УНИВЕРЗИТЕТ У БЕОГРАДУ

ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

ЗАВРШНИ РАД

Тема: Развој софтверског система за део пословања продавнице применом ASP.NET и React оквира

Ментор:
проф. др Саша Лазаревић

Студент:
Милица Раковић 2016/0054

Београд, 2020. године

УНИВЕРЗИТЕТ У БЕОГРАДУ

ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

ЗАВРШНИ РАД

Тема: Развој софтверског система за део пословања продавнице применом ASP.NET и React оквира

Ментор:
проф. др Саша Лазаревић

Студент:
Милица Раковић 2016/0054

Београд, 2020. године

Развој софтверског система за део пословања продавнице применом ASP.NET и React оквира

Садржај

Списак слика:

Списак табела:

Увод

У последњих неколико година може се уочити нагли пораст потребе за веб апликацијама услед неопходности брзом приступу информација на готово свим уређајима. Предност веб апликација, у односу на десктоп и мобилне апликације, је та што је за њихово извршавање потребна само интернет конекција и претраживач. Управо та чињеница наводи програмере широм света да свој фокус пребаце на развијање управо тих технологија.

Када говоримо о корисничком интерфејсу, његовом изгледу и функционалностима, постоји велики број библиотека које се користе како би са једне стране, побољшале изглед веб пликација, а садруге, програмеру омогућиле олакшан рад. *React* је управо једна таква библиотека, која се веома често користи у комбинацији са *Typescript* програмским језиком, о чему ће више речи бити нешто касније.

У овом раду, поред већ наведених технологија за израду корисничког интерфејса, коришћен је и *ASP .NET Core* оквир за изградњу веб *API*-ја како би се подаци из базе пренели на модел тј. веб страницу.

Практичан део завршног рада представља веб апликацију која омогућава поручивање производа преко интернета. Основни циљ ове апликације јесте да корисницима омогући виртуалну куповину како би искључили време неопходно за физички одлазак до продајног места.

Рад је организован у џџ поглавља. У првом поглављу дат је детаљан преглед коришћених технологија приликом израде веб апликације. Најпре је дат опис једностраничних апликација и *REST*-а, а потом и конкретних технологија коришћених на клијентској, односно серверској страни. У другом поглављу приказан је вербални модел са случајевима коришћења. Затим у трећем поглављу је реч о фази анализе која описује структуру и понашање софтверског система. Четврто поглавље говори о пројектовању конкретног система, а пето о имплементацији истог. Након тога следе фаза тестирања у шестом и закључак у седмом поглављу.

1 Преглед коришћених технолпогија

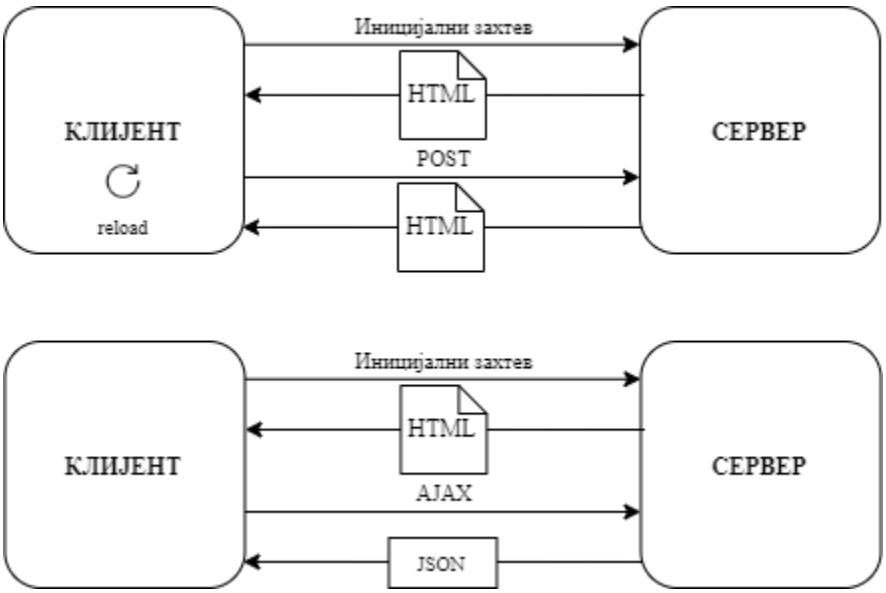
Технологије коришћене у изради завршног рада на тему „Развој софтверског система за део пословања продавнице применом ASP.NET и React оквира“ могу се поделити у две групе:

- **клијентске** или **фронтенд**
- **серверске** или **бекенд**.

1.1 Једностраничне апликације

Једностраничне апликације су апликације које не зајтевају поновно учитавање странице током коришћења и које се извршавају унутра претраживача. Типични примери ових апликација су: Фејсбук, Гугл мапе, Гмејл, Твитер, па чак и Гит Хаб. Највећа предност оваквих апликација је кориснички доживљај (UX) тј. чињеница да корисник не мора да чека учитавање страница већ остаје на истој страни коју покреће JavaScript програмски језик. У складу са тим, брзина се истиче као битна карактеристика оваквих апликација јер је већина ресурса неопходних за функционисање оваквих апликација учитана приликом иницијалног покретања апликације. Једина ствар која се мења јесу подаци који се преносе до и са сервера.

Једностраничне апликације су одличне и када постоји тим који ради заједно. Оне омогућавају бекенд девелоперима да се фокусирају на API, док фронтенд девелопери могу неометано да раде креирајући кориснички интерфејс.



Слика 1 Поређење традиционалног и једностраничног циклуса странице

Процес кеширања података је такође ефикасан – апликација шаље само један захтев, складишти све примљене податке и касније их користи. Ово је посебно значајно када корисник има лошију интернет конекцију.

Међутим, постоје и неки недостаци коришћења једностраничних апликација као што је цурење меморије. С обзиром да апликација може да се извршава дужи временски период, мора се водити рачуна да оваква апликација не заузме више меморије од онога што јој је потребно. Такође, једностраничне апликације постављају додатна оптерећења на веб претраживаче па је могуће да ће, ако корисник има слабији уређај, имати и лошије искуство у поглефу брзине извршавања[1].

У погледу сигурности, проблем настаје приликом иницијалног учитавања. Како се ове апликације у потпуности учитавају податке при покретању, може се догодити да корисник може да приступи подацима којима не би смео.

Тренутна три оквира која се користе за израду фронтенд једностраничних апликација су *Angular, Vue.js* и *React*.

1.2 REST (*Representational state transfer*)

REST је први пут представљен 2000. године од стране Роја Филдинга и представља архитектуру за дистрибуиране системе. *REST API (REST Application Programming Interface)* је тип веб сервиса који омогућава кориснички управљаним или аутоматизованим клијентима да приступе ресурсима који обликују модел (подацима и функцијама).

Основна архитектурална ограничења на којима се заснива[2]:

- униформни интерфејс: фундаменталан за дизај *REST*-а и његова ограничења дефинишу интерфејсе између клијента и сервера
- клијент-сервер: одвајањем корисничког интерфејса од складишта побољшава се портабилност корисничког интерфејса преко више платформи и долази до боље скалабилости услед поједностављења серверских компоненти јер сервер не мора да воду рачуна о корисничком интерфејсу и стањима клијента. Такође, због своје архитектуре, клијент и сервер се могу развијати независно један од другог али и бити модификовани
- без стања: сваки захтев од клијента ка серверу мора да садржи све информације неопходне за разумевање захтева
- могућност кеширања: ограничења везана за кеширање захтевају да се подаци у оквиру одговора на захтев имплицитно или експлицитно означе као *cacheable* или *non-cacheable*
- слојевит систем или микросервиси: овакав стил омогућава архитектури да буде састављена од хијерархијских слојева који не могу да виде податке који су изнад њих.

[1] https://huspi.com/blog-open/definitive-guide-to-spa-why-do-we-need-single-page-applications
[2] https://restfulapi.net/

- код на захтев (опционо): у већини случајева шаљу се статичке репрезентације података у форми *XML*-а или *JSON*-а. Међутим, то не мора да буде случај. Поред тога, може се вражати и извршни код како би се подржао одређени део апликације.

Најзначајнији блокови за креирање *API*-ја[3]:

- *Resources (URIs)* – ресурси су објекти описани од стране *API*-ја који имају јединствени идентификатор
- *HTTP methods* – ове методе се користе како би се изменило стање објекта
- *HTTP headers* – користи се за парсирање обавезних аргумената попут оних за аутентикацију, прихватање типова и слично
- *Query parameters* – параметри упита се користе за опционе аргументе или филтрирање
- *Return codes* – треба да у потпуности пресликава значење и семантику основних *HTTP* спецификација, обезбеђујући да је код довољно информативан и допуњен додатним информацијама.

Веома често долази до мешања *REST* и *Web API*-ја. *Web API* је фрејмворк отвореног кода који се користи за писање *HTTP API*-ја. Представља концепт али не и технологију који се може изградити коришћењем различитих технологија попут *.NET*-а или *Java*-а. *Web API*-ји могу бити *REST*, а и не морају. За разлику од њега, *REST API* је програмски интерфејс који је подржан са стране архитектуралног стила *REST*-а.[4]

[3] https://blog.restcase.com/restful-api-basic-guidelines/
[4] https://rapidapi.com/blog/rest-api-vs-web-api/

1.3 Клијентска страна

Приликом израде апликације на клијентској страни коришћена је библотека *React* као и програмски језик *Typescript*. Поред тога коришћене су и разне друге библиотеке које олакшавају рад са компонентама (нпр. *Material UI*), генеришу *PDF* фајлове (*react-to-pdf*) итд.

1.3.1 *JavaScript*[5]

JavaScript је скриптни језик који омогућава имплементацију комплексних ствари на веб страницама – сваки пут када веб страница уради нешто више од самог приказивања статичких информација за то је задужен овај скриптни језик. Укратко речено, *JavaScript* је динамичан и слабо типизиран, интерпретиран програмски језик. Он може да функционише и као процедуралан и као објектно оријентисан језик. Објекти се креирају програмски везивањем метода и пропертија на празне објекте током run time-а. Када се објекат једном креира, може бити коришћен као нацрт или прототип за креирање сличних објеката.

Слика 2 *JavaScript* лого

JavaScript је креиран 1995. године од стране Брендана Ајка и први пут се појавио у тада популарном веб претраживачу Нетскејп. На самом почетку језик је назван *LiveScript* да би само неколико месеци касније био преименован и добио данашњи назив. Иако је иницијално представљен као језик намењен искључиво за веб претраживаче, веома брзо добија своју имплементацију и на серверској страни.

Што се саме синтаксе тиче, она је базирана на *Java* и *C* програмским језицима – многе структуре које се тамо користе су присутне и у JavaScript-. JavaScript подржава објектно-оријентисано програмирање са објектним прототиповима уместо класама. Такође, подржава и функционално програмирање зато што функције могу бити смештене у оквиру промењиве и прослеђиване као било који други објекат.

Типови података које *JavaScript* подржава су:

- Number – све вреднсоти овог типа података су представљене као реални бројеви у 64-битном формату
- String – секвенце 16-битних вредности Уникодних карактера који се морају наћи под једноструким или двоструким наводницима.
- Boolean
    - Object –
    - Function – објекти који треба да се изврше и који представљају суштину разумевања начина на који JavaScript функционише

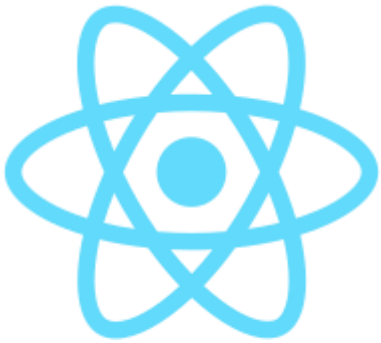
[5] https://www.javascript.com/

    - Array – уређена колекција нумерисаних вредности
    - Date – објекти за рад са датумима и временом
    - RegExp – објекти за рад са регуларним изразима
    - Error – објекти који представљају грешке
- Symbol
- undefined
- null.

JavaScript прави разлику између *undefined* и *null* типа података. *Null* је вредност која указује на намерно непостојање вредности, док представља неиницијализовану променљиву тј. променљиву којој конкретна вредност још увек није додељена.

1.3.2 *React* библиотека[6]

React је декларативна, флексибилна и ефикасна *JavaScript* библиотека отвореног кода за израду брзог и интерактивног корисничког интерфејса за веб и мобилне апликације креирана од стране Фејсбука 2013. године. Тренутно представља једну он најпопуларнијих *JavaScript* фронтенд библиотека која има снажне темеље и велику заједницу иза себе.



Слика 3 *React* лого

Ова библиотека се користи за израду модерних корисничких интерфејса веб апликација, односно једностраничних или мобилних апликација. Комплексиније *React* апликације захтевају коришћење додатних библиотека за управљање стањима, рутирање и интеракцију са API-јима.

Оно што је значајно истаћи везано за *React* библиотеку је да је она компонентно оријентисана. То значи да једном направљен део корисничког интерфејса може бити искоришћен као компонента за други, па тако за креирање једне веће апликације може бити искоришћено више мањих компонената. Главне карактеристике тих компонената је да су независне и међусобно изоловане. Предност коришћења компоненти је та што је могуће изменити било коју компоненту у било ком тренутку, а да при том то не утиче на остатак апликације.

Када изменимо неки податак, *React* аутоматски мења искључиво компоненту која зависи од тог податка. То омогућава претраживачу да не мора поново да учита читаву апликацију већ само делове које је неопходно, односно оне у којима је настала промена.

Компоненте су организоване хијерархијски, у виду стабла, и свака компонента је описана функцијом која прима неко стање, а враћа елемент који описујешта ће се исцртати на екрану. Веће компоненте су састављене од мањих, мање од још мањих и тако долазимо до

[6] https://reactjs.org/

компоненти на листовима стабла које се директно мапирају у *HTML* елементе. Два основна начина за декларисање компоненти у *React*-у су преко функционалних компоненти (*Functionl Component*) и компоненти заснованих на класи (*Class Component*).

- Функционалне компоненте

Основна карактеристика функционалних компонената је да оне искључиво прихватају податке и приказују их у одређеној форми, односно оне су компоненте без стања (*stateless*). Представљају основне *JavaScript* функције које су обично *arrow functions*, мада могу бити креиране и уз помоћ function кључне речи. Прихватају пропертије као аргументе функције и враћају *HTML (JSX).* За функционалне компоненте не постоји render() метода. Још једна предност ових компоненти је коришћење тзв. *React Hooks* као што су рецимо useEffect и useState.

- Компоненте засноване на класи

Ако за функционалне компоненте кажемо да су без стања (*stateless*), нда за компоненте засноване на класи можемо рећи да су „паметне“ (*stateful*) јер оне имплементирају логику и стање. Имају комплекснију логику корисничког интерфејса и морају имати *render*() методу. Свакој компоненти се послеђује props и приступа му се преко this.props.

Оно што је важно напоменути је да су функционалне компоненте све више у употреби због лакоће руковања јер су, као што је већ речено, чисте JavaScript функције без стања. Такође, оне имају боље перформансе и захтевају знатно мање кода због чега су читљивије од оних заснованих на класи.

Два битна појма која долазе са *React*-ом су *JSX* и *Virtual DOM*.

1.3.2.1 *Virtual DOM*

Сваки претраживач приликом учитавања веб странице, прави нешто што се зове ДОМ (*Document Object Model*). ДОМ је стабло *HTML* елемента који описује целу страницу. Након креирања модела, на основу њега се врши исцртавање странице.

У *React*-у за сваки ДОМ објекат постоји одговарајући *virtual DOM* објекат који је репрезентација ДОМ објекта. Виртуалан ДОМ објекат има исте пропертије као и ДОМ објекат али му недостаје моћ да директно мења оно што се налази на екрану. Манипулација ДОМ објектима је спора, за разлику од виртуалних ДОМ објеката код којих се све одвија веома брзо јер они не мењају оно што се налази на екрану.

Када се рендерује *JSX* елемент, сваки виртуелни ДОМ бива измењен. Када је измењен, React упоређује виртуелни ДОМ са пређашњим стањем које је сачувано непосредно пре измене. Упоређивањем та два стања, React утврђује тачно који објекат виртуелног ДОМ-а је измењен и поново приказује искључиво те објекте у стварном ДОМ-у. Овај процес се назива „*diffing*“.

Виртуални ДОМ је концепт у ком је идеална тј. „виртуална“ репрезентација корисничког интерфејса смештена и синхронизована са „реалним“ ДОМ-ом преко библиотеке као што

је *ReactDOM*. Овај приступ омогућава декларативан API *React*-а: када кажете *React*-у у ком стању желите да буде ваш кориснички интерфејс, он обезбеђује да ДОМ има то стање.

1.3.2.2 *JSX*

JSX (*JavaScript Syntax Extension*) је проширење *JavaScript* синтаксе који омогућава начин да се структурне компоненте рендерују коришћењем синтаксе која је блиска многим дивелоперима. Коришћењем *JSX*-а, могу се написати *HTML* структуре у истом фајлу који садржи и *JavaScript* код. Ово чини код лакшим за разумевање и дибаговање јер избегава коришћење комплексне *JavaScript* ДОМ структуре.

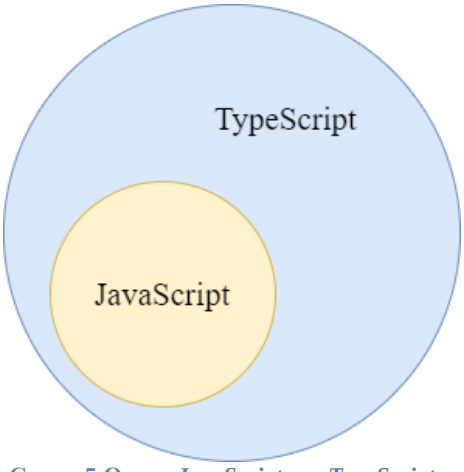
1.3.3 *TypeScript*

Слика 4 TypeScript лого

TypeScript је бесплатан програмски језик, отвореног кода развијен од стране компаније Мајкрософт. Представља заправо проширење *JavaScript* језика, додавајући му типове и објектну оријентисаност. Може се рећи да је он суперсет, односно синтакса *JavaScript*-а у потпуности може бити коришћена у оквиру *TypeScript*-а.

С обзиром да се компајлира у *JavaScript*, он може бити коришћен и за израду фронтенд и бекенд делова апликација.

Програмски језици могу бити статички или динамички типизирани. Код језика са статичким типизирањем, тип промењљиве мора бити познат у време компајлирања. То значи да заједно уз декалрацију промењљиве треба да стоји и њен тип. Што се језика са динамичким типизирањем тиче (нпр. *Python*), ово није неопходно. *TypeScript* подржава статичко типизирање, док *JavaScript* не. Типови у *TypeScript*-у могу бити имплицитно и експлицитно додељени, међутим строго се препоручује додељивање типова.



Слика 5 Однос *JavaScript*-а и *TypeScript*-а

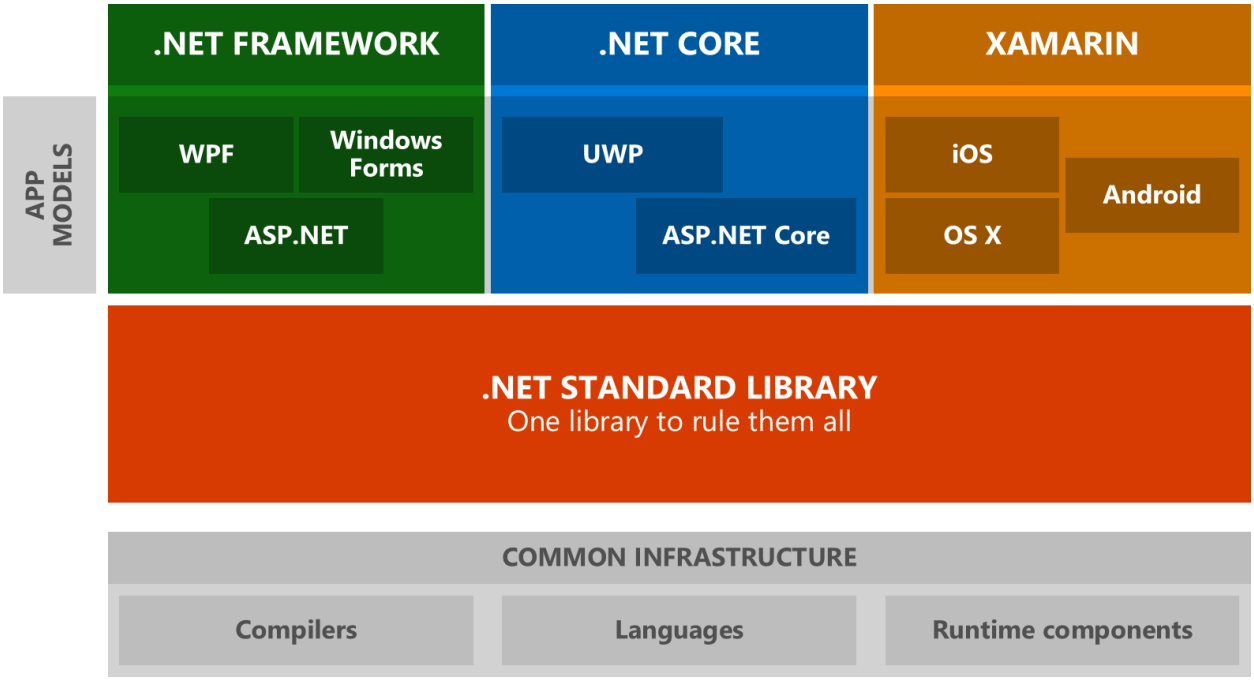
Потреба за развојем оваквог програмског језика настала је услед недостатака *JavaScript*-а. Једна студија показује да се чак 15% грешака насталих приликом коришћењем *JavaScript*-а може уочити коришћењем *TypeScript* програмског језика. Неки од разлога зашто је пожељно користити овај програмски језик, а не чист *JavaScript* су[7]:

- *TypeScript* је поузданији јер контролише *JavaScript* и спречава појаву одређених типова грешака, а најчешће оног који настаје по питању типова
- *TypeScript* чини *JavaScript* код једноставнијим за коришћење, читање и дебаговање
- *TypeScript* је експлицитан и скреће пажњу на то како је систем изграђен и како различити делови система комуницирају једни са другим
- објектно – оријентисано програмирање које омогућава писање робусног и чистог кода
- подржава интерфејсе, *generics,* наслеђивање и модификаторе приступа
- статичка провера типа
- мање кода у односу на *JavaScript*
- компајлирање у односу на интерпретирање које је присутно код *JavaScript*-а.

[7] https://www.typescriptlang.org/

1.4 Серверска страна

На следећој слици приказана је структура целокупне *.NET* платформе.



Слика 6 Приказ *.NET* платформе

1.4.1 *.NET Core*

.NET Core је скуп компоненти за извршавање, скуп библиотека али и скуп компонената компајлера који се могу користити за извршавање различитих задатака на различитим уређајима[8]. Исти код покреће на различитим оперативним системима и архитектурама укључујући x64, x86 и *ARM*. С обзиром да представља скуп компонената отвореног кода, омогућава флексибилан рад на различитим платформама. *.NET Core* подразумева неколико технологија међу којима су *.NET Core*, *ASP.NET Core* і *Entity Framework Core.*

Поред тога, имплементира модерне парадигме попут асинхроног програмирања

1.4.2 *ASP.NET Core*

ASP.NET Core се користи за израду веб апликација на Виндовс, Мак или Линукс оперативном систему. Слободан је и представља оквир отвореног кода који се може извршавати на више платформи. Користи се за израду апликација на локалном уређају али и на облаку (*cloud-based*) као што ус веб апликације, интернет ствари (*Internet of Things*) и бекенд делова апликација.

[8] https://docs.microsoft.com/sr-latn-rs/lifecycle/faq/dotnet-core

Као и *.NET Core* изузетно је модуларан уз минималне трошкове али уз то постоје и напредније опције које могу бити додате уз помоћ *NuGet* пакета уколико апликација то захтева. Резултат овога су високе перформансе, захтев за мањом меморијом и лакоћа одржавања.

Најновија верзија тј. *ASP.NET Core 3.x* се покреће само на *.NET Core*-у, што је новина у односу на претходну верзију *ASP.NET Core 2.x* која се покретала и на *.NET Framework*.

Главне предности *ASP.NET Core* се огледају у следећем[9]:

- могућност покретања на различитим платформама
- брзина и високе перформансе
- интеграција са модерним оквирима за развој корисничког интерфејса као што су *AngularJS, ReactJS, Bootstrap* итд.
- *ASP.NET Core* апликације могу бити хостоване на различитим платформама користећи било који веб сервер попут *IIS*-а, *Apache*-а или *Kestrel*-а (није зависан само од стандардног *IIS* сервера)
- подржава тзв. *dependency injection* дизајн патерн
- омогућава дељење кофа односно омогућава кориснику да изгради библиотеку коју је могуће користити са другим оквирима
- пружа могућности за креирање веб *API*-ја и веб апликација.

1.4.3 *C#*

C# је програмски језик који је развијен од стране Мајкрософт компаније 2000. године као саставни део *.NET Framework* развојног окружења. Представља објектно-оријентисани језик који омогућава развој поузданих и робусних апликација које раде у оквиру .NET екосистема тј. композиције свих имплементација *.NET*-а укључујући и *.NET Core* и *.NET Framework*.

Изворни код написан у *C#* програмском језику се компајлира у *Intermediate language* (*IL*). *IL* код и ресурси као што су битмапе и стрингови бивају складиштени на диск у извршни фајл који се зове асембли (*.exe* или *.dll* екстензије). Када програм бива извршен, асембли се учитава у *CLR*. Након тога, ако је све како треба, *CLR* извршава *Just-In-Time* компајлирање како би превео *IL* код у машински код. *CLR* омогућава и друге сервисе као што је управљање грешкама и ресурсима али и скупљање отпада (*garbage collection*).

Синтакса *C#* програмског језика је у многоме слична као она која се среће у *C, C++* или *Java* програмском језику. Како је, као што је већ напоменуто, припадник објектно-оријентисане групе језика, *C#* подржава концепте енкапсулације, наслеђивања и полиморфизма. Све променљиве и методе, укључујући и *Main* методу, су енкапсулиране у оквиру класе. Класа сама по себи може бити потомак неке надкласе али може и да имплементира читав низ интерфејса.

[9] https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-3.1

2 Кориснички захтеви

У овом поглављу биће представљен студијски пример овог софтверског система.

2.1 Вербални опис

Софтверски систем за део пословања продавнице представља реалан систем који у себи садржи податке о својим производима али и корисницима који те производе могу да поруче.

Администратор система задужен је за унос нових производа у систем али и измену постојећих. Такође, он има право да обрише производ који се више неће дистрибуирати. Поред операција везаних за производе, задатак администратора је и креирање нових типова производа и произвођача као и акције везане за рад са поруџбеницама.

Корисници, са друге стране, поред тога што креирају свој профил, унесене производе од стране администратора могу да претражују и да на основу личних преференци креирају своју поруџбину. Такође, они могу да се обрате администратору путем форме за контакт.

За обе стране, корисника и администратора, неопходно је омогућити пријаву и одјаву са система чиме се обезбеђују различити нивои ауторизације.

2.2 Случајеви коришћења

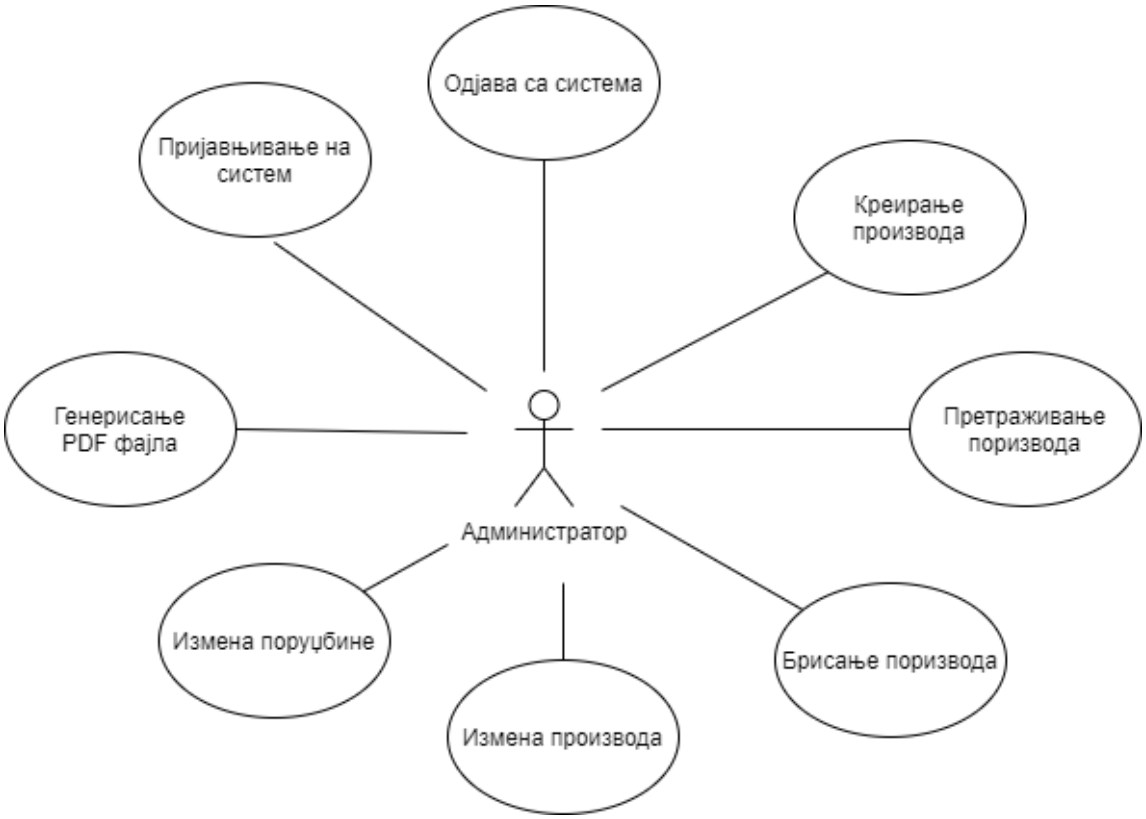
На основу вербалног описа система могу се издвојити следећи случајеви коришћења:

1. Пријављивање на систем
2. Одјава са система
3. Креирање производа
4. Претраживање производа
5. Брисање производа
6. Измена производа
7. Креирање корисничког налога
8. Измена корисничког налога
9. Креирање поруџбине
10. Измена поруџбине
11. Котактирање администратора
12. Генерисање *PDF* фајла

2.2.1 Дијаграми случајева коришћења

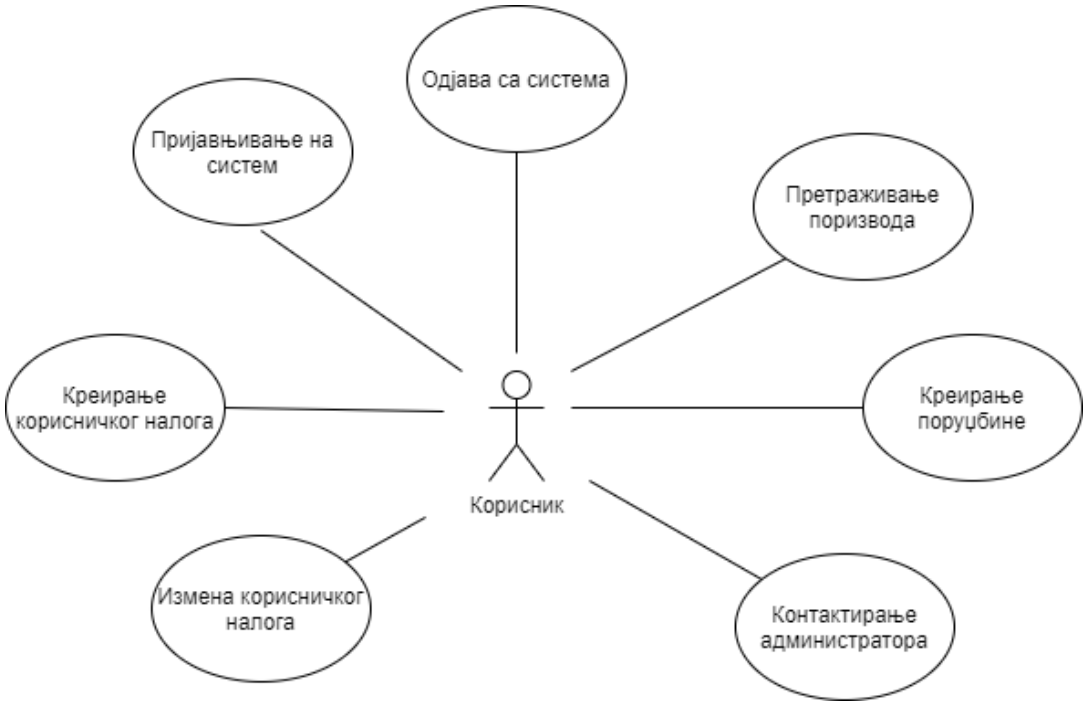
На основу случајева коришћења могу се креирати дијаграми случајева коришћења.

Дијаграм случаја коришћења за администратора приказан је на слици испод:



Слика 7 Дијаграм случаја коришћења за администратора

За самог корисника дијаграм случаја коришћења може се видети на слици испод:



Слика 8 Дијаграм случаја коришћења за корисника

2.2.2 СК1: Случај коришћења - Пријављивање на систем

Назив СК
Пријављивање на систем

Актори СК
Администратор/корисник

Учесници СК
Администратор/корисник и систем (програм)

Предуслов: Систем је укључен и приказује форму за пријављивање.

Основни сценарио СК

1. Администратор/корисник уноси податке за пријављивање. (АПУСО)
2. Корисник контролише да ли је коректно унео корисничко име и лозинку (АНСО)
3. Администратор/корисник позива систем да га пријави. (АПСО)
4. Систем проверава да ли су унети подаци у реду. (СО)
5. Систем приказује администратору/кориснику доступне опције. (ИА)

Алтернативна сценарија
5.1 Уколико систем не може да верификује администратора/корисника он приказује поруку: „Не постоји корисник са унетим корисницчим именом и лозинком. ”. (ИА)

2.2.3 СК2: Случај коришћења - Одјава са система

Назив СК
Одјава са система

Актори СК
Администратор/корисник

Учесници СК
Администратор/корисник и систем (програм)

Предуслов: Систем је укључен и администратор/корисник је улогован под својом шифром. Систем приказује опцију за одјаву са система.

Основни сценарио СК

1. Администратор/корисник позива систем да га одјави. (АПСО)
2. Систем одјављује администратора/корисника са система и приказује почетну страну.(ИА)

Алтернативна сценарија
3.1 Уколико систем не може да одјави администратора/корисника са система: „Није могуће одјавити се са система.”. (ИА)

2.2.4 СК3: Случај коришћења - Креирање производа

Назив СК
Креирање производа

Актори СК
Администратор

Учесници СК
Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је улогован под својом шифром. Учитан је списак произвођача и списак типова поризвода. Систем приказује форму за рад са производом.

Основни сценарио СК

1. Администратор уноси податке о производу. (АПУСО)
2. Администратор контролише да ли је коректно унео податке о производу. (АНСО)
3. Администратор позива систем да креира нови производ са задатим подацима. (АПСО)
4. Систем креира производ са задатим подацима. (СО)
5. Систем приказује администратору поруку: „Производ је успешно креиран“. (ИА)

Алтернативна сценарија
4.1 Уколико систем не може да креира производ он приказује администратору поруку: „Производ није сачуван”. (ИА)

2.2.5 СК4: Случај коришћења - Претраживање производа

Назив СК
Претраживање производа

Актори СК
Администратор/корисник

Учесници СК
Администратор/корисник и систем (програм)

Предуслов: Систем је укључен и администратор/корисник је улогован под својом шифром. Систем приказује форму за рад са производима.

Основни сценарио СК

1. Администратор/корисник уноси критеријум по ком претражује производе. (АПУСО)
2. Администратор/корисник позива систем да пронађе пориизводе на основу задате вредности. (АПСО)
3. Систем тражи производе по задатом критеријуму у учитава податке о њима. (СО)
4. Систем приказује кориснику пронађене производе. (ИА)

Алтернативна сценарија
4.1 Уколико систем не може да пронађе производе он приказује администратору/кориснику празну табелу производа. (ИА)

2.2.6 СК5: Случај коришћења - Брисање производа

Назив СК
Брисање производа

Актори СК
Администратор

Учесници СК
Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је улогован под својом шифром. Систем приказује форму за рад са производом.

Основни сценарио СК

1. Администратор уноси критеријум по ком претражује производе. (АПУСО)
2. Администратор позива систем да пронађе производе на основу задате вредности. (АПСО)
3. Систем тражи производе по задатом критеријуму и учитава податке о њима. (СО)
4. Систем приказује кориснику пронађене производе. (ИА)
5. Администратор бира производ који жели да обрише. (АПУСО)
6. Администратор позива систем да обрише производ. (АПСО)
7. Систем брише производ. (СО)
8. Систем приказује администратор поруку: „Успешно је обрисан производ.” (ИА)

Алтернативна сценарија
4.1 Уколико систем не може да пронађе производе он приказује администратору/кориснику празну табелу производа. Прекида се извршење сценариа.(ИА)
8.1 Уколико систем не може да обрише производ он приказује администратор поруку: „Није могуће обрисати производ”. (ИА)

2.2.7 СК6: Случај коришћења - Измена производа

Назив СК
Измена производа

Актори СК
Администратор

Учесници СК
Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је улогован под својом шифром. Учитан је асортиман проивода, листа произвођача и листа типова производа. Систем приказује форму за рад са производом.

Основни сценарио СК

1. Администратор бира производ који жели да измени. (АПУСО)
2. Администратор мења податке о производу. (АПУСО)
3. Администратор контролише да ли је коректно унео податке о производу. (АНСО)
4. Администратор позива систем да запамти податке о производу. (АПСО)
5. Систем памти податке о производу. (СО)
6. Систем приказује администратору поруку: „Успешно сте изменили производ.” (ИА)

Алтернативна сценарија
6.1 Уколико систем не може да запамти податке о производу он приказује администратору поруку „Није могуће изменити производ”. (ИА)

2.2.8 СК7: Случај коришћења - Креирање корисничког налога

Назив СК
Креирање корисничког налога

Актори СК
Корисник

Учесници СК
Корисник и систем (програм)

Предуслов: Систем је укључен. Систем приказује форму за креирање корисничког налога.

Основни сценарио СК

1. Посетилац уноси податке о новом кориснику. (АПУСО)
2. Посетилац контролише да ли је коректно унео податке о кориснику. (АНСО)
3. Посетилац позива систем да креира новог корисника са задатим подацима. (АПСО)

4. Систем креира новог корисника. (СО)
5. Систем приказује посетиоцу поруку: „Успешно креиран налог“. (ИА)

Алтернативна сценарија
5.1 Уколико систем не може да креира корисника он приказује постиоцу одговарајућу поруку. (ИА)

2.2.9 СК8: Случај коришћења - Измена корисничког налога

Назив СК
Измена корисничког налога

Актори СК
Корисник

Учесници СК
Корисники систем (програм)

Предуслов: Систем је укључен и корисник је улогован под својом шифром. Учитани су подаци о кориснику. Систем приказује форму за рад са профилом корисника.

Основни сценарио СК

1. Систем тражи податке о кориснику. (СО)
2. Систем приказује кориснику пронађене податке. (ИА)
3. Корисник бира податке који жели да измени. (АПУСО)
4. Корисник мења податке. (АПУСО)
5. Корисник контролише да ли је коректно унео податке. (АНСО)
6. Корисник позива систем да запамти податке. (АПСО)
7. Систем памти податке о кориснику. (СО)
8. Систем приказује администратору поруку: „Успешно сте изменили податке.” (ИА)

Алтернативна сценарија
2.1 Уколико систем не може да нађе податке о кориснику он приказује кориснику поруку: „Нису пронањени подаци о кориснику”. Прекида се извршење сценариа. (ИА)
8.1 Уколико систем не може да запамти податке о кориснику он приказује кориснику поруку „Није могуће изменити податке”. (ИА)

2.2.10 СК9: Случај коришћења - Креирање поруџбине

Назив СК
Креирање поруџбине

Актори СК
Корисник

Учесници СК
Корисник и систем (програм)

Предуслов: Систем је укључен и корисник је улогован под својом шифром. Учитан је асортиман проивода. Систем приказује форму за рад са поруџбином.

Основни сценарио СК

1. Корисник уноси податке за нову поруџбину. (АПУСО)
2. Корисник контролише да ли је коректно унео податке у нову поруџбину. (АНСО)
3. Корисник позива систем да креира нову поруџбину. (АПСО)
4. Систем креира поруджбину са задатим подацима. (СО)
5. Систем приказује кориснику поруку: „Поруџбина је сачувана”. (ИА)

Алтернативна сценарија
4.1 Уколико систем не може да креира нову поруџбину он приказује кориснику поруку: „Систем не може да креира нову поруџбину ”. (ИА)

2.2.11 СК10: Случај коришћења - Измена поруџбине

Назив СК
Измена поруџбине

Актори СК
Администратор

Учесници СК
Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је улогован под својом шифром. Учитане су поруџбине и листа опција за измену статуса поруџбине. Систем приказује форму за рад са поруџбином.

Основни сценарио СК

1. Администратор бира поруџбину који жели да измени. (АПУСО)
2. Администратор мења податке о поруџбини. (АПУСО)

3. Администратор контролише да ли је коректно унео податке о поруџбини. (АНСО)
4. Администратор позива систем да запамти податке о поруџбини. (АПСО)
5. Систем памти податке о поруџбини. (СО)
6. Систем приказује администратору поруку: „Успешно сте изменили поруџбину.” (ИА)

Алтернативна сценарија
6.1 Уколико систем не може да запамти податке о поруџбини он приказује администратору поруку „Није могуће изменити одабрани производ”. (ИА)

2.2.12 СК11: Случај коришћења - Котактирање администратора

Назив СК
Контактирање администратора

Актори СК
Корисник

Учесници СК
Корисник и систем (програм)

Предуслов: Систем је укључен и корисник је улогован под својом шифром. Систем приказује форму за контакт.

Основни сценарио СК

1. Корисник уноси податке у форми за контакт. (АПУСО)
2. Корисник контролише да ли је коректно унео податке у форму. (АНСО)
3. Корисник позива систем да пошаље поруку администратору. (АПСО)
4. Систем шаље поруку администратору. (СО)
5. Систем приказује кориснику поруку: „Порука је послата”. (ИА)

Алтернативна сценарија
4.1 Уколико систем не може да пошаље поруку он приказује кориснику поруку: „Није могуће послати поруку ”. (ИА)

2.2.13 СК12: Случај коришћења - Генерисање *PDF* фајла

Назив СК
Генерисање *PDF* фајла

Актори СК
Администратор

Учесници СК
Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је улогован под својом шифром. Учитан је списак произвођача и списак типова поризвода. Систем приказује форму за рад са производом.

Основни сценарио СК

1. Администратор позива систем да генерише нови *PDF* фајл . (АПСО)
2. Систем генерише фајл. (СО)
3. Систем приказује администратору генерисан фајл. (ИА)

Алтернативна сценарија
4.1 Уколико систем не може да генерише фајл он приказује администратору поруку: „Није могуће генерисати фајл”. (ИА)

3 Фаза анализе

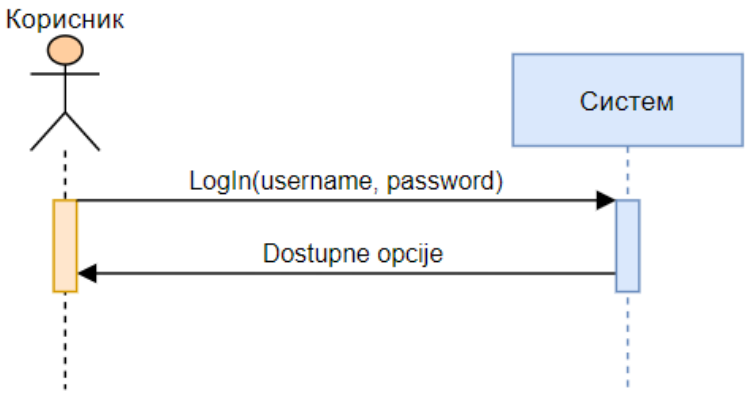
Фаза анализе описује логичку структуру и понашање софтверског система. Понашање софтверског система је описано помоћу системских дијаграма секвенци и преко системских операција. Структура софтерског система се описује помоћу концептуалног и релационог модела[10].

3.1 Дијаграми секвенци

Системски дијаграм секвенци приказује, за издвојени сценарио СК, догађаје у одређеном редоследу, који успостављају интеракцију између актора и софтверског система[11].

3.1.1 ДС1: Дијаграми секвенци случаја коришћења – Пријављивање на систем

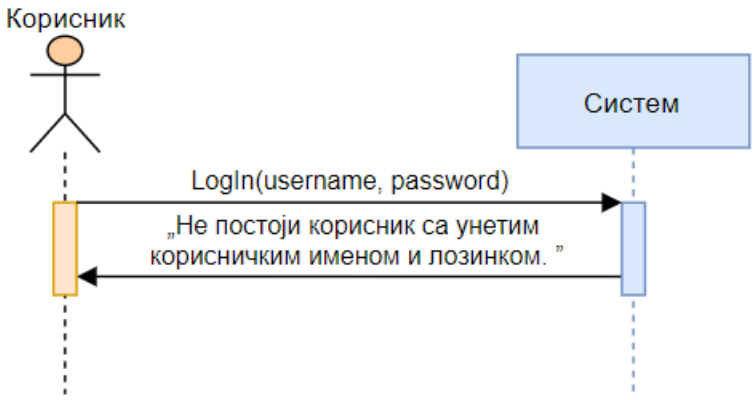
1. Администратор/корисник позива систем да га пријави. (АПСО)
2. Систем приказује администратору/кориснику доступне опције. (ИА)



Слика 9 Основни случај пријављивања на систем

Алтернативна сценарија
2.1 Уколико систем не може да верификује администратора/корисника он приказује поруку: „Не постоји корисник са унетим корисницчим именом и лозинком.”. (ИА)

[10]Vlajić, S. (2015). Projektovanje softvera (skripta). Beograd, Srbija: FON
[11] влајић исто



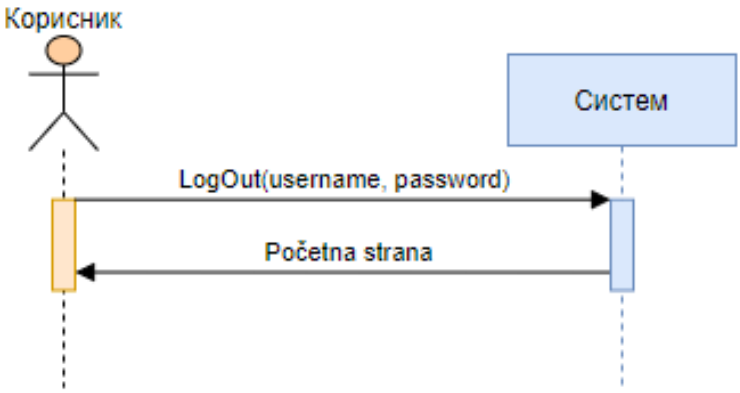
Слика 10 Алтернативни случај пријављивања на систем за случај 2.1

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

1. *signal* **LogIn**(username, password);

3.1.2 ДС2: Дијаграми секвенци случаја коришћења – Одјава са система

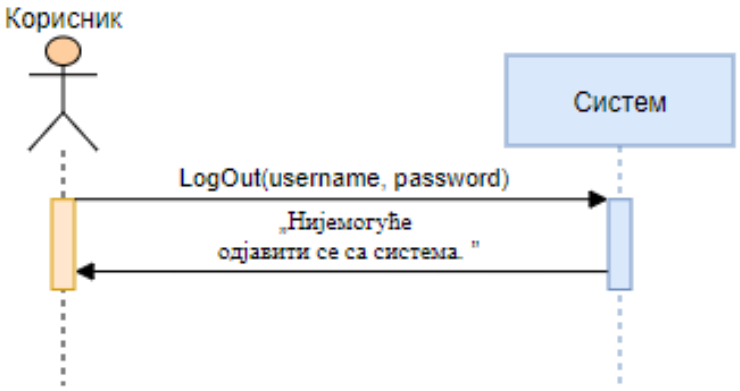
1. Администратор/корисник позива систем да га одјави. (АПСО)
2. Систем одјављује администратора/корисника са система и приказује почетну страну.(ИА)



Слика 11 Основни случај одјаве са система

Алтернативна сценарија

2.1 Уколико систем не може да одјави администратора/корисника са система: „Није могуће одјавити се са система.”. (ИА)



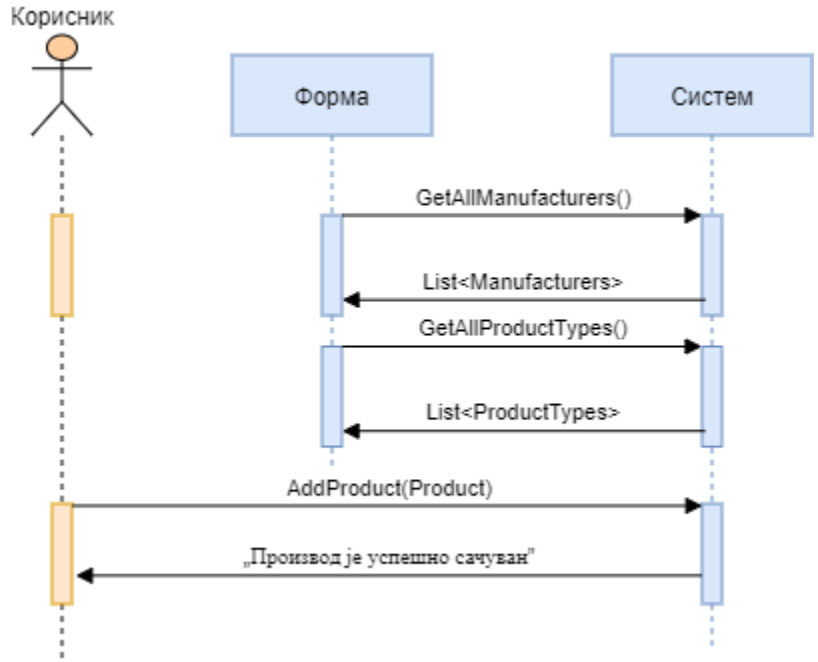
Слика 12 Алтернативни случај одјаве са система

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

1. *signal* **LogOut**(username, password);

3.1.3 ДС3: Дијаграми секвенци случаја коришћења – Креирање производа

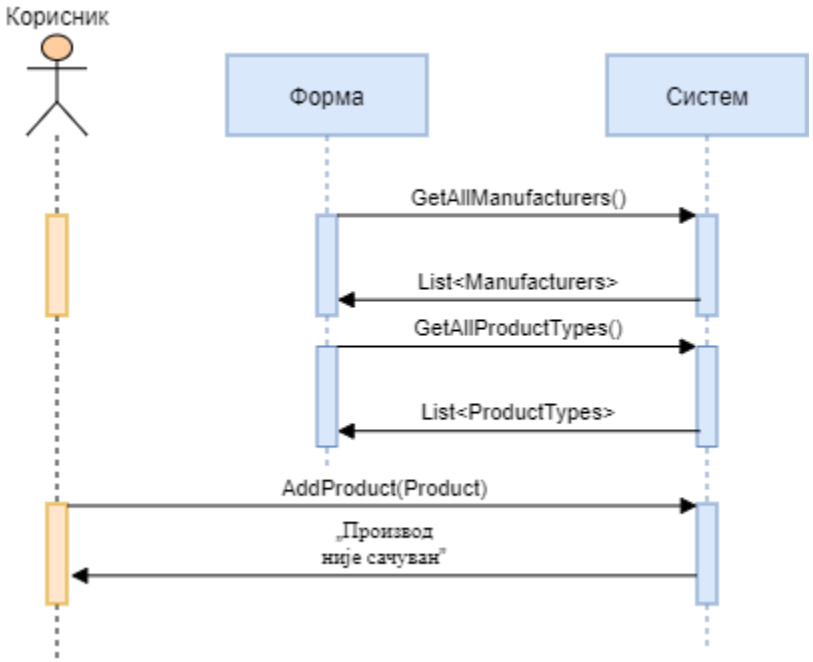
1. Администратор позива систем да креира нови производ са задатим подацима. (АПСО)
2. Систем приказује администратору поруку: „Производ је успешно сачуван“. (ИА)



Слика 13 Основни случај креирања производа

Алтернативна сценарија

2.1 Уколико систем не може да креира производ он приказује администратору поруку: „Производ није сачуван”. (ИА)



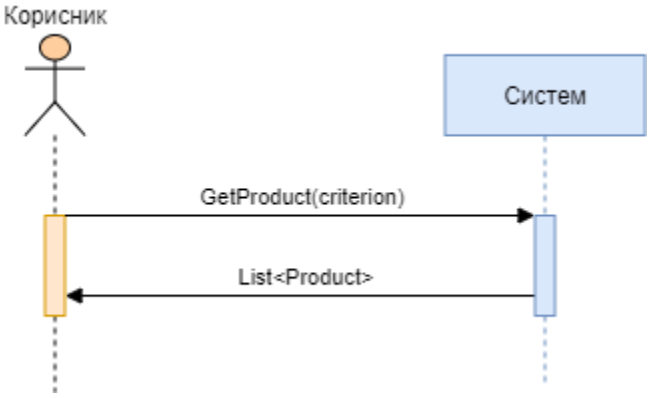
Слика 14 Алтернативни случај креирања производа

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

1. *signal* **GetAllManufacturers**();
2. *signal* **GetAllProductTypes**();
3. *signal* **AddProduct**(Product);

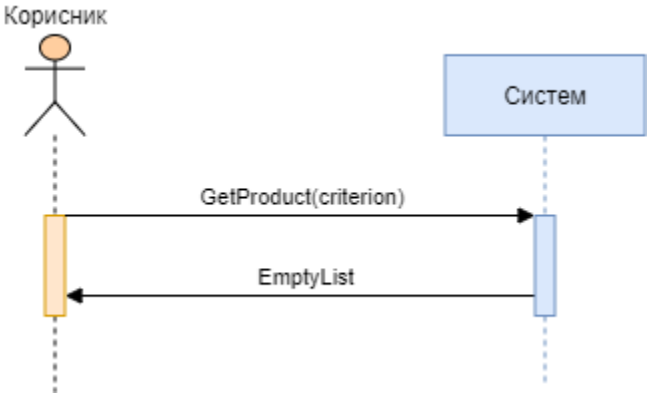
3.1.4 ДС4: Дијаграми секвенци случаја коришћења – Претраживање производа

1. Систем тражи производе по задатом критеријуму у учитава податке о њима. (СО)
2. Систем приказује кориснику пронађене производе. (ИА)



Слика 15 Основни случај претраживања производа

Алтернативна сценарија
2.1 Уколико систем не може да пронађе производе он приказује администратору/кориснику празну табелу производа. (ИА)



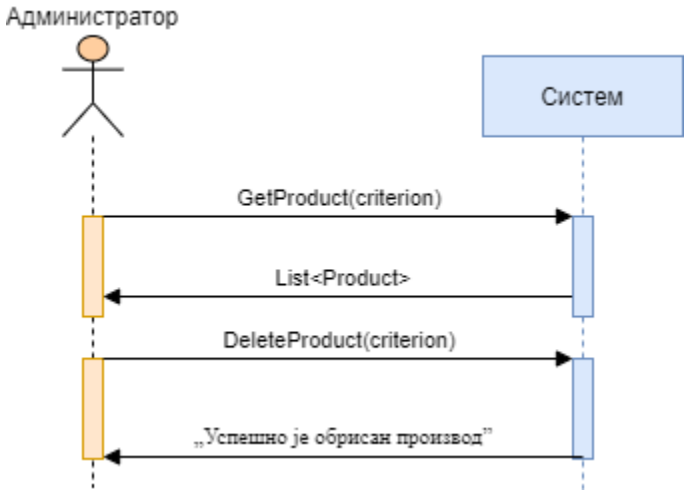
Слика 16 Алтернативни случај претраживања производа

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

1. *signal* **GetProduct**(criterion);

3.1.5 ДС5: Дијаграми секвенци случаја коришћења – Брисање производа

1. Администратор позива систем да пронађе производе на основу задате вредности. (АПСО)
2. Систем приказује кориснику пронађене производе. (ИА)
3. Администратор позива систем да обрише производ. (АПСО)
4. Систем приказује администратор поруку: „Успешно је обрисан производ.“(ИА)



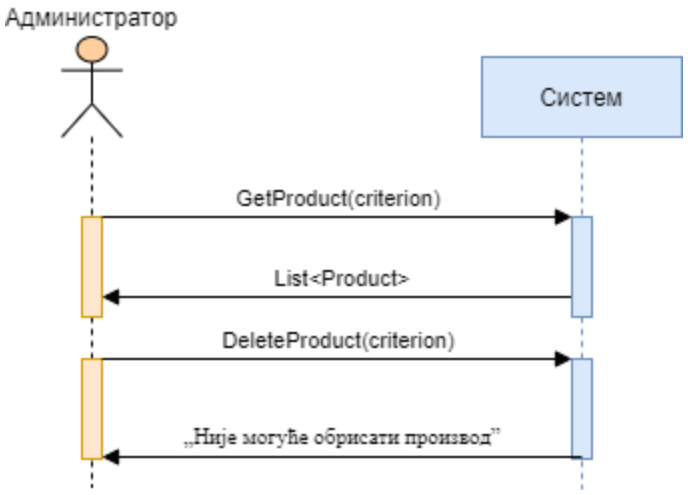
Слика 17 Основни случај брисања производа

Алтернативна сценарија

2.1 Уколико систем не може да пронађе производе он приказује администратору/кориснику празну табелу производа. Прекида се извршење сценариа. (ИА)

Слика 18 Алтернативни случај брисања производа sa случај 2.1

4.1 Уколико систем не може да обрише производ он приказује администратор поруку: „Није могуће обрисати производ”. (ИА)



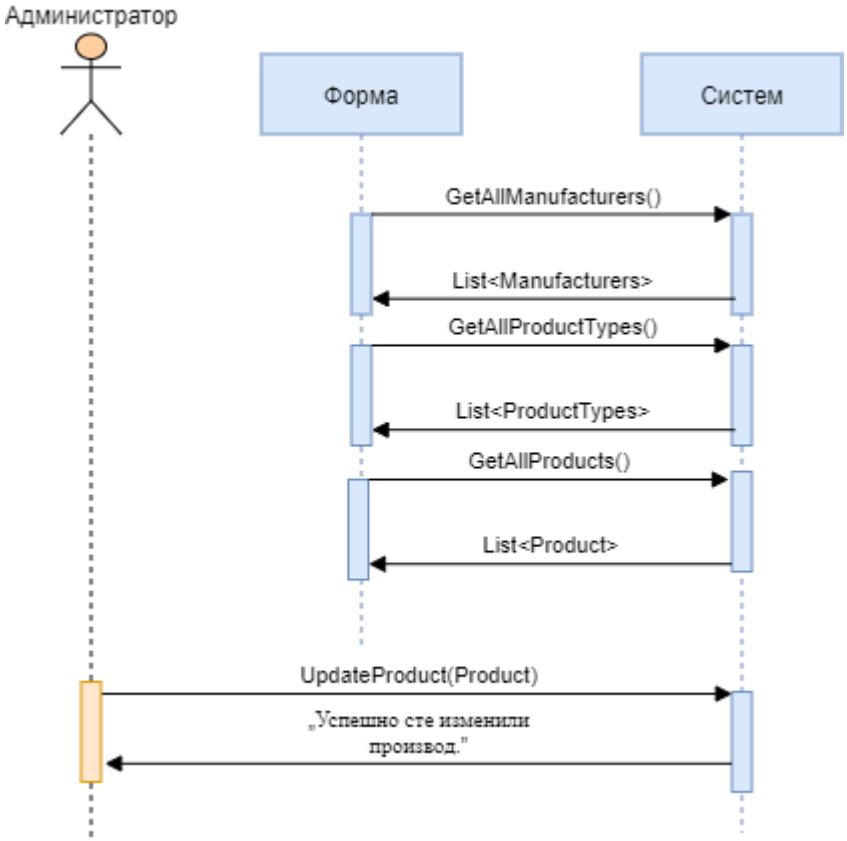
Слика 19 Алтернативни случај брисања производа sa случај 4.1

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

1. *signal* **GetProduct**(criterion);
2. *signal* **DeleteProduct**(criterion);

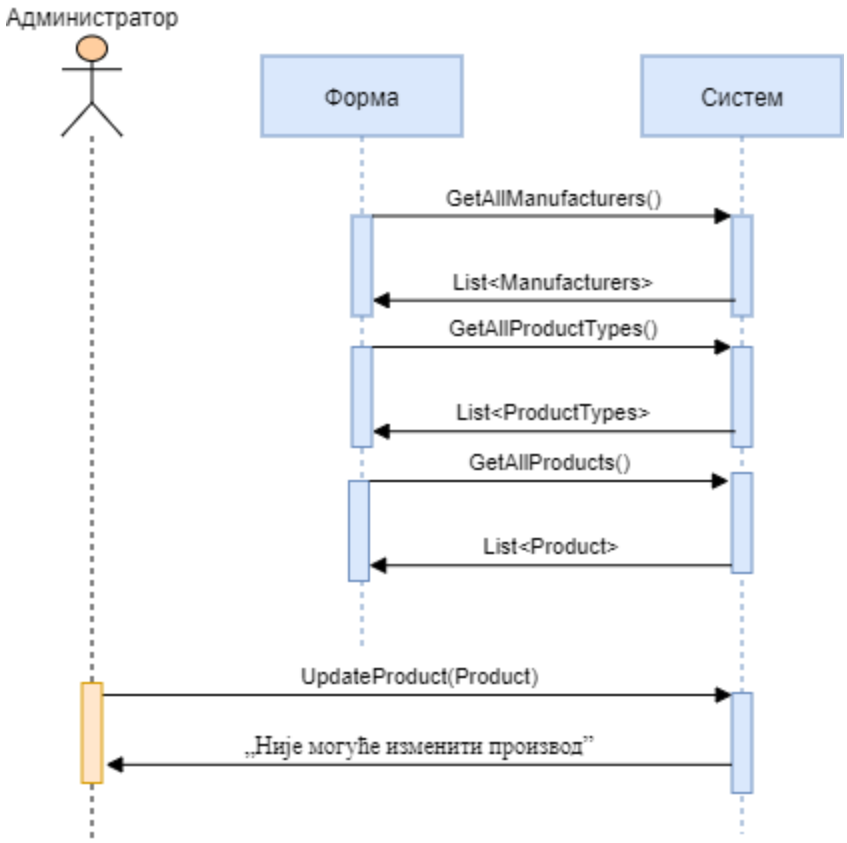
3.1.6 ДС6: Дијаграми секвенци случаја коришћења – Измена производа

1. Администратор позива систем да запамти податке о производу. (АПСО)
2. Систем приказује администратору поруку: „Успешно сте изменили производ” (ИА)



Слика 20 Основни случај измене производа

Алтернативна сценарија
2.1 Уколико систем не може да запамти податке о производу он приказује администратору поруку „Није могуће изменити производ.“. (ИА)



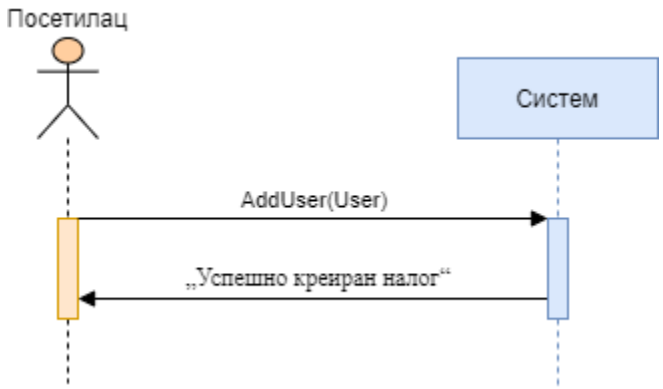
Слика 21 Алтернативни случај измене производа

Са наведених секвенцних дијаграма уочавају се 4 системске операције које треба пројектовати:

1. *signal* **GetAllManufacturers**();
2. *signal* **GetAllProductTypess**();
3. *signal* **GetAllProducts**();
4. *signal* **UpdateProduct**(Product);

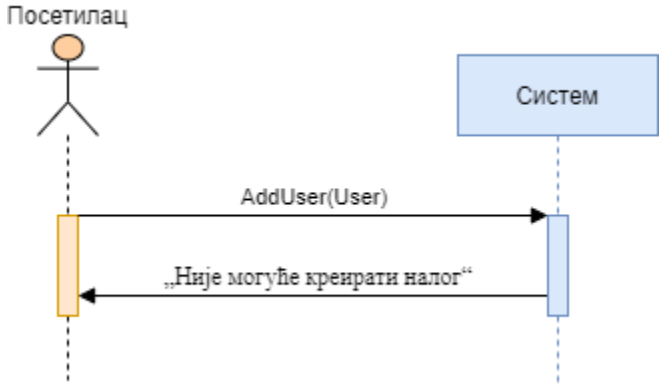
3.1.7 ДС7: Дијаграми секвенци случаја коришћења – Креирање корисничког налога

1. Посетилац позива систем да креира новог корисника са задатим подацима. (АПСО)
2. Систем приказује посетиоцу поруку: (ИА)



Слика 22 Основни случај креирања корисничког налога

Алтернативна сценарија
2.1 Уколико систем не може да креира корисника он приказује постиоцу поруку: „Није могуће креирати налог“. (ИА)



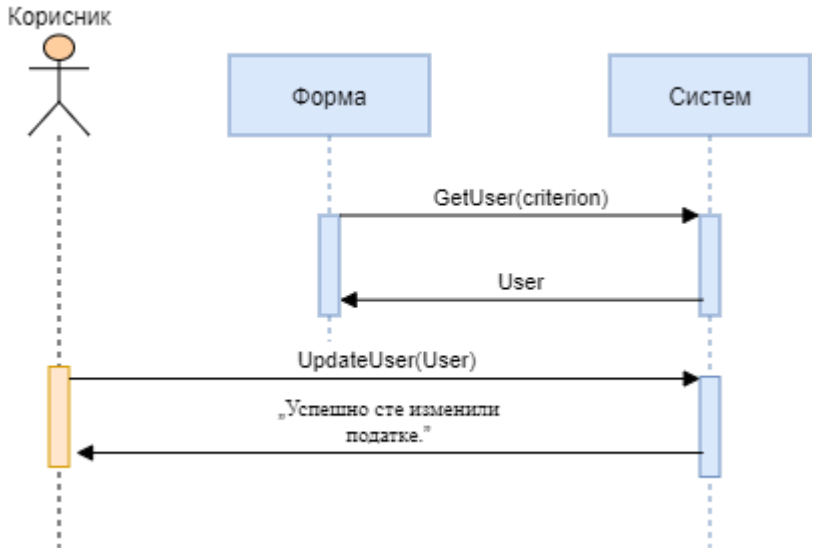
Слика 23 Алтернативни случај креирања корисичког налога

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

1. *signal* **AddUser**(User);

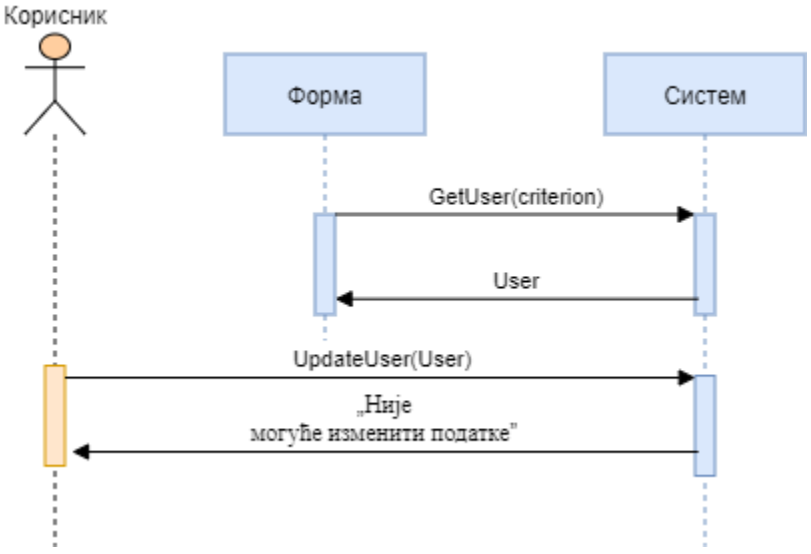
3.1.8 ДС8: Дијаграми секвенци случаја коришћења – Измена корисничког налога

1. Корисник позива систем да запамти податке. (АПСО)
2. Систем приказује администратору поруку: „Успешно сте изменили податке.” (ИА)



Слика 24 Основни случај измене корисничког налога

Алтернативна сценарија
2.1 Уколико систем не може да запамти податке о кориснику он приказује кориснику поруку „Није могуће изменити податке”. (ИА)



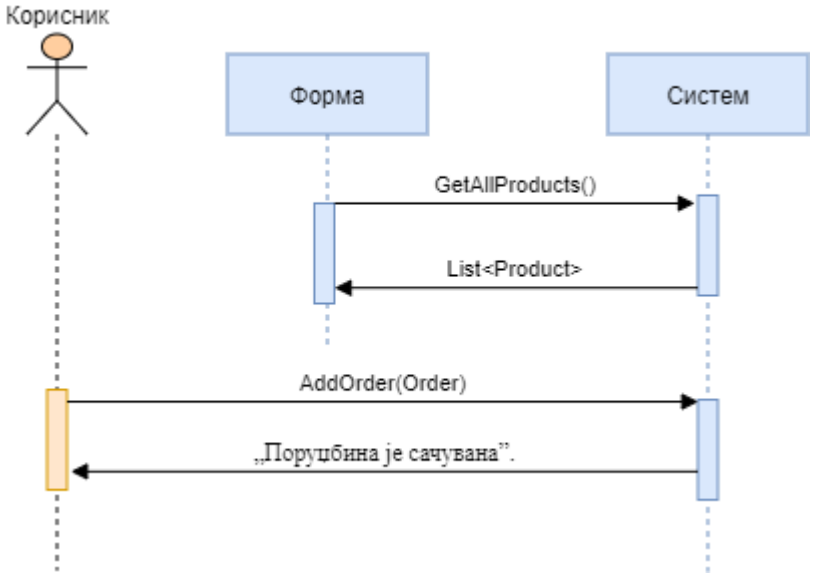
Слика 25 Алтернативни случај измене корисничког налога

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

1. *signal* **GetUser**(criterion);
2. *signal* **UpdateUser**(User);

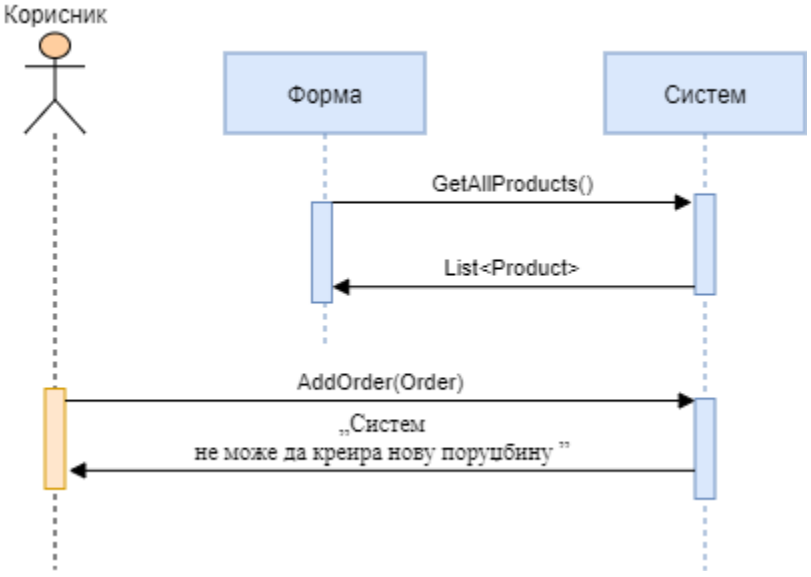
3.1.9 ДС9: Дијаграми секвенци случаја коришћења – Креирање поруџбине

1. Корисник позива систем да креира нову поруџбину. (АПСО)
2. Систем приказује кориснику поруку: „Поруџбина је сачувана”. (ИА)



Слика 26 Основни случај креирања поруџбине

Алтернативна сценарија
2.1 Уколико систем не може да креира нову поруџбину он приказује кориснику поруку: „Систем не може да креира нову поруџбину ”. (ИА)



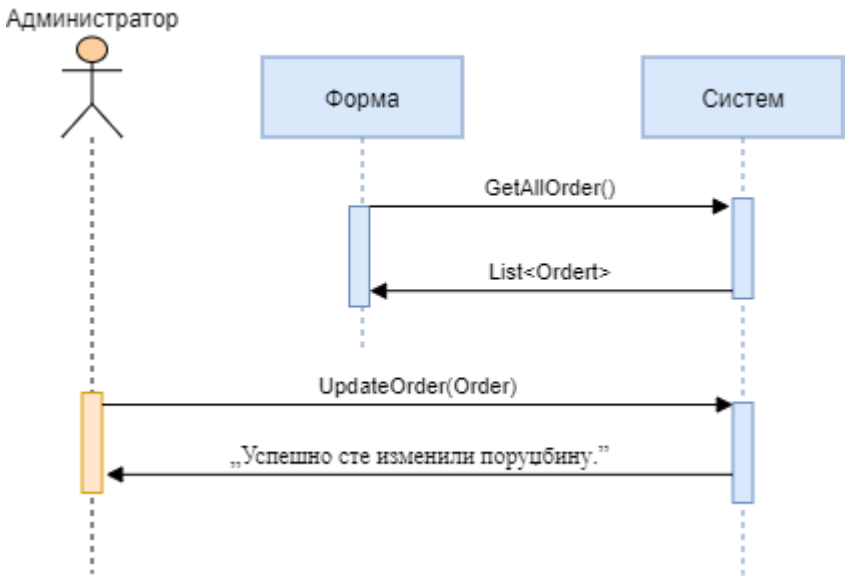
Слика 27 Алтернативни случај креирања поруџбине

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

1. *signal* **GetAllProducts**();
2. *signal* **AddOrder**(Order);

3.1.10 ДС10: Дијаграми секвенци случаја коришћења – Измена поруџбине

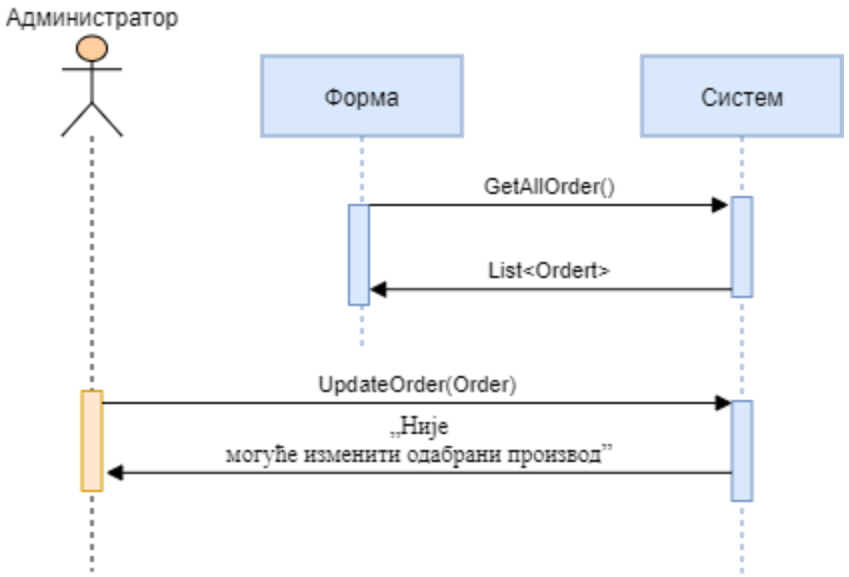
1. Администратор позива систем да запамти податке о поруџбини. (АПСО)
2. Систем приказује администратору поруку: „Успешно сте изменили поруџбину.” (ИА)



Слика 28 Основни случај измене поруџбине

Алтернативна сценарија

2.1 Уколико систем не може да запамти податке о поруџбини он приказује администратору поруку „Није могуће изменити одабрани производ”. (ИА)



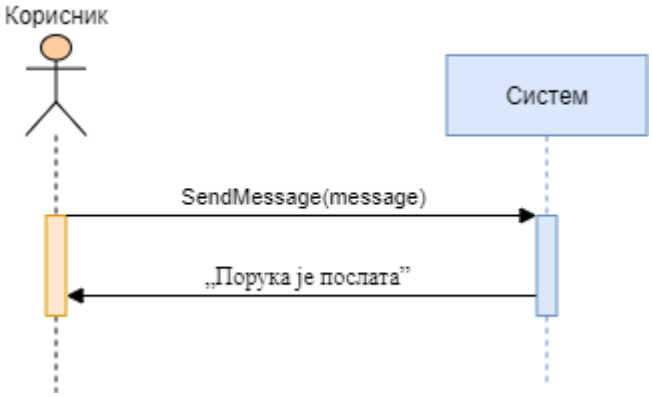
Слика 29 Алтернативни случај измене поруџбине

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

1. *signal* **GetAllOrders**();
2. *signal* **UpdateOrder**(Order);

3.1.11 ДС11: Дијаграми секвенци случаја коришћења – Котактирање администратора

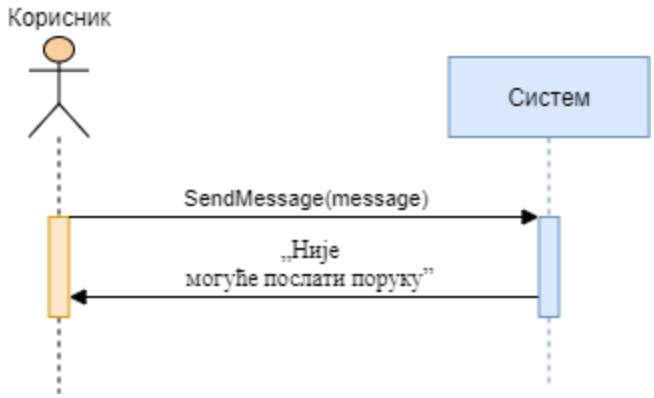
1. Корисник позива систем да пошаље поруку администратору. (АПСО)
2. Систем приказује кориснику поруку: „Порука је послата”. (ИА)



Слика 30 Основни случај контактирања администратора

Алтернативна сценарија

2.1 Уколико систем не може да пошаље поруку он приказује кориснику поруку: „Није могуће послати поруку”. (ИА)



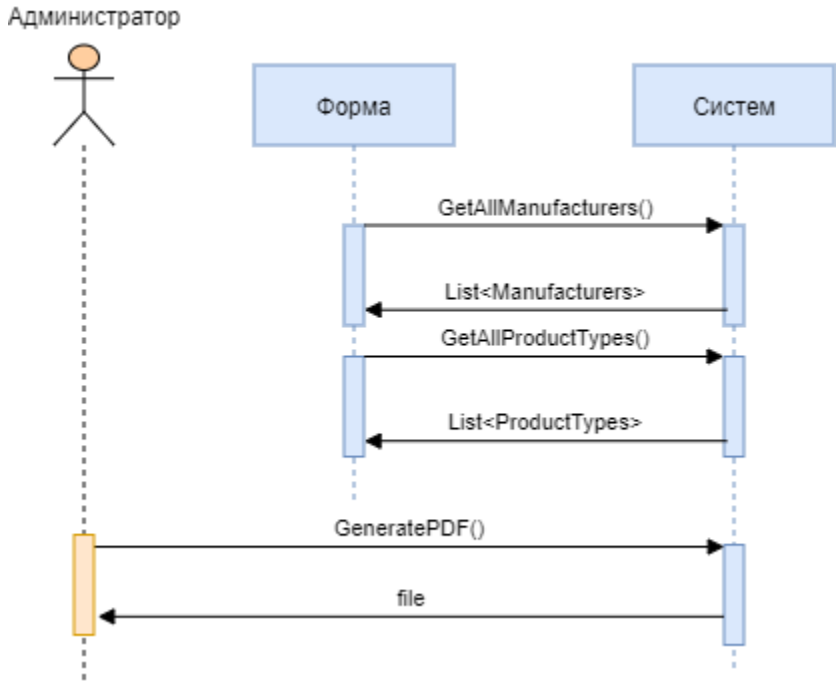
Слика 31 Алтернативни случај контактирања администратора

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

1. *signal* **SendMessage**(message);

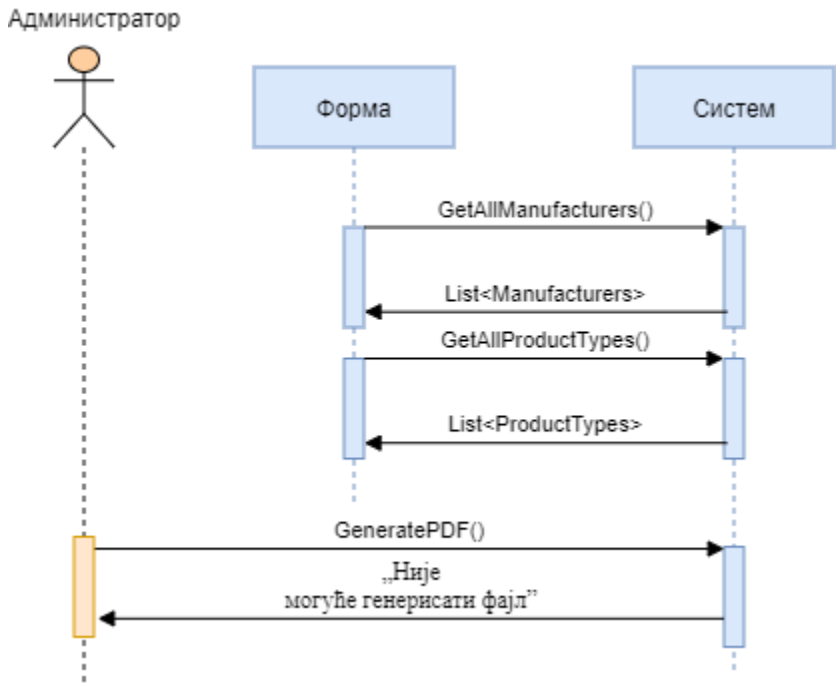
3.1.12 ДС12: Дијаграми секвенци случаја коришћења – Генерисање *PDF* фајла

1. Администратор позива систем да генерише нови *PDF* фајл . (АПСО)
2. Систем приказује администратору генерисан фајл. (ИА)



Слика 32 Основни случај генерисања фајла

Алтернативна сценарија
4.1 Уколико систем не може да генерише фајл он приказује администратору поруку: „Није могуће генерисати фајл”. (ИА)



Слика 33 Алтернативни случај генерисања фајла

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

1. *signal* **GetAllManufacturers**();
2. *signal* **GetAllProductTypess**();
3. *signal* **GeneratePDF**();

3.1.13 Закључак на основу дијаграма секвенци случаја коришћења

Као резултат анализе сценарија добијено је укупно 17 системских операција које треба пројектовати:

1. *signal* **LogIn**(username, password);
2. *signal* **LogOut**(username, password);
3. *signal* **GetAllManufacturers**();
4. *signal* **GetAllProductTypes**();
5. *signal* **GetAllOrders**();
6. *signal* **GetAllProducts**();
7. *signal* **AddProduct**(Product);
8. *signal* **GetProduct**(criterion);
9. *signal* **DeleteProduct**(criterion);
10. *signal* **UpdateProduct**(Product);
11. *signal* **AddUser**(User);
12. *signal* **GetUser**(criterion);
13. *signal* **UpdateUser**(User);
14. *signal* **AddOrder**(Order);
15. *signal* **UpdateOrder**(Order);
16. *signal* **SendMessage**(message);
17. *signal* **GeneratePDF**();

3.2 Понашање софтверског система – Дефинисање уговора о системским операцијама

3.2.1 Уговор УГ1: *LogIn*

Операција: LogIn(username, password): signal;
Веза са СК: СК1
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *User*.
Постуслов: Корисник је пријављен на систем.

3.2.2 Уговор УГ2: *LogOut*

Операција: LogOut (username, password): signal;
Веза са СК: СК2
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *User*.
Постуслов: Корисник је одјављен са система.

3.2.3 Уговор УГ3: *GetAllManufacturers*

Операција: GetAllManufacturers(): signal;
Веза са СК: СК3, СК6, СК12
Предуслов: /
Постуслов: /

3.2.4 Уговор УГ4: *GetAllProductTypes*

Операција: GetAllProductTypes(): signal;
Веза са СК: СК3, СК6, СК11
Предуслов: /
Постуслов: /

3.2.5 Уговор УГ5: *GetAllOrders*

Операција: GetAllOrders(): signal;
Веза са СК: СК10
Предуслов: /
Постуслов: /

3.2.6 Уговор УГ6: *GetAllProducts*

Операција: GetAllProducts(): signal;
Веза са СК: СК6, СК9
Предуслов: /
Постуслов: /

3.2.7 Уговор УГ7: *AddProduct*

Операција: AddProduct(Product): signal;
Веза са СК: СК3
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *Product.*
Постуслов: Производ је креиран.

3.2.8 Уговор УГ8: *GetProduct*

Операција: GetProduct (criterion): signal;
Веза са СК: СК4, СК5
Предуслов: /
Постуслов: /

3.2.9 Уговор УГ9: *DeleteProduct*

Операција: DeleteProduct (criterion): signal;
Веза са СК: СК5
Предуслов: Морају бити задовољена структурна ограничења над објектом *Product.*
Постуслов: Објекат је обрисан.

3.2.10 Уговор УГ10: *UpdateProduct*

Операција: UpdateProduct(Product): signal;
Веза са СК: СК6
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *Product.*
Постуслов: Измењен производ је запамћен.

3.2.11 Уговор УГ11: *AddUser*

Операција: AddUser(User): signal;
Веза са СК: СК7
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *User.*
Постуслов: Корисник је креиран.

3.2.12 Уговор УГ12: *GetUser*

Операција: GetUser (criterion): signal;
Веза са СК: СК8
Предуслов: /
Постуслов: /

3.2.13 Уговор УГ13: *UpdateUser*

Операција: UpdateUser (User): signal;
Веза са СК: СК8
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *User*.
Постуслов: Измењен корисник је запамћен.

3.2.14 Уговор УГ14: *AddOrder*

Операција: AddOrder (Order): signal;
Веза са СК: СК9
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *Order*.
Постуслов: Поруџбина је креирана.

3.2.15 Уговор УГ15: *UpdateOrder*

Операција: UpdateOrder(Order): signal;
Веза са СК: СК10
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *Order*.
Постуслов: Измењена је поруџбина.

3.2.16 Уговор УГ16: *SendMessage*

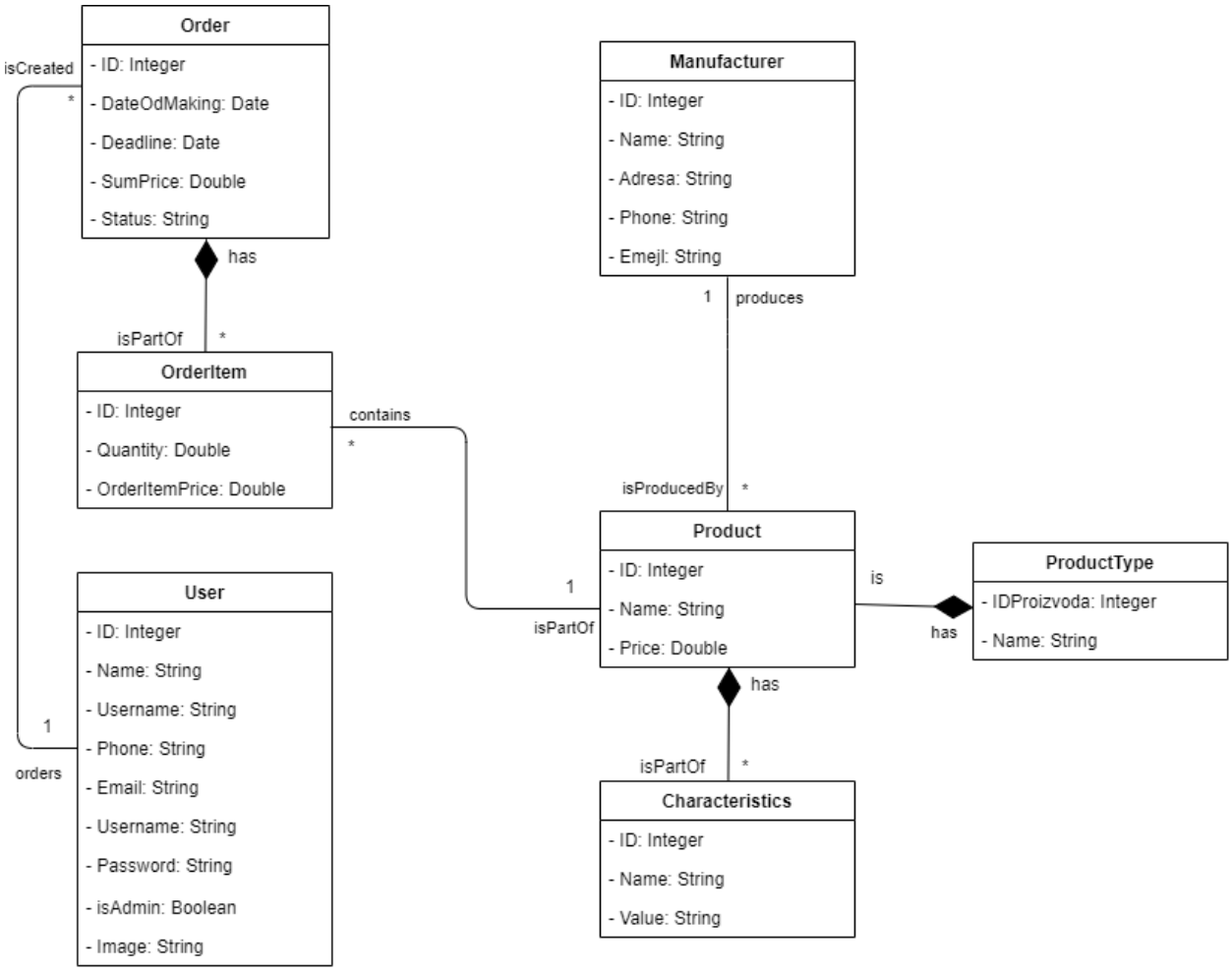
Операција: SendMessage(message): signal;
Веза са СК: СК11
Предуслов: Морају бити задовољена структурна ограничења над објектом *message*.
Постуслов: Порука је послата.

3.2.17 Уговор УГ17: *GeneratePDF*

Операција: GeneratePDF (): signal;
Веза са СК: СК12
Предуслов: /
Постуслов: Фајл је генерисан.

3.3 Структура софтверског система – Концептуални модел

На следећој слици приказан је концептуални модел софтверског система:



Слика 34 Концептуални модел

obicna veza kod pt

3.4 Структура софтверског система - Релациони модел

На основу концептуалног модела може се сачинити релациони модел података:

User(ID, Name, Surname, Phone, Email, Username, Password, isAdmin, Image)
ProductType(ID, Name)
Manufacturer(ID, Name, Phone, Email, Adress)
Product(ID, *ProductTypeID*, Name, Price, *ManufacturerID*)
Order(ID, DateOfMaking, Deadline, SumPrice, Status, *UserID*)
OrderItem(ID, *OrderID*, Quantity, OrderItemPrice, *ProductID*)
Characteristics(ID, *ProductID*, Name, Value)

Табела *User*		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	INSERT / UPDATE Cascade Order DELETE Restrict Order
	ID	Integer	Not null and >0			
	Name	String	Not null			
	Surname	String	Not null			
	Phone	String	Not null			
	Email	String	Not null			
	Username	String	Not null			
	Password	String	Not null			
	isAdmin	Boolean				
	Image	String				

Табела 1 Табела *User*

Табела *ProductType*		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	INSERT / UPDATE Cascade OrderItem DELETE Cascade OrderItem
	ID	Integer	Not null and >0			
	Name	String	Not null			

Табела 2 Табела *ProductType*

Табела *Manufacturer*		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	INSERT / UPDATE Cascade Product DELETE Restrict Product
	ID	Integer	Not null and >0			
	Name	String	Not null			
	Phone	String	Not null			
	Email	String	Not null			
	Adress	String	Not null			

Табела 3 Табела *Manufacturer*

Табела *Product*		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	INSERT Restricted User UPDATE Restricted Manufacturer, ProductType Cascade OrderItem DELETE Restrict OrderItem
	ID	Integer	Not null and >0			
	Name	String	Not null			
	Price	Double	Not null			
	ProductTypeID	Integer	Not null and >0			
	ManufacturerID	Integer	Not null and >0			

Табела 4 Табела *Product*

Табела *Characteristics*		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибут	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	INSERT Restricted User UPDATE Restricted User Cascade OrderItem DELETE Cascade OrderItem
	ID	Integer	Not null and >0			
	ProductID	Integer	Not null and >0			
	Name	String				
	Value	String				

Табела 5 Табела *Characteristics*

Табела *Order*		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	INSERT Restricted User UPDATE Restricted User Cascade OrderItem DELETE Cascade OrderItem
	ID	Integer	Not null and >0			
	DateOfMaking	Date	Not null			
	Deadline	Date	Not null			
	SumPrice	Double				
	Status	String				
	UserID	Integer	Not null and >0			

Табела 6 Табела *Orders*

Табела *OrderItem*		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Име	Тип атрибута	Вредност атрибута	Међузав. атрибута једне табеле	Међузав. атрибута више табела	INSERT Restricted User UPDATE Restricted User Cascade OrderItem DELETE Cascade OrderItem
	ID	Integer	Not null and >0			
	OrderID	Integer	Not null and >0			
	Quantity	Integer	Not null			
	OrderItemPrice	Double				
	ProductID	Integer	Not null and >0			

Табела 7 Табела *OrderItem*

4 Фаза пројектовања

4.1 Пројектовање корисничког интерфејса

Кориснички интефејс се састоји од екранских форми које су одговорне за прихватање и података и догађаја које униосе актори али и прослеђивање тих података ка серверу. Такође, клијентска страна апликације, на којој се налази кориснички интерфејс, задужена је за приказивање података који стижу са сервера кориснику.

Сценарији коришћења екранских форми су директно повезани са сценаријима случајева коришћења.

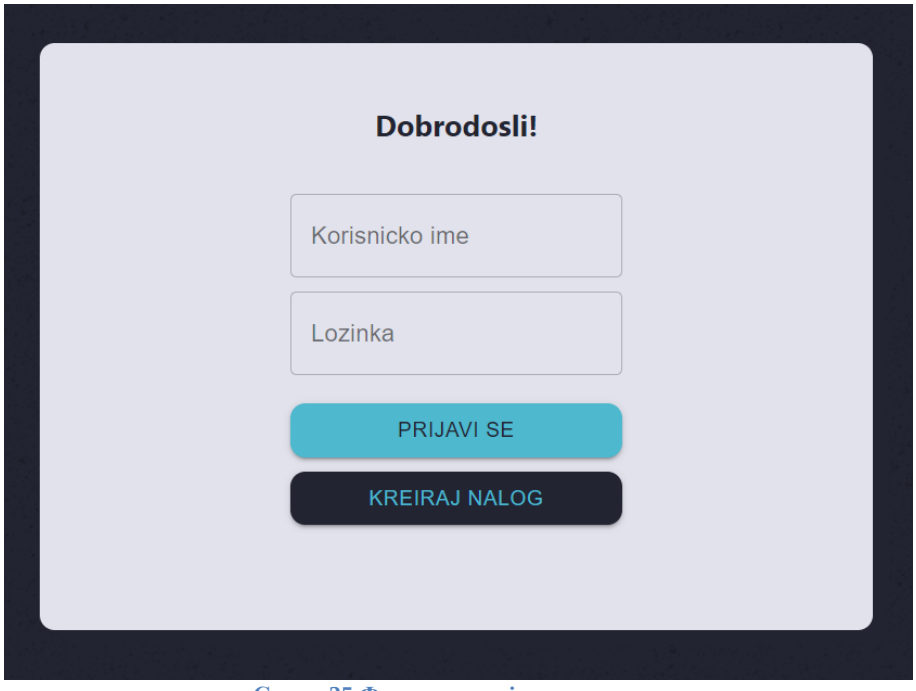
4.1.1 СК1: Случај коришћења - Пријављивање на систем

Назив СК
Пријављивање на систем

Актори СК
Администратор/корисник

Учесници СК
Администратор/корисник и систем (програм)

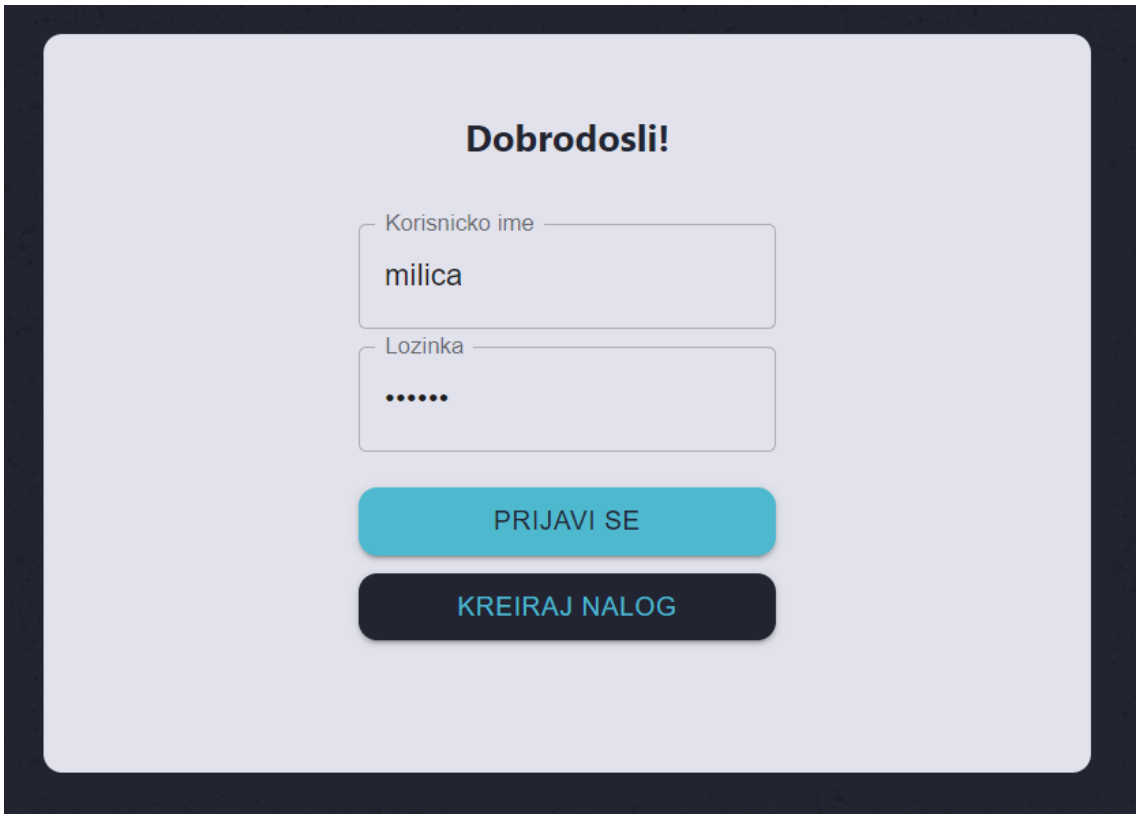
Предуслов: Систем је укључен и приказује форму за пријављивање.



Слика 35 Форма за пријаву на систем

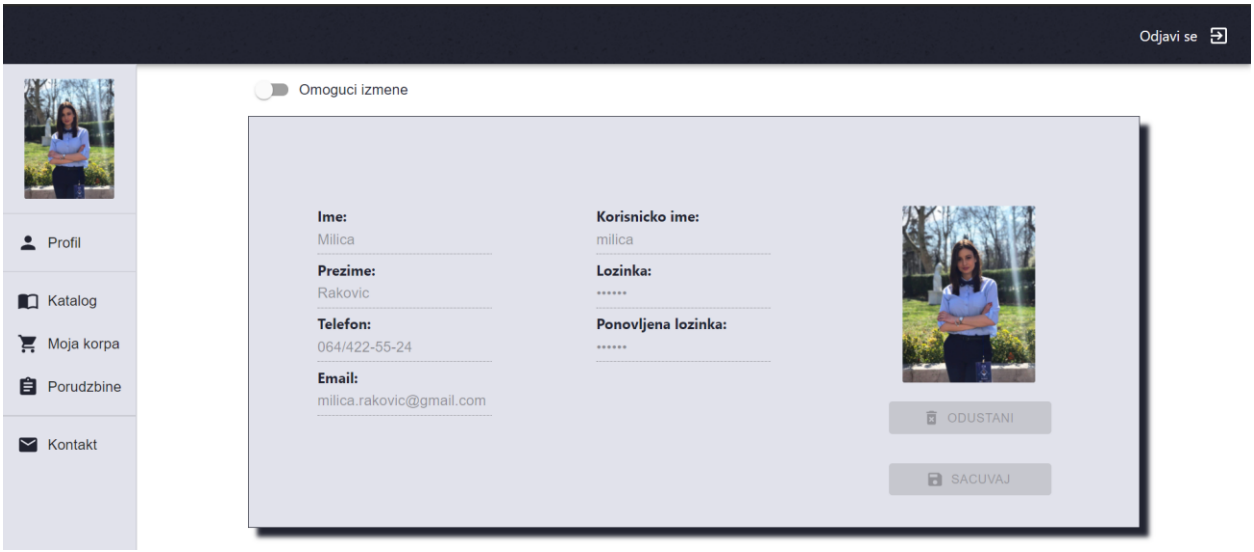
Основни сценарио СК

1. Администратор/корисник уноси податке за пријављивање. (АПУСО)



Слика 36 Попуњена форма за пријаву на систем

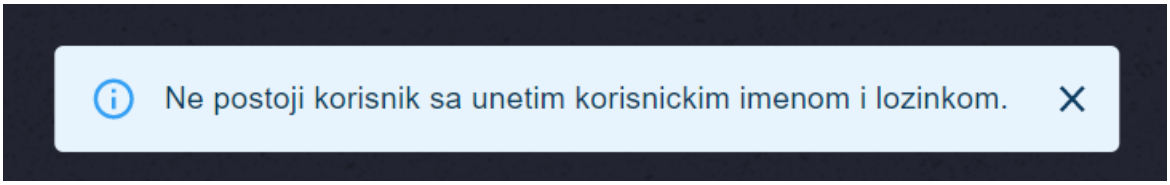
2. Корисник контролише да ли је коректно унео корисничко име и лозинку (АНСО)
3. Администратор/корисник позива систем да га пријави. (АПСО)
4. Систем проверава да ли су унети подаци у реду. (СО)
5. Систем приказује администратору/кориснику доступне опције. (ИА)



Слика 37 Изглед екранске форме након успешног пријављивања на систем

Алтернативна сценарија

5.1 Уколико систем не може да верификује администратора/корисника он приказује поруку: „Не постоји корисник са унетим корисницчим именом и лозинком. ”. (ИА)



Слика 38 Алтернативни сценарио приликом пријаве на систем уколико корисник не постоји

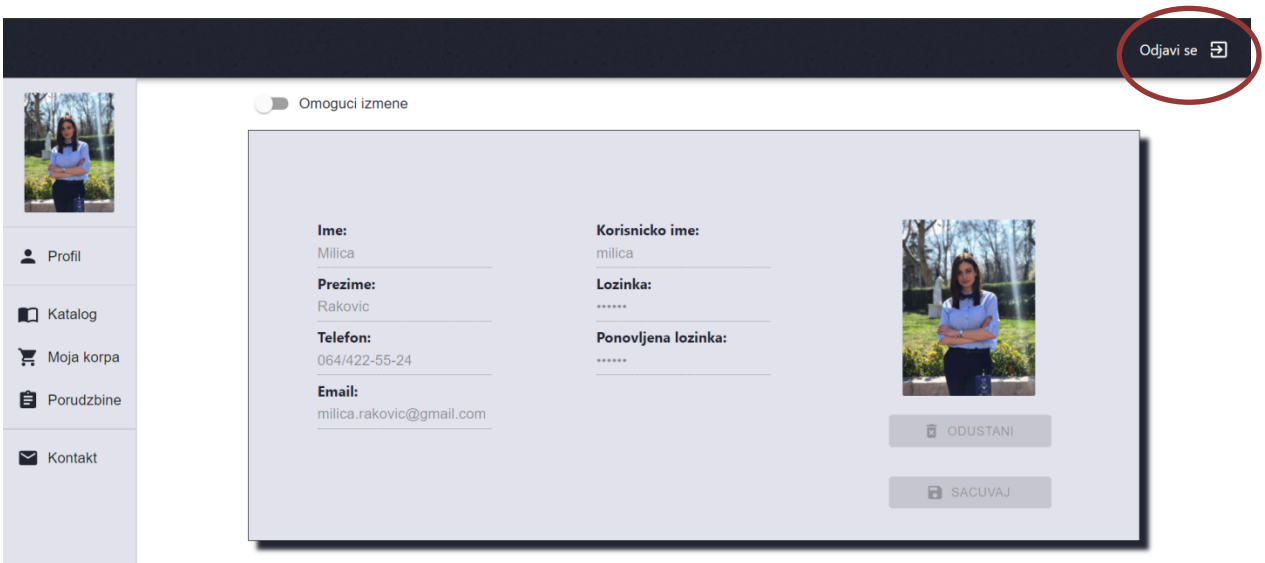
4.1.2 СК2: Случај коришћења - Одјава са система

Назив СК
Одјава са система

Актори СК
Администратор/корисник

Учесници СК
Администратор/корисник и систем (програм)

Предуслов: Систем је укључен и администратор/корисник је улогован под својом шифром. Систем приказује опцију за одјаву са система.



Слика 39 Основни сценарио одјаве са система

Основни сценарио СК

1. Администратор/корисник позива систем да га одјави. (АПСО)
2. Систем одјављује администратора/корисника са система и приказује почетну страну.(ИА)

Алтернативна сценарија
2.1 Уколико систем не може да одјави администратора/корисника са система: „Није могуће одјавити се са система.”. (ИА)

4.1.3 СК3: Случај коришћења - Креирање производа

Назив СК
Креирање производа

Актори СК
Администратор

Учесници СК
Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је улогован под својом шифром. Учитан је списак произвођача и списак типова поризвода. Систем приказује форму за рад са производом.

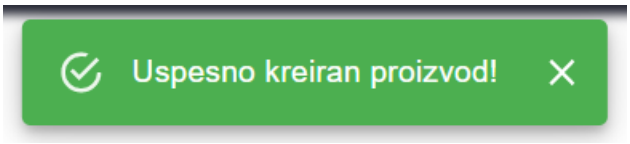
Слика 40 Форма за креирање производа

Основни сценарио СК

1. Администратор уноси податке о производу. (АПУСО)

Слика 41 Попуњена форма за креирање производа

2. Администратор контролише да ли је коректно унео податке о производу. (АНСО)
3. Администратор позива систем да креира нови производ са задатим подацима. (АПСО)
4. Систем креира производ са задатим подацима. (СО)
5. Систем приказује администратору поруку: „Производ је успешно креиран“. (ИА)



Слика 42 Успешно креиран производ

Алтернативна сценарија
5.1 Уколико систем не може да креира производ он приказује администратору поруку: „Није могуће креирати производ”. (ИА)

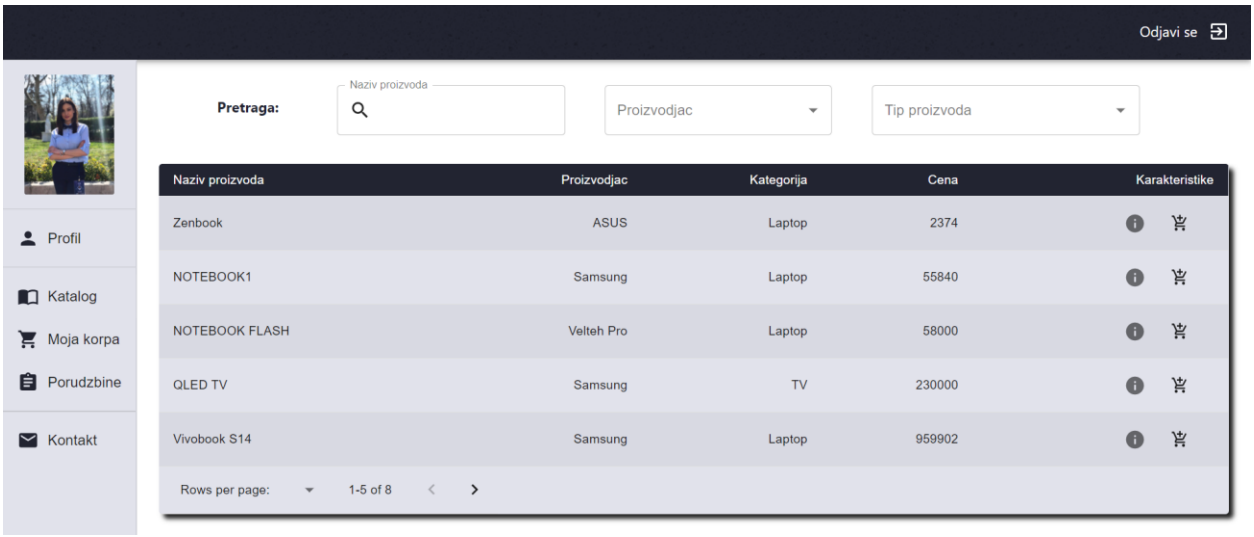
4.1.4 СК4: Случај коришћења - Претраживање производа

Назив СК
Претраживање производа

Актори СК
Администратор/корисник

Учесници СК
Администратор/корисник и систем (програм)

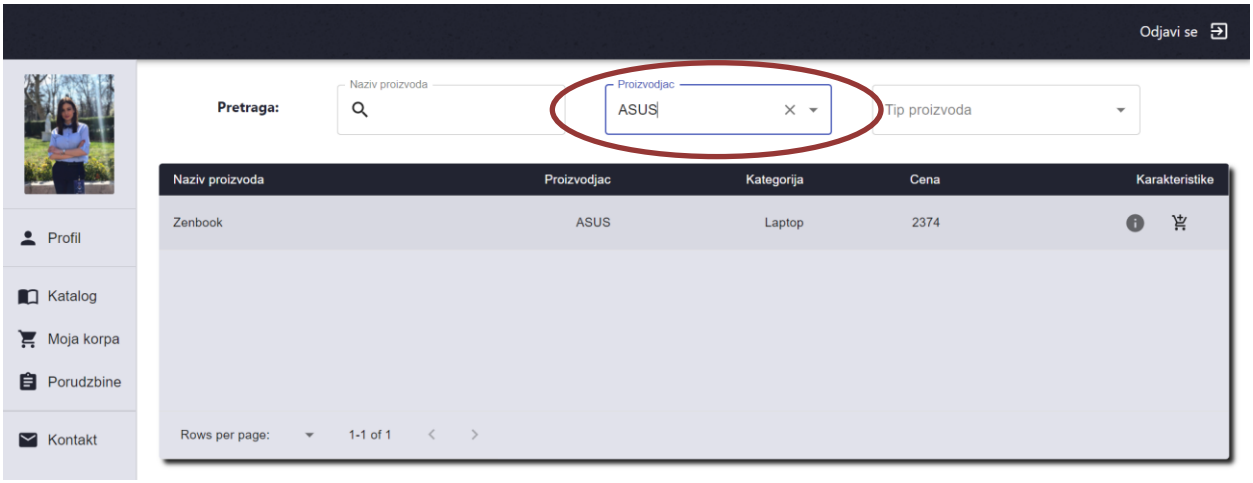
Предуслов: Систем је укључен и администратор/корисник је улогован под својом шифром. Систем приказује форму за рад са производима.



Слика 43 Форма за рад са производима

Основни сценарио СК

1. Администратор/корисник уноси критеријум по ком претражује производе. (АПУСО)



Слика 44 Претрага производа по критеријуму

2. Администратор/корисник позива систем да пронађе пориизводе на основу задате вредности. (АПСО)
3. Систем тражи производе по задатом критеријуму у учитава податке о њима. (СО)
4. Систем приказује кориснику пронађене производе. (ИА)

Алтернативна сценарија
4.1 Уколико систем не може да пронађе производе он приказује администратору/кориснику празну табелу производа. (ИА)

4.1.5 СК5: Случај коришћења - Брисање производа

Назив СК
Брисање производа

Актори СК
Администратор

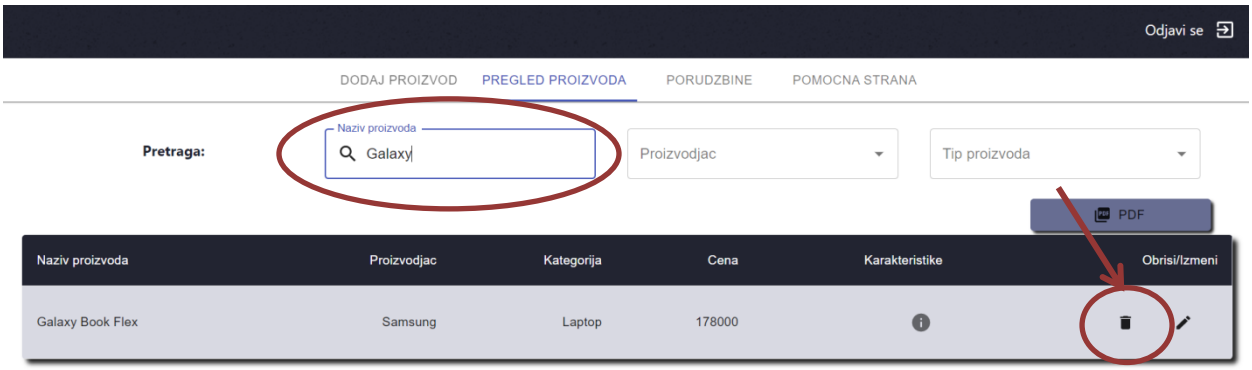
Учесници СК
Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је улогован под својом шифром. Систем приказује форму за рад са производом.

Основни сценарио СК

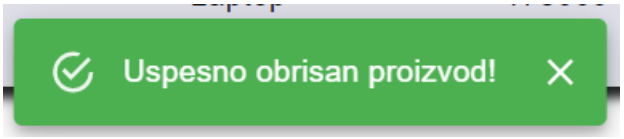
1. Администратор уноси критеријум по ком претражује производе. (АПУСО)
2. Администратор позива систем да пронађе производе на основу задате вредности. (АПСО)
3. Систем тражи производе по задатом критеријуму и учитава податке о њима. (СО)
4. Систем приказује кориснику пронађене производе. (ИА)

5. Администратор бира производ који жели да обрише. (АПУСО)
6. Администратор позива систем да обрише производ. (АПСО)



Слика 45 Форма за брисање производа на основу критеријума

7. Систем брише производ. (СО)
8. Систем приказује администратор поруку: „Успешно обрисан производ.” (ИА)



Слика 46 Успешно обрисан производ

Алтернативна сценарија
4.1 Уколико систем не може да пронађе производе он приказује администратору/кориснику празну табелу производа. Прекида се извршење сценариа.(ИА)
8.1 Уколико систем не може да обрише производ он приказује администратор поруку: „Није могуће обрисати производ”. (ИА)

4.1.6 СК6: Случај коришћења - Измена производа

Назив СК
Измена производа

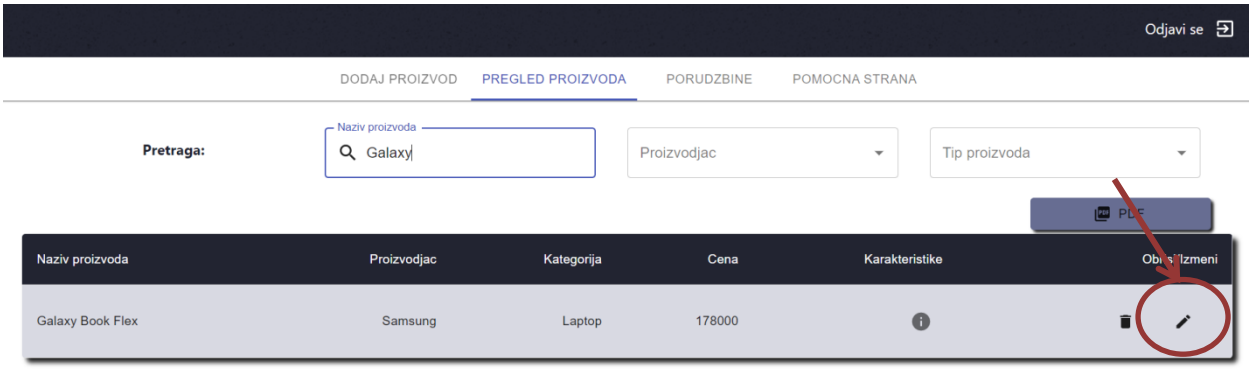
Актори СК
Администратор

Учесници СК
Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је улогован под својом шифром. Учитан је асортиман проивода, листа произвођача и листа типова производа. Систем приказује форму за рад са производом.

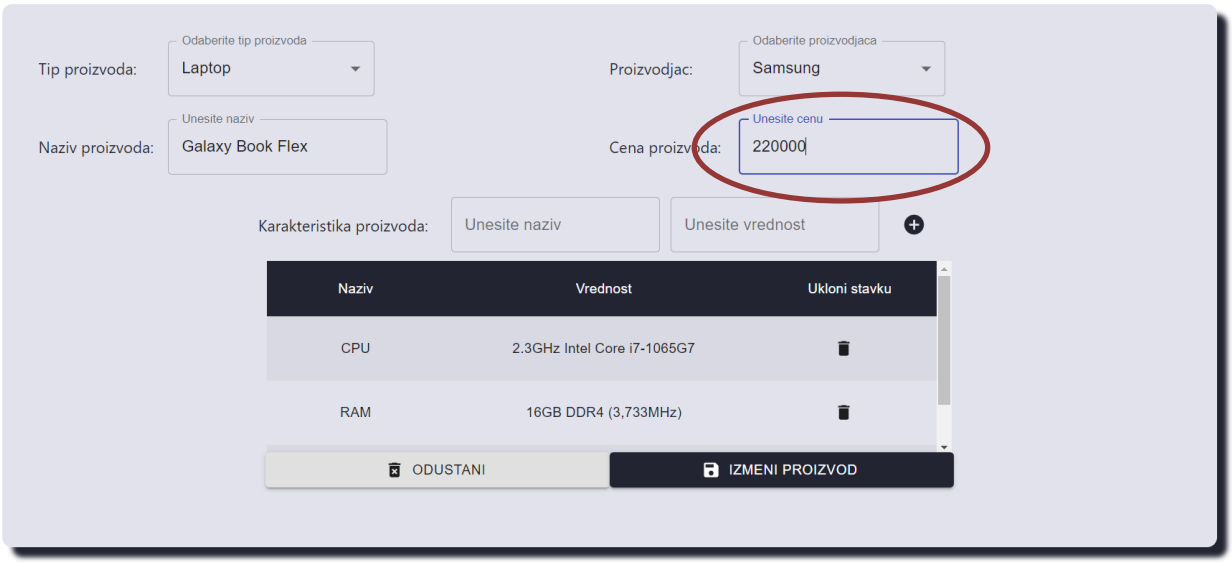
Основни сценарио СК

1. Администратор бира производ који жели да измени. (АПУСО)



Слика 47 Форма за измену производа на основу критеријума

2. Администратор мења податке о производу. (АПУСО)



Слика 48 Унос нових вредности за одабрани произвид

3. Администратор контролише да ли је коректно унео податке о производу. (АНСО)
4. Администратор позива систем да запамти податке о производу. (АПСО)
5. Систем памти податке о производу. (СО)
6. Систем приказује администратору поруку: „Успешно сте изменили производ.” (ИА)

Алтернативна сценарија
6.1 Уколико систем не може да запамти податке о производу он приказује администратору поруку „Није могуће изменити производ”. (ИА)

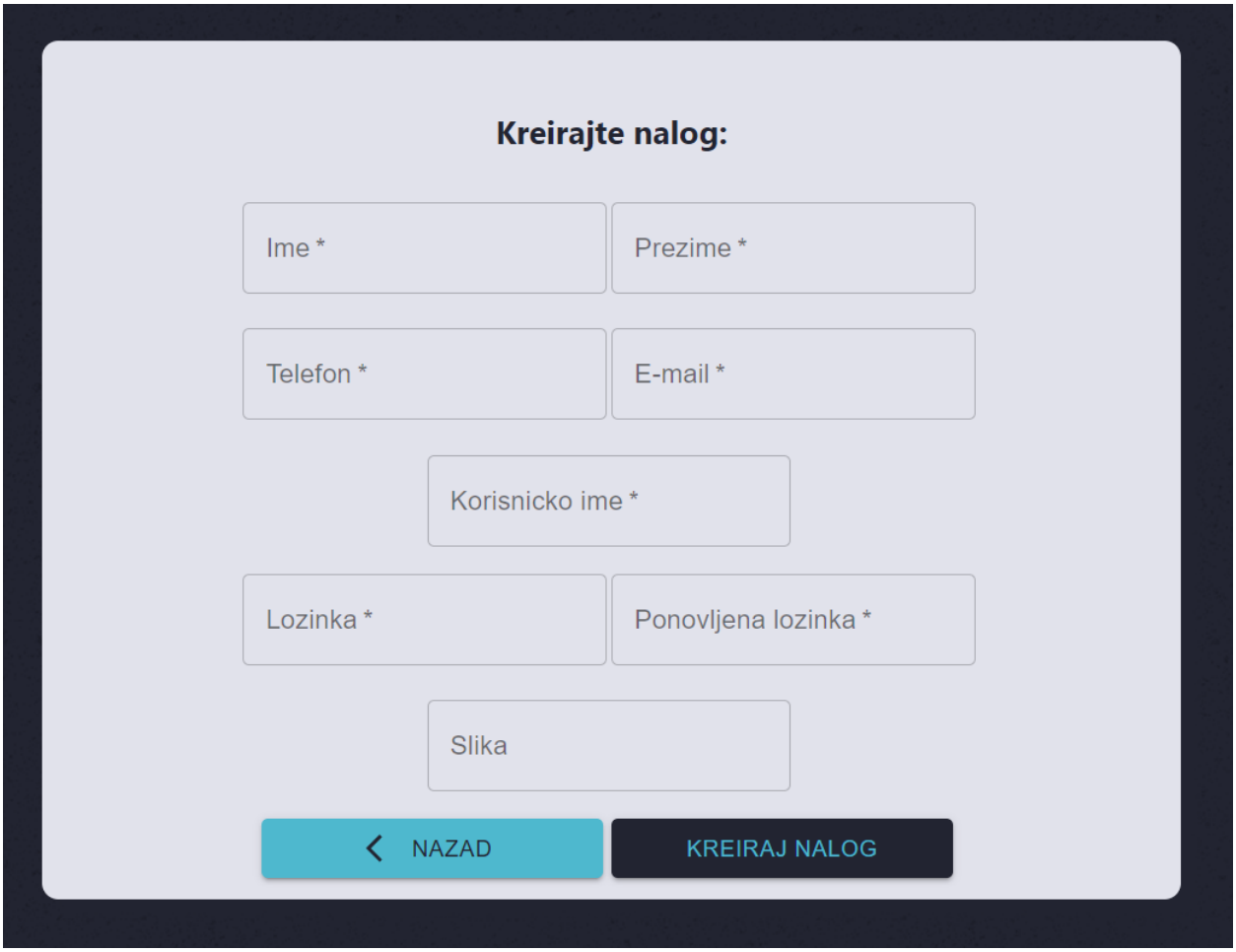
4.1.7 СК7: Случај коришћења - Креирање корисничког налога

Назив СК
Креирање корисничког налога

Актори СК
Корисник

Учесници СК
Корисник и систем (програм)

Предуслов: Систем је укључен. Систем приказује форму за креирање корисничког налога.



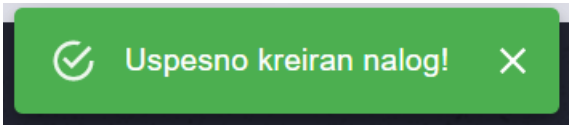
Слика 49Приказ форме за креирање корисничког налога

Основни сценарио СК

1. Посетилац уноси податке о новом кориснику. (АПУСО)

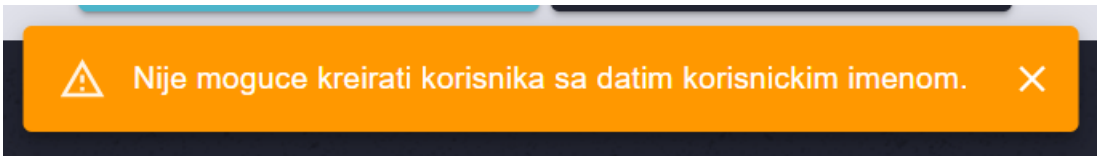
Слика 50 Приказ попуњене форме за креирање корисничког налога

2. Посетилац контролише да ли је коректно унео податке о кориснику. (АНСО)
3. Посетилац позива систем да креира новог корисника са задатим подацима. (АПСО)
4. Систем креира новог корисника. (СО)
5. Систем приказује посетиоцу поруку: „Успешно креиран налог“. (ИА)



Слика 51 Успешно креиран налог.

Алтернативна сценарија
5.1 Уколико систем не може да креира корисника он приказује постиоцу одговарајућу поруку. (ИА)



Слика 52 Алтернативни случај коришћења креирања корисничког налога

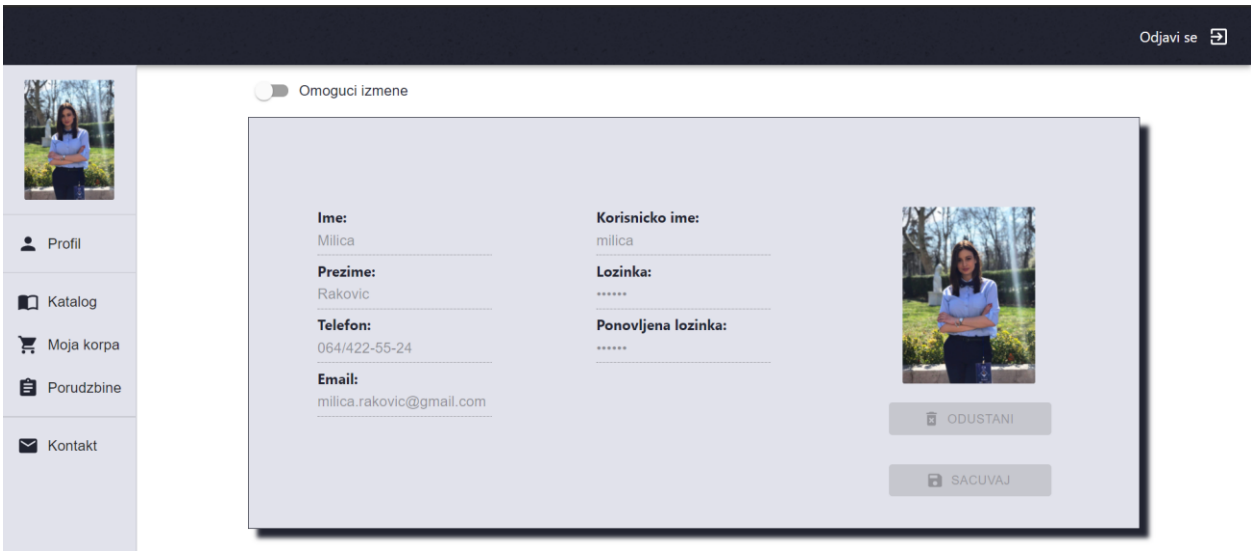
4.1.8 СК8: Случај коришћења - Измена корисничког налога

Назив СК
Измена корисничког налога

Актори СК
Корисник

Учесници СК
Корисници систем (програм)

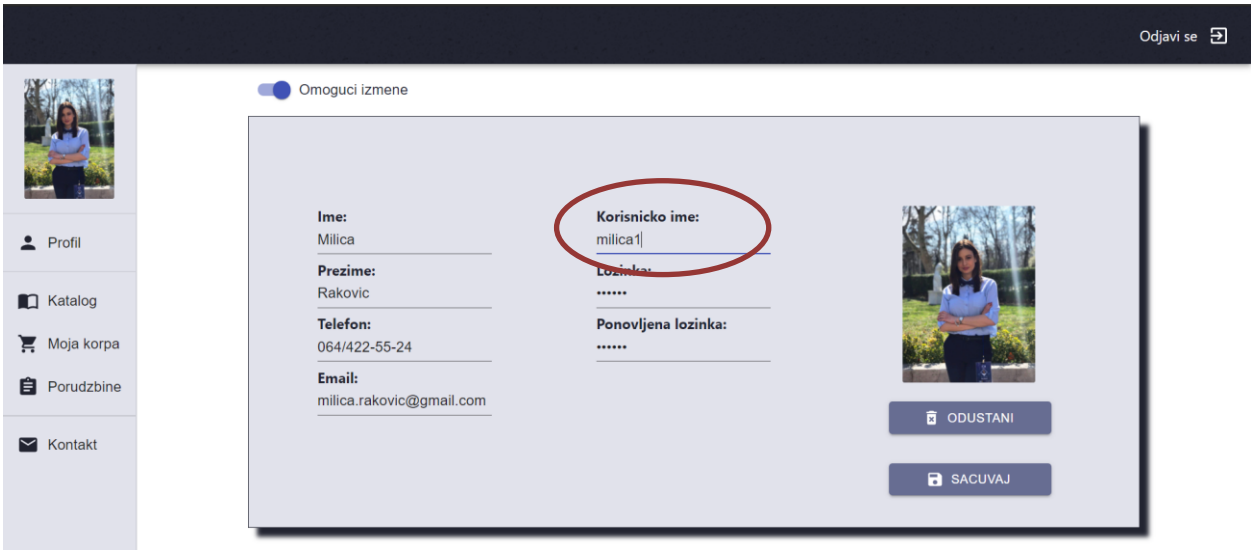
Предуслов: Систем је укључен и корисник је улогован под својом шифром. Учитани су подаци о кориснику. Систем приказује форму за рад са профилом корисника.



Слика 53 Приказ података о кориснику

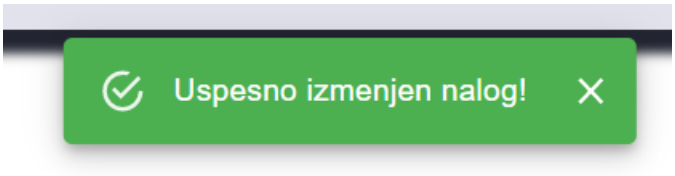
Основни сценарио СК

1. Систем тражи податке о кориснику. (СО)
2. Систем приказује кориснику пронађене податке. (ИА)
3. Корисник бира податке који жели да измени. (АПУСО)
4. Корисник мења податке. (АПУСО)



Слика 54 Унос нових података за корисника

5. Корисник контролише да ли је коректно унео податке. (АНСО)
6. Корисник позива систем да запамти податке. (АПСО)
7. Систем памти податке о кориснику. (СО)
9. Систем приказује администратору поруку: „Успешно измењен налог.” (ИА)



Слика 55 Успешно измењен налог.

Алтернативна сценарија
2.1 Уколико систем не може да нађе податке о кориснику он приказује кориснику поруку: „Нису пронађени подаци о кориснику”. Прекида се извршење сценариа. (ИА)
9.1 Уколико систем не може да запамти податке о кориснику он приказује кориснику поруку „Није могуће изменити податке”. (ИА)

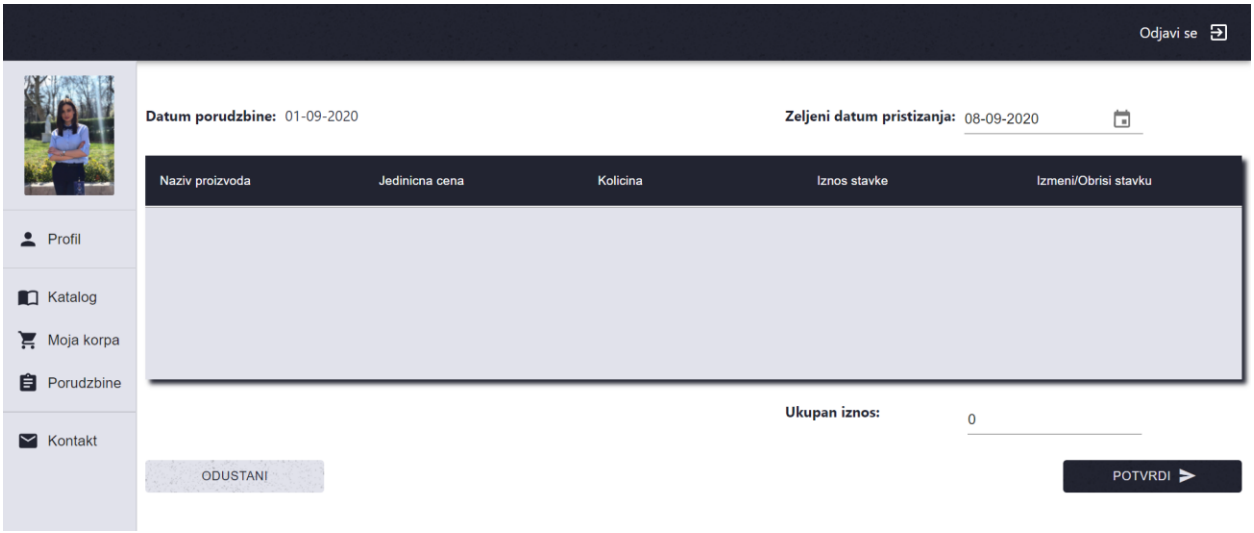
4.1.9 СК9: Случај коришћења - Креирање поруџбине

Назив СК
Креирање поруџбине

Актори СК
Корисник

Учесници СК
Корисник и систем (програм)

Предуслов: Систем је укључен и корисник је улогован под својом шифром. Учитан је асортиман проивода. Систем приказује форму за рад са поруџбином.



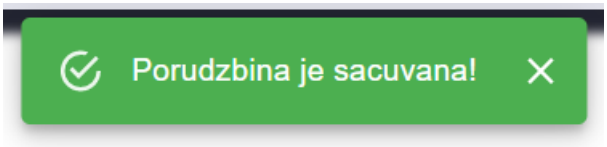
Слика 56 Форма за креирање поруџбине

Основни сценарио СК

1. Корисник уноси податке за нову поруџбину. (АПУСО)

Слика 57 Попуњена форма за креирање поруџбине

2. Корисник контролише да ли је коректно унео податке у нову поруџбину. (АНСО)
3. Корисник позива систем да креира нову поруџбину. (АПСО)
4. Систем креира поруджбину са задатим подацима. (СО)
5. Систем приказује кориснику поруку: „Поруџбина је сачувана”. (ИА)



Слика 58 Успешно сачувана поруџбина

Алтернативна сценарија
4.1 Уколико систем не може да креира нову поруџбину он приказује кориснику поруку: „Систем не може да креира нову поруџбину ”. (ИА)

4.1.10 СК10: Случај коришћења - Измена поруџбине

Назив СК
Измена поруџбине

Актори СК
Администратор

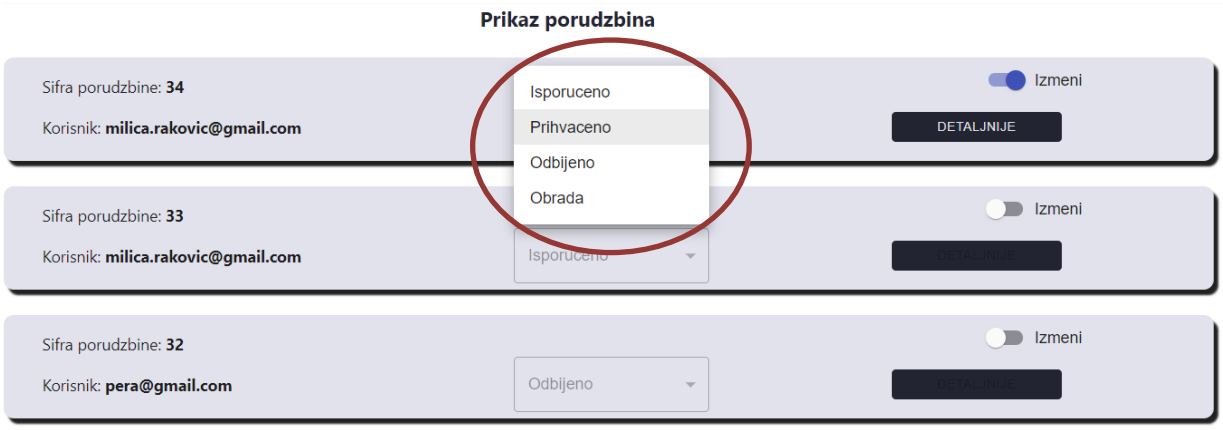
Учесници СК
Администратор и систем (програм)

Предуслов: Систем је укључен и администратор је улогован под својом шифром. Учитане су поруџбине и листа опција за измену статуса поруџбине. Систем приказује форму за рад са поруџбином.

Слика 59 Приказ форме за рад са поруџбинама

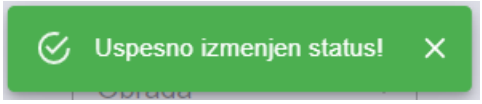
Основни сценарио СК

1. Администратор бира поруџбину који жели да измени. (АПУСО)
2. Администратор мења податке о поруџбини. (АПУСО)



Слика 60 Измена статуса поруџбине

3. Администратор контролише да ли је коректно унео податке о поруџбини. (АНСО)
4. Администратор позива систем да запамти податке о поруџбини. (АПСО)
5. Систем памти податке о поруџбини. (СО)
6. Систем приказује администратору одговарајућу поруку. (ИА)



Слика 61 Успешно измењен статус поруџбине

Алтернативна сценарија
6.1 Уколико систем не може да запамти податке о поруџбини он приказује администратору поруку „Није могуће изменити одабрани производ”. (ИА)

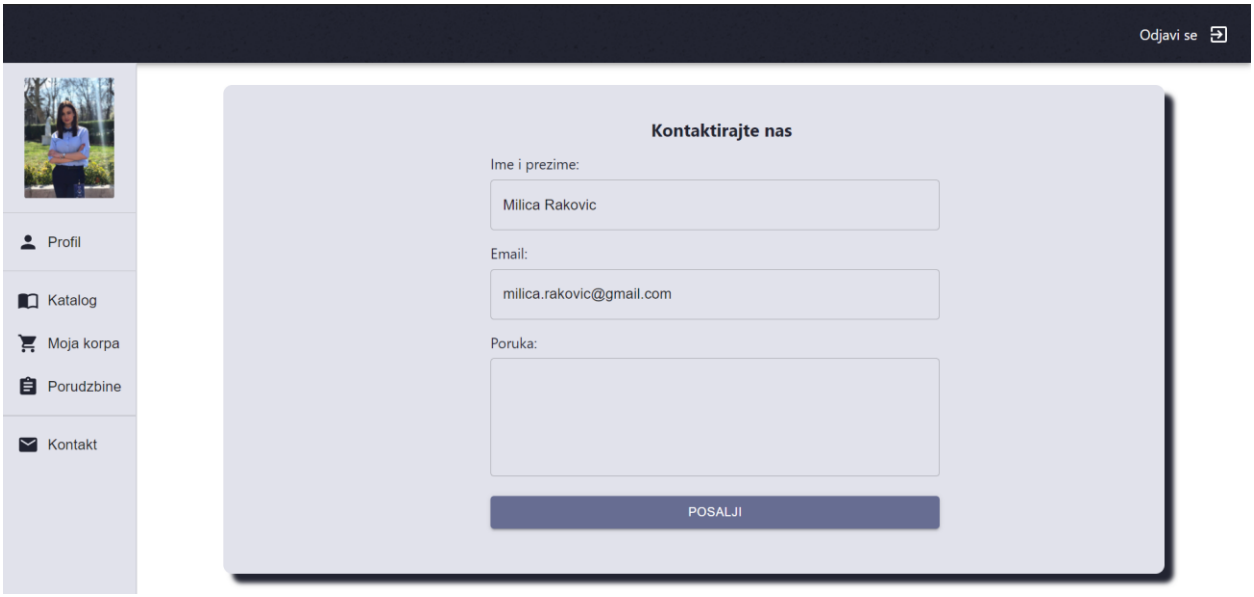
4.1.11 СК11: Случај коришћења - Котактирање администратора

Назив СК
Контактирање администратора

Актори СК
Корисник

Учесници СК
Корисник и систем (програм)

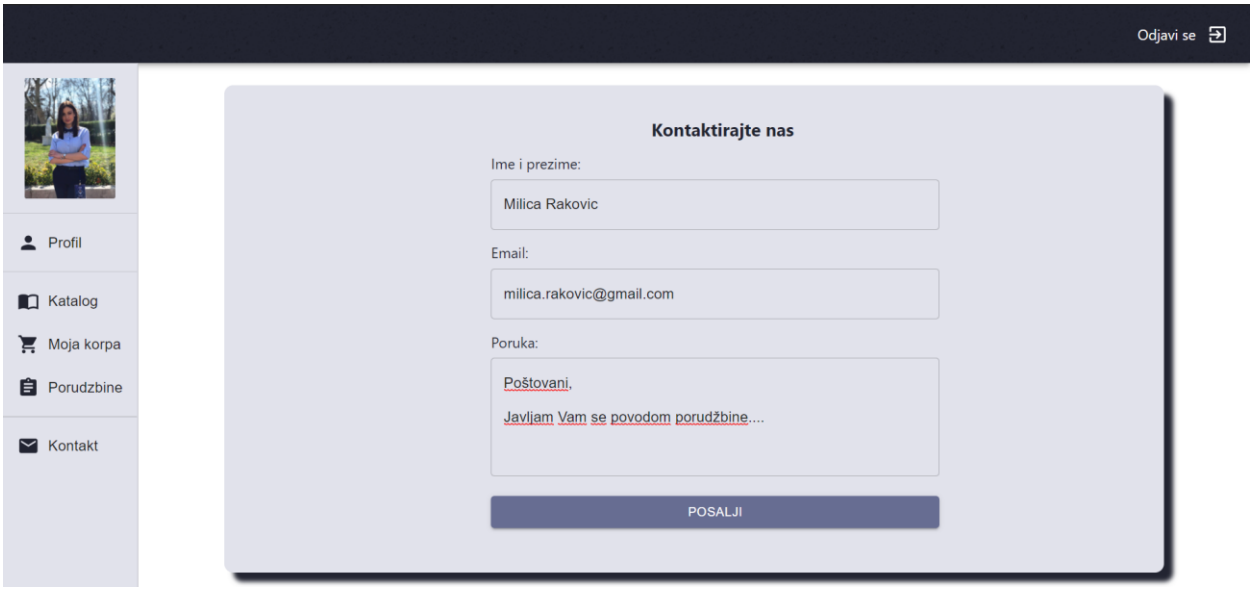
Предуслов: Систем је укључен и корисник је улогован под својом шифром. Систем приказује форму за контакт.



Слика 62 Форма за контактирање администратора

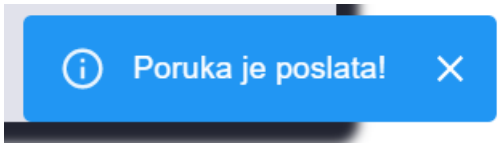
Основни сценарио СК

1. Корисник уноси податке у форми за контакт. (АПУСО)



Слика 63 Попуњена форма за контактирање администратора

2. Корисник контролише да ли је коректно унео податке у форму. (АНСО)
3. Корисник позива систем да пошаље поруку администратору. (АПСО)
4. Систем шаље поруку администратору. (СО)
5. Систем приказује кориснику поруку: „Порука је послата”. (ИА)



Слика 64 Успешно послата порука

Алтернативна сценарија
4.1 Уколико систем не може да пошаље поруку он приказује кориснику поруку: „Није могуће послати поруку ”. (ИА)

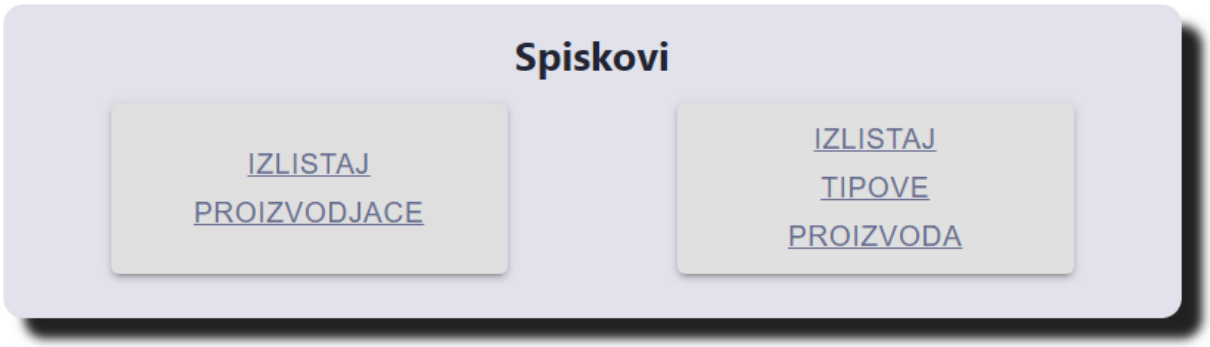
4.1.12 СК12: Случај коришћења - Генерисање *PDF* фајла

Назив СК
Генерисање *PDF* фајла

Актори СК
Администратор

Учесници СК
Администратор и систем (програм)

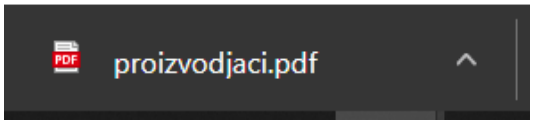
Предуслов: Систем је укључен и администратор је улогован под својом шифром. Учитан је списак произвођача и списак типова поризвода. Систем приказује форму за рад са производом.



Слика 65 Приказ екрана за рад са *PDF* фајловима

Основни сценарио СК

1. Администратор позива систем да генерише нови *PDF* фајл . (АПСО)
2. Систем генерише фајл. (СО)
3. Систем приказује администратору генерисан фајл. (ИА)



Слика 66 Успешно генерисан фајл

Алтернативна сценарија
3.1 Уколико систем не може да генерише фајл он приказује администратору поруку: „Није могуће генерисати фајл”. (ИА)

4.2 Пројектовање апликационе логике

4.3 Пословна логика

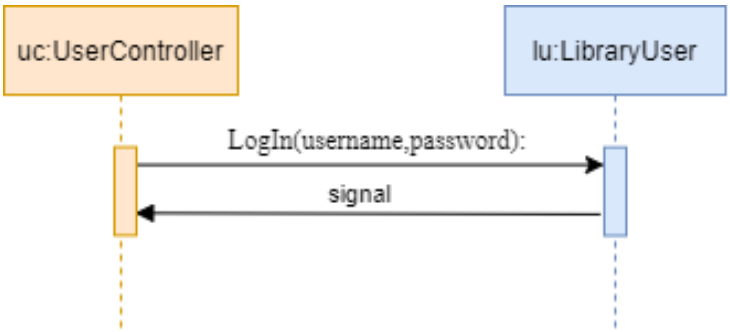
Пословна логика је описана са структуром (доменским класама) и понашањем (системским операцијама).

Пре извршења системске операције проверава се предуслов уколико постоји и отвара се трансакција. Уколико дође до изузетака приликом извршења системске операције, поништава се трансакција (*rollback*), у супротном се потврђује (*commit*).

За сваку системску операцију потребно је направити концептуална решења која су директно повезана са логичким проблемима. За сваку од ових уговора пројектује се концептуално решење.

4.3.1 Уговор УГ1: *LogIn*

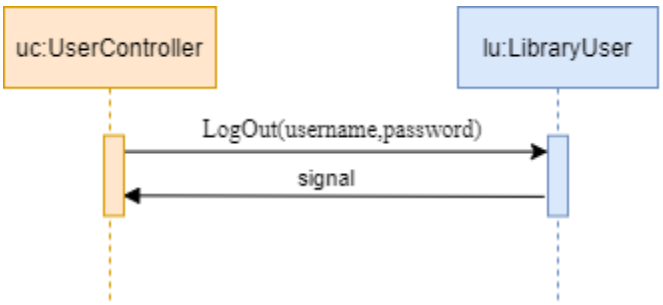
Операција: LogIn(username, password): signal;
Веза са СК: СК1
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *User*.
Постуслов: Корисник је пријављен на систем.



Слика 67 Пријава на систем

4.3.2 Уговор УГ2: *LogOut*

Операција: LogOut (username, password): signal;
Веза са СК: СК2
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *User*.
Постуслов: Корисник је одјављен са система.



Слика 68 Одјава са система

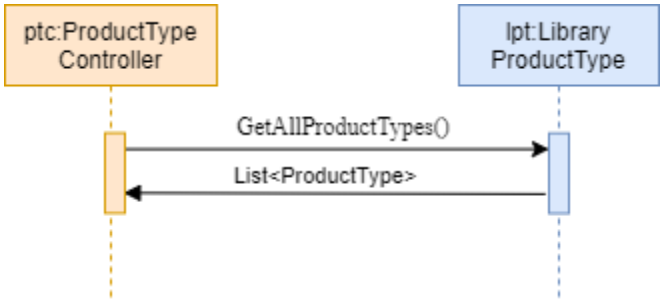
4.3.3 Уговор УГ3: *GetAllManufacturers*

Операција: GetAllManufacturers(): signal;
Веза са СК: СК3, СК6, СК12
Предуслов: /
Постуслов: /

Слика 69 Учитавање произвођача

4.3.4 Уговор УГ4: *GetAllProductTypes*

Операција: GetAllProductTypes(): signal;
Веза са СК: СК3, СК6, СК11
Предуслов: /
Постуслов: /



Слика 70 Учитавање типова производа

4.3.5 Уговор УГ5: *GetAllOrders*

Операција: GetAllOrders(): signal;
Веза са СК: СК10
Предуслов: /
Постуслов: /

Слика 71 Учитавање поруџбина

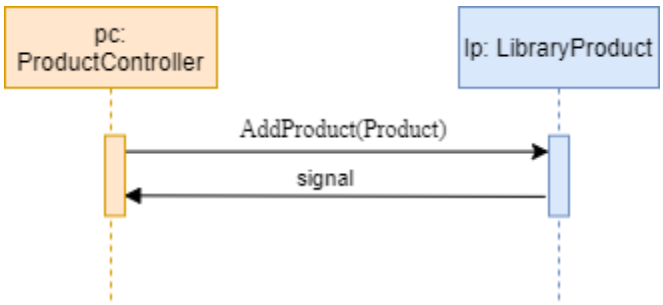
4.3.6 Уговор УГ6: *GetAllProducts*

Операција: GetAllProducts(): signal;
Веза са СК: СК6, СК9
Предуслов: /
Постуслов: /

Слика 72 Учитавање производа

4.3.7 Уговор УГ7: *AddProduct*

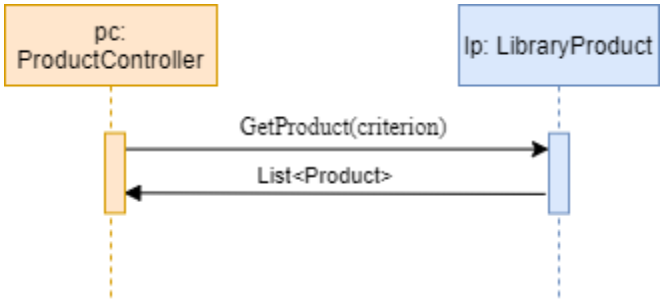
Операција: AddProduct(Product): signal;
Веза са СК: СК3
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *Product.*
Постуслов: Производ је креиран.



Слика 73 Додавање производа

4.3.8 Уговор УГ8: *GetProduct*

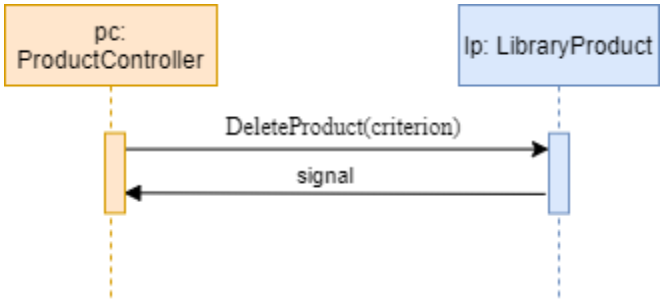
Операција: GetProduct (criterion): signal;
Веза са СК: СК4, СК5
Предуслов: /
Постуслов: /



Слика 74 Учитавање производа на основу критеријума

4.3.9 Уговор УГ9: *DeleteProduct*

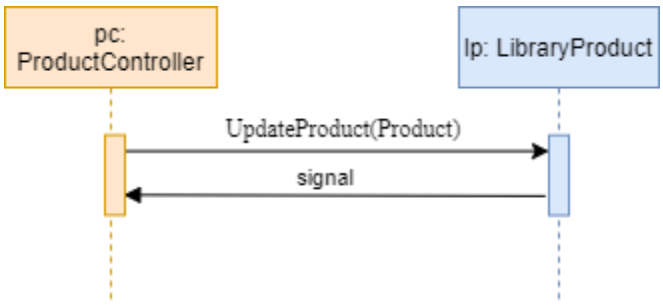
Операција: DeleteProduct (criterion): signal;
Веза са СК: СК5
Предуслов: Морају бити задовољена структурна ограничења над објектом *Product.*
Постуслов: Објекат је обрисан.



Слика 75 Брисање производа

4.3.10 Уговор УГ10: *UpdateProduct*

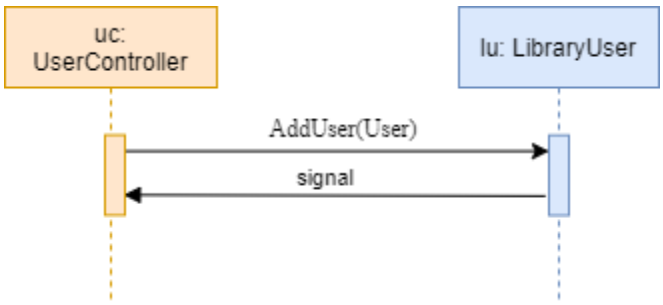
Операција: UpdateProduct(Product): signal;
Веза са СК: СК6
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *Product.*
Постуслов: Измењен производ је запамћен.



Слика 76 Измена производа

4.3.11 Уговор УГ11: *AddUser*

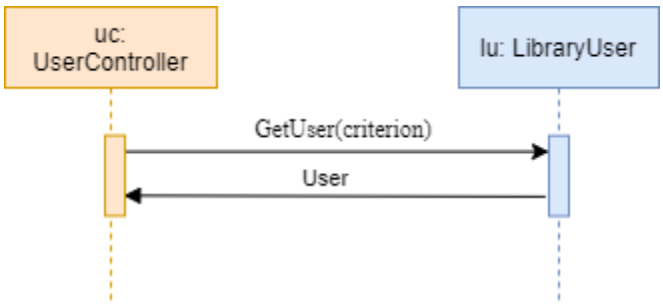
Операција: AddUser(User): signal;
Веза са СК: СК7
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *User.*
Постуслов: Корисник је креиран.



Слика 77 Додавање корисника

4.3.12 Уговор УГ12: *GetUser*

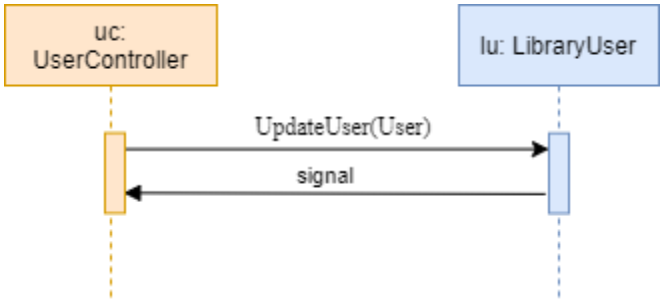
Операција: GetUser (criterion): signal;
Веза са СК: СК8
Предуслов: /
Постуслов: /



Слика 78 Учитавање корисника

4.3.13 Уговор УГ13: *UpdateUser*

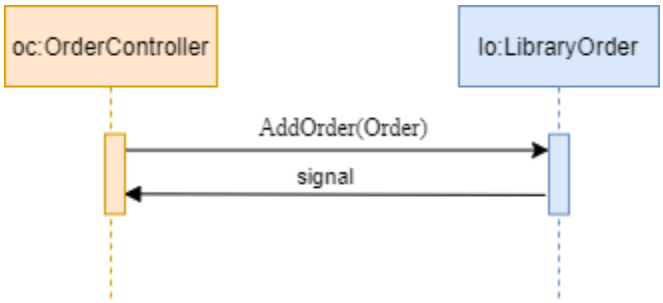
Операција: UpdateUser (User): signal;
Веза са СК: СК8
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *User*.
Постуслов: Измењен корисник је запамћен.



Слика 79 Измена корисника

4.3.14 Уговор УГ14: *AddOrder*

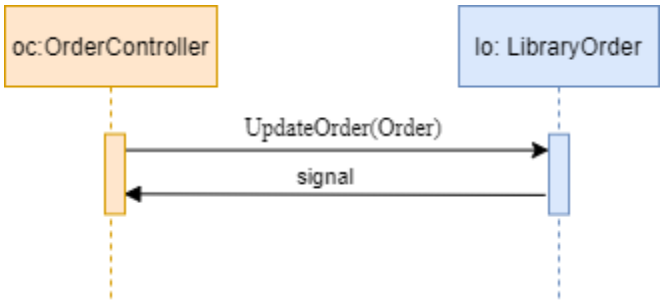
Операција: AddOrder (Order): signal;
Веза са СК: СК9
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *Order*.
Постуслов: Поруџбина је креирана.



Слика 80 Додавање поруџбине

4.3.15 Уговор УГ15: *UpdateOrder*

Операција: UpdateOrder(Order): signal;
Веза са СК: СК10
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *Order*.
Постуслов: Измењена је поруџбина.



Слика 81 Измена поруџбине

4.3.16 Уговор УГ16: *SendMessage*

Операција: SendMessage(message): signal;
Веза са СК: СК11
Предуслов: Морају бити задовољена структурна ограничења над објектом *message*.
Постуслов: Порука је послата.

4.3.17 Уговор УГ17: *GeneratePDF*

Операција: GeneratePDF (): signal;
Веза са СК: СК12
Предуслов: /
Постуслов: Фајл је генерисан.
OVO SE ODVIJA SAMO NA FRONTU. KAKO NACRTATI?

4.4 Пројектовање складишта података

На основу софтверских класа структуре пројектоване су табеле (складишта података) релационог система за управљање базом података:

DESKTOP-CHSMP58\...dbo.Manufacturer

Column Name	Data Type	Allow Nulls
ID	int	☐
Name	nvarchar(MAX)	☐
Phone	nvarchar(MAX)	☑
Email	nvarchar(MAX)	☑
Adress	nvarchar(MAX)	☑
		☐

Табела 8 *Manufacturer*

DESKTOP-CHSMP58\L...o.Characteristics

Column Name	Data Type	Allow Nulls
ID	int	☐
Name	nvarchar(MAX)	☐
Value	nvarchar(MAX)	☐
ProductID	int	☑
		☐

Табела 9 *Characteristics*

DESKTOP-CHSMP58\...mski - dbo.Order

Column Name	Data Type	Allow Nulls
ID	int	☐
DateOfMaking	datetime2(7)	☐
Deadine	datetime2(7)	☐
SumPrice	float	☐
UserID	int	☐
Status	nvarchar(MAX)	☑
		☐

Табела 10 *Order*

DESKTOP-CHSMP58\...i - dbo.OrderItem

Column Name	Data Type	Allow Nulls
ID	int	☐
Quantity	int	☐
OrderItemPrice	float	☐
ProductID	int	☑
OrderID	int	☑
		☐

Табела 11 *OrderItem*

DESKTOP-CHSMP58\...ski - dbo.Product

Column Name	Data Type	Allow Nulls
ID	int	☐
Name	nvarchar(MAX)	☐
Price	float	☐
ManufacturerID	int	☑
ProductTypeID	int	☑
		☐

Табела 12 *Product*

DESKTOP-CHSMP58\...dbo.ProductType

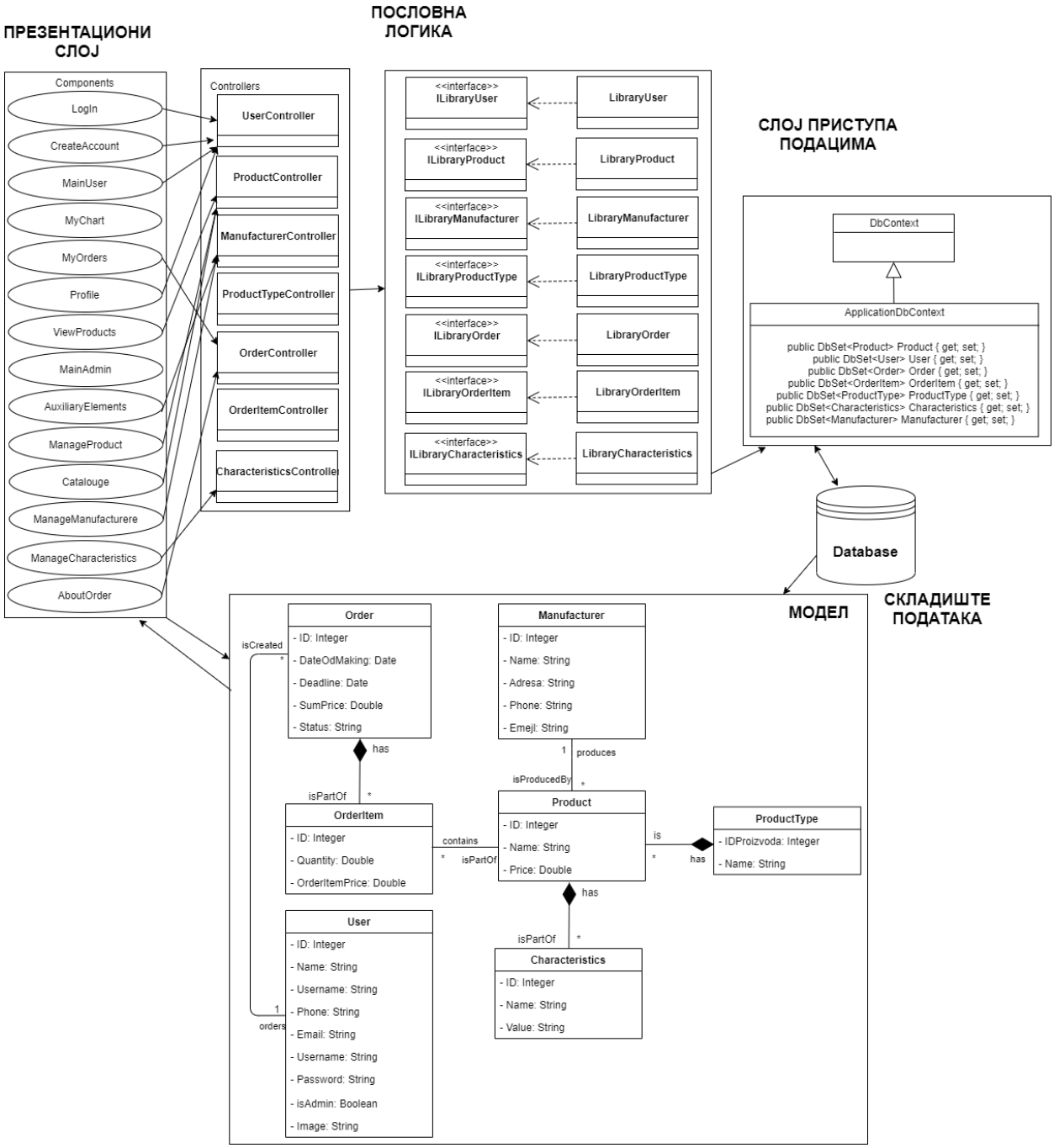
Column Name	Data Type	Allow Nulls
ID	int	☐
Name	nvarchar(MAX)	☑
		☐

Табела 13 *ProductType*

DESKTOP-CHSMP58\...lomski - dbo.User

Column Name	Data Type	Allow Nulls
ID	int	☐
Name	nvarchar(MAX)	☑
Surname	nvarchar(MAX)	☑
Phone	nvarchar(MAX)	☑
Email	nvarchar(MAX)	☑
Username	nvarchar(MAX)	☐
Password	nvarchar(MAX)	☐
IsAdmin	bit	☐
Image	nvarchar(MAX)	☑
		☐

Табела 14 *User*



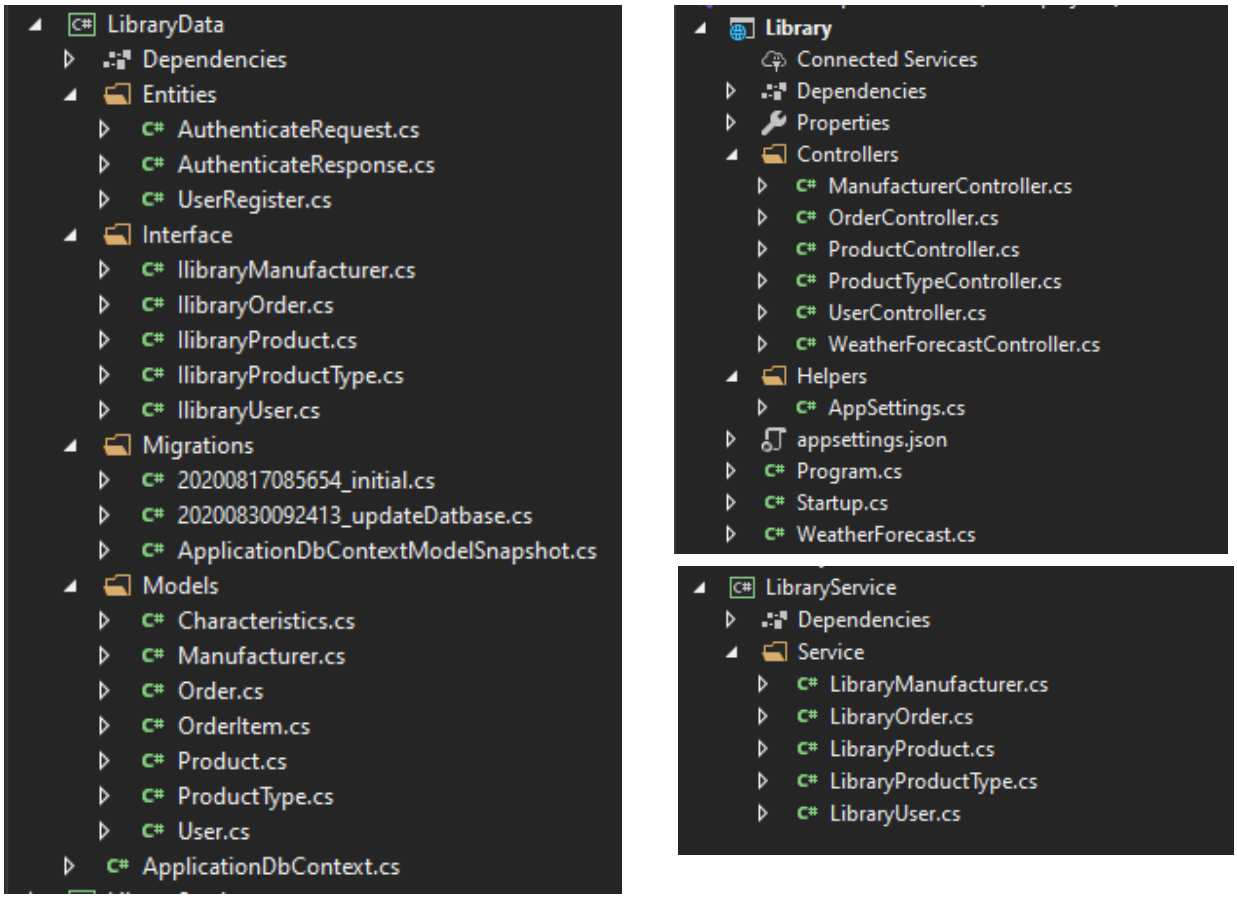
Слика 82 Коначна архитектура спфтверскпг система

5 Фаза имплементације

Софтверски систем је пројектован као једностранична, веб клијент-сервер апликација. Серверска страна развијена је у програмском језику C#. Коришћен је Visual Studio 2019 као развојно окружења саме апликације, а SQL Server Object Explorer као систем за управљање базом података. Клијентска страна изграђена је уз коришћење Typescrip програмског језика и *React* библиотеке у *Visual Studio Code* развојном окружењу.

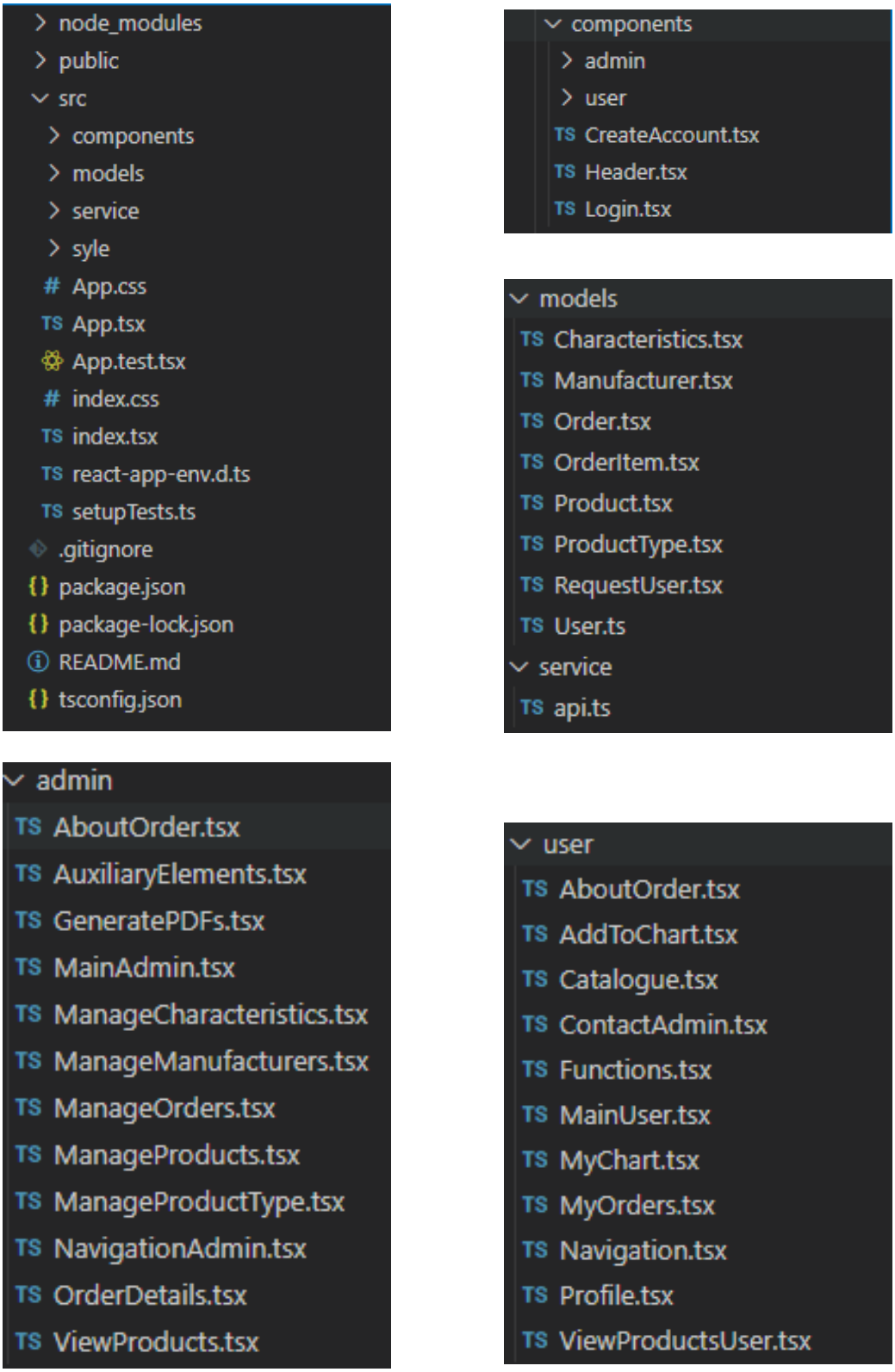
5.1 Структура софтверског система

На серверској страни имплементиране су следеће класе:



Слика 83 Приказ пројекта на серверској страни апликације

На клијентској страни класе које су имплементиране су приказане на слици испод:



Слика 84 Приказ пројекта на клијентској страни апликације

5.2 Имплементација апликационе логике

У оквиру поглављу биће приказана имплементација апликационе логике која је коришћена за изради овог софтверског система.

5.2.1 Комуникација са клијентом

Полазна тачка у комуникацији са клијентом јесте *Main* метода која се налази у *Program.cs* класи:

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateHostBuilder(args).Build().Run();
    }

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

Комуникација са клијентима остварује се преко *HTTP* захтева. Захтев који стигне од клијента прослеђује се до тзв. *Middleware*-а, а потом до одговарајућег контролера. Контролер потом обрађује захтев и позива функције неопходне за извршавање захтева.

У Startup класи у методи *ConfigureServices* дефинишу се потребни сервиси који су неопходни за сам софтверски систем. У наставку биће приказана имплементација једног таквог сервиса за аутентикацију клијента уз помоћ *JSON Web Token*-а.

```
[HttpPost("authenticate")]
        [AllowAnonymous]
        public async Task<ActionResult<AuthenticateResponse>>
Authenticate([FromBody]AuthenticateRequest request)
        {
            var user = _user.Authentificate(request.Username, request.Password);
            if (user is null) return NotFound();

            var tokenString = CreateToken(user);
            AuthenticateResponse response = CreateResponse(tokenString, user);
            return Ok(response);
        }

        private string CreateToken(User user)
        {
            var tokenHandler = new JwtSecurityTokenHandler();
            var key = Encoding.ASCII.GetBytes(_appSettings.Secret);
            var tokenDescriptor = new SecurityTokenDescriptor
            {
```

```
                Subject = new ClaimsIdentity(new Claim[]
                {
                    new Claim(ClaimTypes.Name, user.ID.ToString()),
                }),
                Expires = DateTime.UtcNow.AddDays(7),
                SigningCredentials = new SigningCredentials(new
SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
            };
            var token = tokenHandler.CreateToken(tokenDescriptor);
            var tokenString = tokenHandler.WriteToken(token);
            return tokenString;
        }
```

5.2.2 Пословна логика

Пословна логика је описана са структуром и понашањем[12]. Структуру чине доменске класе, а понашање софтверског система чине системске операције. У наставку дат је пример једне доменске класе, а потом и системске операције:

```
    public class ProductType
    {
        public int ID { get; set; }
        public string Name { get; set; }
    }
```

Уговор УГ1: LogIn
Операција: LogIn(username, password): signal;
Веза са СК: СК1
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *User*.
Постуслов: Корисник је пријављен на систем.

```
public User Authentificate(string username, string password)
        {
            try
            {
                var user = _db.User.SingleOrDefault((u) => u.Username.Equals(username) &&
u.Password == password);
                if (user is null) return null;


                return user;
            }
            catch (Exception)
            {

                throw;
            }
        }
```

[12] скрипта

Уговор УГ2: GetUser
Операција: GetUser (criterion): signal;
Веза са СК: СК8
Предуслов: /
Постуслов: /

```
public IEnumerable<User> GetUser(int UserID)
        {
            try
            {
                return _db.User.Where(u => u.ID == UserID);
            }
            catch (Exception)
            {
                return null;
            }
        }
```

Уговор УГ3: GetAllManufacturers
Операција: GetAllManufacturers(): signal;
Веза са СК: СК3, СК6, СК12
Предуслов: /
Постуслов: /

```
        public IEnumerable<Manufacturer> GetAllManufacturer()
        {
            try
            {
                return _db.Manufacturer;
            }
            catch
            {
                return null;
            }
        }
```

Уговор УГ4: GetAllProductTypes
Операција: GetAllProductTypes(): signal;
Веза са СК: СК3, СК6, СК11
Предуслов: /
Постуслов: /

```
        public IEnumerable<ProductType> GetAllProductType()
        {
            try
            {
                return _db.ProductType;
            }
            catch
            {
                return null;
            }
        }
```

Уговор УГ5: GetAllOrder
Операција: GetAllOrder(): signal;
Веза са СК: СК10
Предуслов: /
Постуслов: /

```
public IEnumerable<Order> GetAllOrder()
{
    try
    {
        return _db.Order
            .Include(u => u.User)
            .Include(i => i.OrderItems).ThenInclude(p => p.Product);
    }
    catch (Exception)
    {
        return null;
    }
}
```

Уговор УГ6: GetAllProducts
Операција: GetAllProducts(): signal;
Веза са СК: СК6, СК9
Предуслов: /
Постуслов: /

```
public IEnumerable<Product> GetAllProduct()
{
    try
    {
        return _db.Product.Include(m => m.Manufacturer).Include(t =>
t.ProductType).Include(c=>c.Characteristics);
    }
    catch (Exception)
    {

        return null;
    }
}
```

Уговор УГ7: AddProduct
Операција: AddProduct(Product): signal;
Веза са СК: СК3
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *Product*.
Постуслов: Производ је креиран.

```
public bool AddProduct(Product product)
{
    try
    {
```

```
                var productType = _db.ProductType.SingleOrDefault(t => t.ID ==
product.ProductType.ID);
                var manufacturer = _db.Manufacturer.SingleOrDefault(m => m.ID ==
product.Manufacturer.ID);

                if (productType == null || manufacturer == null) return false;

                product.ProductType = productType;
                product.Manufacturer = manufacturer;
                _db.Add(product);
                _db.SaveChanges();
                return true;
            }
            catch (Exception)
            {
                return false;
            }
        }
```

Уговор УГ8: GetProduct
Операција: GetProduct (criterion): signal;
Веза са СК: СК4, СК5
Предуслов: /
Постуслов: /

```
        public Product GetProduct(int ProductID)
        {
            try
            {
               return
_db.Product.Include(m=>m.Manufacturer).Include(t=>t.ProductType).Include(c =>
c.Characteristics).SingleOrDefault(x => x.ID == ProductID);

            }
            catch (Exception)
            {
                return null;
            }
        }
```

Уговор УГ9: DeleteProduct
Операција: DeleteProduct (criterion): signal;
Веза са СК: СК5
Предуслов: Морају бити задовољена структурна ограничења над објектом *Product*.
Постуслов: Објекат је обрисан.

```
        public bool DeleteProduct(int ProductID)
        {
            try
            {
                var product = GetProduct(ProductID);

                if (product == null) return false;
                bool deleted = DeleteCharacteristicsForProduct(product.Characteristics);
```

```
            if (deleted)
            {
                _db.Entry(product).State = EntityState.Deleted;
                _db.SaveChanges();
                return true;
            }
            return false;
        }
        catch(Exception e)
        {
            return false;
        }
    }
```

Уговор УГ10: UpdateProduct
Операција: UpdateProduct(Product): signal;
Веза са СК: СК6
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *Product*.
Постуслов: Измењен производ је запамћен.

```
    public bool UpdateProduct(int ProductID, Product product)
    {
        try
        {
            var manufacturer = _db.Manufacturer.SingleOrDefault(m => m.ID ==
product.Manufacturer.ID);
            var type = _db.ProductType.SingleOrDefault(t => t.ID ==
product.ProductType.ID);

            if (manufacturer == null || type == null) return false;

            var pr = _db.Product
                .Include(m => m.Manufacturer)
                .Include(t => t.ProductType)
                .Include(c=>c.Characteristics)
                .SingleOrDefault(p => p.ID == ProductID);

            if (pr == null) return false;

            pr.Manufacturer = manufacturer;
            pr.ProductType = type;
            pr.Name = product.Name;
            pr.Price = product.Price;
            var oldChar = pr.Characteristics;
            pr.Characteristics = ManageCharacteristics(oldChar,
product.Characteristics);
            _db.Entry(pr).CurrentValues.SetValues(product);
            _db.SaveChanges();
            return true;            }
        catch (Exception)
        {
            return false;
        }
    }
```

Уговор УГ11: AddUser
Операција: AddUser(User): signal;
Веза са СК: СК7
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *User*.
Постуслов: Корисник је креиран.

```
public bool AddUser(User user)
{
    try
    {
        if (IsThereAUser(user.Username) == true) return false;
        _db.Add(user);
        _db.SaveChanges();
        return true;
    }
    catch (Exception)
    {

        return false;
    }
}
```

Уговор УГ12: UpdateUser
Операција: UpdateUser (User): signal;
Веза са СК: СК8
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *User*.
Постуслов: Измењен корисник је запамћен.

```
public bool UpdateUser(User updatedUser)
{
    try
    {
        var user = _db.User.SingleOrDefault(u => u.ID == updatedUser.ID);
        if (user == null) return false;

        _db.Entry(user).CurrentValues.SetValues(updatedUser);
        _db.SaveChanges();
    }
    catch (Exception)
    {
        return false;
    }
    return true;
}
```

Уговор УГ13: AddOrder
Операција: AddOrder (Order): signal;
Веза са СК: СК9
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *Order*.

Постуслов: Поруџбина је креирана.

```
public bool AddOrder(Order order)
{
    try
    {
        var user = _db.User.SingleOrDefault(u => u.ID == order.User.ID);

        if (user == null) return false;
        var items = AddOrderItems(order.OrderItems);

        order.User = user;
        order.OrderItems = items;

        _db.Add(order);
        _db.SaveChanges();
        return true;
    }
    catch (Exception)
    {
        return false;
    }
}
```

Уговор УГ15: UpdateOrder
Операција: UpdateOrder(Order): signal;
Веза са СК: СК10
Предуслов: Морају бити задовољена структурна и вредносна ограничења над објектом *Order*.
Постуслов: Измењена је поруџбина.

```
public bool UpdateOrder(int OrderID, Order order)
{
    try
    {
        var user = _db.User.SingleOrDefault(u => u.ID == order.User.ID);

        var oldOrder = GetOrder(OrderID);
        if (oldOrder == null || user == null) return false;



        _db.Entry(oldOrder).CurrentValues.SetValues(order);
        _db.SaveChanges();
        return true;

    }
    catch (Exception e)
    {
        return false;
    }
}
```

За сваку доменску класу имплементирана је сервис класа и интерфејс где се налазе системске операције. Сваки сервис референцира одговарајући објекат класе и на тај начин се остварује комуникација са базом података. Пример једног сервиса и интерфејса дат је у наставку, а обрађен је за случај рада са типовима производа тј. за рад са класом *ProductType* приказаном раније.

```
public interface IlibraryProductType
{
  IEnumerable<ProductType> GetAllProductType();
  bool AddProductType(ProductType type);
  bool DeleteProductType(int ProductTypeID);
}

public class LibraryProductType : IlibraryProductType
    {
        private ApplicationDbContext _db;

        public LibraryProductType(ApplicationDbContext db)
        {
            _db = db;
        }

        public bool AddProductType(ProductType type)
        {
            try
            {
                _db.ProductType.Add(type);
                _db.SaveChanges();
                return true;
            }
            catch
            {
                return false;
            }
        }

        public bool DeleteProductType(int ProductTypeID)
        {
            try
            {
                var type = _db.ProductType.Where(t => t.ID == ProductTypeID);
                if (type == null) return false;

                _db.Entry(type).State = EntityState.Deleted;
                _db.SaveChanges();
                return true;
            }
            catch
            {
                return false;
            }
        }

        public IEnumerable<ProductType> GetAllProductType()
        {
            try
```

```
            {
                return _db.ProductType;
            }
            catch
            {
                return null;
            }
        }
    }
```

5.2.3 Имплементација слоја приступа подацима

Како би се омогућио рад са базом података, неопходно је направити модел који мапира ентитете и односе дефинисане у моделу у табеле базе. Класа која се користи за интеракцију са базом је *DbContext*. Начин рада са контекстом налаже да се креира класа која је наслеђује, у овом случају *ApplicationDbContext*

```
    public class ApplicationDbContext : DbContext
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) :
base(options)
        {

        }
        public DbSet<Product> Product { get; set; }
        public DbSet<User> User { get; set; }
        public DbSet<Order> Order { get; set; }
        public DbSet<OrderItem> OrderItem { get; set; }
        public DbSet<ProductType> ProductType { get; set; }
        public DbSet<Characteristics> Characteristics { get; set; }
        public DbSet<Manufacturer> Manufacturer { get; set; }
    }
}
```

Поред контекста, дефинише се и конекциони стринг у оквиру *appsettings.json* фајла:

```
  "ConnectionStrings": {
    "DefaultConnection": "Server = (localdb)\\MSSQLLocalDB; Database = Diplomski;
Integrated Security = True; Trusted_connection = True "
  }
```

5.2.4 Имплементација презентационог слоја

Уз помоћ презентационог слоја остварује се интеракција између корисника и система. Презентациони слој се састоји из корисничког интерфејса и презентационе логике. У данашњем времену све се више пажње обраћа на кориснички интерфејс и целокупно искуство корисника (*User Interface & User Experience*). Преко корисничког интерфејса, корисник задаје акције које треба да се изврше па је од велике важности да он буде интуитиван и лак за употребу.

У наредном делу следи приказ дела имплементације корисничког интерфејса коришћењем *React* библиотеке и *TypeScript* програмског језика:

```
interface Props {
  manufacturers: Manufacturer[];
  productTypes: ProductType[];
  onAddManufacturer: (manufacturer: Manufacturer) => any;
}

export default function AuxiliaryElements(props: Props) {
  const classes = useStylesAuxiliary();
  return (
    <Grid className={classes.root}>
      <Grid item xs={5}>
        <ManageManufacturers
          manufacturers={props.manufacturers}
          onAddManufacturer={props.onAddManufacturer}
        />
      </Grid>
      <Grid item xs={1}></Grid>
      <Grid item xs={5}>
        <ManageProductType />
        <GeneratePDFs
          manufacturers={props.manufacturers}
          productTypes={props.productTypes}
        />
      </Grid>
    </Grid>
  );
}
```

Компонента *ManageManufacturers*, позвана у примеру изнад, служи за манипулацију подацима за произвођаче. Преко интерфејса, њој се прослеђује метода *AddManufacturer* која извршава *HTTP Post* захтев ка серверу.

```
export async function AddManufacturer(manufacturer: Manufacturer) {
  const res = await fetch(baseUrl + `/manufacturer`, {
    method: 'POST',
    body: JSON.stringify(manufacturer),
    headers: {
      'Content-Type': 'application/json',
    },
  });
  return await res.json();
}
```

У контролеру апликационе логике врши се пријем захтева, након чега се захтев шаље сервису на обраду. С обзиром да је у коду изнад приказана метода за додавање новог произвођача, следи пример пријема тог захтева:

```
// POST: api/Manufacturer
[HttpPost]
public ActionResult Post([FromBody] Manufacturer manufacturer)
{
    if (_manufacturer.AddManufacturer(manufacturer)) return Ok();
    return NotFound();
}
```

6 Фаза тестирања

Након имплементације уследило је тестирање сваког од случајева коришћења. Тестирање је обављено тако што су уношени валидни и невалидни подаци. Утврђене грешке или слабије функционалности су исправљене како би софтверски систем што боље функционисао.

OPIS FAZE

7 Закључак

8 Литература

Влајић, С. (2015). Пројектовање софтвера (Скрипта). Београд