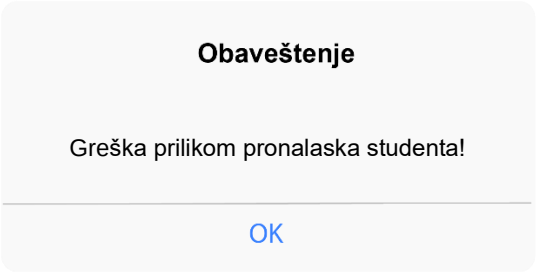
8. Систем приказује администратору податке о изабраном студенту.(ИА)



Слика 41 Подаци о студенту

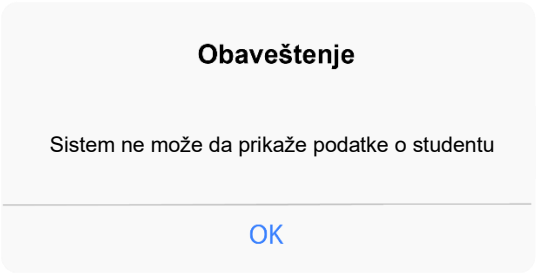
Алтернативна сценарија

4.1 Уколико систем не може да пронађе студента по задатим вредностима приказује се порука: ”Грешка приликом проналаска студента“. (ИА)



Слика 42Неуспешна претрага студента

8.1. Уколико систем не може да прикаже податке о изабраном студенту приказује се порука „Систем не може да прикаже податке о студенту“(ИА)



Слика 43Неуспешна претрага студента

СК3: Случај коришћења - Евидентирање уплате

Назив СК

Евидентирање уплате

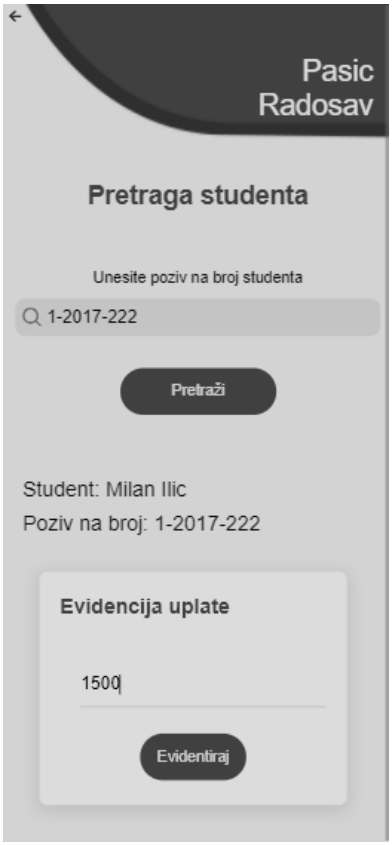
Актори СК

Администратор

Учесници СК

Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је пријављен на систем. Систем приказује форму за рад са студентима. Учитана је листа студената.



Слика 44 Евиденција упкате

Основни сценарио СК

1. Администратор уноси вредност по којој претражује студента. (АПУСО)

2. Администратор позива систем да нађе студенте по задатој вредности. (АПСО)

Опис акције: Администратор кликом на дугме „Pretraži“ позива системску операцију VratiStudentePoKriterijumu(Kriterijum, List<Student>)

3. Систем тражи студенте по задатој вредности. (СО)

4. Систем приказује администратору нађене студенте. (ИА)

5. Администратор бира студента чије податке жели да измени.(АПУСО)

6. Администратор позива систем да учита податке о студенту. (АПСО)

Опис акције: Администратор кликом на картицу студента позива системску операцију PrikaziPodatkeOStudentu(Student)

7. Систем учитава податке о студенту.(СО)

8. Систем приказује администратору задатог студента.(ИА)

9. Администратор мења податке о студенту.(АПУСО)

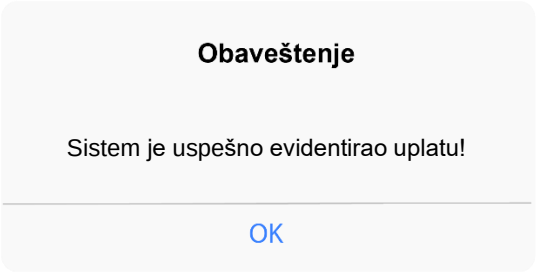
10. Администратор проверава да ли је исправно унео измењене податке.(АНСО)

11. Администратор позива систем да запамти податке о студенту.(АПСО)

Опис акције: Администратор кликом на дугме „Evidentiraj“ позива системску операцију IzmeniStudenta(Student)

12. Систем памти измењене податке о студенту.(СО)

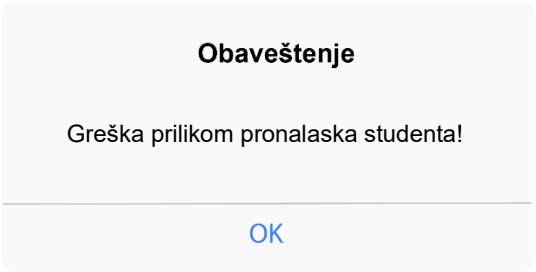
13. Систем приказује администратору поруку:“Систем је успешно евидентирао уплату.“(ИА)



Слика 45 Евиденција уплате успешна

Алтернативна сценарија

4.1. Уколико систем не може да пронађе задатог студента приказује се порука: ”Грешка приликом проналаска студента”. Прекида се извршење сценарија. (ИА)

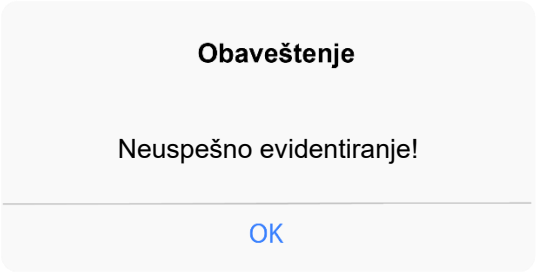


Слика 46Неуспешна претрага студента

8.1. Уколико систем не пронађе информације о студенту, систем приказује администратору поруку:“Систем не може да прикаже податке о студенту“. Прекида се извршење сценарија.(ИА)

Слика 47Неуспешна претрага студента

13.1. Уколико систем не може да сачува измене о студенту, систем приказује следећу поруку:“Неуспешно евидентирање.“(ИА)



Слика 48 Неуспешна уплата

СК4: Случај коришћења - Претрага соба по критеријуму

Назив СК

Претрага соба по критеријуму

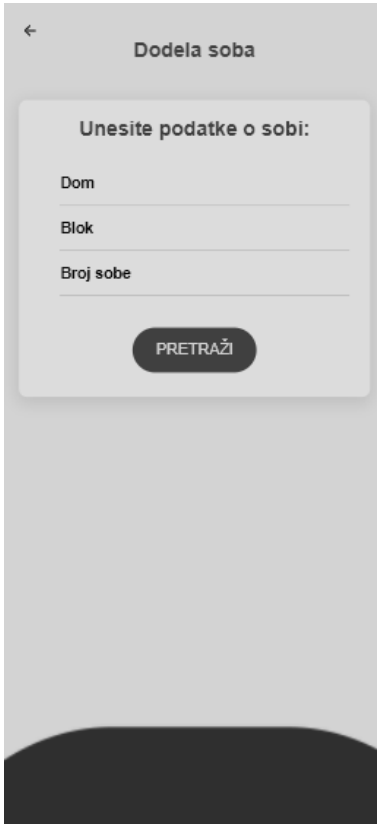
Актори СК

Администратор

Учесници СК

Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је пријављен на систем. Систем приказује форму за рад са собама. Учитана је листа соба.



Слика 49 Форма за претрагу соба

Основни сценарио СК

1. Администратор уноси вредност по којој претражује собу. (АПУСО)
2. Администратор позива систем да нађе собе по задатој вредности. (АПСО)

Опис акције: Администратор кликом на дугме „Pretraži“ позива системску операцију VratiSobePoKriterijumu(Kriterijum,List<Soba>)

3. Систем тражи собе по задатој вредности. (СО)
4. Систем приказује администратору нађене собе. (ИА)
5. Администратор бира собу чије податке жели да види.(АПУСО)
6. Администратор позива систем да прикаже податке о изабраној соби.(АПСО)

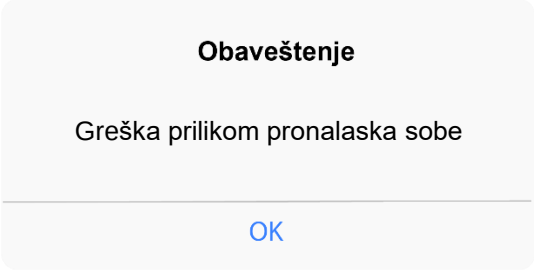
Опис акције: Администратор кликом на картицу собе позива системску операцију PrikaziPodatkeOSobi(Soba)

7. Систем проналази податке о изабраној соби.(СО)
8. Систем приказује администратору податке о изабраној соби.(ИА)

Слика 50 Резултат претраге собе

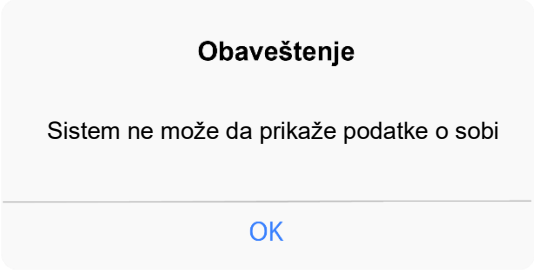
Алтернативна сценарија

4.1 Уколико систем не може да пронађе собу по задатим вредностима приказује се порука: ”Грешка приликом проналаска собе“. (ИА)



Слика 51 Грешка приликом проналаска собе

8.1. Уколико систем не може да прикаже податке о изабраној соби приказује се порука „Систем не може да прикаже податке о соби“(ИА)



Слика 52 Грешка приликом приказивања собе

СК5: Случај коришћења - Додељивање собе студенту

Назив СК

Додељивање собе студенту

Актори СК

Администратор

Учесници СК

Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је пријављен на систем. Систем приказује форму за доделу соба. Учитане су листа студената и листа соба.

Слика 53 Форма за доделу собе

Основни сценарио СК

1. Администратор уноси вредност по којој претражује собу. (АПУСО)
2. Администратор позива систем да нађе собе по задатој вредности. (АПСО)

Опис акције: Администратор кликом на дугме „Pretraži“ позива системску операцију VratiSobePoKriterijumu(Kriterijum,List<Soba>)

3. Систем тражи собе по задатој вредности. (СО)
4. Систем приказује администратору нађене собе. (ИА)
5. Администратор бира собу чије податке жели да види.(АПУСО)
6. Администратор позива систем да прикаже податке о изабраној соби.(АПСО)

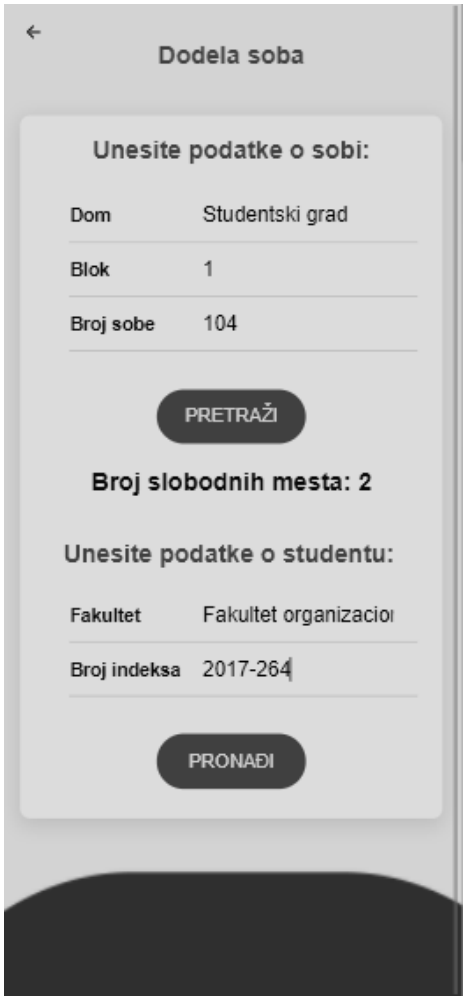
Опис акције: Администратор кликом на картицу собе позива системску операцију PrikaziPodatkeOSobi(Soba)

7. Систем проналази податке о изабраној соби.(СО)
8. Систем приказује администратору податке о изабраној соби.(ИА)
9. Администратор уноси вредност по којој претражује студента. (АПУСО)
10. Администратор позива систем да нађе студенте по задатој вредности. (АПСО)

Опис акције: Администратор кликом на дугме „Pronađi“ позива системску операцију VratiStudentePoKriterijumu(Kriterijum,List<Student>)

11. Систем тражи студенте по задатој вредности. (СО)

12. Систем приказује администратору нађене студенте. (ИА)



Слика 54 Претрага студента

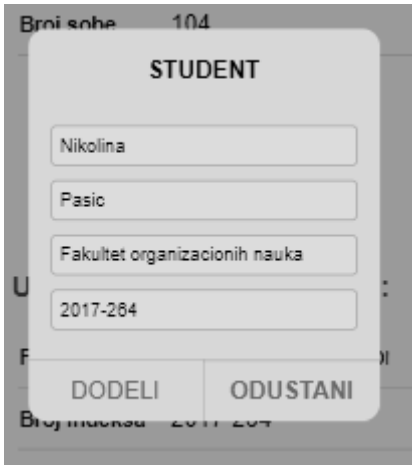
13. Администратор бира студента чије податке жели да измени.(АПУСО)

14. Администратор позива систем да учита податке о студенту. (АПСО)

Опис акције: Администратор кликом на картицустудента позива системску операцију PrikaziStudenta (Student)

15. Систем учитава податке о студенту.(СО)

16. Систем приказује администратору задатог студента.(ИА)



Слика 55 Подаци о студенту

17. Администратор мења податке о студенту.(АПУСО)

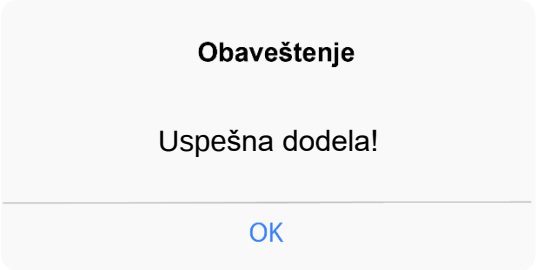
18. Администратор проверава да ли је исправно унео измењене податке.(АНСО)

19. Администратор позива систем да запамти податке о студенту.(АПСО)

Опис акције: Администратор кликом на дугме „Dodeli“ позива системску операцију DodeliSobu(Student,Soba)

20. Систем памти измењене податке о студенту.(СО)

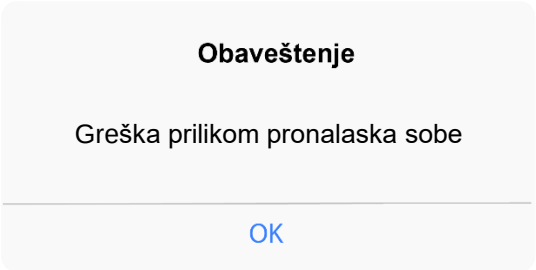
21. Систем приказује администратору поруку:“Успешна додела!“(ИА)



Слика 56 Успешна додела

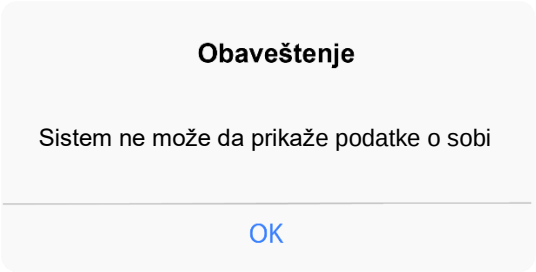
Алтернативна сценарија

4.1. Уколико систем не може да пронађе задату собу приказује се порука: ”Грешка приликом проналаска собе”. Прекида се извршење сценарија. (ИА)



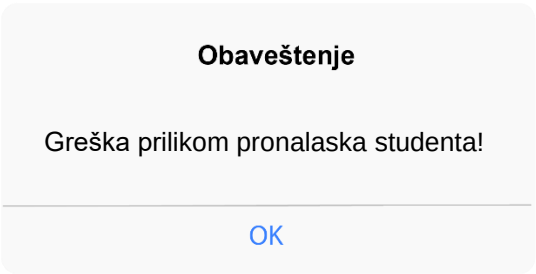
Слика 57 Грешка приликом проналаска собе

8.1. Уколико систем не пронађе информације о соби, систем приказује администратору поруку:“Систем не може да прикаже податке о соби“. Прекида се извршење сценарија.(ИА)



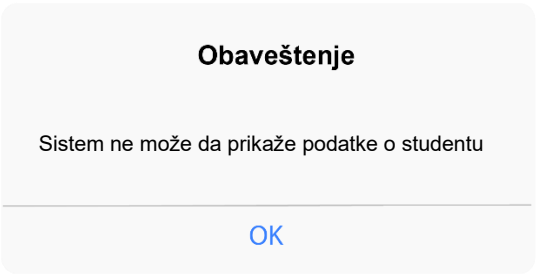
Слика 58 Грешка приликом приказивања собе

12.1. Уколико систем не може да пронађе задатог студента приказује се порука: ”Грешка приликом проналаска студента”. Прекида се извршење сценарија. (ИА)



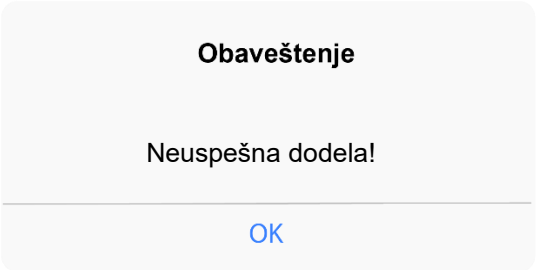
Слика 59 Грешка приликом проналаска студента

16.1. Уколико систем не пронађе информације о студенту, систем приказује администратору поруку:“Систем не може да прикаже податке о студенту“. Прекида се извршење сценарија.(ИА)



Слика 60 Грешка приликом приказивања студента

21.1. Уколико систем не може да сачува измене о студенту, систем приказује следећу поруку:“Неуспешна додела.“(ИА)



Слика 61 Неуспешна додела собе

СК6: Случај коришћења - Куповина оброка

Назив СК:

Куповина оброка

Аутори СК:

Студент

Учесници СК:

Студент и систем

Предуслов: Систем је укључен. Систем приказује страну за куповину оброка. Учитани су подаци о студенту.

Слика 62 Куповина оброка

Основни сценарио СК:

1. Студент уноси количину оброка које жели да купи (АПУСО)

2. Студент контролише да ли је коректно унео количину оброка. (АНСО)

3. Студент позива систем да запамти измењене податке на студентском налогу. (АПСО)

Опис акције: Студент кликом на дугме „Potvrdi“ позива системску операцију IzmeniStudenta(Student)

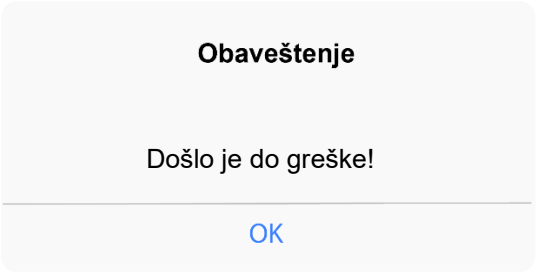
4. Систем памти измењене податке на студентском налогу (СО)

5. Систем приказује кориснику поруку: “Успешно сте купили оброке”. (ИА)

Слика 63 Избор оброка и куповина

Алтернативни сценарио СК:

5.1 Уколико систем не може да измени податке о корисничком налогу, систем приказује кориснику поруку: “Дошло је до грешке ”. (ИА)



Слика 61 Грешка приликом куповине

СК7: Случај коришћења - Резервација машине

Назив СК:

Резервација машине

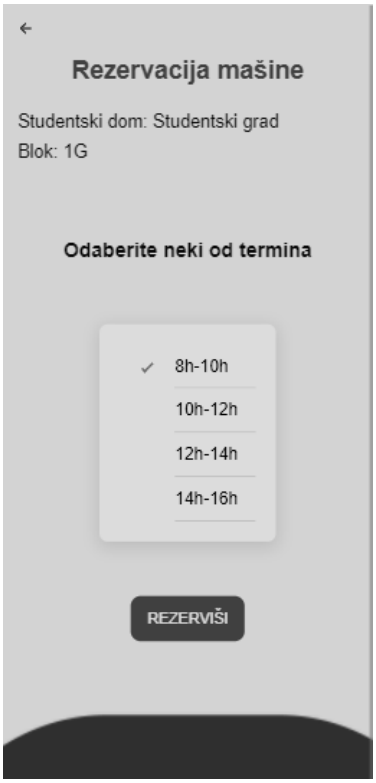
Аутори СК:

Студент

Учесници СК:

Студент и систем

Предуслов: Систем је укључен. Систем приказује страну за резервацију машине. Учитани су подаци о студенту и листа резервација.



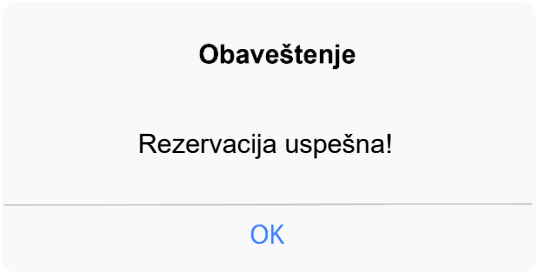
Слика 64 Форма за резервацију машине

Основни сценарио СК:

1. Студент уноси термин у којем жели да резервише машину. (АПУСО)
2. Студент контролише да ли је коректно унео податке о термину. (АНСО)
3. Студент позива систем да изврши резервацију. (АПСО)

Опис акције: Студент кликом на дугме „Rezerviši“ позива системску операцију SacuvajRezervaciju(Rezervacija)

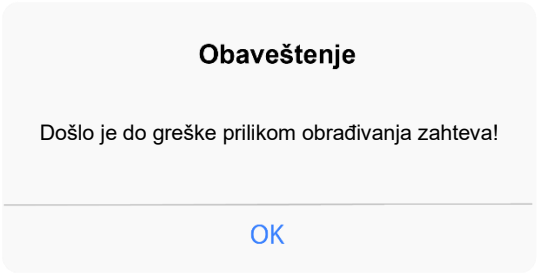
4. Систем памти податке о резервацији. (СО)
5. Систем приказује админу поруку: “Резервација успешна!“. (ИА)



Слика 65 Успешна резервација

Алтернативна сценарија

5.1. Уколико систем не може да запамти податке о резервацији, он приказује студенту поруку: “Дошло је до грешке приликом обрађивања захтева!” .(ИА)



Слика 66 Грешка приликом резервације

СК8: Случај коришћења - Уплата станарине

Назив СК:

Уплата станарине

Аутори СК:

Студент

Учесници СК:

Студент и систем

Предуслов: Систем је укључен. Систем приказује страну за плаћање станарине. Учитани су подаци о студенту и соби.

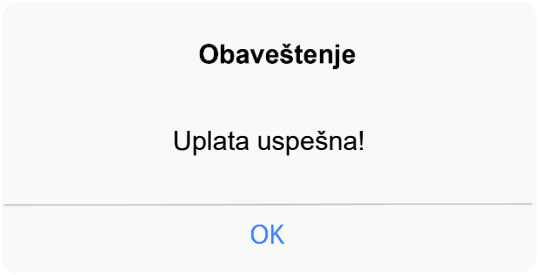
Слика 67 Форма за уплату станарине

Основни сценарио СК:

1. Студент уноси износ цене станарине. (АПУСО)
2. Студент контролише да ли је коректно унео податке о станарини. (АНСО)
3. Студент позива систем да запамти уплату. (АПСО)

Опис акције: Студент кликом на дугме „Plati stanarinu“ позива системску операцију SacuvajUplatu(UplataStanarine)

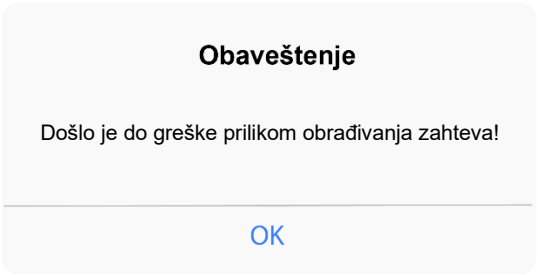
4. Систем памти податке о уплати. (СО)
5. Систем приказује админу поруку: “Уплата успешна!“. (ИА)



Слика 68 Успешна уплата станарине

Алтернативна сценарија:

5.1. Уколико систем не може да запамти податке о уплати, он приказује студенту поруку: “Дошло је до грешке приликом обрађивања захтева!” .(ИА)



Слика 69 Грешка приликом плаћања станарине

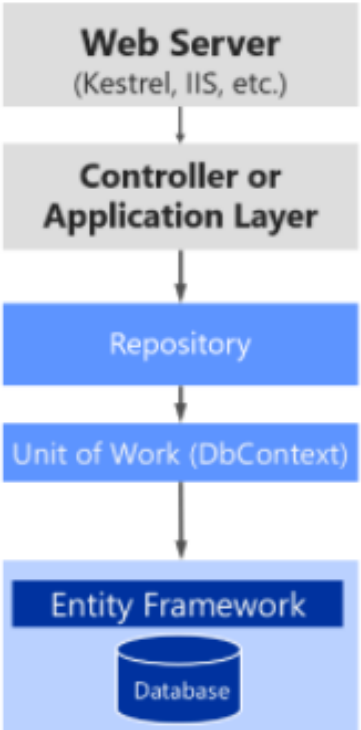
6.4 Пројектовање апликационе логике

Апликациона логика садржи класе које су неопходне за имплементацију пословне логике, а то су:

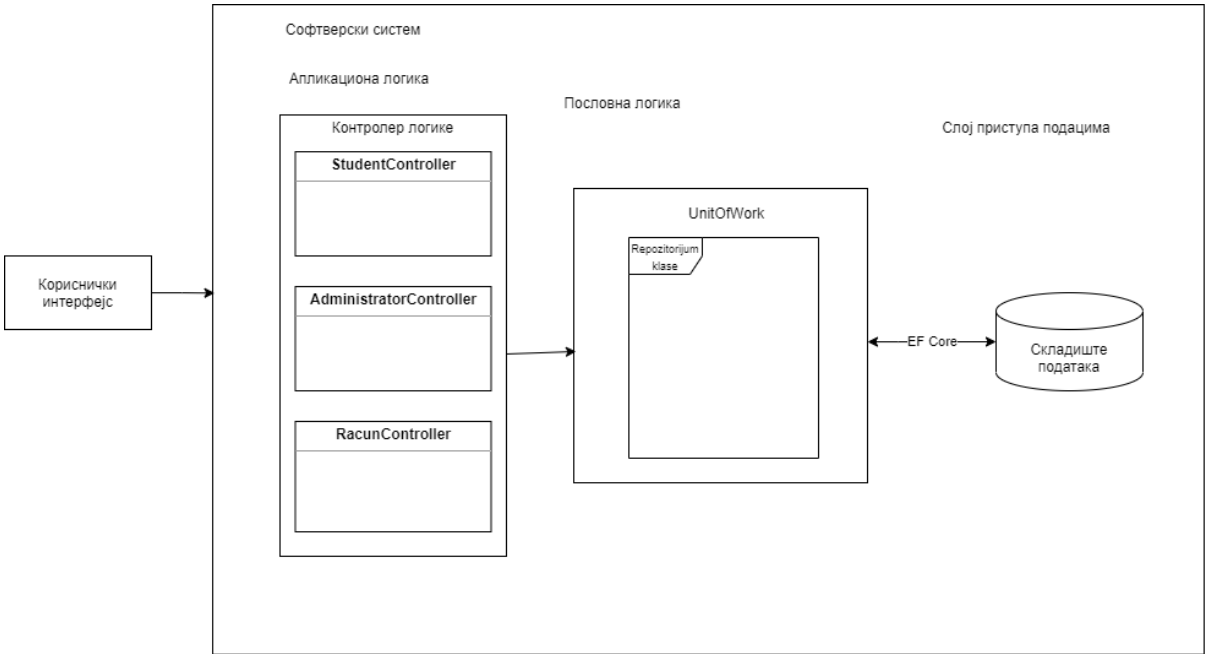
- **Контролери** – прихватају захтеве и прослеђују их сервисима на обраду
- **UnitOfWork** класа – осигурава да када се користе више Repository инстанца, оне деле један објекат класе
- **DatabaseContext** како не би дошло до конфилкта са базом, координира складиштење измена и решавање проблема када је у питању конкурентност
- **Repository** класе – служе за комуникацију са базом
- **Entity** објекти – представљају табеле у бази података, резултат објектно-релационог мапирања

6.4.1 Контролер апликационе логике

Контролер прихвата захтеве који пристижу са клијентске стране и прослеђује их даље на обраду. Након обраде, контролер прихвата одговор и враћа назад клијенту одгвор.



Слика 70 Путања корисничког захтева



Слика 71Архитектура софтверског система након пројектовања контролера и класа које чине апликациону логику

6.4.2 Пословна логика

Пословна логика је описана структуром (доменским класама) и понашањем (системским операцијама). За сваки од уговора системских операција дефинисаних у фази анализе пројектује се концептуално решење.

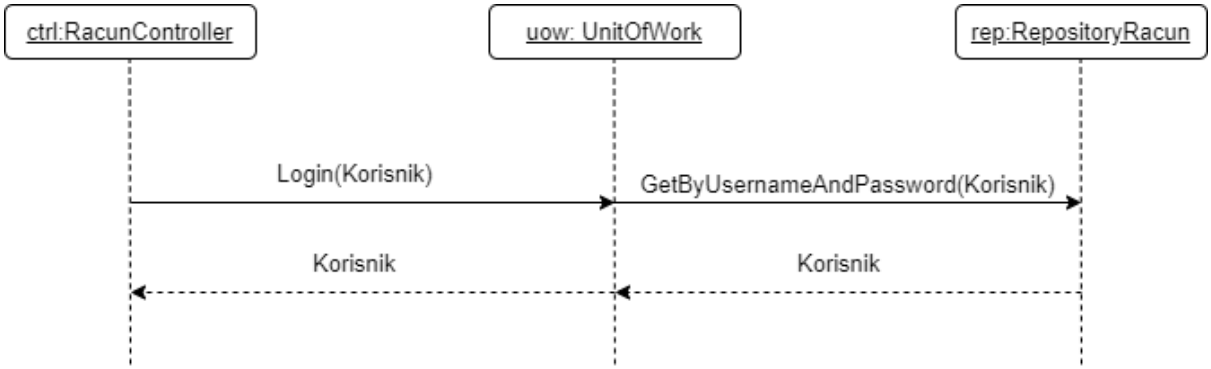
Уговор УГ1: PrijaviKorisnika

Операција: PrijaviKorisnika(username, password)

Веза са СК: СК1

Предуслови: Вредносна и структурна ограничења над објектом Студент морају бити задовољена

Постуслов: Корисник је пријављен на систем



Слика 72Дијаграм секвенци: Уговор - Пријави корисника

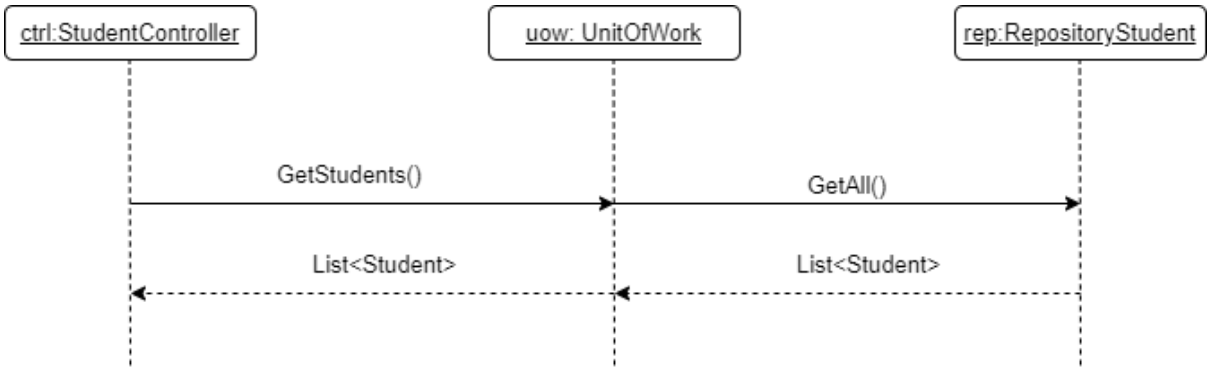
Уговор УГ2: UcitajListuStudenata

Операција: UcitajListuStudenata(List<Student>)

Веза са СК: СК2, СК3, СК5

Предуслови:

Постуслов:



Слика 73Дијаграм секвенци:Уговор - Учитај листу студената

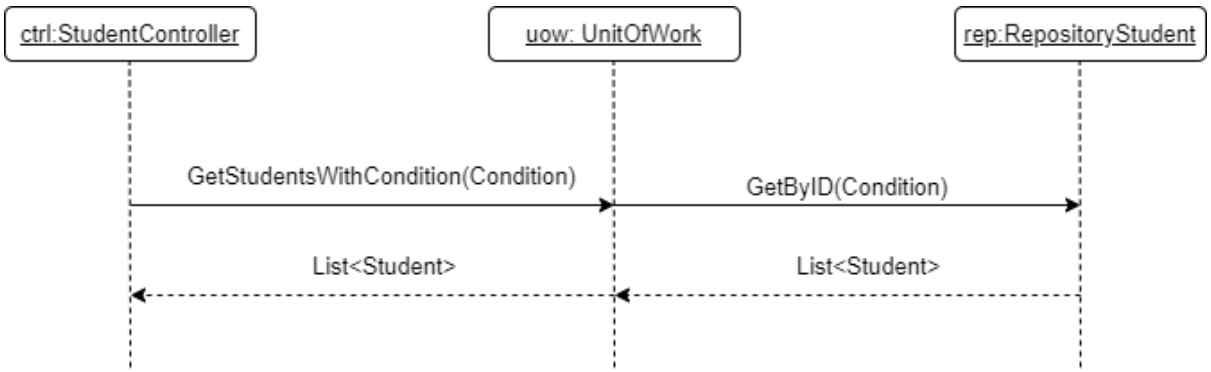
Уговор УГ3: VratiStudentePoKriterijumu

Операција:VratiStudentePoKriterijumu (Kriterijum,List<Student>)

Веза са СК: СК2, СК3, СК5

Предуслови:

Постуслови: Враћена је листа студената по критеријуму



Слика 74Дијаграм секвенци: Уговор- Врати студенте по критеријуму

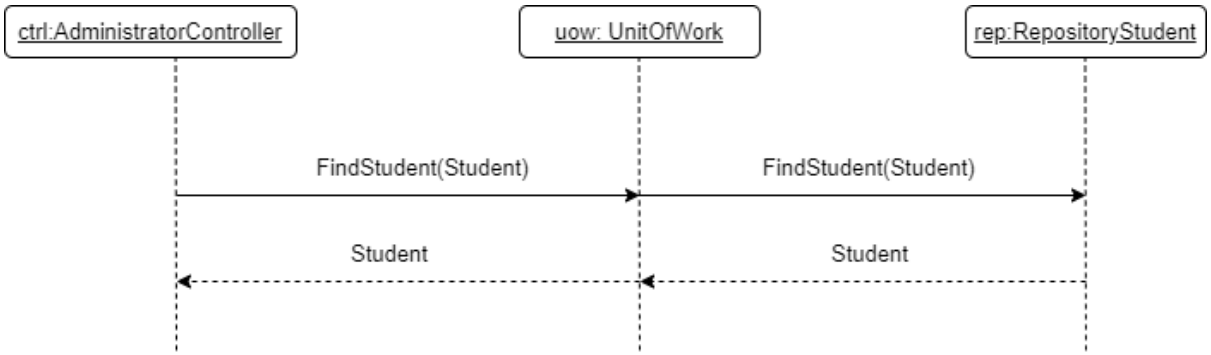
Уговор УГ4: PrikaziPodatkeOStudentu

Операција: PrikaziPodatkeOStudentu(Student)

Веза са СК: СК2, СК3, СК5

Предуслови:

Постуслови: Подаци о студенту су приказани



Слика 75Дијаграм секвенци: Уговор- Прикажи податке о студенту

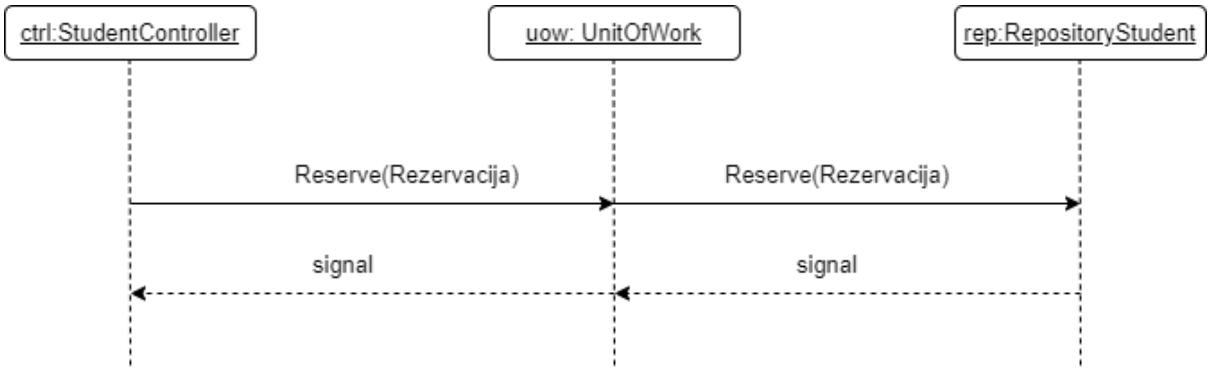
Уговор УГ5: SacuvajRezervaciju

Операција: SacuvajRezervaciju (Rezervacija)

Веза са СК: СК7

Предуслови: Вредносна и структурна ограничења над објектом Rezervacija морају бити задовољена

Постуслови: Подаци о резервацији су сачувани



Слика 76 Дијаграм секвенци: Уговор- Сачувај резервацију

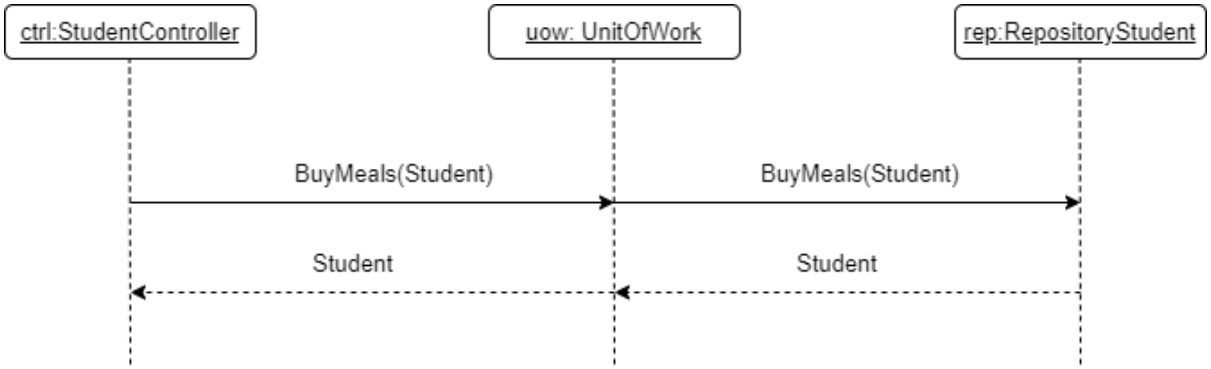
Уговор УГ6: IzmeniStudenta

Операција: IzmeniStudenta (Student)

Веза са СК: СК3, СК5, СК6

Предуслови: Вредносна и структурна ограничења над објектом Student морају бити задовољена

Постуслови: Подаци о студенту су измењени



Слика 77 Дијаграм секвенци: Уговор- Измени студента

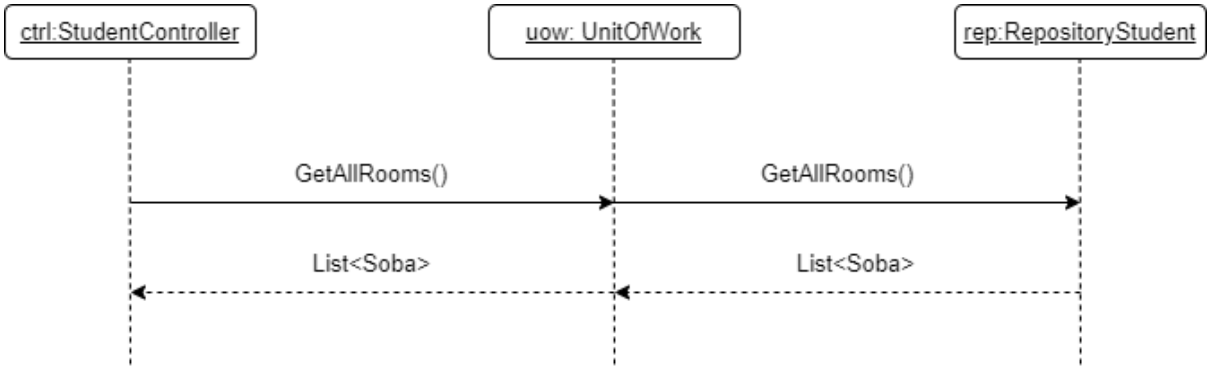
Уговор УГ7: UcitajListuSoba

Операција: UcitajListuSoba(List<Soba>)

Веза са СК: СК4, СК5

Предуслови:

Постуслови:



Слика 78 Дијаграм секвенци: Уговор- Учитај листу соба

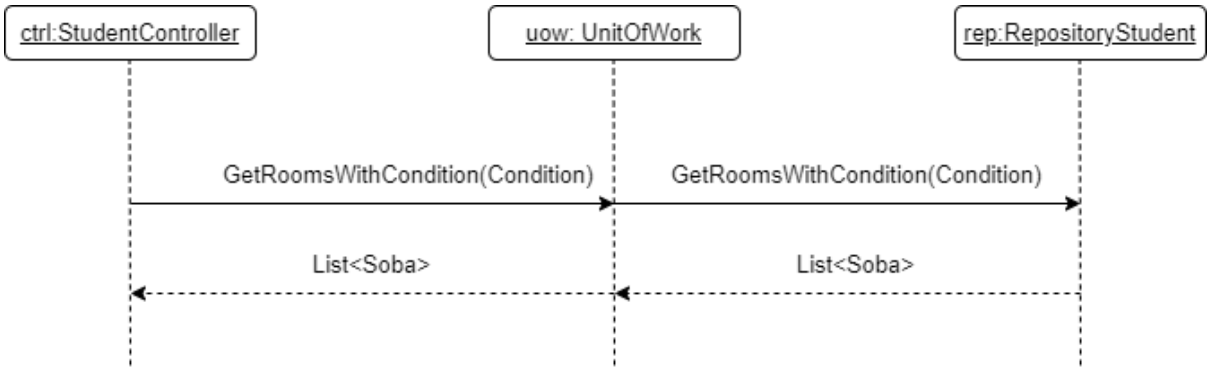
Уговор УГ8: VratiSobePoKriterijumu

Операција: VratiSobePoKriterijumu(Kriterijum,List<Soba>)

Веза са СК: СК4, СК5

Предуслови:

Постуслови: Враћена је листа соба по критеријуму



Слика 79 Дијаграм секвенци: Уговор- Врати собе по критеријуму

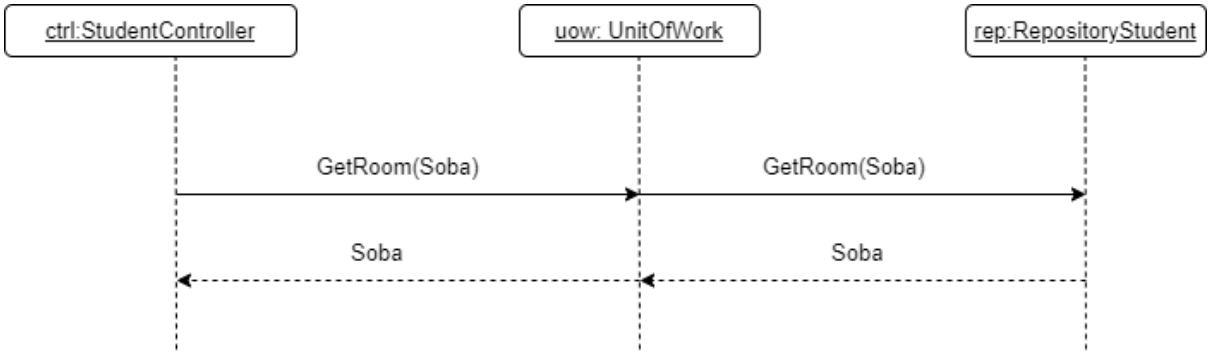
Уговор УГ9: PrikaziPodatkeOSobi

Операција: PrikaziPodatkeOSobi (Soba)

Веза са СК: СК4, СК5

Предуслови:

Постуслови: Подаци о соби су приказани



Слика 80 Дијаграм секвенци: Уговор- Прикааз података о соби

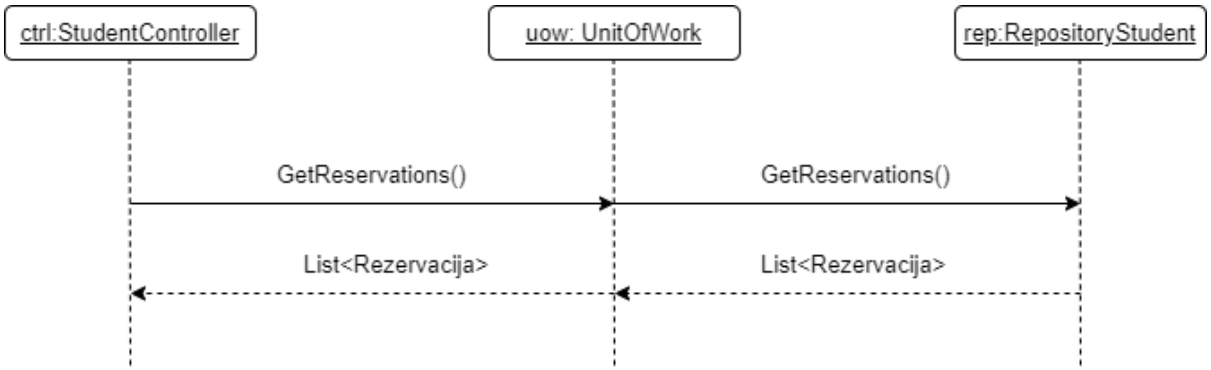
Уговор УГ10: UcitajListuRezervacija

Операција: UcitajListuRezervacija(List<Rezervacija>)

Веза са СК: СК7

Предуслови:

Постуслови:



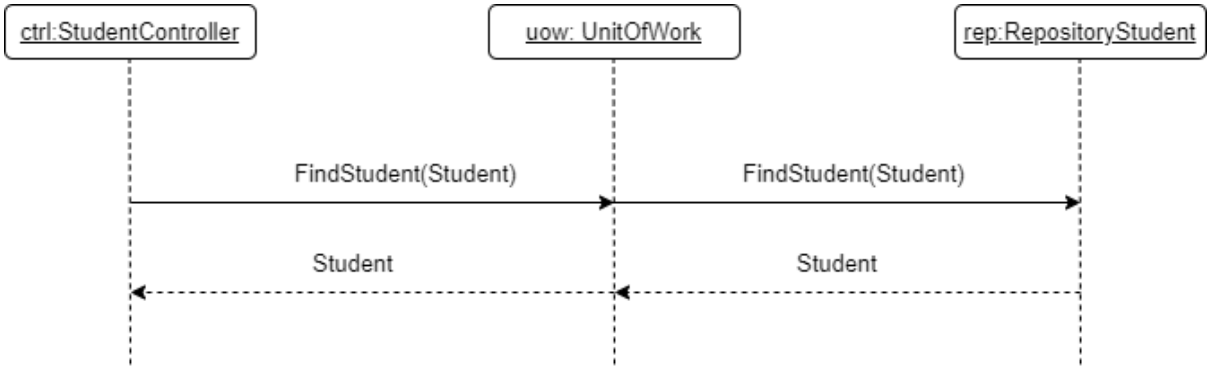
Уговор УГ11: VratiStudenta

Операција: VratiStudenta(Student)

Веза са СК: СК6, СК7, СК8

Предуслови:

Постуслови:



Слика 81 Дијаграм секвенци: Уговор- Врати студента

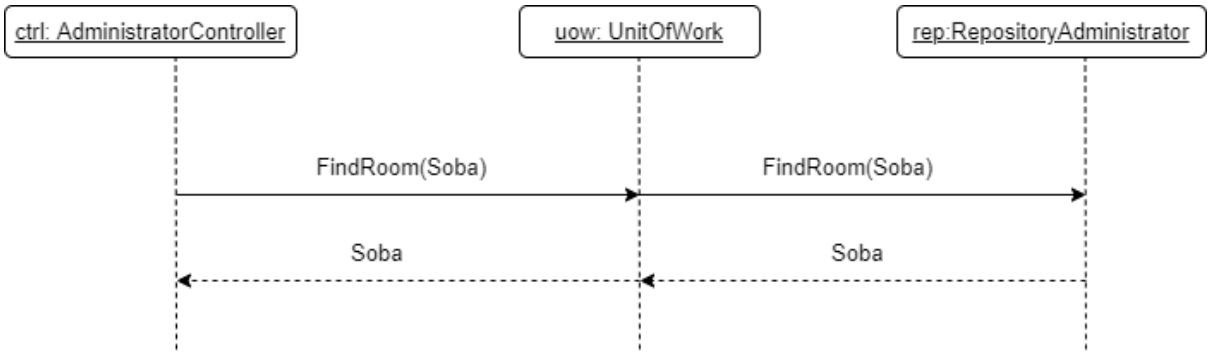
Уговор УГ12: VratiSobu

Операција: VratiSobu(Soba)

Веза са СК: СК8

Предуслови:

Постуслови:



Слика 82 Дијаграм секвенци: Уговор- Врати собу

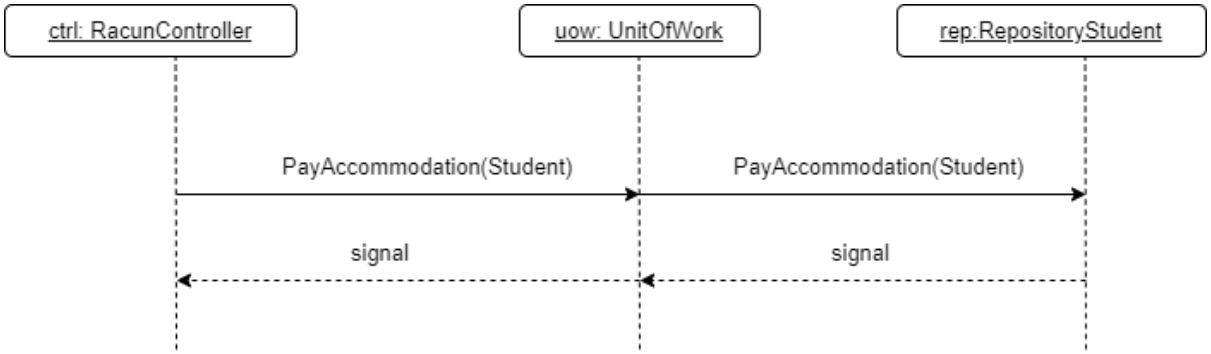
Уговор УГ13: SacuvajUplatu

Операција: SacuvajUplatu(UplataStanarine)

Веза са СК: СК8

Предуслови: Вредносна и структурна ограничења над објектом UplataStanarine морају бити задовољена

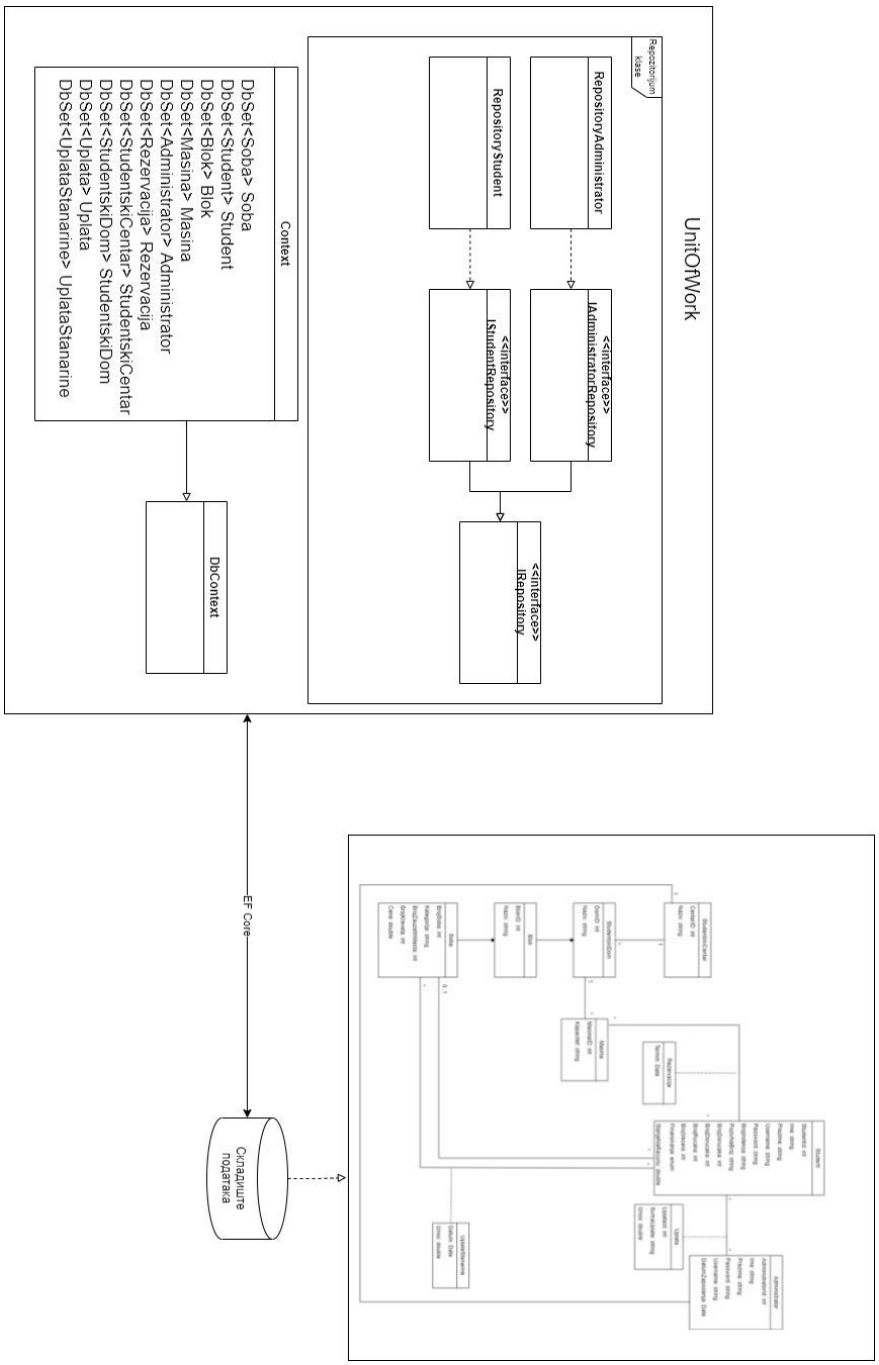
Постуслови: Подаци о уплати су сачувани



Слика 83 Дијаграм секвенци: Уговор- сачувај уплату

6.4.3 Комуникација између пословне логике и складишта података

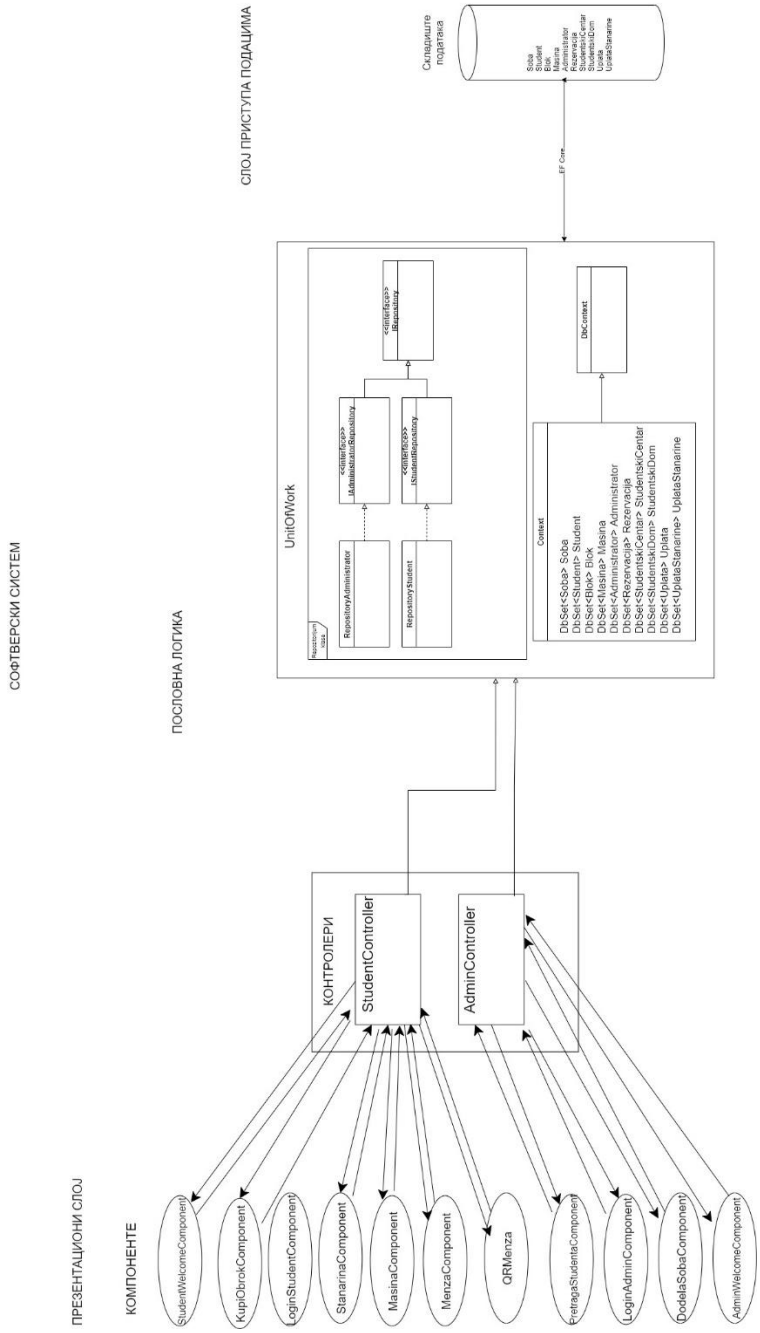
Комуникација између пословне логике и складишта података је апстрахована коришћењем Repository патерна односно увођењем Repository класа. Repository преко DbContext класе остварују комуникацију са складиштем података, у овом случају Sql релационом базом података. Као систем за управљање базом коришћен је Sql Server. Објекат класе DbContext се инстанцира кроз конструктор у оквиру сваке класе. Класе садрже мотедоре којима се приступа подацима и врше основне CRUD операције, уз помоћ Entity Framework Core-а. Свака инстанцирана класа контекста представља сесију са базом.



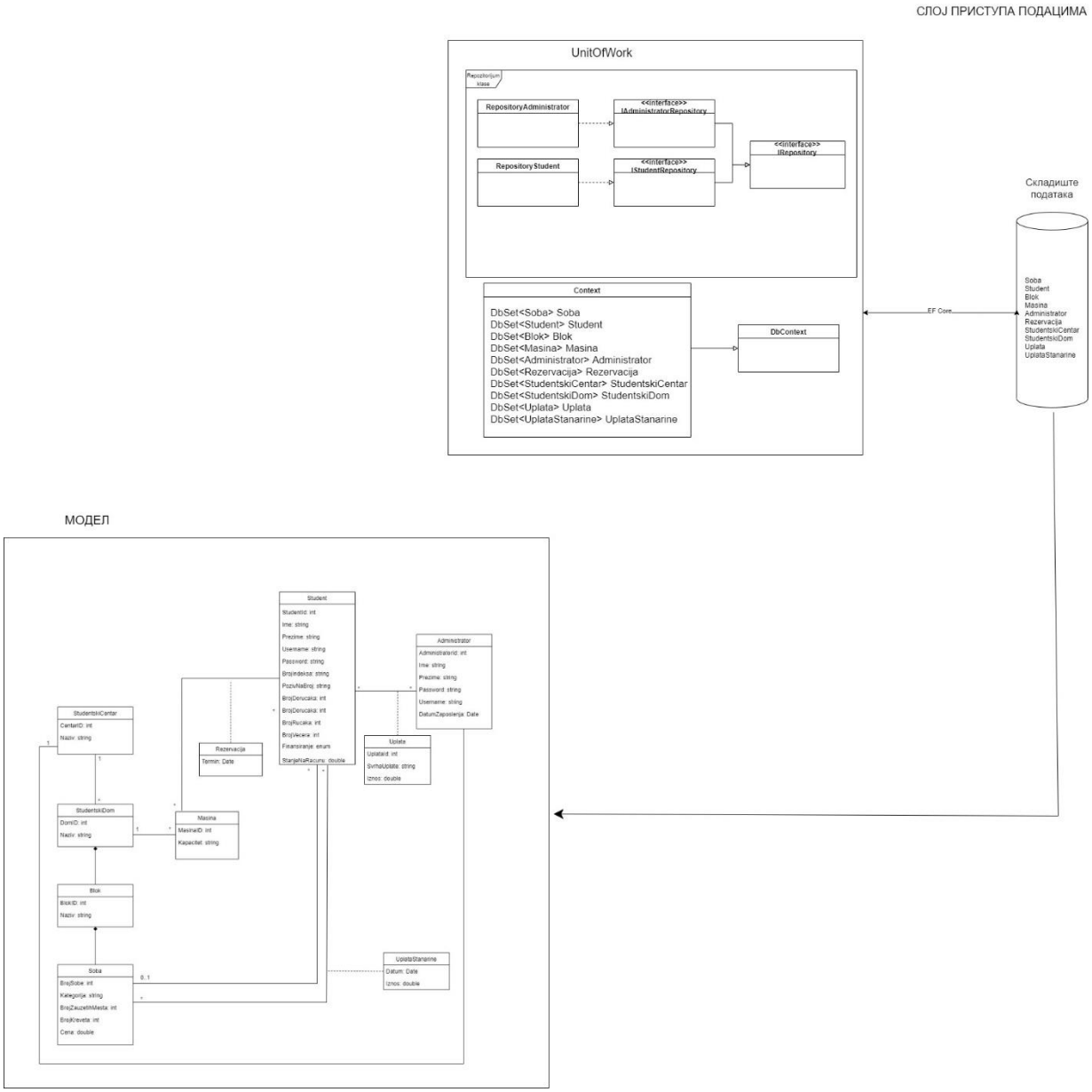
Слика 84Пројектовање апстрактног слоја између пословне логике и складишта података

6.6 Коначан изглед архитектуре софтверског система

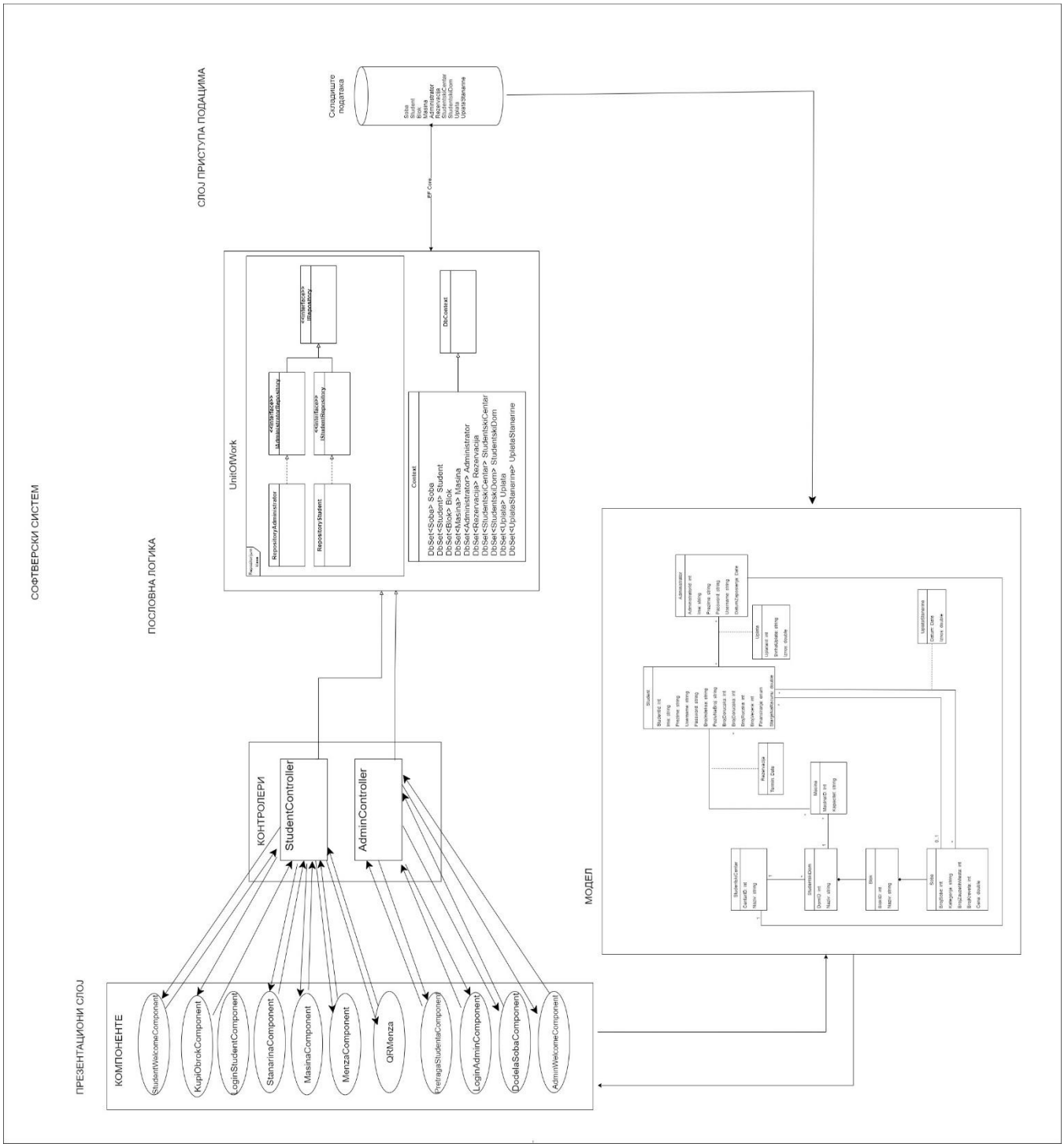
Коначна архитектура софтверског система се добија на основу фазе пројектовања. Презентациони слој чини frontend апликација које је издељена у компоненте. Све захтеве који стижу са корисничког интерфејса примају и обрађују контролери. Контролери имају референцу на UnitOfWork класу, а она даље има референцу на Repository класе. Repository класе служе за комуникацију са базом података. Табеле у бази су креиране на основу концептуалних класа помоћу објектно релационог мапирања



Слика 85 Веза између слоја пословне логике и слоја приступа подацима



Слика 86 Слој приступа подацима и база података



Слика 87Коначна архитектура софтверског система

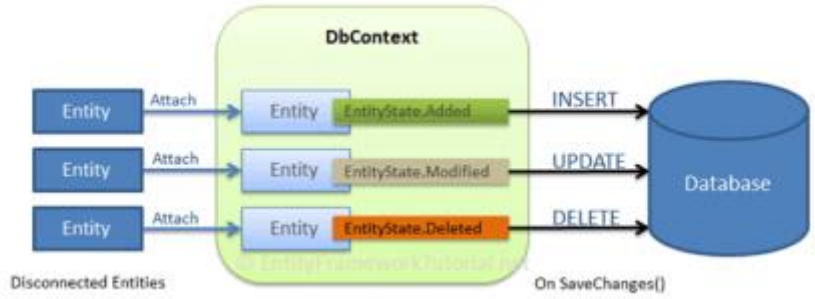
7. Имплементација

Рад представља развој софтверског система за студентски центар уз коришћење ASP.NET Core развојног оквира и EntityFrameworkCore објектно - релационог мапера на серверској страни и комбинације Angular и Ionic оквира на клијентској страни.

Као развојно окружење коришћен је Microsoft Visual Studio 2019. Систем за управљање базом података је SQL сервер.

7.1 Слој приступа подацима

За рад са базом података користи се Entity Framework Core као објектно релациони мапер и користи се code first приступ. За директну комуникацију са складиштем података користи се класа DbContext. Представља сесију са базом преко које се изводе основне CRUD операције. Неопходно је направити модел који мапира ентитете и односе дефинисане у моделу у табеле базе. Да бисмо радили са контекстом потребно је направити класу која га наслеђује а то је у студијском примеру класа Context.



Слика 88 Интеракција између DbContext-а и базе података

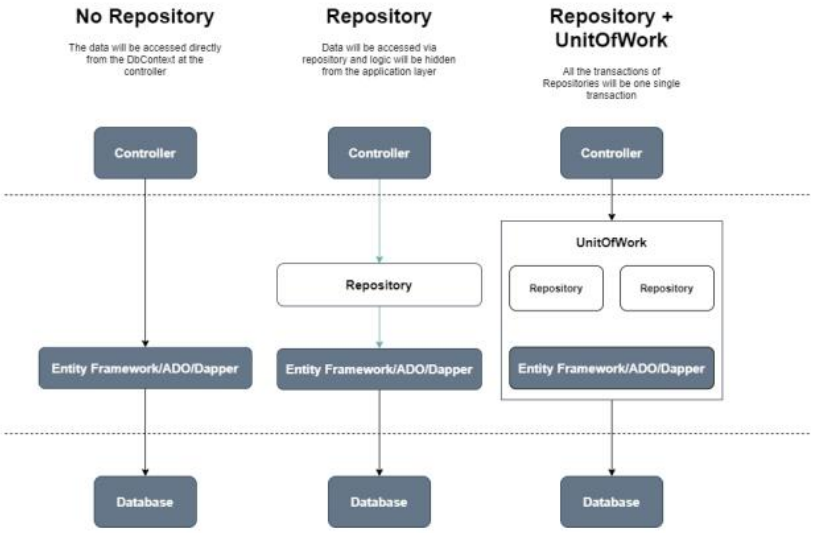
https://www.entityframeworktutorial.net/images/efcore/save-data-in-disconnected-scenario.png

```
public class Context:DbContext
{
public DbSet<Soba> Soba { get; set; }
public DbSet<Student> Student { get; set; }
public DbSet<Blok> Blok { get; set; }
public DbSet<Masina> Masina { get; set; }
public DbSet<Administrator> Administrator { get; set; }
public DbSet<Rezervacija> Rezervacija { get; set; }
public DbSet<StudentskiCentar> StudentskiCentar{ get; set; }
public DbSet<StudentskiDom> StudentskiDom { get; set; }
public DbSet<Uplata> Uplata { get; set; }
public DbSet<UplataStanarine> UplataStanarine { get; set; }
…
}
```

Unit of Work и Repository pattern

Главна намена UnitOfWork и Repository патерна је стварање слоја апстракције између пословне логике и слоја приступа подацима. UnitOfWork као јединица рада представља логичку трансакцију која групише већи број позива ка бази. Када UnitOfWork јединица рада заврши са својим радом, може се позвати метода Save() којом се потврђују промене у бази. На овај начин повећава се ниво апстракције и одваја пословна логика од директног приступа подацима.

UnitOfWork класа садржи референце ка Repository класама. За сваки тип у објектом моделу креирана је мапирајућа класа која имплементира интерфејс који приказује све операције над базом података које се могу извршити над типом. Овакве класе зову се репозиторијуми. Репозиторијум посредује између пословне логике и слоја приступа подацима, енкапсулира скуп објеката сачуваних на неком медијуму и дозвољене операције над њима, на објектно-оријентисани начин, такође одвајајући пословну логику од директног приступа подацима.



Слика 89 Repository и UnitOfWork

На основу структуре софтверских класа пројектоване су табеле (складишта података) релационог система за управљање базом података. У овом раду је коришћен SQL Server.

	Name	Data Type	Allow Nulls	Default
🔑	AdministratorId	int	☐	
	Ime	nvarchar(MAX)	☑	
	Prezime	nvarchar(MAX)	☑	
	Username	nvarchar(MAX)	☑	
	Password	nvarchar(MAX)	☑	
	DatumZaposlenja	datetime2(7)	☐	
	StudentskiCentarId	int	☐	
			☐	

Слика 90 Администратор

	Name	Data Type	Allow Nulls	Default
🔑	BlokId	int	☐	
🔑	StudentskiCentarId	int	☐	
🔑	StudentskiDomId	int	☐	
	Naziv	nvarchar(MAX)	☑	
			☐	

Слика 91Блок

	Name	Data Type	Allow Nulls	Default
🔑	MasinaId	int	☐	
	Kapacitet	nvarchar(MAX)	☑	
	StudentskiCentarId	int	☐	
	StudentskiDomId	int	☐	

Слика 92 Машина

	Name	Data Type	Allow Nulls	Default
🔑	Termin	datetime2(7)	☐	
🔑	StudentId	int	☐	
🔑	MasinaId	int	☐	

Слика 93 Резервација

	Name	Data Type	Allow Nulls	Default
🔑	BrojSobe	int	☐	
🔑	StudentskiCentarId	int	☐	
🔑	StudentskiDomId	int	☐	
🔑	BlokId	int	☐	
	Kategorija	nvarchar(MAX)	☑	
	Cena	float	☐	
	BrojZauzetihMesta	int	☐	
	BrojKreveta	int	☐	

Слика 94 Соба

	Name	Data Type	Allow Nulls
🔑	StudentId	int	☐
	Ime	nvarchar(MAX)	☑
	Prezime	nvarchar(MAX)	☑
	Username	nvarchar(MAX)	☑
	Password	nvarchar(MAX)	☑
	BrojIndeksa	nvarchar(MAX)	☑
	PozivNaBroj	nvarchar(MAX)	☑
	BrojDorucaka	int	☐
	BrojRucaka	int	☐
	BrojVecera	int	☐
	Finansiranje	nvarchar(MAX)	☑
	StanjeNaRacunu	float	☐
	BrojSobe	int	☑
	StudentskiCentarId	int	☑
	StudentskiDomId	int	☑
	BlokId	int	☑
	Fakultet	nvarchar(MAX)	☑

Слика 95 Студент

	Name	Data Type	Allow Nulls	Default
🔑	StudentskiCentarId	int	☐	
	Naziv	nvarchar(MAX)	☑	

Слика 96 Студентски центар

	Name	Data Type	Allow Nulls	Default
🔑	StudentskiDomId	int	☐	
🔑	StudentskiCentarId	int	☐	
	Naziv	nvarchar(MAX)	☑	

Слика 97 Студентски дом

	Name	Data Type	Allow Nulls	Default
🔑	UplataId	int	☐	
🔑	AdministratorId	int	☐	
🔑	StudentId	int	☐	
	SvrhaUplate	nvarchar(MAX)	☑	
	Iznos	float	☐	

Слика 98 Уплата

	Name	Data Type	Allow Nulls	Default
🔑	Datum	datetime2(7)	☐	
🔑	StudentId	int	☐	
🔑	BrojSobe	int	☐	
🔑	StudentskiCentarId	int	☐	
🔑	StudentskiDomId	int	☐	
🔑	BlokId	int	☐	
	Iznos	float	☐	

Слика 99 Уплата станарине

7.2 Имплементација екранских форми

На основу пројектовања екранских форми, коришћењем технологија на клијентској страни, у наставку за сваки слујчај коришћења је дата имплементација екранских форми.

СК1: Случај коришћења - Пријављивање на систем

Назив СК

Пријављивање на систем

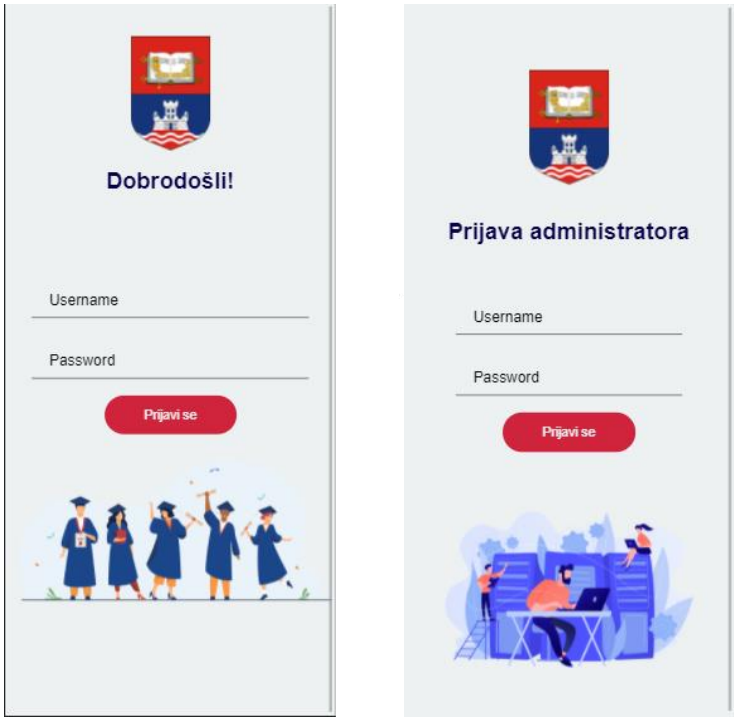
Актори СК

Корисник (администратор или студент)

Учесници СК

Корисник и систем (програм)

Предуслов: Систем је укључен. Систем приказује форму за пријављивање на систем.



Слика 37Пријављивање корисника на систем

Основни сценарио СК

1. Корисник уноси корисничко име и лозинку. (АПУСО)

2. Корисник контролише да ли је коректно унео корисничко име и лозинку. (АНСО)

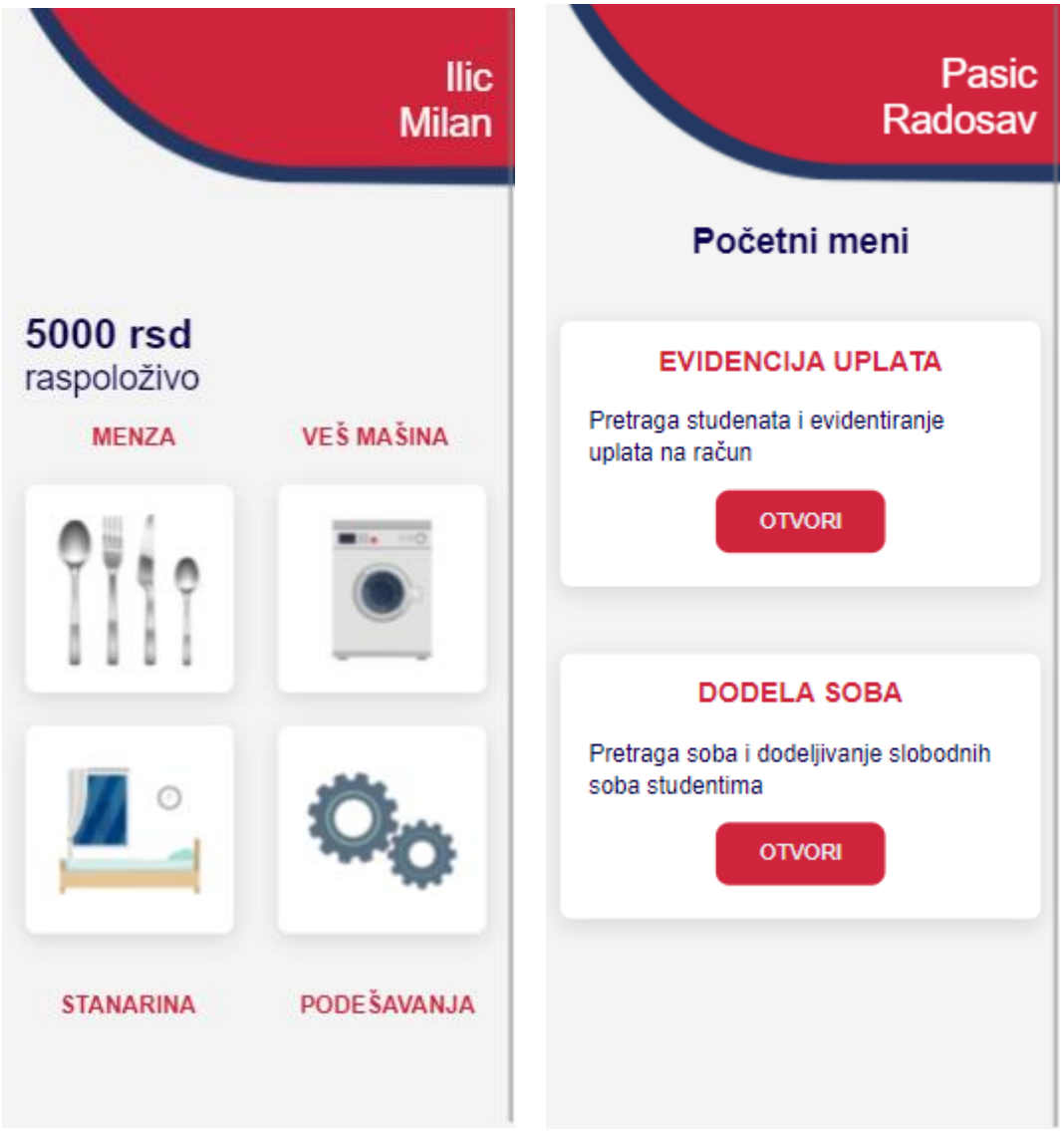
3. Корисник позива систем да се пријави на систем (провери податке). (АПСО)

Опис акције: Корисник кликом на дугме „Prijavi se“ позива системску операцију PrijaviKorisnika(username,password)

4. Систем проверава податке о кориснику. (СО)

5. Систем приказује кориснику почетну страну и поруку:

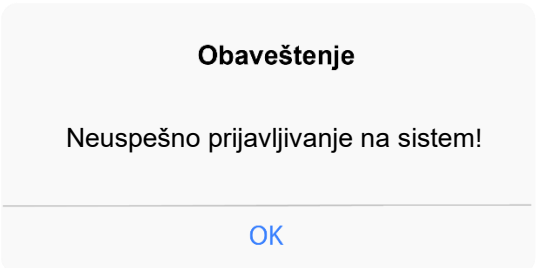
“Успешно пријављивање!“. (ИА)



Слика 100 Почетна страна- студент- администратор

Алтернативна сценарија

5.1. Уколико систем не може да нађе корисника, он приказује кориснику поруку: “ Неуспешно пријављивање на систем!”. (ИА)



Слика 101Неуспешно пријављивање

СК2: Случај коришћења - Претрага студентских налога по критеријуму

Назив СК

Претрага студентских налога по критеријуму

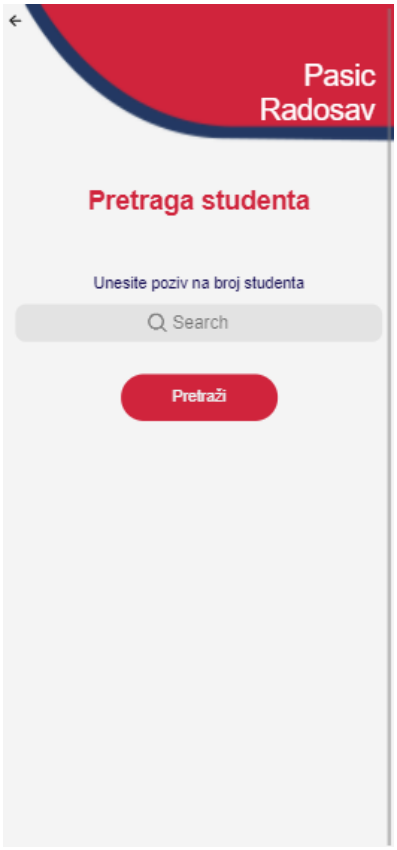
Актори СК

Администратор

Учесници СК

Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је пријављен на систем. Систем приказује форму за рад са студентима. Учитана је листа студената.



Слика 102Форма за рад са студентима

Основни сценарио СК

1. Администратор уноси вредност по којој претражује студента. (АПУСО)

2. Администратор позива систем да нађе студенте по задатој вредности. (АПСО)

Опис акције: Администратор кликом на дугме „Pretraži“ позива системску операцију VratiStudentePoKriterijumu(Kriterijum, List<Student>)

3. Систем тражи студенте по задатој вредности. (СО)

4. Систем приказује администратору нађене студенте. (ИА)

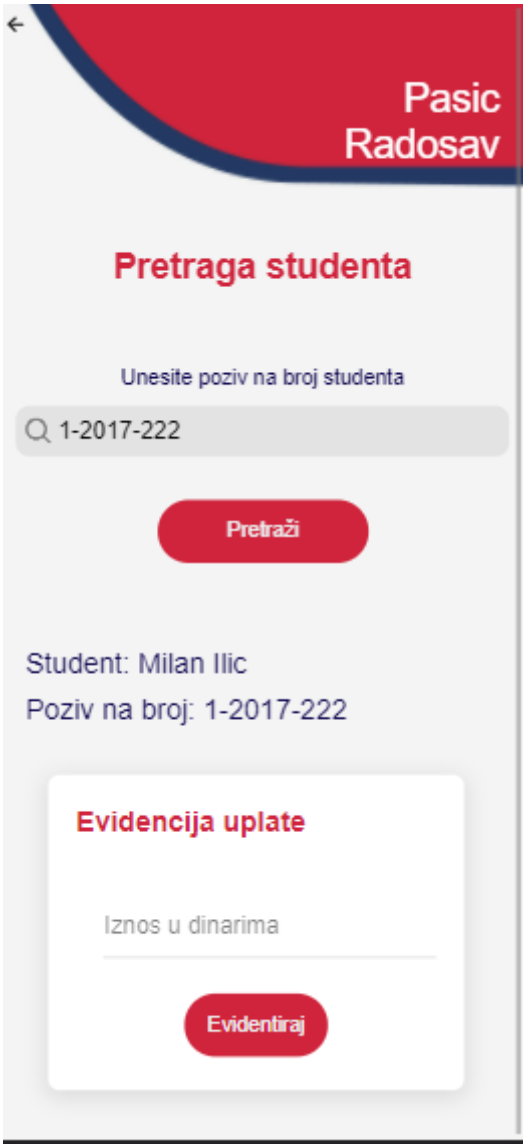
5. Администратор бира студента чије податке жели да види.(АПУСО)

6. Администратор позива систем да прикаже податке о изабраном студенту.(АПСО)

Опис акције: Администратор кликом на картицу студента позива системску операцију PrikaziPodatkeOStudentu(Student)

7. Систем проналази податке о изабраном студенту.(СО)

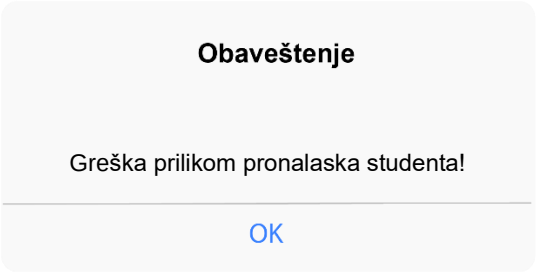
8. Систем приказује администратору податке о изабраном студенту.(ИА)



Слика 103 Подаци о студенту

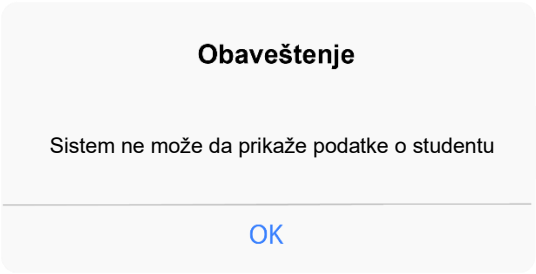
Алтернативна сценарија

4.1 Уколико систем не може да пронађе студента по задатим вредностима приказује се порука: ”Грешка приликом проналаска студента“. (ИА)



Слика 104Неуспешна претрага студента

8.1. Уколико систем не може да прикаже податке о изабраном студенту приказује се порука „Систем не може да прикаже податке о студенту“(ИА)



Слика 105Неуспешна претрага студента

СК3: Случај коришћења - Евидентирање уплате

Назив СК

Евидентирање уплате

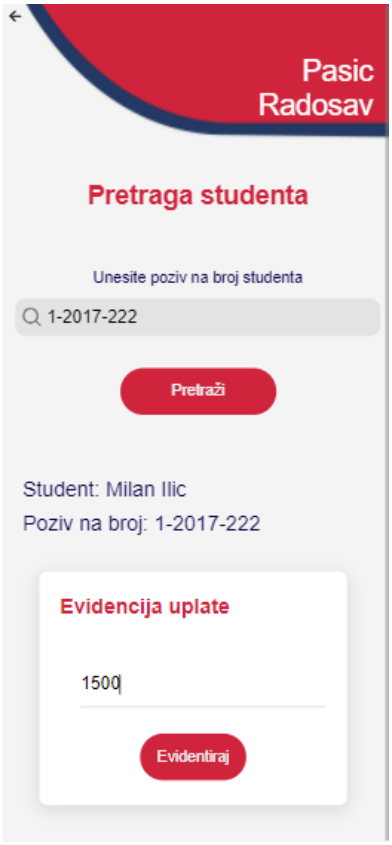
Актори СК

Администратор

Учесници СК

Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је пријављен на систем. Систем приказује форму за рад са студентима. Учитана је листа студената.



Слика 106 Евиденција упката

Основни сценарио СК

1. Администратор уноси вредност по којој претражује студента. (АПУСО)

2. Администратор позива систем да нађе студенте по задатој вредности. (АПСО)

Опис акције: Администратор кликом на дугме „Pretraži“ позива системску операцију VratiStudentePoKriterijumu(Kriterijum, List<Student>)

3. Систем тражи студенте по задатој вредности. (СО)

4. Систем приказује администратору нађене студенте. (ИА)

5. Администратор бира студента чије податке жели да измени.(АПУСО)

6. Администратор позива систем да учита податке о студенту. (АПСО)

Опис акције: Администратор кликом на картицу студента позива системску операцију

PrikaziPodatkeOStudentu(Student)

7. Систем учитава податке о студенту.(СО)

8. Систем приказује администратору задатог студента.(ИА)

9. Администратор мења податке о студенту.(АПУСО)

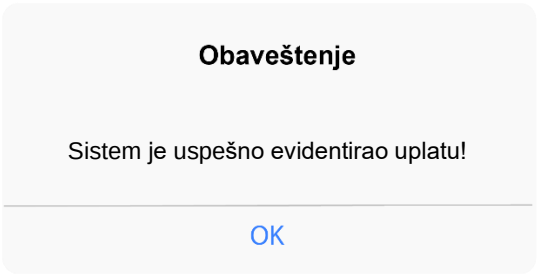
10. Администратор проверава да ли је исправно унео измењене податке.(АНСО)

11. Администратор позива систем да запамти податке о студенту.(АПСО)

Опис акције: Администратор кликом на дугме „Evidentiraj“ позива системску операцију IzmeniStudenta(Student)

12. Систем памти измењене податке о студенту.(СО)

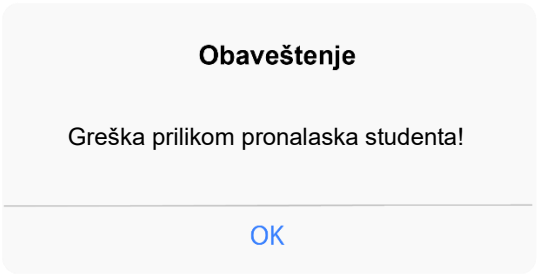
13. Систем приказује администратору поруку:“Систем је успешно евидентирао уплату.“(ИА)



Слика 107 Евиденција уплате успешна

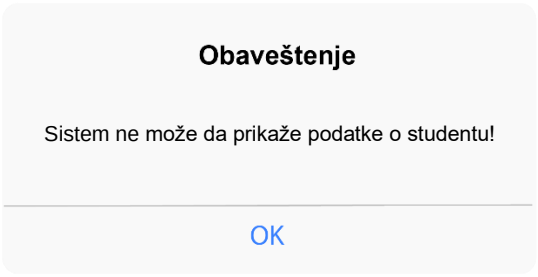
Алтернативна сценарија

4.1. Уколико систем не може да пронађе задатог студента приказује се порука: ”Грешка приликом проналаска студента”. Прекида се извршење сценарија. (ИА)



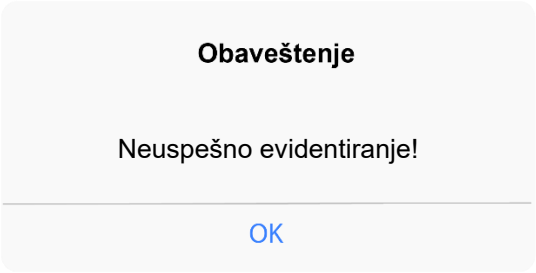
Слика 108Неуспешна претрага студента

8.1. Уколико систем не пронађе информације о студенту, систем приказује администратору поруку:“Систем не може да прикаже податке о студенту“. Прекида се извршење сценарија.(ИА)



Слика 109Неуспешна претрага студента

13.1. Уколико систем не може да сачува измене о студенту, систем приказује следећу поруку:“Неуспешно евидентирање.“(ИА)



Слика 110 Неуспешна уплата

СК4: Случај коришћења - Претрага соба по критеријуму

Назив СК

Претрага соба по критеријуму

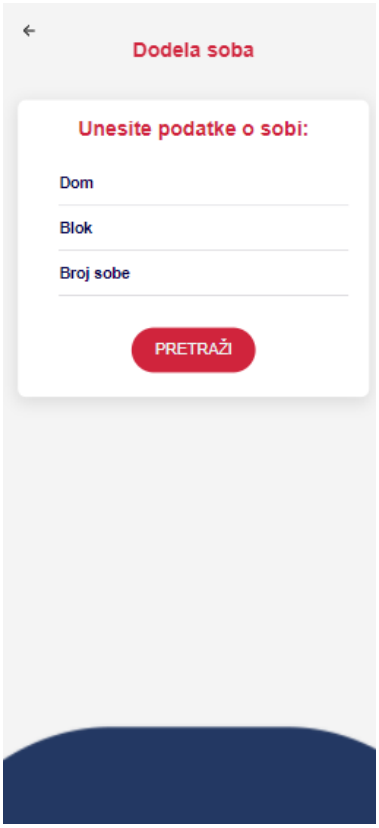
Актори СК

Администратор

Учесници СК

Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је пријављен на систем. Систем приказује форму за рад са собама. Учитана је листа соба.



Слика 111 Форма за претрагу соба

Основни сценарио СК

1. Администратор уноси вредност по којој претражује собу. (АПУСО)
2. Администратор позива систем да нађе собе по задатој вредности. (АПСО)

Опис акције: Администратор кликом на дугме „Pretraži“ позива системску операцију VratiSobePoKriterijumu(Kriterijum,List<Soba>)

3. Систем тражи собе по задатој вредности. (СО)
4. Систем приказује администратору нађене собе. (ИА)
5. Администратор бира собу чије податке жели да види.(АПУСО)
6. Администратор позива систем да прикаже податке о изабраној соби.(АПСО)

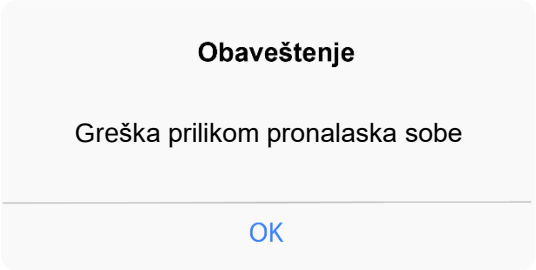
Опис акције: Администратор кликом на картицу собе позива системску операцију PrikaziPodatkeOSobi(Soba)

7. Систем проналази податке о изабраној соби.(СО)
8. Систем приказује администратору податке о изабраној соби.(ИА)

Слика 112 Резултат претраге собе

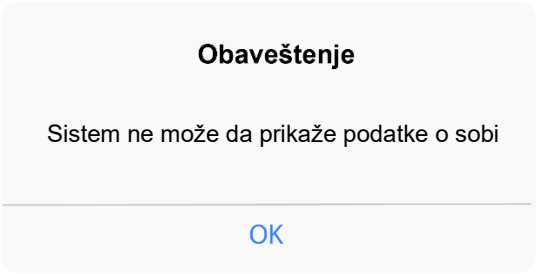
Алтернативна сценарија

4.1 Уколико систем не може да пронађе собу по задатим вредностима приказује се порука: ”Грешка приликом проналаска собе“. (ИА)



Слика 113 Грешка приликом проналаска собе

8.1. Уколико систем не може да прикаже податке о изабраној соби приказује се порука „Систем не може да прикаже податке о соби“(ИА)



Слика 114 Грешка приликом приказивања собе

СК5: Случај коришћења - Додељивање собе студенту

Назив СК

Додељивање собе студенту

Актори СК

Администратор

Учесници СК

Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је пријављен на систем. Систем приказује форму за доделу соба. Учитане су листа студената и листа соба.

Слика 115 Форма за доделу собе

Основни сценарио СК

1. Администратор уноси вредност по којој претражује собу. (АПУСО)
2. Администратор позива систем да нађе собе по задатој вредности. (АПСО)

Опис акције: Администратор кликом на дугме „Pretraži“ позива системску операцију VratiSobePoKriterijumu(Kriterijum,List<Soba>)

3. Систем тражи собе по задатој вредности. (СО)
4. Систем приказује администратору нађене собе. (ИА)
5. Администратор бира собу чије податке жели да види.(АПУСО)
6. Администратор позива систем да прикаже податке о изабраној соби.(АПСО)

Опис акције: Администратор кликом на картицу собе позива системску операцију PrikaziPodatkeOSobi(Soba)

7. Систем проналази податке о изабраној соби.(СО)
8. Систем приказује администратору податке о изабраној соби.(ИА)
9. Администратор уноси вредност по којој претражује студента. (АПУСО)
10. Администратор позива систем да нађе студенте по задатој вредности. (АПСО)

Опис акције: Администратор кликом на дугме „Pronađi“ позива системску операцију

VratiStudentePoKriterijumu(Kriterijum,List<Student>)

11. Систем тражи студенте по задатој вредности. (СО)

12. Систем приказује администратору нађене студенте. (ИА)

Слика 116 Претрага студента

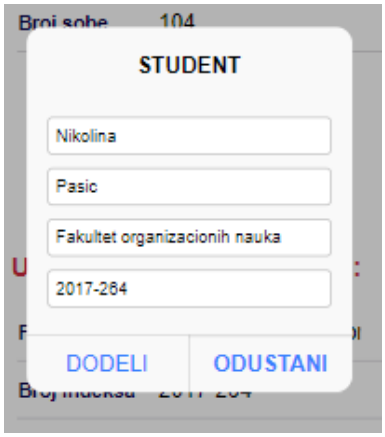
13. Администратор бира студента чије податке жели да измени.(АПУСО)

14. Администратор позива систем да учита податке о студенту. (АПСО)

Опис акције: Администратор кликом на картицустудента позива системску операцију PrikaziStudenta (Student)

15. Систем учитава податке о студенту.(СО)

16. Систем приказује администратору задатог студента.(ИА)



Слика 117 Подаци о студенту

17. Администратор мења податке о студенту.(АПУСО)

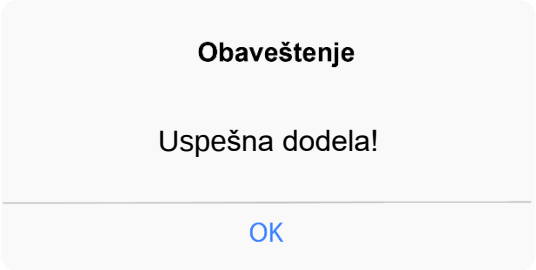
18. Администратор проверава да ли је исправно унео измењене податке.(АНСО)

19. Администратор позива систем да запамти податке о студенту.(АПСО)

Опис акције: Администратор кликом на дугме „Dodeli“ позива системску операцију DodeliSobu(Student,Soba)

20. Систем памти измењене податке о студенту.(СО)

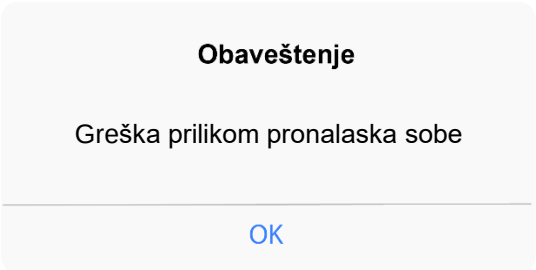
21. Систем приказује администратору поруку:“Систем је успешно извршио доделу.“(ИА)



Слика 118 Успешна додела

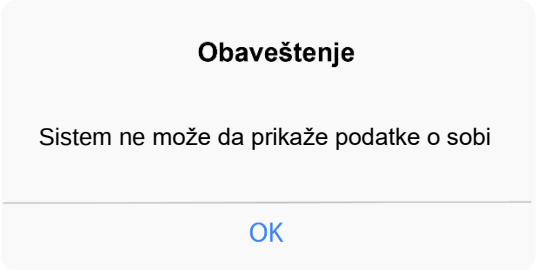
Алтернативна сценарија

4.1. Уколико систем не може да пронађе задату собу приказује се порука: ”Грешка приликом проналаска собе”. Прекида се извршење сценарија. (ИА)



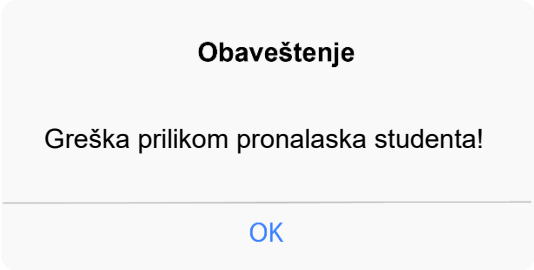
Слика 119 Грешка приликом проналаска собе

8.1. Уколико систем не пронађе информације о соби, систем приказује администратору поруку:“Систем не може да прикаже податке о соби“. Прекида се извршење сценарија.(ИА)



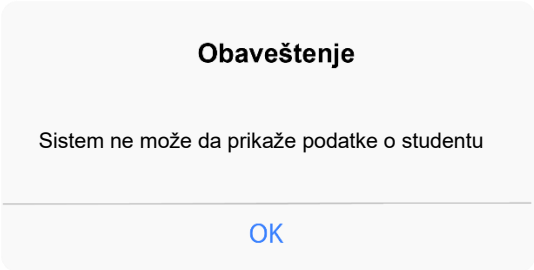
Слика 120 Грешка приликом приказивања собе

12.1. Уколико систем не може да пронађе задатог студента приказује се порука: ”Грешка приликом проналаска студента”. Прекида се извршење сценарија. (ИА)



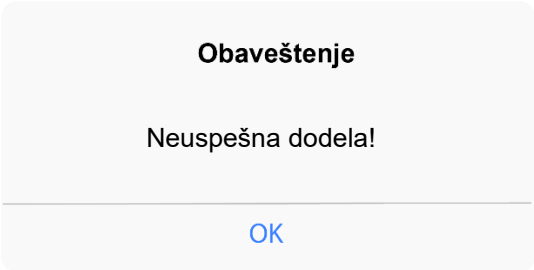
Слика 121 Грешка приликом проналаска студента

16.1. Уколико систем не пронађе информације о студенту, систем приказује администратору поруку:“Систем не може да прикаже податке о студенту“. Прекида се извршење сценарија.(ИА)



Слика 122 Грешка приликом приказивања студента

21.1. Уколико систем не може да сачува измене о студенту, систем приказује следећу поруку:“Неуспешна додела.“(ИА)



Слика 123 Неуспешна додела собе

СК6: Случај коришћења - Куповина оброка

Назив СК:

Куповина оброка

Аутори СК:

Студент

Учесници СК:

Студент и систем

Предуслов: Систем је укључен. Систем приказује страну за куповину оброка. Учитани су подаци о студенту.

Слика 124 Куповина оброка

Основни сценарио СК:

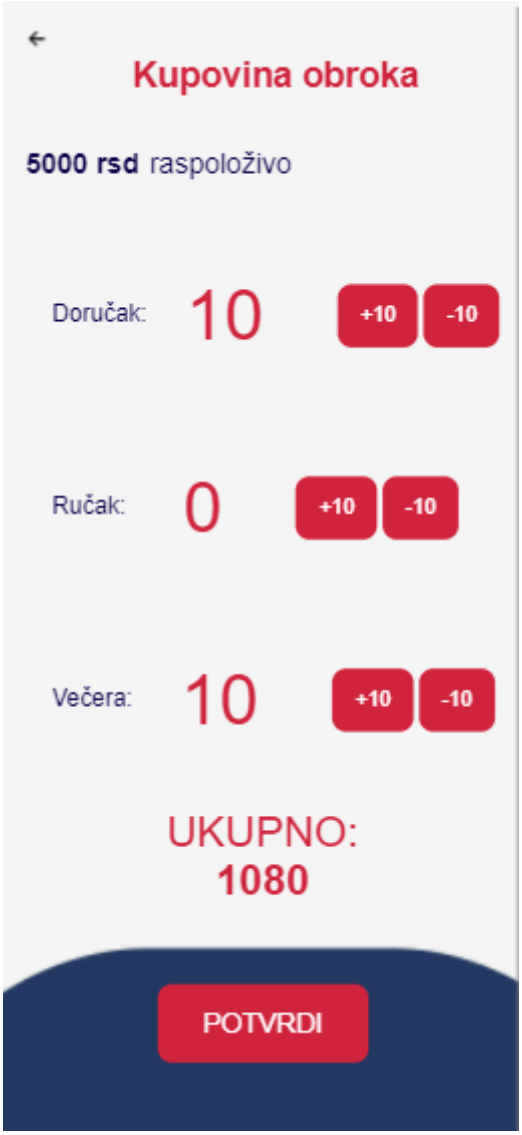
1. Студент уноси количину оброка које жели да купи (АПУСО)
2. Студент контролише да ли је коректно унео количину оброка. (АНСО)

3. Студент позива систем да запамти измењене податке на студентском налогу. (АПСО)

Опис акције: Студент кликом на дугме „Potvrdi“ позива системску операцију IzmeniStudenta(Student)

4. Систем памти измењене податке на студентском налогу (СО)

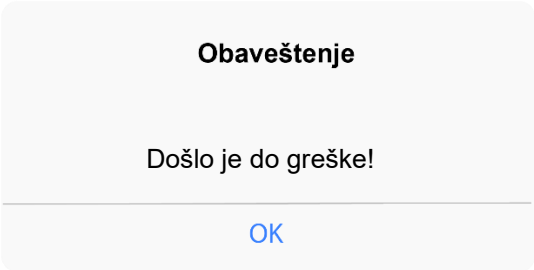
5. Систем приказује кориснику поруку: “Успешно сте купили оброке”. (ИА)



Слика 125 Избор оброка и куповина

Алтернативни сценарио СК:

5.1 Уколико систем не може да измени податке о корисничком налогу, систем приказује кориснику поруку: “Дошло је до грешке ”. (ИА)



Слика 61 Грешка приликом куповине

СК7: Случај коришћења - Резервација машине

Назив СК:

Резервација машине

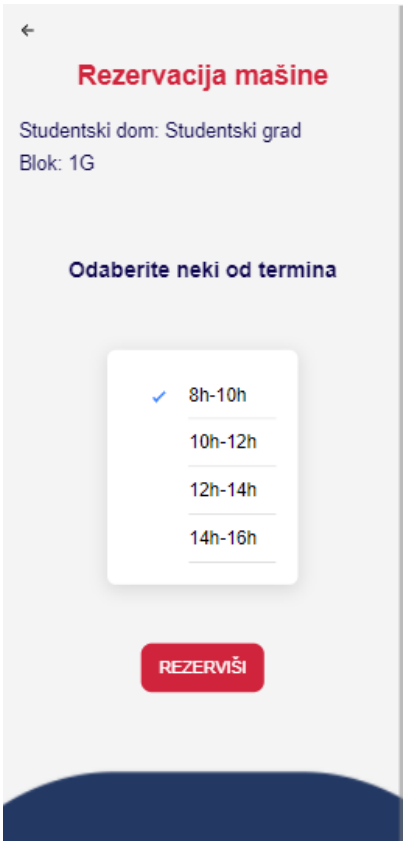
Аутори СК:

Студент

Учесници СК:

Студент и систем

Предуслов: Систем је укључен. Систем приказује страну за резервацију машине. Учитани су подаци о студенту и листа резервација.



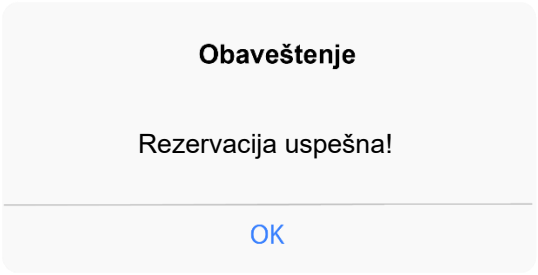
Слика 126 Форма за резервацију машине

Основни сценарио СК:

1. Студент уноси термин у којем жели да резервише машину. (АПУСО)
2. Студент контролише да ли је коректно унео податке о термину. (АНСО)
3. Студент позива систем да изврши резервацију. (АПСО)

Опис акције: Студент кликом на дугме „Rezerviši“ позива системску операцију SacuvajRezervaciju(Rezervacija)

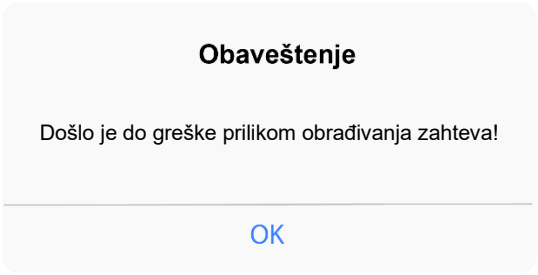
4. Систем памти податке о резервацији. (СО)
5. Систем приказује админу поруку: “Резервација успешна!“. (ИА)



Слика 127 Успешна резервација

Алтернативна сценарија

5.1. Уколико систем не може да запамти податке о резервацији, он приказује студенту поруку: “Дошло је до грешке приликом обрађивања захтева!” .(ИА)



Слика 128 Грешка приликом резервације

СК8: Случај коришћења - Уплата станарине

Назив СК:

Уплата станарине

Аутори СК:

Студент

Учесници СК:

Студент и систем

Предуслов: Систем је укључен. Систем приказује страну за плаћање станарине. Учитани су подаци о студенту и соби.

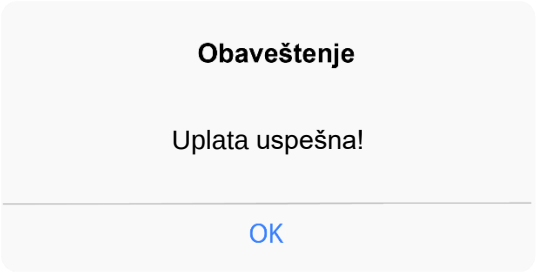
Слика 129 Форма за уплату станарине

Основни сценарио СК:

1. Студент уноси износ цене станарине. (АПУСО)
2. Студент контролише да ли је коректно унео податке о станарини. (АНСО)
3. Студент позива систем да запамти уплату. (АПСО)

Опис акције: Студент кликом на дугме „Plati stanarinu“ позива системску операцију SacuvajUplatu(UplataStanarine)

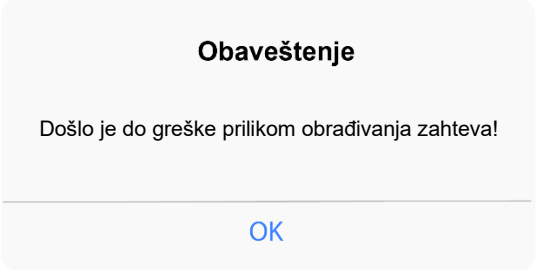
4. Систем памти податке о уплати. (СО)
5. Систем приказује админу поруку: “Уплата успешна!“. (ИА)



Слика 130 Успешна уплата станарине

Алтернативна сценарија:

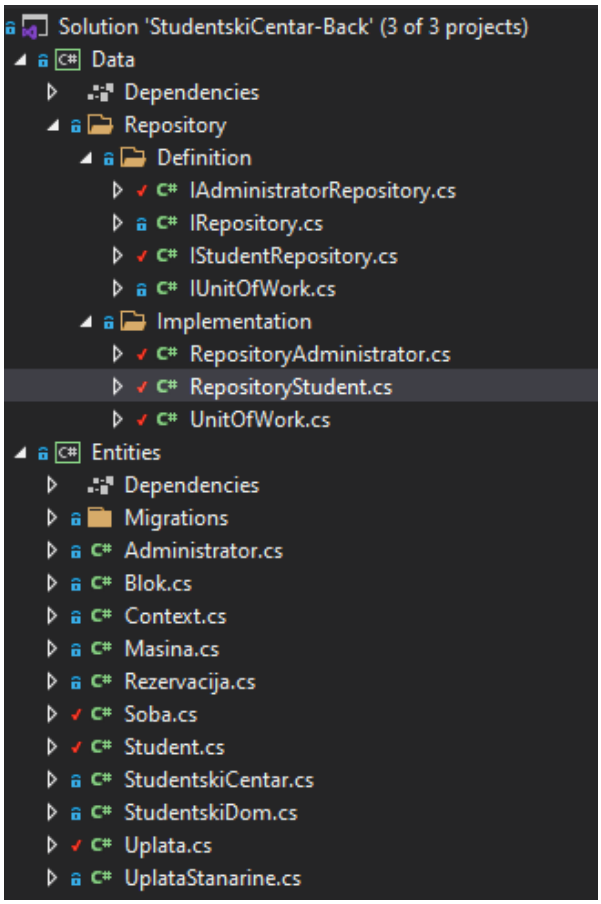
5.1. Уколико систем не може да запамти податке о уплати, он приказује студенту поруку: “Дошло је до грешке приликом обрађивања захтева!” .(ИА)

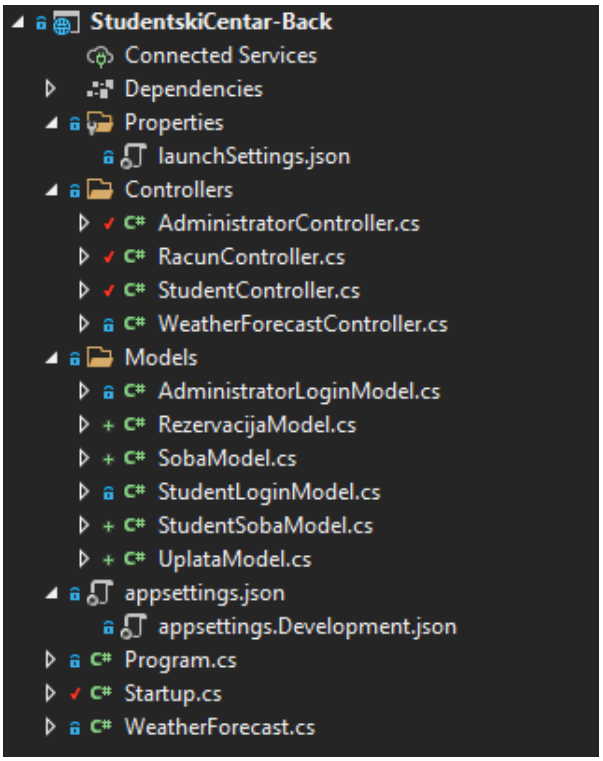


Слика 131 Грешка приликом плаћања станарине

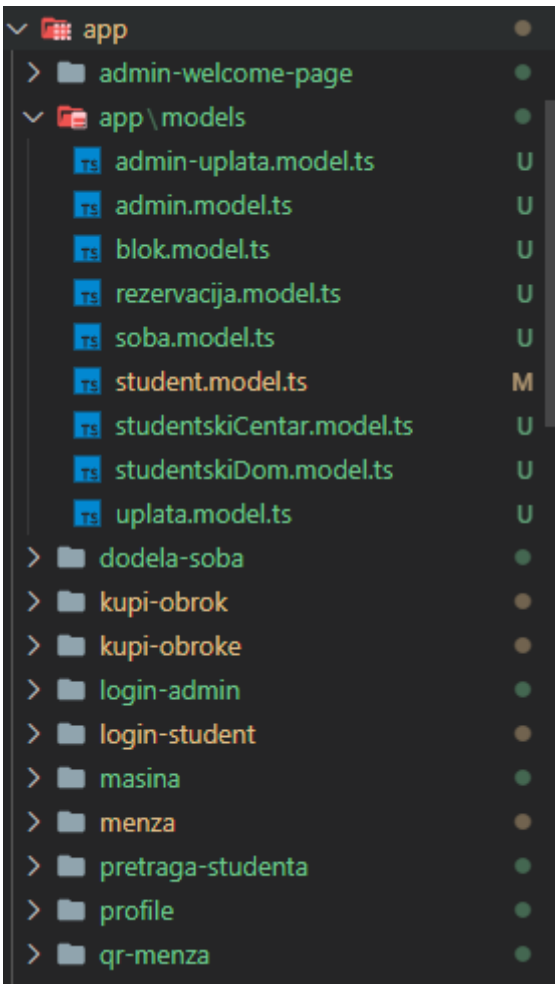
7.3 Структура софтверског решења

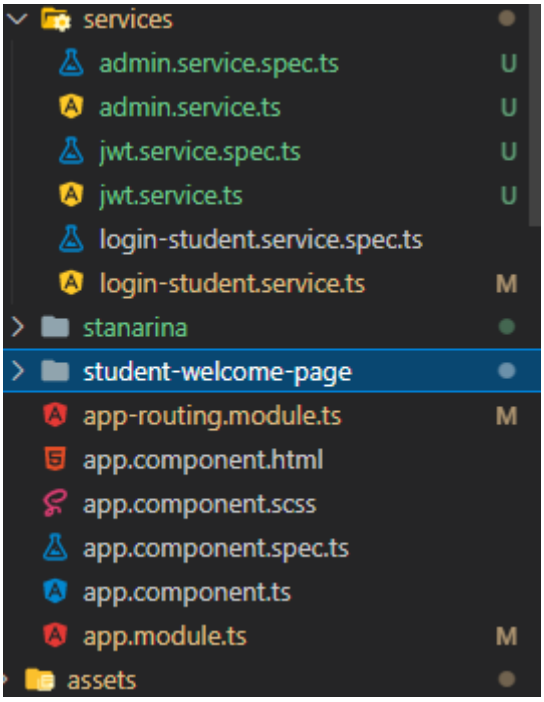
На серверској страни имплементиране су следеће класе:





На клијентској страни су имплементиране следеће класе:





7.4 Имплементација пословне логике

Уговор УГ1: PrijaviKorisnika

Операција: PrijaviKorisnika(username, password)

Веза са СК: СК1

Предуслови: Вредносна и структурна ограничења над објектом Студент морају бити задовољена

Постуслов: Корисник је пријављен на систем

```
public async Task<ActionResult<Student>> Login([FromBody] StudentLoginModel
student)
        {
            Student s = new Student();
            s.Password = student.Password;
            s.Username = student.Username;
            return await _unitOfWork.Student.GetByUsernameAndPassword(s);
        }
```

Уговор УГ2: UcitajListuStudenata

Операција: UcitajListuStudenata(List<Student>)

Веза са СК: СК2, СК3, СК5

Предуслови:

Постуслов:

```
  public async Task<List<Student>> GetStudents()
        {return await _unitOfWork.Student.GetAllAsync();}
```

Уговор УГ3: VratiStudentePoKriterijumu

Операција:VratiStudentePoKriterijumu (Kriterijum,List<Student>)

Веза са СК: СК2, СК3, СК5

Предуслови:

Постуслови: Враћена је листа студената по критеријуму

```
  public async Task<Student> GetStudentById(String ime)
        { return await _unitOfWork.Student.GetStudent(ime);}
```

Уговор УГ4: PrikaziPodatkeOStudentu

Операција: PrikaziPodatkeOStudentu(Student)

Веза са СК: СК2, СК3, СК5

Предуслови:

Постуслови: Подаци о студенту су приказани

```
  public async Task<ActionResult<Student>> ReturnStudent([FromBody]
StudentSobaModel model)
```

```
{
    Student s = new Student();
    s.Fakultet = model.Fakultet;
    s.BrojIndeksa = model.BrojIndeksa;
    s.Soba = new Soba();
    s.Soba = model.Soba;
    s.StudentskiCentar = new StudentskiCentar();
    s.StudentskiCentarId = model.Soba.StudentskiCentarId;
    s.StudentskiDom = new StudentskiDom();
    s.StudentskiDom = model.Soba.StudentskiDom;
    s.Blok = new Blok();
    s.Blok = model.Soba.Blok;
    var signal = await _unitOfWork.Administrator.ReturnStudent(s);
    if (signal != null) return Ok(signal);
    return null;
}
```

Уговор УГ5: SacuvajRezervaciju

Операција: SacuvajRezervaciju (Rezervacija)

Веза са СК: СК7

Предуслови: Вредносна и структурна ограничења над објектом Rezervacija морају бити задовољена

Постуслови: Подаци о резервацији су сачувани

```
public int Reserve([FromBody] RezervacijaModel model) {
        DateTime t = DateTime.Today;
        switch (model.Termin)
        {
            case "8h-10h":
                t = new DateTime(t.Year, t.Month, t.Day, 8, 0, 0);
                break;
            case "10h-12h":
                t = new DateTime(t.Year, t.Month, t.Day, 10, 0, 0);
                break;
            case "12h-14h":
                t = new DateTime(t.Year, t.Month, t.Day, 12, 0, 0);
                break;
            case "14h-16h":
                t = new DateTime(t.Year, t.Month, t.Day, 14, 0, 0);
                break;
```

```
            }
            Rezervacija rezervacija = new Rezervacija();
            rezervacija.Termin = t;
            rezervacija.StudentId = model.Student.StudentId;
            rezervacija.Masina = null;
          int signal= _unitOfWork.Student.Reserve(rezervacija);
            return signal;
        }
```

Уговор УГ6: IzmeniStudenta

Операција: IzmeniStudenta (Student)

Веза са СК: СК3, СК5, СК6

Предуслови: Вредносна и структурна ограничења над објектом Student морају бити задовољена

Постуслови: Подаци о студенту су измењени

```
public async Task<ActionResult> BuyMeals([FromBody]Student student)
        {
            try
            {
                var student1 = await _unitOfWork.Student.BuyMeals(student);
                return Ok(student1);
            }
            catch (Exception)
            {
                return NotFound("Doslo je do greske!");
                throw;
            }
        }
```

Уговор УГ7: UcitajListuSoba

Операција: UcitajListuSoba(List<Soba>)

Веза са СК: СК4, СК5

Предуслови:

Постуслови:

```
  public async Task<List<Soba>> GetRooms()
        {
            return await _unitOfWork.Student.GetAllRoomsAsync();
        }
```

Уговор УГ8: VratiSobePoKriterijumu

Операција: VratiSobePoKriterijumu(Kriterijum,List<Soba>)

Веза са СК: СК4, СК5

Предуслови:

Постуслови: Враћена је листа соба по критеријуму

```
  public async Task<Student> GetRoomsWhere(String kategorija)
        {
            return await _unitOfWork.Student.GetRoomsWhere(kategorija);
        }
```

Уговор УГ9: PrikaziPodatkeOSobi

Операција: PrikaziPodatkeOSobi (Soba)

Веза са СК: СК4, СК5

Предуслови:

Постуслови: Подаци о соби су приказани

```
public async Task<ActionResult<Soba>> FindRoom([FromBody] SobaModel model)
        {
            Soba s = new Soba();
            s.BrojSobe = model.BrojSobe;
            s.Blok = new Blok();
            s.Blok.Naziv = model.Blok;
            s.StudentskiDom = new StudentskiDom();
            s.StudentskiDom.Naziv = model.StudentskiDom;
            var soba = await _unitOfWork.Student.GetRoom(s);
            if (soba != null) return Ok(soba);
            return null;
        }
```

Уговор УГ10: UcitajListuRezervacija

Операција: UcitajListuRezervacija(List<Rezervacija>)

Веза са СК: СК7

Предуслови:

Постуслови:

```
  public async Task<List<Soba>> GetReservations()
        {
            return await _unitOfWork.Student.GetReservationsAsync();
        }
```

Уговор УГ11: VratiStudenta

Операција: VratiStudenta(Student)

Веза са СК: СК6, СК7, СК8

Предуслови:

Постуслови:

```
public async Task<ActionResult<Student>> ReturnStudent([FromBody] StudentSobaModel model)
        {
            Student s = new Student();
            s.Fakultet = model.Fakultet;
            s.BrojIndeksa = model.BrojIndeksa;
            s.Soba = new Soba();
            s.Soba = model.Soba;
            s.StudentskiCentar = new StudentskiCentar();
            s.StudentskiCentarId = model.Soba.StudentskiCentarId;
            s.StudentskiDom = new StudentskiDom();
            s.StudentskiDom = model.Soba.StudentskiDom;
            s.Blok = new Blok();
            s.Blok = model.Soba.Blok;
            var signal = await _unitOfWork.Administrator.ReturnStudent(s);
            if (signal != null) return Ok(signal);
            return null;
        }
```

Уговор УГ12: VratiSobu

Операција: VratiSobu(Soba)

Веза са СК: СК8

Предуслови:

Постуслови:

```
public async Task<ActionResult<Soba>> FindRoom([FromBody] SobaModel model)
         {
             Soba s = new Soba();
             s.BrojSobe = model.BrojSobe;
             s.Blok = new Blok();
             s.Blok.Naziv = model.Blok;
             s.StudentskiDom = new StudentskiDom();
             s.StudentskiDom.Naziv = model.StudentskiDom;
             var soba = await _unitOfWork.Administrator.GetRoom(s);
             if (soba != null) return Ok(soba);
             return null;
```

```
        }
```

Уговор УГ13: SacuvajUplatu

Операција: Веза са СК: СК8

Предуслови: Вредносна и структурна ограничења над објектом UplataStanarine морају бити задовољена

Постуслови: Подаци о уплати су сачувани

```
public  ActionResult PayAccommodation([FromBody] Student student)
        {
            try
            {
                _unitOfWork.Student.PayAccommodation(student);
                return Ok("Success");
            }
            catch (Exception)
            {
                return NotFound("Doslo je do greske!");
                throw;
            }
        }
```

8. Фаза тестирања

Покретањем апликације и уносом неисправних података, вршене су провере исправност валидације. Ако је корисник погрешно унео податке, систем ће му приказати поруку о погрешном уносу. Уношени су и правилни подаци, како би се тестирали сви случајеви коришћења. Уколико је дошло до одређеног недостатка приликом тестирања, они су уклоњени и исправљени.

Након одређеног броја тестова, закључак је да апликација правилно функционише и испуњава све потребне захтеве.

9. Закључак

У раду *Развој софтверског система за студентски центар применом ASP.NET Core и Ionic оквира* описана је архитектура комплексних апликација кроз комплетан поступак за креирање софтверских система.

У теоријском делу рада дат је приказ основних концепата свих технологија коришћених како на серверској тако и на клијентској страни. Практични део представља студијски пример развоја софтверског система који је реализован имплементацијом поменутих технологија.

Инспирација за креирање оваквог софтверског система јавила се због застарелог начина функционисања система за праћење студентских налога у студентским центрима у Србији. Вођена личним негативним искуством при коришћењу истог, као и трендовима у свету мобилних апликација имплементирала сам апликацију која би омогућила првенствено студентима лакше коришћење услуга студентских центара.

Апликација наравно има много простора за унапређење. Било би неопходно консултовати се са администраторима тренутног софтверског система који постоји у студентским центрима и видети каквим подацима у бази они располажу и како би се они уклопили у софтверски систем који је развијен у овом раду. Затим, евидентирање студентских уплата које је имплементирано преко функције администратора могло би значајно да се унапреди склапањем партнерстава са сервисом који би омогућио безбедно плаћање путем интернета.

10. Литература

Benjamin Perkins, J. V. (2018). *ASP.NET and ASP.NET Core.*

Brind, M. (2016). *https://www.learnentityframeworkcore.com/dbcontext*. Retrieved from https://www.learnentityframeworkcore.com.

Cheng, F. (2018). *Build Mobile Apps with Ionic 4 and Firebase.*

https://restfulapi.net/. (2017). Retrieved from https://restfulapi.net/rest-architectural-constraints/.

Leonard Richardson, S. R. (2007). *RESTful Web Services.*

Microsoft. (2021). Retrieved from Introduction to .NET: https://docs.microsoft.com/en-us/dotnet/core/introduction

Price, M. J. (2019). *C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development Fourth Edition.* Packt.

Smith, J. (2018). *Entity Framework Core in Action.*

Wilken, J. (2018). *Ionic in Action HYBRID MOBILE APPS WITH IONIC AND ANGULARJS.*

Влајић, С. (2019). *Пројектовање софтвера - скрипта.* Београд.