Универзитет у Београду

Факултет организационих наука

ЗАВРШНИ РАД

Тема: Развој софтверског система за подсистем набавке књига у књижари употребом ASP.NET Core оквира

Ментор:

проф. др Саша Лазаревић

Студент:

Алекса Милетић 2017/0159

Београд, 2021. године

Садржај

Списак слика:

Списак табела:

1 УВОД

ASP.NET (Active Server Pages .NET) је оквир који је намењен за прављење динамичких веб страница, апликација и сервиса. Ова технологија је креирана од старне компаније Microsoft, која је прву верзију објавила 2002. године.

Убрзаним развојем интернета веб апликације постају све значајније за маркетинг и пословање компаније. Интернет технологије и уопште технологије које омогућавају приступ и изградњу интернет страница су широко распорстрањене. Веб апликације користе комбинацију серверске скрипте (ASP) да раде са базом и да врате одређене информације и клијентске скрипте (JavaScript и HTML) да би представиле информације кориснику. Ово омогућава корисницима да буду у интеракцији са компанијом користећи онлајн форме, корпу за куповину у онлајн продавници, управљање садржајем и многе друге ствари.

У наставку биће приказан развој подсистема за набавку књига, али и кориснички део управљања садржајем. У 2020. години, због пандемије вируса Covid-19, велики број фирми је прешао на онлајн режим рада. Неке делатности су доживеле успон, као на пример достава хране, онлајн куповина ствари итд., а људи су постали слободнији и више упознати са могућностима које интернет пружа. Оваква апликација може помоћи како корисницима, тако и компанијама да лакше спроводе своје пословне активности и уштеде време и ресурсе потребне за наручивање књига. Слична решења су се показала добро у пракси те постоји велика потражња за оваквим софтвером.

У другом поглављу овог рада је дат приказ .NET технологије за развој веб апликација у коме је описан и C# програмски језик који је коришћен на серверској страни. У трећем поглављу је описан JavaScript као програмски језик који је коришћен за програмирање функционалности изгледа сајта. У четвртом поглављу је описан Bootstrap радни оквир који је коришћен за стилизовање компоненти сајта. У петом поглављу је коришћен JQuery који омогућава додатне функционалности за лакше манипулисање DOM-о и једноставно коришћење AJAX технологије описане у шестом поглављу. У седмом поглављу је описан API јер је коришћен за преузимање књига које администратор користи за унос у базу. У осмом поглављу је описан ASP .NET Core оквир, а у деветом је описан Entity Framework Core који олакшава рад са базом података. Софтерски систем је описан помоћу Ларманове методе. Наредна поглавља описују фазе Ларманове методе, а то су: фаза прикупљања захтева, фаза анализе, фаза пројектовања, фаза имплементације, фаза тестирања. На крају су представљени закључак и коришћена литература.

2 .NET платформа

.NET платформа је окружење отвореног кода, развијена од стране Microsoft-а, за креирање апликација различитог типа, као што су (Microsoft, Introduction to .NET, 2021):

- Web апликације, web API-ји и микросервиси
- Мобилне апликације
- Игрице
- Машинско учење
- Конзолне апликације...

.NET је направљен са циљем да се на исти начин креирају различити типови апликација. Платформу чине језици, библиотеке и CLR[1] који се користе за креирање апликације и њихово покретање. Платформе су: .NET Core која се може покренути на било ком оперативном систему (Windows, Linux, macOS), .NET Framework се користи при креирању web сајтова, сервиса и апликација за Windows, Xamarin/mono за мобилне уређаје.

Три главна језика за писање у .NET-у су: C#, F# и Visual Basic (Microsoft, Introduction to .NET, 2021).

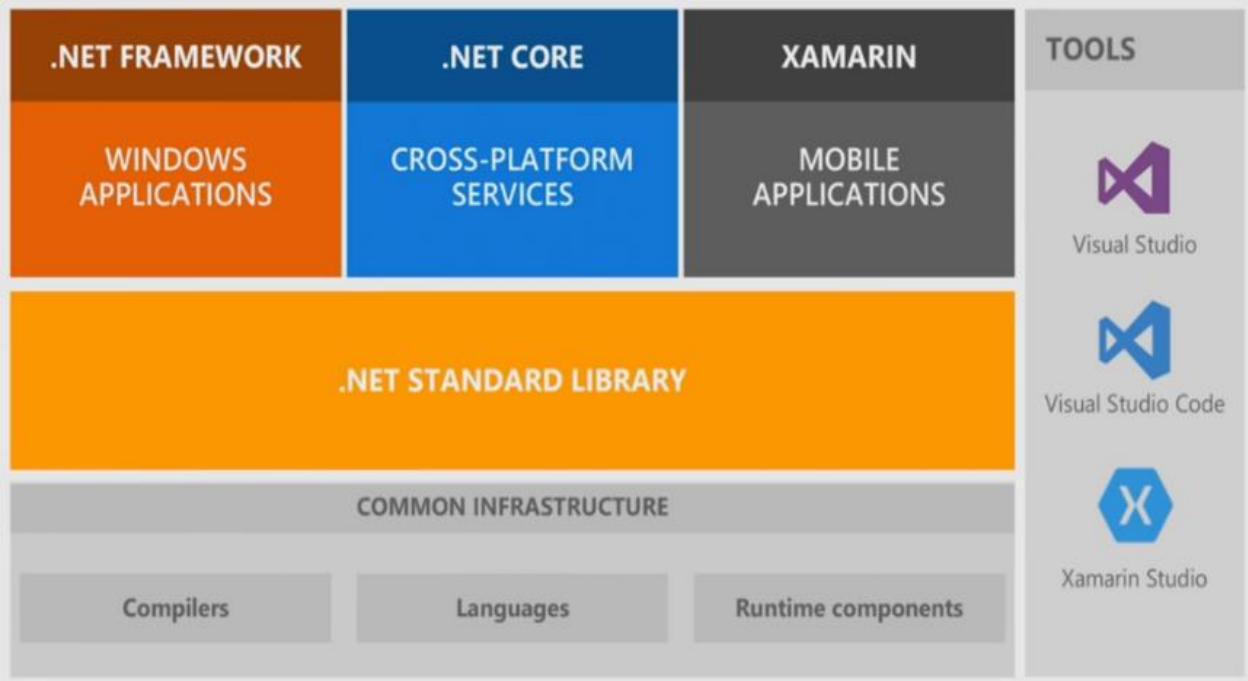
- C# је једноставан, модеран, објектно оријантисан језик јаког типа.
- F# је функционални, објектно оријентисани и императивни програмирамски језик.
- Visual Basic је језик са једноставном синтаксом за креирање објектно оријентисаних апликација сигурног типа.

.NET Standard је базиран на API-ју који је заједнички за све .NET имплементације. Направљен је за успостављање веће униформности у .NET екосистему. Свака имплементација може да има своје API-је који су специфични за оперативни систем на којем требају да се извршавају. На пример: .NET Framework је направљен само за Windows, .NET Core може на свим оперативним системима. (Microsoft, Introduction to .NET, 2021)

Common Language Infrastructure (CLI) је стандард направљен од стране Microsoft-а који омогућава коришћење различитих језика високог нивоа на различитим компјутерским платформама без поновног писања кода за сваку специфичну архитектуру. CLI омогућује да програм написан у било ком програмском језику, који подржава CLS[2], буде покренут на било ком оперативном систему користећи заједнички програм за извршавање уместо специфичног за сваки језик. (Jain, 2017)

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

[2] CLS је скуп функција, које дефинише CLI и које се сматрају одговарајућим преко језичких граница. Подскуп се зове *Common Language Specification.*



Слика 1 - .NET платформа[3]

2.1 Композиција .NET Core окружења

Композиција .NET Core окружења је сачињена од следећих елемената:

CLI Tools је скуп алата за развој и примену. CLI је крос-платформ алат за креирање, преуређивање пакета, прављење, покретање и објављивање .NET апликација. При креирању овог пројекта коришћен је Visual Studio. Visual Studio има уграђен CLI за отварање фајлова, инсталацију проширења (екстензија), промену језика у ком се пише и дијагностику излаза путем опција командне линије. Остали IDE-јеви (Integrated development environment) вишег нивоа, едитори и алати могу да користе CLI за подршку .NET Core апликација. (TutorialTeacher, 2020)

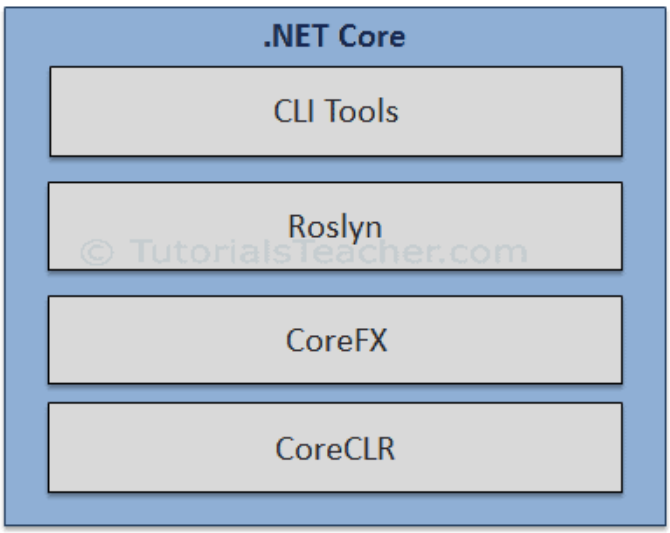
Roslyn је компајлер за C# и Visual Basic. Компајлер процесира код који програмер напише, пратећи правила која се често разликују од начина на који људи читају и разумеју код. Основно разумевање модела који користе компајлери од суштинског је значаја за разумевање API-ја који се користе када се креирају алати засновани на Roslyn-у. (Microsoft, Understand the .NET Compiler Platform SDK model, 2020)

CoreFX је скуп framework библиотека, односно основна библиотека класа за .NET Core. Садржи типове за колекцију, конзолу, JSON, XML, *async* и многе друге.

CoreCLR је група технологија које су основа за runtime, као што су RyuJIT, Garbage Collector и многе друге. Разлика између CoreCLR-а и CLR-а је то што је CoreCLR крос-платформ.

[3] [Digital image]. https://devblogs.microsoft.com/wp-content/uploads/sites/31/2019/04/netcore.png

.NET Core Framework је састављен од следећих делова:



Слика 2 - Композиција .NET Core окружења[4]

2.2 C# програмски језик

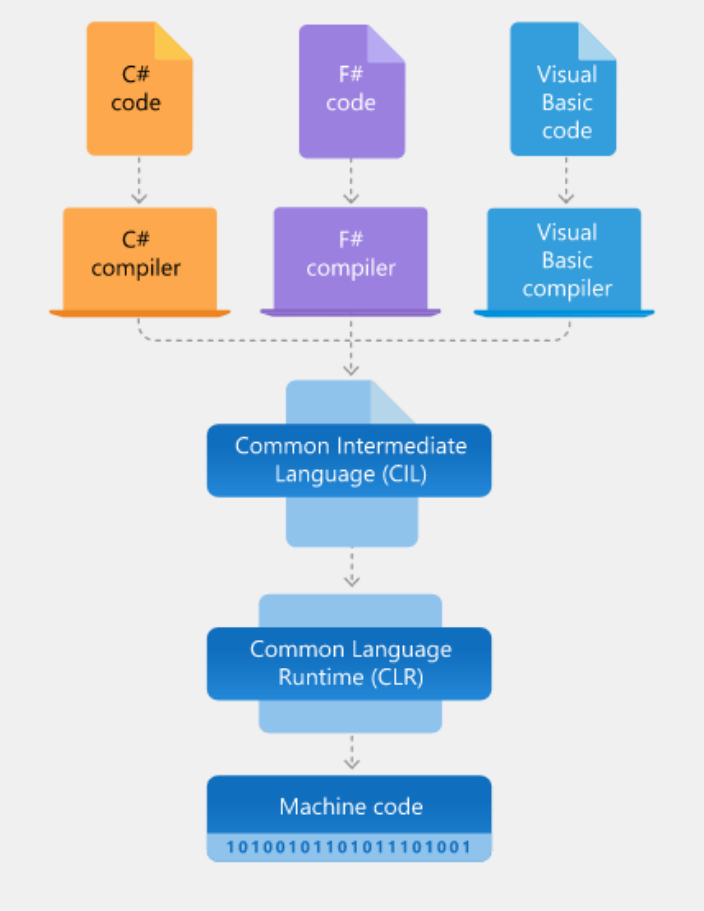
C# је модеран, објектно оријентисан, *type safety*[5] програмски језик, оријентисан на компоненте. Направљен је од стране Microsoft-а као део .NET Framework-а. Такође је језик ECMA-334 стандарда, који је Microsoft направио да буде крос-платформ језик. Он је *strongly typed* језик, што значи да када се варијабла једном декларише њен тип се не може променити.

Омогућава програмерима да изграде многе врсте поузданих и робусних апликација које раде у .NET екосистему. Неке од важних C# карактеристика које омогућавају креирање робустних и издржљивих апликација су (Microsoft, A tour of the C# language, 2021):

- **Garbage collection** аутоматски чисти меморију заузету нодоступним некоришћеним објектима.
- **Nullable types** представљају све вредности основног типа плус додатна null вредност.
- **Хватање грешке (Exception handling)** који помажуда се разреши нека неочекивана ситуација која се окине кад је програм покренут.
- **Lambda expressions** подржава технике функционалног програмирања.
- **Language Integrated Query (LINQ)** је синтакса која креира заједничу шему за рад са подацима из било ког извора.
- Језик подржава **асинхроне операторе** који омогућавају да се код извршава паралелно.

[4] [Digital image]. https://www.tutorialsteacher.com/Content/images/core/dotnet-core-components.png
[5] Само операције дефинисане за одређени тип могу да се примене над тим типом.

Извршни код написан у C# програмском језику се комплајлира у Intermediate language (IL). IL као и ресурси (битмапе и стрингови) су складиштени на диск у извршни фајл који се зове асембли (.exe или .dll екстензије). Када се програм изврши, асембли се учитава у CLR; а он извршава Just-In-Time компајлирање да преведе IL код у машински код.



Слика 3 - Извршавање кода у C#[6]

[6] [Digital image]. https://dotnet.microsoft.com/static/images/illustrations/swimlane-architecture-framework.svg?v=ZuTW7j9pS1oiuMqx3E-Xvb9OEM_8ajDEcHbecyjRtLA

3 JavaScript

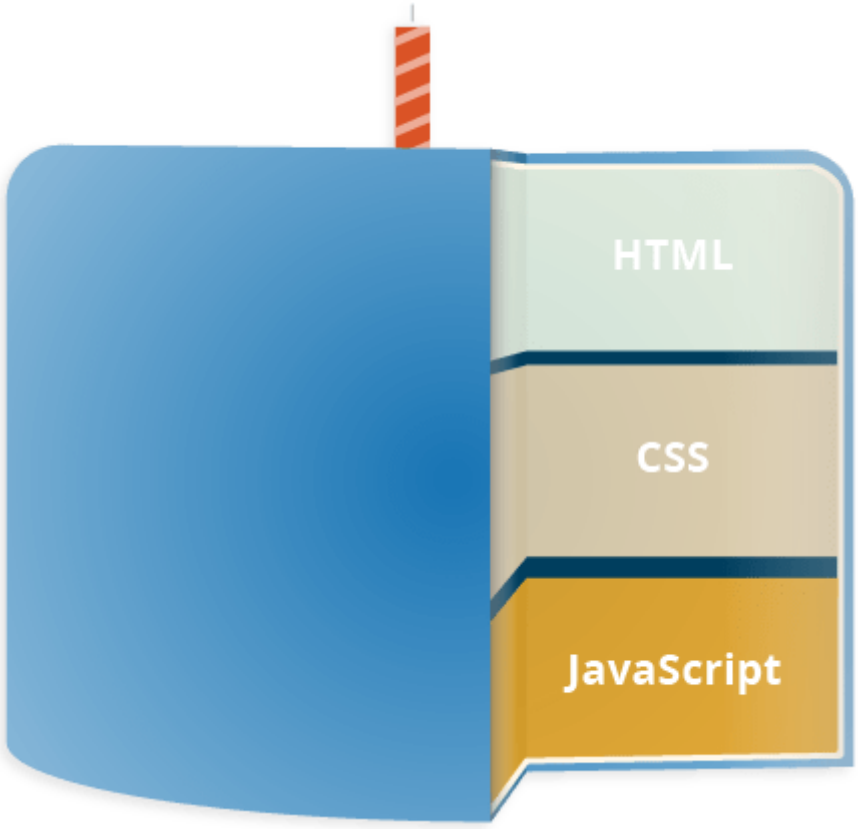
Следеће технологије су коришћене при креирању корисничког интерфејса:

HTML је маркап језик који користимо да дефинишемо структуру и дамо значење веб садржају, нпр. дефинисање параграфа, хединга, табела...

CSS је језик за стилизовање који користимо да би стилизовали HTML компоненте, нпр. подешавање позадине, боје, фонта, постављање садржаја у више колона...

JavaScript је скриптни језик који омогућава да се динамички ажурира садржај, контролише мултимедија, анимирају слике...

Програмски или скриптни језик JavaScript је језик који омогућава да се имплеметирају комплексне карактеристике на веб страницама приказујући садржаје, интерактивне мапе, анимиране 2D/3D графове и многе друге ствари. Он је трећи слој веб технологија (док су прве две HTML и CSS).



Слика 4 - HTML CSS JavaScript торта[7]

JavaScript је, за разлику од веровања многих, compile-time језик, користећи између осталог just-in-time компајлер. Такође је и динамички језик који подржава више програмских парадигми, између осталог објектно-оријентисану, императивну и декларативну (функционалну парадигму). Најчешће се извршава на клијентској страни,

[7] [Digital image]. https://i1.wp.com/developerstips.com/wp-content/uploads/2017/07/cake.png?fit=442%2C429&ssl=1

креирајући интерактивне странице и управљајући самим елементима странице, али и као серверска апликација користећи Node окружење.

Администратори веб сајтова су хтели да креирају богатије корисничке интерфејсе, углавном због жеље да уштеде простор на серверу за једноставне задатке које је требало обавити више пута, као што је валидација образаца. Настале су две опције: Java аплети и LiveScript језик, који је осмислио Брендан Еицх у компанији Netscape 1995. године, а који је касније уведен у прегледач Netscape 2.0 под називом JavaScript.

Аплети нису масовно прихваћени, али JavaScript јесте. Заједница администратора веб сајтова је прихватила могућност употребе кратких исечака кода који су уграђени у HTML документа и измену статичких елемената веб сајта. Затим је Microsoft објавио Internet Explorer 3.0 са JScript језиком, који је био обрнута инжењерска верзика JavaScript-а. Са неким IE функцијама. На крају су се стандардизовале различите имплементације језика и тако је настао језик ECMAScript. Европска асоцијација произвођача рачунара(ECMA) је креирала стандард ECMA-262, у коме се описани основни делови JavaScript програмског језика, без прегелдача и функције за веб. (Antani & Stefanov, 2017)

JavaScript можемо да посматрамо као појам који обухвата следећа три дела (Antani & Stefanov, 2017):

- **ECMAScript** - основни језик – променљиве, функције, петље и тако даље. ECMAScript језик није повезан са прегледачем и може да се користи у многим другим окружењима.
- **објектни модел документа (DOM)** - Овај модел обезбеђује начин за коришћење HTML и XML докумената. На почетку је JavaScript омогућавао ограничени приступ оним елементима који се могу скриптовати на страници, углавном обрасцима, линковима и сликама. Касније је JavaScript језик проширен да би сви елементи могли да се скриптују. Због тога је **World Wide Web конзорцијум (W3C)** креирао језички независан (који више није повезан са JavaScript-ом) DOM стандард за минипулацију структурним документима.
- **Објектни модел прегледача (BOM) –** Ово је скуп објеката који је повезан са окружењем прегелдача и никада није био део ни једног стандарда, све док HTML5 није започео стандардизацију неких уобичајених објеката који постоје у прегледачима.

Свака вредност која се користити је одређеног типа. У JavaScript-у постоје **основни типови** (number, string, bool, undefined, null) података и **сложени типови** (објекти). Основни типови података (Antani & Stefanov, 2017):

1. **number** – Обухвата бројеве са покретним зарезом, као и целе бројеве. На пример, све ове вредности су бројеви – 1, 100 и 3,14.
2. **string** – Састоји се од било ког броја знакова – на пример: a, један, један 2.
3. **bool** – Може да има вредности true или false
4. **undefined –** Када се приступа променљивој која не постоји, добија се посебна вредност undefined. То је default-на врадност и када се декларише промељива

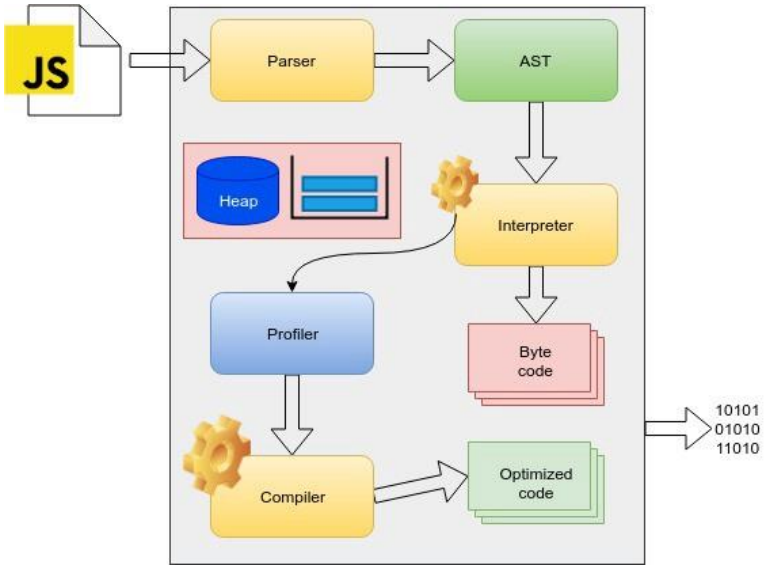
без да се иницијализује. JavaScript иницијализује променљиву преко вредности undefined.

5. **null – Означава празну вредност.** Null тип података се разликује од undefined типа по томе што променљива садржи вредност null, али је ипак дефинисана.

У JavaScript-у се све дешава унутар **Execution Context-а**. То је окружење у коме се извршава JavaScript код. Под окружењем се мисли на this, променљиве, објекти и функције којима код има приступ у одређеном тренутку. Постоје три типа Execution Context-а (Cohen, 2021):

- **Global execution context** – Ово је контекст у коме JavaScript код креће извршавање када се фајл учита у претраживач. Цео глобалан код односно код који није унутар функција и објеката се извршава у глобалном контексту.
- **Functional execution context** – Функционални контекст је дефинисан као контекст који прави JavaScript engine сваки пут када дође до позива функције. Свака функција има свој контекст извршавања. Функционални контекст има приступ коду глобалног контекста али не и обрнуто.
- **Execution context унутар eval функције** – Креира се када је позвана eval функција.

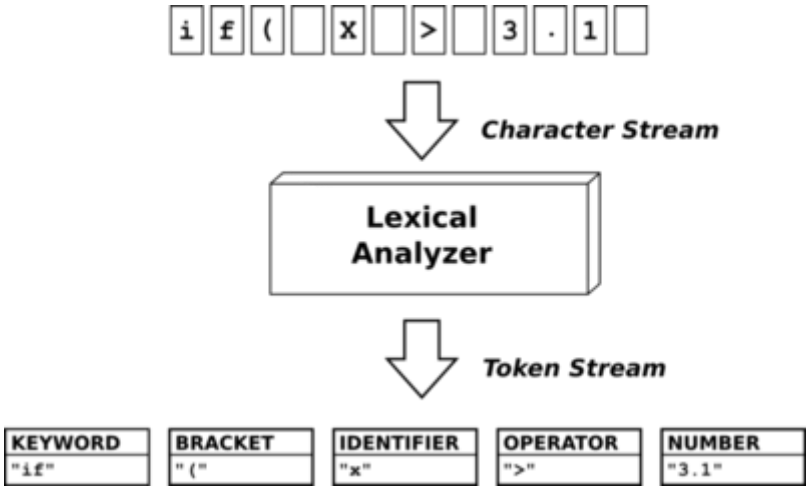
JavaScript код се у позадини интерпретира и компајлира. **Интерпретер** преводи код у машински код, инструкцију по инструкцију, док **компајлер** узмима цео код који претвара у машински код. Постоје два начина на који се JavaScript код може претворити у машински код. Када се код искомпајлира, машина лакше разуме шта треба обавити пре него што се код изврши, што чини извршавање бржим, али је потребно неко време унапред за овај процес. Са друге стране, када се код интерпретира, извршавање почиње одмах и брже се извршава код, али због недостатка оптимизације то може успорити велике апликације.



Слика 5 - Превођење JavaScript кода у машински код[8]

[8] [Digital image]. https://miro.medium.com/max/703/1*ab5fIXSXiqsOJ7i5ztkoUg.jpeg

JavaScript фајл приступа *engine*-у и **парсер** обавља лексичке анализе тако што разбија код на токене да би открио њихово значење. Ови токени праве **апстрактно синтаксно стабло** (Abstract Syntax Tree).



Слика 6 - Парсер JavaScript кода[9]

Апстрактно стабло игра главну улогу у семантичкој анализи где компајлер одобрава исправно коришћење елемената језика и кључних речи. Касније је поново користи апстрактно стабло за генерисање бајтокода или машинског кода.

Сада се JavaScript **интерпретира** од стране интерпретера који се зове *Ignition*, али и **компајлира** од стране JIT компајлера који се зове *TurboFan*. Стабло се предаје интерпретеру који брзо генерише неоптимизујући машински код и извршавање може почети без чекања. **Профајлер (profiler)** посматра код током извршавања и оптимизује оно што може да се оптимизује, нпр. ако се код извршава 1000 пута али сваки пут враћа исто решење. Профајлер сваки неоптимизовани код шаље компајлеру ради оптимизације и генерисања машинског кода, а он замењује све дупликате у претходно генерисаном неоптимизованом коду. Пошто профајлер и компајлер константно размењују бајткод, перформансе JavaScript се постепено побољшавају. (Patel, 2019)

[9] [Digital image]. https://miro.medium.com/max/500/1*OYiEF_Ww7vPwBZHCSUMc9g.png

4 Bootstrap

Bootstrap представља front-end радни оквир (framework) који се употребљава за израду веб страница и веб апликација. У питању је колекција CSS, HTML правила и JavaScript екстензије које користе најновије технике претраживања, пружа врло модерну типографију, форме, дугмад, табеле, навигацију као и остале елементе неопходне да би се правила страница високог квалитета. Највећа предност коју нуди јесте да подржи развој потпуно респонзивних, тј. прилагодљивих веб страница на различитим уређајима, у зависности од њихове величине.

CSS у Bootstrap-у представља формиране класе које треба применити на елементима HTML странице. Такође садржи JavaScript функције које пружају разне могућности као што је на пример Carousel. Bootstrap садржи грид систем који се састоји од 12 колона. Основне компоненте грид система Bootstrap-а су Container, Row и Col. Container је родитељски елемент у кога се смештају редови, а сваки од тих редова се састоји од колона. Сви остали елементи се смештају унутар ове три компоненте.

5 JQuery

JQuery је популарна JavaScript библиотека која се користи на клијентској страни. Коришћен је у хиљаде веб сајтова за разне задатке, од једноставних ефеката за прелаз преко елемената до манипулације DOM-ом па до валидација у AJAX-у. ASP.NET Core користи JQuery за валидације на страни клијента. JQuery је библиотека оријентисана на DOM и такође пружа функционалности везане за AJAX. JQuery има много додатака других произвођача који убрзавају рад и поједностављују развој на страни клијента.

У овом пројекту JQuery је коришћен за креирање пагинације и при креирању налога за корисника да се створе визуелни ефекти да ли су правилно унети имејл адреса, име, презиме, шифра... У наредном сегменту биће представљен код који је коришћен за обрађивање поља за унос текста уз помоћ JQuery-ја.

```
$("#signUp input").keyup(function () {
    switch ($(this).attr("name")) {
        case "Email":
            RegexCheck("email", $("[name='Email']").val(), $("[name='Email']"));
            break;
        default:
            $(this).removeClass("is-invalid");
            $(this).addClass("is-valid");
            break;
        case "Password":
            ConfirmPassword();
            RegexCheck("password", $("[name='Password']").val(),
$("[name='Password']"));
            break;
        case "ConfirmPassword":
            ConfirmPassword();
```

```
            break;
    }
});
```

Селектор **$** у JQuery-ју селектује читав HTML документ и омогућава да се на њега примене JQuery функције. Након $-а следе заграде **( )** унутар којих се пише HTML елемент који треба селектовати. У овом примеру је селектован елемент чији је id *„#signUp“* па унутар њега елемент *input*. Након тога следи акција, тј. шта треба урадити са тим елементом. Акција је **keyup** која ће да се окине сваки пут када се откуца неки карактер унутар input поља. Унутар акције се пише функција која ће да се примени сваки пут када се окине *keyup* акција.

```
function RegexCheck(type, value, name) {
    var pattern;
    if (type == "email")
        pattern = /^([\w\.\-]+)@([\w\-]+)((\.(\w){2,3})+)$/;
    else
        pattern = /^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[!@#$%^&*])(?=.{6,})/;
    if (value.match(pattern)) {
        name.removeClass("is-invalid");
        name.addClass("is-valid");
    } else {
        name.removeClass("is-valid");
        name.addClass("is-invalid");
        if (type == "password") {

            $("#emptyPassword").hide();
            $("#password").show();
        }
        else {
            $("#empty").hide();
            $("#email").show();
        }
    }
}
```

Уколико је нешто откуцано у *input* пољу, чије је име „Email“, позваће се функција RegexCheck(tip, vrednost, ime). Ова функција проверава да ли унети текст одговара шаблону који је дефинисан на следећи начин: *(/^([\w\.\-]+)@([\w\-]+)((\.(\w){2,3})+)$/)*. Уколико одговара склониће се класа *„is-invalid“*, са тог елемента, а додаће се *„is-valid“*.

6 AJAX

Ajax је акроним за Asynchronous JavaScript and XML. Модерне веб апликације преферирају JSON уместо XML-а као формат за трансфер података. Ajax је начин комуницирања између клијентског браузера и сервера који користи постојеће технологије као што су HTML, XML, JSON и JavaScript.

Узмимо за пример страницу која има у себи форму и када се поднесе захтев, подаци се шаљу до сервера. Подаци се обрађују на серверској страни и резултат се враћа назад до браузера.

Без Ajax-а, комуникација би имала овакав ток (Joshi, 2019):

1. Корисник шаље захтев за страницом за унос података.
2. Сервер шаље захтевану страницу браузеру.
3. Корисник попуњава форму и шаље податке до сервера тако што потврди форму.
4. Када је форма потврђена, корисник не може да ради са формом зато што браузер чека сервер да пошаље одговор.
5. Сервер обрађује те податке и шеље одговор назад до браузера.
6. Када се одговор учита у браузеру, корисник може поново да користи страницу.

Овако описане операције представљају *синхроно* извршавање, јер се унос података (активност на клијентској страни) и обрада података (активност на серверској страни) дешавају једно после другог.

Ако би иста страница била направљена помоћу Ajax-а, редослед операција би био следећи (Joshi, 2019):

1. Корисник шаље захтев за страницом за унос података.
2. Сервер шаље захтевану страницу браузеру.
3. Корисник попуњава форму и шаље податке до сервера тако што потврди форму.
4. Овај пут форма је потврђено коришћењем Ajax-а. Ajax ће се као асинхрона операција извршавати у позадини кроз JavaScript код.
5. Корисникова интеракција са страницом није онемогућена иако браузер чека одговор од сервера.
6. Сервер обрађује те податке и шеље одговор назад до браузера.
7. Браузер може обавестити корисника о одговору како би се могле предузети даље радње (ако их има).

На овај начин се у Ajax комуникацији може *паралално* одвијати интеракција корисника са страницом и обрада сервера. Током Ajax комуникације мора се изабрати формат података као што је XML или JSON. Овако описана Ajax комуникација покреће се помоћу JavaScript-а и објекта XMLHttpRequest који обезбеђује браузер.

JQuery библиотека обједињује све Ajax функционалности у скуп метода, од којих је најважнија $.ajax(). Остале методе зависе од $.ajax() методе да би обављале свој посао. (Joshi, 2019) Примером из овог пројекта може се сагледати $.ajax() метода.

```
$.ajax({
    url: '@Url.ActionLink("ReturnBooks", "Book")',
```

```
    data: { pagiNumber: pageNumber, price: price, genres: genres },
    method: "get",
    success: function (data) {
        $("#loop").html("");
        PasteThisBooks(data);
    },
    error: function () {
        alert("Error while returning Books!");
    }
});
```

- **url** представља адресу којој се приступа на серверској страни, тачније уз помоћ методе *@Url.ActionLink* се приступа Book контролеру и *ReturnBooks* методи која се налази унутар тог контролера.
- **data** представља параметре који се шаљу методи *ReturnBooks* и они морају бити именовани на исти начин као што су именовани на серверској страни!
- **method** представља који захтев треба послати ка серверу, да ли је то *GET, POST, PUT, DELETE* или неки други.
- **success** је функција која ће да се изврши када се функција *ReturnBooks* правилно изврши. Своју повратну вредност фукнција ће да додели *data* параметру фукнције.
- **error** ће да се изврши уколико функција *ReturnBooks* баци неку грешку која није обрађена.

Одговор на позив Ajax-а може се добити у три формата: **HTML**, **XML** или **JSON**. HTML је најједноставнији тип података који може да се дода на страницу коришћењем Ajax-а.

Предности HTML-а су:

- Једноставан је за писање, захтеве и приказивање.
- Подаци који су послати са сервера иду директно на страницу и браузер их не обрађује.

Недостаци HTML-а су:

- Сервер мора да направи формат HTML-а који је спреман за коришћење на страници.
- Није добро прилагођен за употребу у другим апликацијама осим на вебу. Нема добру преносивост података.
- Захтев мора доћи са истог домена.

XML изгледа слично као HTML, али су називи тагова другачији, зато што описују податке које садрже. Синтакса је стриктнија него код HTML-а. Предности XML-а су:

- Флексибилан формат података и помоћу њега се могу представити комплексне структуре.
- Добро ради са различитим платформама и апликацијама.
- Обрађује се истим DOM методама као и HTML.

Недостаци XML -а су:

- Сматра се опширним језиком зато што тагови додају доста карактера подацима који се шаљу.
- Захтев мора доћи са истог домена као и остатак странице.
- За обраду резултата може бити потребно много кода.

JavaScript Object Notation (JSON) користи сличну синтаксу као што је објектна нотација да би представио податке.

Предности JSON -а су:

- Може бити позван са било ког домена.
- Прецизнији је (мање оприан) него HTML и XML.
- Обично се користи са JavaScript-ом (и добија све ширу употребу у веб апликацијама).

Недостаци JSON -а су:

- Синтакса је стриктна, јер уколико недостаје наводник, зарез или двотачка могу да изазову проблем.
- Зато што је то JavaScript може да садржи злонамерни садржај, зато треба користити JSON који су произвели поуздани извори. (Duckett, 2014)

7 WEB API

Application programming interface (API) је интефејс који софтверски програм представља осталим програмима, људима, и у случају веб API-ја свету путем интернета. Дизајн API-ја се много ослања на програм који стоји иза њега – пословна логика, карактеристике производа, повремене грешке. Иако је API дизајниран да ради са другим програмима, углавном су намењени коришћењу и разумевању за људе који пишу те друге програме.

Web API-ји су главни елементи који омогућавају олакшано и несметано кретање података кроз мрежу за главне пословне платформе на интернету. Он је настао да би се размењивале информације са добављачима података, са циљем да људи у другим компанијама не троше време да сами решавају те проблеме. На пример, уколико би требала да се дода интерактивна мапа на веб страницу без самосталног прављења Google мапе, или уколико би се креирала страница за логовање без самосталног прављења странице за Facebook Login, или креирање четбота који комуницира са корисницима без прављења система за размену порука у реалном времену.

У свим овим случајевима, додатне функције и производи су креирани коришћењем података или интеракцијом са неким платформама. API-ји омогућавају предузећима да брзо развију јединствене производе. Уместо да поново „измисле точак“, старт-апи су у стању да направе разлику у својој понуди производа, користећи предности постојећих технологија и улазећи у друге екосистеме.

Парадигма API-је дефинише као интерфејс који излаже позадинске податке са сервера осталим апликацијама. Током година настало је више API парадигми: REST, RPC, GraphQL, WebHooks и WebSockets су једни од најпознатијих стандарда данас.

7.1 Request–Response APIs

Request–response API-ји представљају интерфејс преко HTTP веб сервера. API-ји дефинишу скупове *endpoint-а*. Клијент креира HTTP захтев за податке ка *endpoint-има,* а сервер враћа одговор у JSON или XML формату. Сервиси користе три уобичајене парадигме да представе request–response API-е: : **REST**, **RPC** и **GraphQL**.

7.1.1 **Representational State Transfer (REST)**

REST је софтверска архитектура за обезбеђивање стандарда између рачунарских система на веб-у, што олакшава комуникацију између система. У последње време REST је најпопуларнија парадигма за развој API-ја. REST парадигму користе произвођачи *Google, Stripe, Twitter u GitHub*. У REST-у се све заснива на *resource*-има, што представља ентитет који може бити идентификован, именован, адресиран или обрађен на вебу. REST API-ји излажу податке као *resource*-е и користе стандардне

HTTP методе да представе трансакције креирање, читање, ажурирање и брисање илити **CRUD** (Create, Read, Update и Delete) у односу на ове *resource*-е.

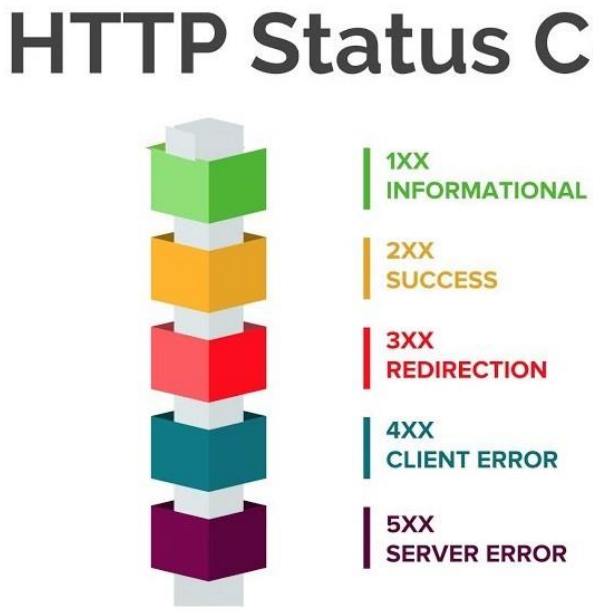
Правила за REST API су:

- *Resource*-и су део URL-а, нпр. */књиге*
- За сваки resource имплементирају се две URL-а: један за колекцију */књиге*, а други за одређени елемент */књиге/1*
- За *resource*-е се уместо глагола користе именице. Нпр. уместо */вратиИнформацијеОКњигама/1* користи се */књиге/1*
- HTTP методе попут GET, POST, UPDATE и DELETE наговештавају серверу акцију која ће да се изврши. Различите HTTP методе позване над истим URL-ом пружају различите функционалности:
    - ***Create*** - Користити **POST** за креирање нових *resource*-а.
    - ***Read*** – Користити **GET** за читање *resource*-а. GET захтев никад не мења стање *resource*-а. Немају спољне ефекте, намењени су само за читање.
    - ***Update*** – Користити **PUT** за измену *resource*-а и **PATCH** за парцијално ажурирање постојећих *resource*-а.
    - ***Delete*** – Користити **DELETE** за брисање постојећих *resource*-а.

Operation	HTTP verb	URL: /users	URL: /users/U123
Create	POST	Create a new user	Not applicable
Read	GET	List all users	Retrieve user U123
Update	PUT or PATCH	Batch update users	Update user U123
Delete	DELETE	Delete all users	Delete user U123

Слика 7 - CRUD операције, HTTP глаголи и REST конвенција

- Стандардни HTTP одговор враћа статус код указујући на успешност или на неуспех. Кодови који почињу са **200** представљају успешно извршавање, кодови који почињу са **300** указују на то да је ресурс померен (нека акција мора бити испуњена да би се испунио захтев), кодови који почињу са **400** указују на грешку на клијентској страни (нпр. недостаје захтевани параметар или превише захтева) и кодови који почињу са **500** указују на грешку насталу на серверској страни.



Слика 8 - HTTP статус кодови[10]

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

Уколико *resource* постоји само у оквиру неког другог *resource*-а онда то није resource највишег нивоа у URL-у него представља *subresource*. Нпр. GitHub API користи *subresource-е* да представи однос у различитим API-јима:

- POST /repos/:owner/:repo/issues - Креира проблем
- GET /repos/:owner/:repo/issues/:number – Проналази проблем
- GET /repos/:owner/:repo/issues – Приказује све проблеме
- PATCH /repos/:owner/:repo/issues/:number – Исправља проблем

Поред CRUD операција у REST архитектури постоје и non-CRUD операције. Оне се користе у следећим случајевима:

- Извршава акцију као део поља у *resource-u*. GitHub-ов API користи „archived“ као улазни параметар, да представи архивирање у складишту.
- Третира акцију као *subresource*. GitHub-ов API користи шему за закључавање и откључавање проблема. *PUT /repos/:owner/:repo/issues/:number/lock* закључава проблем.
- Неке операције, као што је претрага, су тешке за уклапање у REST парадигму. Тада се корсити глагол које представља акцију у API URL-у. *GET*

[10] [Digital image]. https://miro.medium.com/max/920/1*w_iicbG7L3xEQTArjHUS6g.jpeg

/search/code?q=:query: проналази фајлове на GitHub-у, који одговарау задатом упиту.

7.1.2 **RPC и GraphQL**

Remote Procedure Call (RPC) је један од најједноставнијих API парадигми, у којој клијент извршава код на другом серверу. Код REST архитектуре се све врти око *resource-a*, док су код RPC-а то акције. Клијенти шаљу назив методе и аргументе до сервера и као одговор добијају JSON или XML.

RPC API прати два правила (Jin, Sahni, & Shevat, 2018):

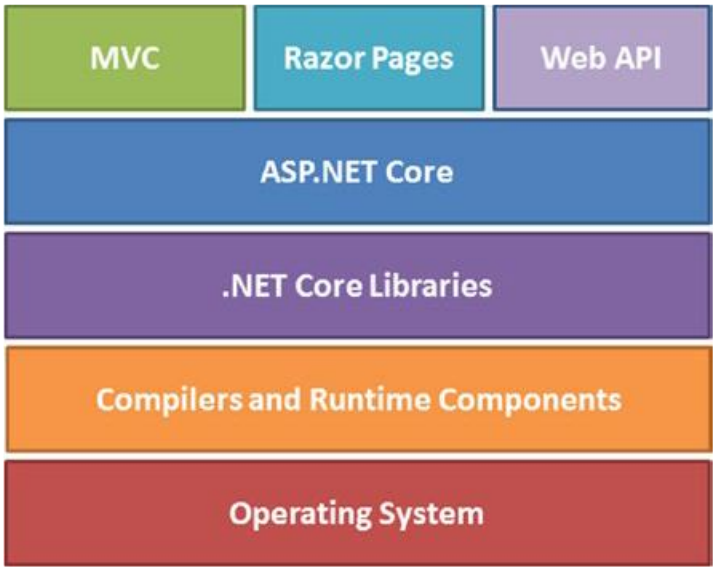
- Крајње тачке садрже назив операције која треба да се изврши.
- Позиви API-ја се праве од HTTP глагола који је најприкладнији за ту акцију: GET за захтеве који не смеју да се мењају,већ само служи да прочита неке податке и POST за остале.

GraphQL је језик упита за API-је, атакође представља и представља време које је потребно да се подаци врате назад до корисника. Направио га је Facebook 2012 године, а јавно га је објавио 2915 године и усвојили су га API произвођачи као што су GitHub, Yelp и Pinterest. GraphQL омогућава клијентима да дефинишу структуру захтеваних података, а сервер ће вратити баш ту структуру. За разлику од REST и RPC API-ја, GraphQL API-ји захтевају само једну крајњу тачку. Нису потребни различити HTTP глаголи да се опише операција. Уместо тога, у телу JSON-а се наводи да ли ће да се изврши упит или мутација. (Jin, Sahni, & Shevat, 2018)

8 ASP.NET Core оквир

ASP.NET Core је оквир за креирање модерних веб апликација и сервиса. Део је .NET Core-а и он је крос-платформ оквир и оквир отвореног кода. То значи да може да развије и примени веб апликације на све модерне оперативне системе као што су Windows, Linux и macOS. Главне функционалности овог оквира су (Joshi, 2019):

- Крос-платформ оквир. Могу да се направе и покрену веб апликације на Windows, Linux и macOS-у.
- Отвореног је кода са великим учешћем заједнице.
- Уграђен оквир *убризгавања зависности* (dependency injection)
- ASP.NET Core укључује уграђен Kestrel веб сервис. Kestrel се може користити сам или се хостовати као веб апликације преко IIS, Nginx или Apache-а.
- Више развојних опција за кориснички интерфејс укључујући MVC, Razor странице и Blazor. Такође се могу користити и JavaScript оквири на клијентској страни за креирање изгледа страница.
- Обједињени модел програмирања за MVC веб апликације и Web API-је.
- Модуларни *pipeline* високих перформанси погодан за модерне апликације оптимизоване за *cloud*.



Слика 9- Слојеви .NET Core-а и ASP.NET Core-а

Најнижи слој је оперативни систем. Зато што је .NET Core крос-платформа то могу бити Windows, Linux и macOS. Код веб апликације остаје исти без обзира на основу оперативног система. За развој и покретање .NET Core апликација су потребни компајлери програмских језика и друге неопходне компоненте за време извршавања. Следећи слој је сачињен од .NET Core библиотека и услуга на нивоу оквира. Ове библиотеке обезбеђују неколико функционалности укључујући типове података и фајлова IO. ASP.NET Core користи .NET Core библиотеке и стога је приказан на врху

ове лествице. ASP.NET Core има 3 главне опције за развој апликације: ASP.NET Core MVC, ASP.NET Core Razor Pages и ASP.NET Core Web API-ји. Ове опције чине врх лествице. ASP.NET Core је изграђен узимајући у обзир модуларност. Најше ASP.NET Core апликација користи функције које се налазе у NuGet пакетима. (Joshi, 2019)

Део ASP.NET Core странице за приказ књиге:

```
@model Book
@section Style{<link href="~/css/BookShowItemCSS.css" rel="stylesheet" />}
<div class="col-lg-3 col-md-3 col-sm-12 leftblock">
    <img id="bookimg" src="@Model.Image" alt="Alternate Text" />
</div>
<div class="col-lg-9 col-sm-12 rightblock">
    <div class="col-lg-9 col-sm-12 book">
        <h3>@Model.Title</h3>
        <p>@Model.Description</p>
        <div class="genres">
            <h5>Genres:</h5>
            @foreach (Genre item in Model.Genres)
            {<p>@item.Name</p>}
        </div>
    </div>
    <div class="col-lg-3 col-sm-12 autors">
        <h5>Autors:</h5>
        @foreach (Autor item in Model.Autors)
        {<p>@item</p>}
    </div>
</div>
```

Прво се уочава **@model Book** који се дефинише увек на почетку и служи да учита објекат књиге при учитавању странице. **@section{}** је блок у кога се убацују линк или *script* тагови и они ће да буду учитани на старници само када се приступи овом .cshtml фајлу. Карактер **@** означава да је у питању C# код који *Razor* страница зна да прочита и као излаз даје само HTML. **@Model.Title, @item.Name, @Model.Image** представљају параметризоване податке.

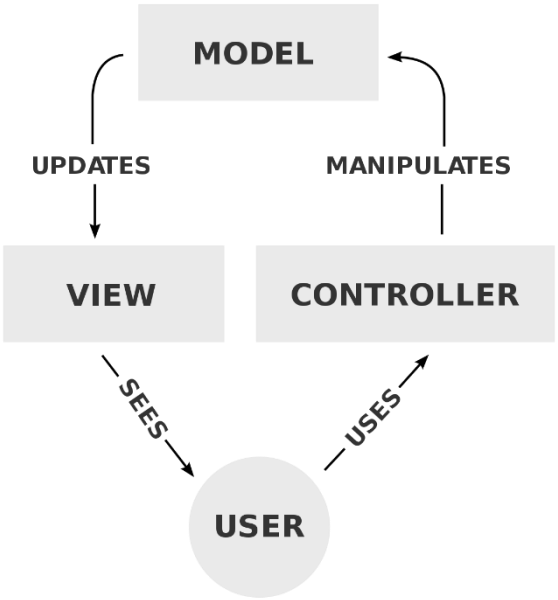
Значајна страница је **Layout** страница која је „родитељ“ свим осталим страницама. Она је та која садржи заједничке делове странице нпр. head и body, а онда се помоћу ***@RenderBody()*** учитава остатак странице који се налази у посебном .cshtml фајлу. Не мора да постоји само једна *Layout* страница, већ могу да се мењају зависно од делова који треба да буду приказани увек или уклопљени са странице када се учита. У наставку биће представљен исечак кода описаних елемената.

```
@{
    Layout = "_Layout";
}

<div class="container">
    <main role="main" class="pb-3" id="main">
        @RenderBody()
    </main>
</div>
```

8.1 ASP.NET MVC архитектура

ASP.NET Core Razor странице креирају веб апликације које користе *Model-View-Controller (MVC)* као шаблон. Model-View-Controller шаблон дели функционалности апликације на три компоненте: model, view и controller. Разлика између MVC и MVVM је то што у MVVM-у нема класе контролер која контролише акције, већ је све смештено у једној класи.



Слика 10 - MVC архитектура[11]

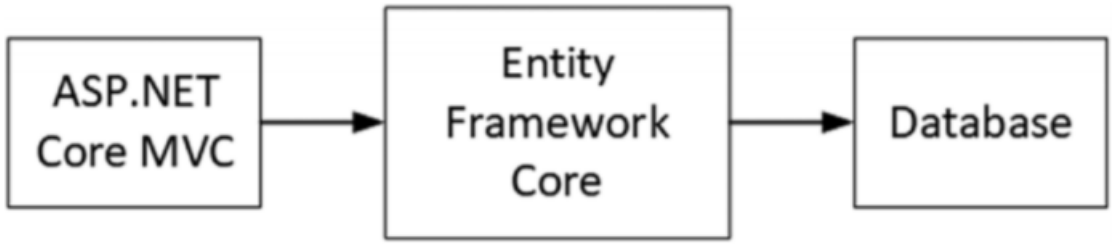
- **Models** – Објекти из модела су део апликације који имплементирају доменску логику. Често објекти модела враћају и чувају стање модела у бази.
- **Views** – То су компоненте које приказују корисничи интерфејс апликације. Кориснички интерфејс је састављен од модела.
- **Controllers** – То су модели који обрађују корисничке захтеве, манипулишу моделом и на крају бирају приказ за приказивање корисничког интерфејса.

MVC шаблон помаже при креирању апликација тако што одваја различите аспекте апликације (улазну логику, пословну логику и логику корисничког интерфејса) и међусобно их повезује. Овај начин раздвајања помаже код сложених проблема, зато што омогућава коринику да се фокусира на имплементацију и решавање једног простијег проблема. MVC шаблон олакшава тестирање апликације подстичући на коришћење *развоја вођеног тестовима* за креирање апликација.

[11] *[Digital image]*. https://upload.wikimedia.org/wikipedia/commons/thumb/a/a0/MVC-Process.svg/1200px-MVC-Process.svg.png

9 Entity Framework Core

Entity Framework Core има кључан задатак, а то је убацивање .NET објеката у базу података и враћање истих из базе. Он се понаша као мост између ASP.NET Core MVC апликације и базе података.



Слика 11 - Entity Framework Core као мост између ASP.NET Core MVC и базе података

Убацивање података у базу може да буде веома комплексан процес. Базе података не постоје одвојено, оне су креиране и њима се управља помоћу *сервера базе података*. Сервери базе података производе трајно складиште за велики број апликација и зато су потребни јаки софтвери са високим перформансама. Постоје различити типови сервера базе података, а Entity Framework Core ради са релационим сервером базе података. Релациони сервер базе података ради са релационим базама података у којима се подаци смештају у редове табела. Релационе базе података прихватају команде написане у SQL-у, који одобрава операције над подацима, као што је чување или брисање података. Да би складиштио податке у базу података, Entity Framework Core мора да претвори објекат у форму која може да се сачува у базу података и може да се запише помоћу SQL команде. Да би вратио податке из базе података, Entity Framework Core мора бити у могућности да окрене процес, тј. мора сам да направи SQL упит који ће да затражи податке из базе који представљају тражени објекат и да попуни особине тог .NET објекта. Entity Framework Core користи LINQ упите да би постављао упите над базом, што рад са колекцијама у бази података чини сличним раду колекцијама у меморији. (Freeman, 2018)

Да би се подаци сачували у базу података, Entity Framework Core треба посебно да идентификује сваки објекат, што захтева одабир својстава који ће се користити као примарни кључ. Да би Entity Framework Core препознао неку особину као примарни кључ, она у називу мора да има „Id“ или да се дефинише преко fluent API-ја или преко конвенције. Пример класе Жанр из апликације:

```
public int GenreId { get; set; }
public string Name { get; set; }
public List<Book> Books { get; set; }
public override string ToString()
{
   return Name;
}
```

Када се користи Entity Framework Core да сачува прост модел података, као пример изнад, класа *објашњења базе подата* ће то једноставно урадити. Постоје три кључне карактеристике класе за *објашњење базе подата*. (Freeman, 2018)

Прва карактеристика је да је основна класа ***DbContext*** класа, која се налази у Microsoft.EntityFrameworkCore именованом простору. Оно што чини класу за обајшњење базе података и омогућава приступ Entity Framework Core функционалностима је коришћење *DbContext* класе. (Freeman, 2018)

Друга карактеристика је да конструктор прима *DbContextOptions<T>* објекат (T је класа обајшњења) који мора бити прослеђен конструктору основне класе косристећи кључну реч **base** као на примеру:

```
public DataContext(DbContextOptions<DataContext> opts) : base(opts) { }
```

Параметар конструктора ће омогућити Entity Framework Core-у информације о конфигурацији које су му потребне за повезивање са сервером базе података. Уколико се не проследи параметар конструктру или се не проследи објекат, биће бачена грешка. (Freeman, 2018)

Трећа карактеристика је особина чији је тип *DbSet<T>*, где T представља класу која ће да се сачува у бази података.

```
public DbSet<Genre> Genre { get; set; }
```

Модел класе података је Жанр, тако да особина враћа *DbSet<Genre>* објекат. Особина мора бити дефинисана са get и set члановима. *Set* члан омогућава Entity Framework Core-у да додели податке објекту, а get члан омогућава приступ тим подацима у целој апликацији. (Freeman, 2018)

За override функције *OnModelCreating* користи се *ModelBuilder* API за конфигурисање модела. Ово је најмоћнија метода за конфигурацију која дозвољава да се наведе конфигурација без мењања класа ентитета. Fluent API конфигурација има највећу предност и override-ованје конвенције и анотације података.

```
protected override void OnModelCreating(ModelBuilder modelBuilder){}
```

Инстанце *DbContext*-а се могу конструисати на обичан .NET начин, на пример са *new* у C#-у. Конфигурација се изводи override-овањем методе *OnConfiguring* или прослеђивањем опција конструктору као на пример:

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=EShop;");}
```

9.1 Linq технологија

Linq је акроним за Language Integrated Query (језички интегрисани упит) који је део програмског језика. Linq пружа једноставан начин за писање или постављање упита подацима са наведеном синтаксом, као када би се користила **where** клаузула за писање упита у SQL-у. То је синтакса која се примењује у циљу испитивања података. У следећем одељку ће бити приказан пример Linq упита који је коришћен за претраживање конкретног купца:

```
public Customer Find(SignInViewModel model)
{
    Customer customer = uow.RepostiryCustomer.Find(c => c.Email == model.Email &&
c.Password == model.Password);
    if (customer is null)
        throw new CustomerNullException("Wrong credentials");
    return customer;
}

public Customer Find(Predicate<Customer> p) => shopContext.Customer.ToList().Find(p);
```

Функцији *Find* се прослеђује модел са подацима на основу којих се проверава да ли задати корисник постоји у бази. У тој функцији се позива друга функција *Find* из класе *RepostiryCustomer*. Тој функцији се прослеђује **предикат** који представља један *Lambda* израз. У методи *Find* из класе *RepostiryCustomer* се прво приступа *shopContext*-у који је *DataContext* класа, а онда *DbSet*-у *Customer*. *Linq* као резултат враћа *IEnumerable<Customer>* и због тога мора да се претвори у листу помоћу *ToList()* упита. Над том листом се позива *Linq* упит коме се прослеђује *Lambda* израз који је примљен као параметар функције.

10 Студијски пример

Развој студијског примера заснован је на Ларамановој методи развоја софтверског система. Ларманова метода се састоји из следећих фаза (Влајић, 2019):

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

Фаза прикупљања захтева – У овој фази се идентификују и дефинишу кориснички захтеви везани за пројекат и потребно је прикупити што више потребних информазија односно захтева од корисника. Захтеви се описују преко UML модела случаја коришћења, а он се састоји од скупа случаја коришћења, актора и веза измешу актора и случаја коришћења. Случај коришћења описује скуп сценарија, односно скуп жељених коришћења система од стране актора. Случај коришћења има више алтернативних и један основни сценарио. Сценарио представља секвенцу акције које описују интеракцију актера и система. Једно акцију сценарија може извршити актер или систем, па их стога можемо поделити на две групе (Влајић, 2019):

- **Актор** може изводити три акције:
    1. АПУСО – Актор припрема улаз за системску операцију
    2. АПСО – Актор позива системску операцију
    3. АНСО – Актро извршава несистемску операцију
- **Систем** изводи две акције једну за другом:
    1. СО – Систем извршава системску операцију
    2. ИА – Излазни аргументи који представљају резултат системске операције

Фаза анализе – У овој фази се описује структура и понашање софтверског система. Понашање је описано преко секвенцних дијаграма и преко системских операција. Структура је описана преко концептуалног и релационог модела. Структура софтверског система се описује преко концептуалног модела који је сачињен од концептуалних каласа и асоцијација између тих класа. Секвенцни дијаграм се прави посебно за сваки случај коришћења и као резултат анализе секвенцних дијаграма добијају се системске операције које треба порјектовати у коду. За сваку системску операцију праве се по један уговор. Уговори се састоје из следећих секција (Влајић, 2019):

- **Операција** – назив операције и њени улазни и излазни аргументи
- **Веза са СК** - имена случајева коришћења у којима се позива ова системска операција
- **Предуслов** – пре извршења СО мотају бити испињени одређени предуслови
- **Постуслови** – после извршења СО у систему морају бити задовољени одређени постуслови

Фаза пројектовања – У овој фази се описује физичка структура и понашање софтверског система (архитектура). Пројектовање архитектуре представља трослојну архитектуру која обухвата: пројектовање корисничког интерфејса, апликационе логике и складишта података.

Фаза имплементације – У овој фази се креирају имеплемтационе компоненте које имплементирају у некој технологији. **Фаза тестирања** – У овој фази се тестирају све компоненте које су креиране у фази имплементације. За сваку имплементациону компоненту се пави тест случајева коришћења, тест процедуре и тест компоненте.

11 Прикупљање захтева

Прва фаза у развоју софтверског система Ларамановом методом је прикупљање захтева. Највећу пажњу треба усмерити ка овој фази зато што треба предупредити грешке и добро сагледати проблем. Сви пропусти који нису уочени на време могу да доведу до испољавања потенцијалних потешкоћа.

11.1 Вербални опис

Софтверски систем за набавку књига преко интернета представља систем преко којег администратори могу наручити књиге. На почетној страници администратори могу претраживати књиге које желе да наруче, убацују њихове жанрове, цену и број књига који желе да поруче. У систему постоје две улоге: корисник (купац) и администратор.

Администратор може да прегледа све поруџбине које које су корисници поручили и да девинише статус поруџбине. Такође може да уђе и да погледа сваку поруџбини посебно и то који је књигу корисник наручио, колико књига и која је укупна цена коју мора даплати.

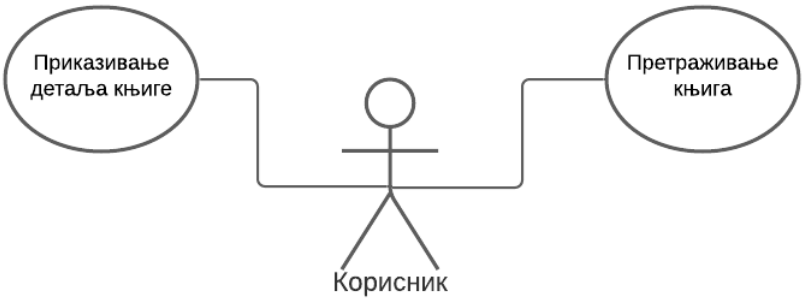
На почетној страници корисничког приказа сајта су приказане најновије књиге у књижари и може погледати најпопуларније књиге на сајту. Сам сајт омогућава корисницима да прегледају све књиге које су доступне у књижари, могу сортирати по различитим критеријума као и претраживати по називу књиге.

Сајт је направљен да корисницима олакша куповину књига, да на брз и ефикасан начин могу прегледати све књиге у књижари као и пратити статус наруџбеница у било ком моменту.

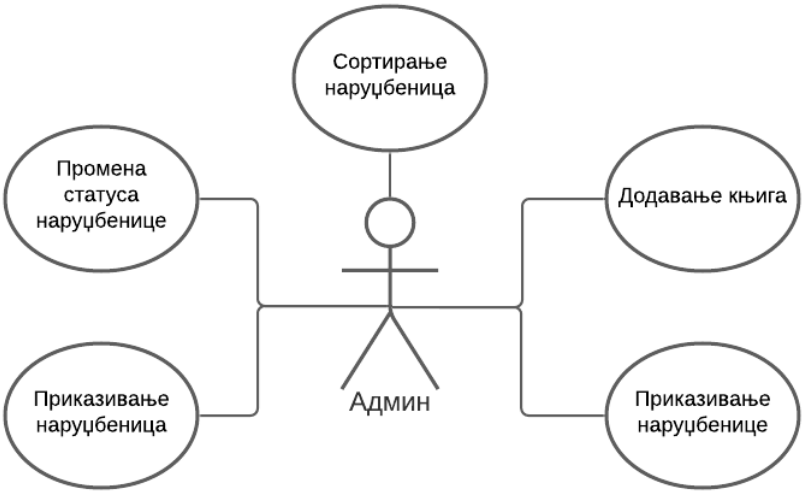
11.2 Опис захтева помоћу модела случаја коришћења

Издвајају се следећи случајеви коришћења:

1. Приказивање детаља књиге
2. Приказивање наруџбеница
3. Промена статуса наруџбеница
4. Додавање књига
5. Приказивање наруџбенице
6. Сортирање наруџбеница
7. Претраживање књига



Слика 12 - Случајеви коришћења корисник рола



Слика 13 - Случајеви коришћења админ рола

11.2.1 **СК1: Случај коришћења – Приказивање детаља књиге**

Назив СК

Приказивање детаља књиге

Актери СК

Корисник

Учесници СК

Корисник и систем

Предуслов: Кориснику је приказана листа књига.

Основни сценарио СК

1. Корисник **бира** књигу коју жели да му се прикаже. (АПУСО)
2. Корисник **позива** систем да му прикаже изабрану књигу. (АПСО)
3. Систем **тражи** књигу по задатој вредности. (СО)
4. Систем **приказује** кориснику књигу. (ИА)

Алтернативна сценарија

4.1. Уколико систем не може да пронађе књигу, кориснику се приказује порука : „Book doesn’t exist“ (ИА)

11.2.2 **СК2: Случај коришћења – Приказивање наруџбеница**

Назив СК

Приказивање наруџбеница

Актери СК

Админ

Учесници СК

Корисник и систем

Основни сценарио СК

1. Админ **позива** систем да прикаже све поружбине. (АПСО)
2. Систем **тражи** поруџбине. (СО)
3. Систем **приказује** админу све поруџбине. (ИА)

Алтернативна сценарија

3.1. Уколико систем не може да нађе поруџбине, админу се приказује порука : „There are no orders yet.“ (ИА)

11.2.3 **СКЗ: Случај коришћења – Промена статуса наруџбеница**

Назив СК

Промена статуса наруџбеница

Актери СК

Админ

Учесници СК

Админ и систем

Предуслов: Админ је регистрован на свој налог и приказана је листа свих поруџбина.

Основни сценарио СК

1. Админ **мења** статусе поруџбина. (АПУСО)
2. Админ **проверава** да ли је добро променио статусе поруџбина. (АНСО)
3. Админ **позива** систем да сачува статусе поруџбина. (АПСО)
4. Систем **памти** поруџбине. (СО)
5. Систем **приказује** админу поруку „Successfuly updated orders“. (ИА)

Алтернативна сценарија

5.1. Уколико систем не може да запамти поруџбине, админу се приказује порука: „Cannot update orders“ (ИА)

11.2.4 **СК4 : Случај коришћења – Додавање књига**

Назив СК

Додавање књига

Актери СК

Админ

Учесници СК

Админ и систем

Предуслов: Админу је регистрован на свој налог. Учитана је листа књига.

Основни сценарио СК

1. Админ **уноси** критеријум по коме претражује књиге. (АПУСО)
2. Админ **проверава** да ли је добро унео критеријум претраге. (АНСО)
3. Админ **позива** систем да претражи књиге по задатом критеријуму. (АПСО)
4. Систем **претражује** књиге по задатом критеријуму. (СО)
5. Систем **приказује** админу листу књига по задатом критеријуму. (ИА)
6. Админ **бира** књиге које жели да сачува. (АПУСО)
7. Админ **проверава** да ли је добро одабрао књиге. (АНСО)
8. Админ **позива** систем да сачува књиге. (АПСО)
9. Систем **памти** књиге. (СО)
10. Систем **приказује** админу поруку „There are no selected books“. (ИА)

Алтернативна сценарија

5.1. Уколико систем не може да нађе књиге по задатом критеријуму, админу се приказује порука: „System cannot find books“. Прекида се извршавање сценарија. (ИА)

10.1. Уколико систем не може да запамти књиге, админу се приказује порука: „System cannot save the books“ (ИА)

11.2.5 **СК5 : Случај коришћења – Приказивање наруџбенице**

Назив СК

Приказивање наруџбенице

Актери СК

Админ или Корисник

Учесници СК

Админ(Корисник) и систем

Предуслов: Админ(Корисник) је регистрован на свој налог. Учитана је листа наруџбеница.

Основни сценарио СК

1. Админ(Корисник) **бира** наруџбеницу коју жели да му се прикаже. (АПУСО)
2. Админ(Корисник) **позива** систем да прикаже ставке наруџбенице. (АПСО)
3. Систем **претражује** одабрану наруџбеницу. (СО)
4. Систем **приказује** админу(кориснику) ставке наруџбенице. (ИА)

Алтернативна сценарија

4.1. Уколико систем не може да нађе ставке наруџбенице, админу(кориснику) се приказује порука: „Cannot find order items“ (ИА)

11.2.6 **СК6 : Случај коришћења – Сортирање наруџбеница**

Назив СК

Сортирање наруџбеница

Актери СК

Админ

Учесници СК

Админ и систем

Предуслов: Админ је регистрован на свој налог. Учитана је листа наруџбеница.

Основни сценарио СК

1. Админ **уноси** критеријум сортирања наруџбеница. (АПУСО)
2. Админ **позива** систем да сортира наруџбенице. (АПСО)
3. Систем **сортира** наруџбенице. (СО)
4. Систем **приказује** админу наруџбенице. (ИА)

Алтернативна сценарија

4.1. Уколико систем не може да сортира наруџбенице, админу се приказује порука : „Cannot sort the orders“ (ИА)

11.2.7 **СК7: Случај коришћења – Претраживање књига**

Назив СК

Претраживање књига

Актери СК

Корисник

Учесници СК

Корисник и систем

Основни сценарио СК

1. Корисник **уноси** критеријум претраживања књига. (АПУСО)
2. Корисник **позива** систем да претражи књиге по неком критеријуму. (АПСО)
3. Систем **претражује** књиге. (СО)
4. Систем **приказује** кориснику књиге. (ИА)

Алтернативна сценарија

4.1. Уколико систем не може да претражи књиге, кориснику се приказује порука: „Cannot search the books“ (ИА)

12 Анализа

Фаза анализе описује логичку структуру и понашање софтверског система, тј. пословну логику софтверског система.

Понашање описујемо помоћу:

1. Системских дијаграма секвенци
2. Уговора о системским операцијама

Структуру описујемо помоћу:

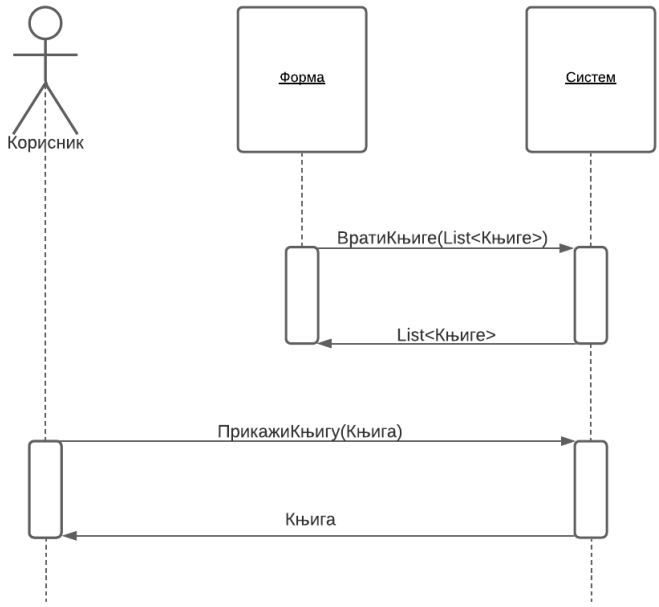
1. Концептуалног модела
2. Релационог модела

12.1 Понашање софтверског система – Системски дијаграми секвенци

Преко UML дијаграма секвенци се може описати понашање система. За сваки случај коришћења из фазе прикупљања захтева се приказује по један дијаграм секвенци. Он приказује догађаје у одређеном редоследу, који успостављају интеракцију између актора и софтверског система.

12.1.1 Дијаграм секвенци случајева коришћења – Приказивање детаља књиге

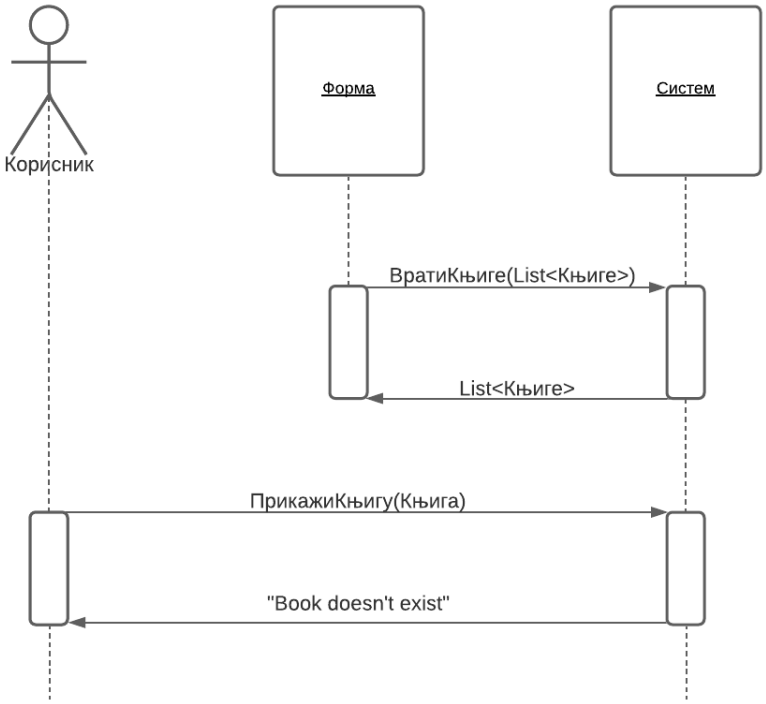
1. Форма **позива** систем да учита листу књига. (АПСО)
2. Систем **враћа** форми листу књига. (ИА)
3. Корисник **позива** систем да му прикаже књигу. (АПСО)
4. Систем **приказује** кориснику књигу. (ИА)



Слика 14 - ДС Приказивање детаља књиге

Алтернативна сценарија

4.1 Уколико систем не може да пронађе књигу приказује корисник поруку „Book doesn’t exist“



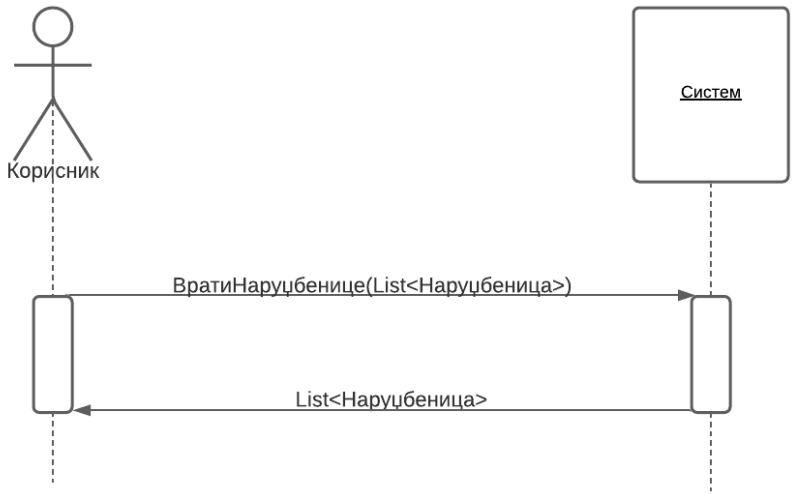
Слика 15 - ДС Књига не постоји

Са наведених дијаграма секвенци уочава се 2 системске операције:

1. Сигнал **ПрикажиКњигу**(Књига)
2. Сигнал **ВратиКњиге**(List<Књига>)

12.1.2 **Дијаграм секвенци случајева коришћења – Приказивање наруџбеница**

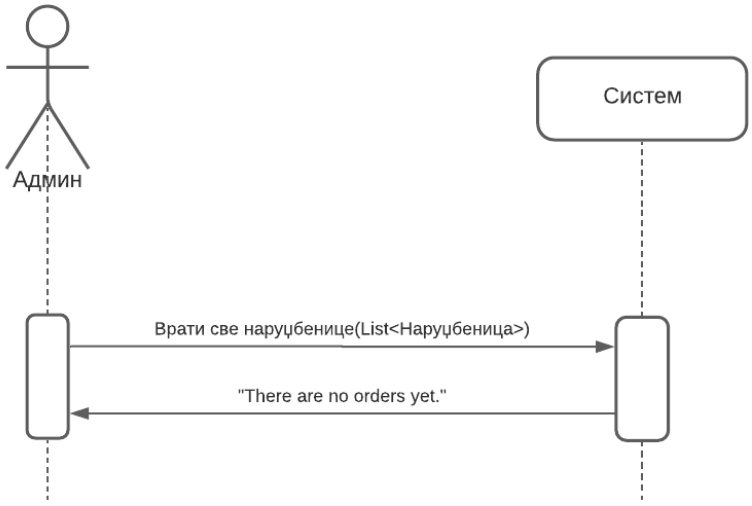
1. Админ позива систем да прикаже све поруџбине (АПСО)
2. Систем приказује админу све поруџбине (ИА)



Слика 16 - ДС Приказивање наруџбеница

Алтернативна сценарија

2.1 Уколико систем не може да нађе поруџбине, кориснику се приказује порука „There are no orders yet“



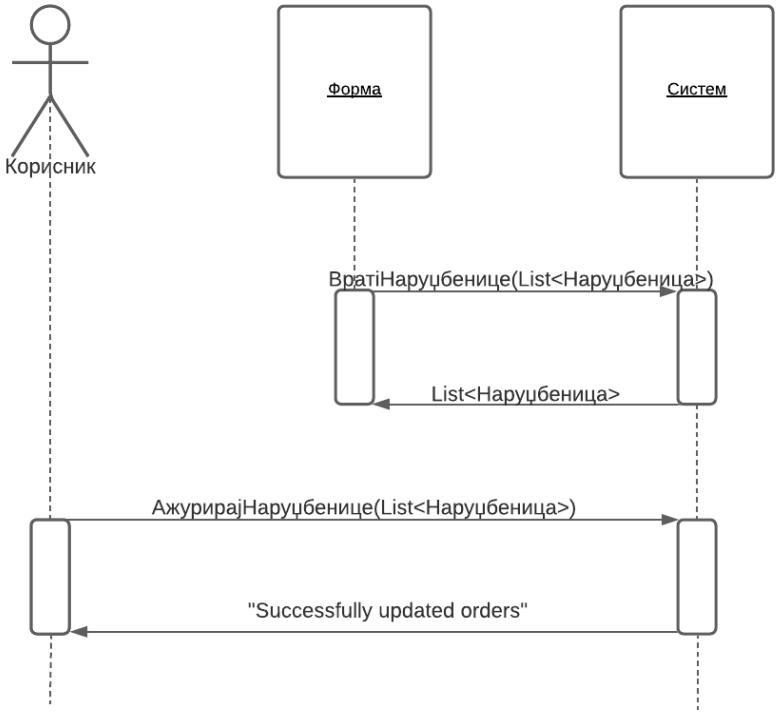
Слика 17 - ДС Не постоји ни једна поруџбина

Са наведених дијаграма секвенци уочава се 1 системска операција:

1. Сигнал **ВратиНаруџбенице**(List<Наруџбеница>)

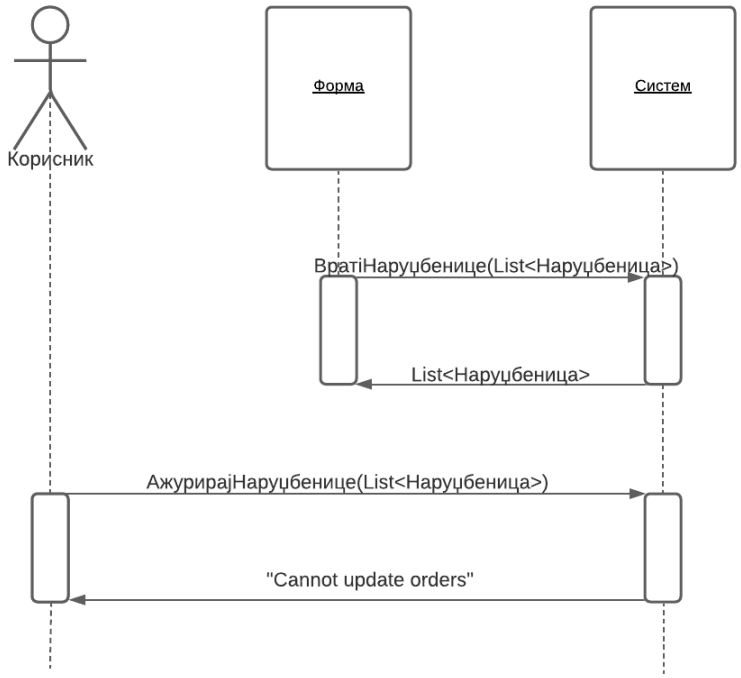
12.1.3 **Дијаграм секвенци случајева коришћења – Промена статуса наруџбеница**

1. Форма **позива** систем да учита све наруџбенице. (АПСО)
2. Систем **враћа** форми све наруџбенице. (ИА)
3. Админ **позива** систем да запамти статусе наруџбеница. (АПСО)
4. Систем **приказује** кориснику поруку „Successfully updated orders“. (ИА)



Слика 18 - ДС Промена статуса наруџбеница

4.1 Уколико систем не може да запамти поруџбине,админу се приказује порука „Cannot update orders“



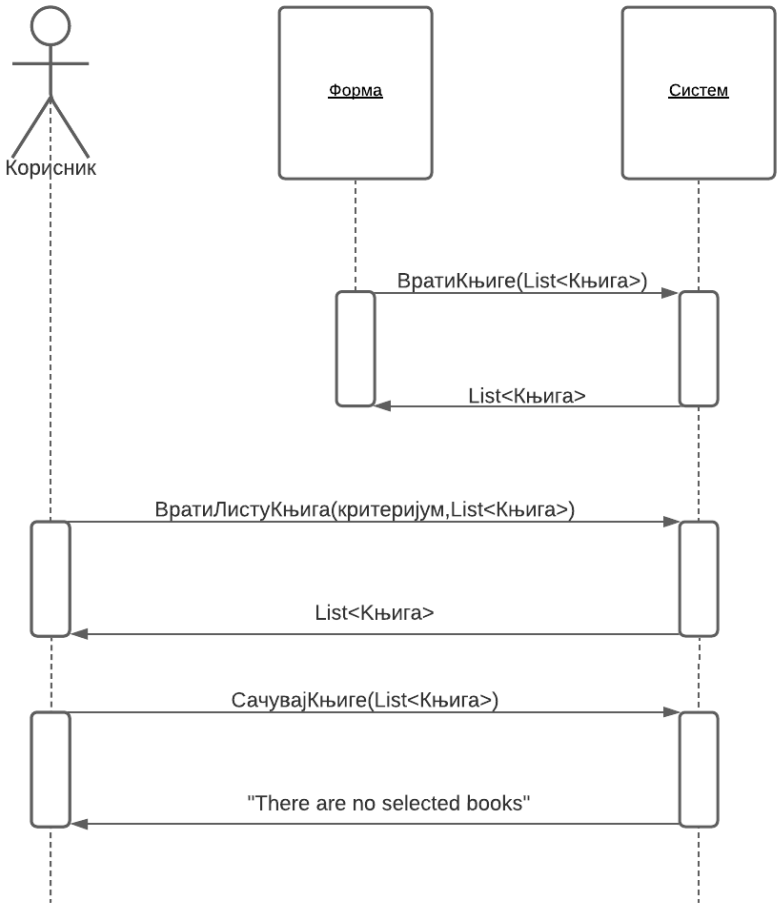
Слика 19 - ДС Наруџбеница се не може ажурирати

Са наведених дијаграма секвенци уочава се 2 системске операције:

1. Сигнал **АжурирајНаруџбенице**(List<Наруџбеница>)
2. Сигнал **ВратиНаруџбенице**(List<Наруџбеница>)

12.1.4 **Дијаграм секвенци случајева коришћења – Додавање књига**

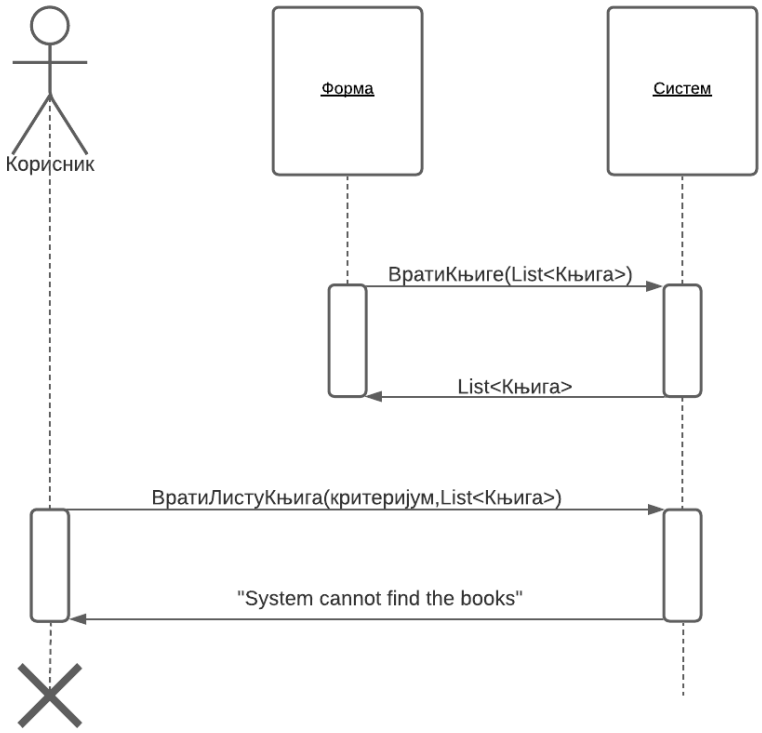
1. Форма **позива** систем да учита листу књига. (АПСО)
2. Систем **враћа** форми листу књига. (ИА)
3. Админ **позива** систем да претражи књиге по задатом критеријуму. (АПСО)
4. Систем **приказује** админу листу књига по задатом критеријуму. (ИА)
5. Админ **позива** систем да сачува књиге. (АПСО)
6. Систем **приказује** админу поруку „There are no selected books“. (ИА)



Слика 20 - ДС Додавање књига

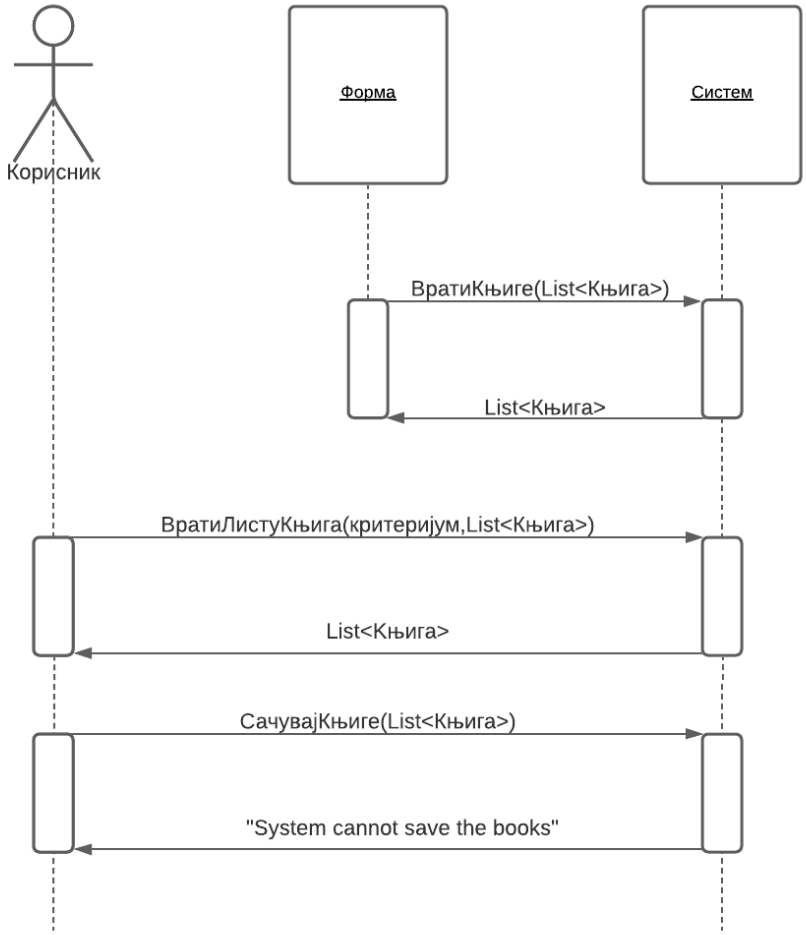
Алтернативна сценарија

4.1 Уколико систем не може да нађе књиге по задатом критеријуму, админу се приказује порука : „System cannot find the books“ (ИА)



Слика 21 - ДС Систем не може пронаћи књиге

6.1. Уколико систем не може да запамти књиге, админу се приказује порука : „System cannot save the books“ (ИА)



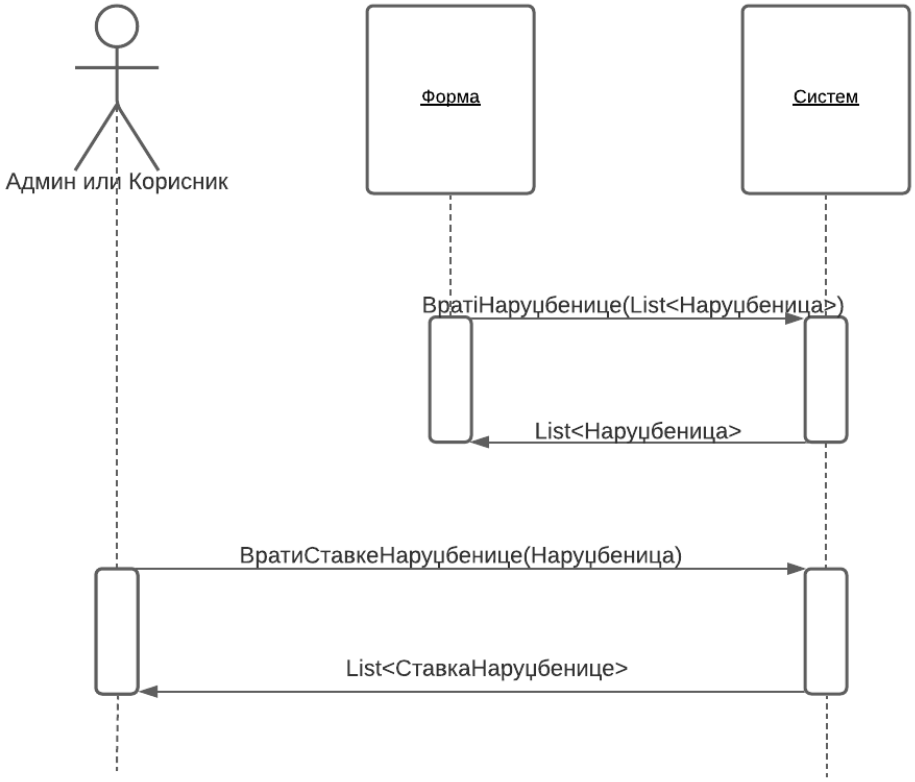
Слика 22 - ДС Систем не може сачувати књиге

Са наведених дијаграма секвенци уочава се 2 системске операције:

1. Сигнал **ВратиКњиге**(List<Књига>)
2. Сигнал **ВратиЛистуКњига**(критеријум,List<Књига>)
3. Сигнал **СачувајКњиге**(List<Књига>)

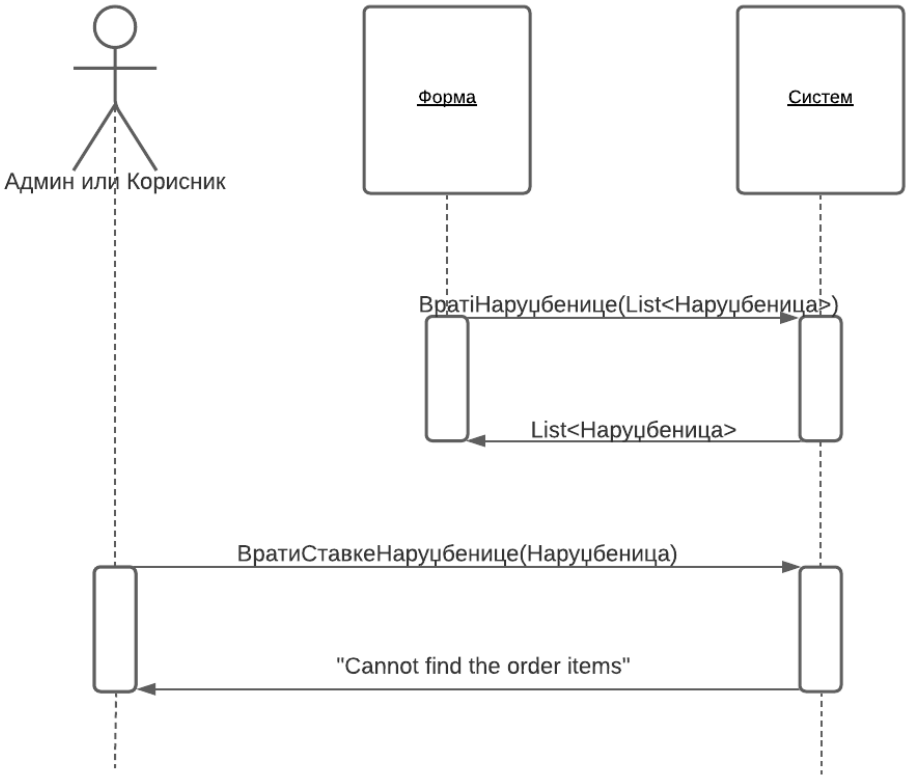
12.1.5 **Дијаграм секвенци случајева коришћења – Приказивање наруџбенице**

1. Форма **позива** систем да учита све наруџбенице. (АПСО)
2. Систем **враћа** форми све наруџбенице. (ИА)
3. Админ(Корисник) **позива** систем да врати ставке наруџбенице. (АПСО)
4. Систем **приказује** админу(кориснику) ставке одабране наруџбенице. (ИА)



Слика 23 - ДС Приказивање наруџбенице

4.1 Уколико систем не може да нађе ставке наруџбенице, приказује администу(кориснику) поруку „Cannot find the order items“



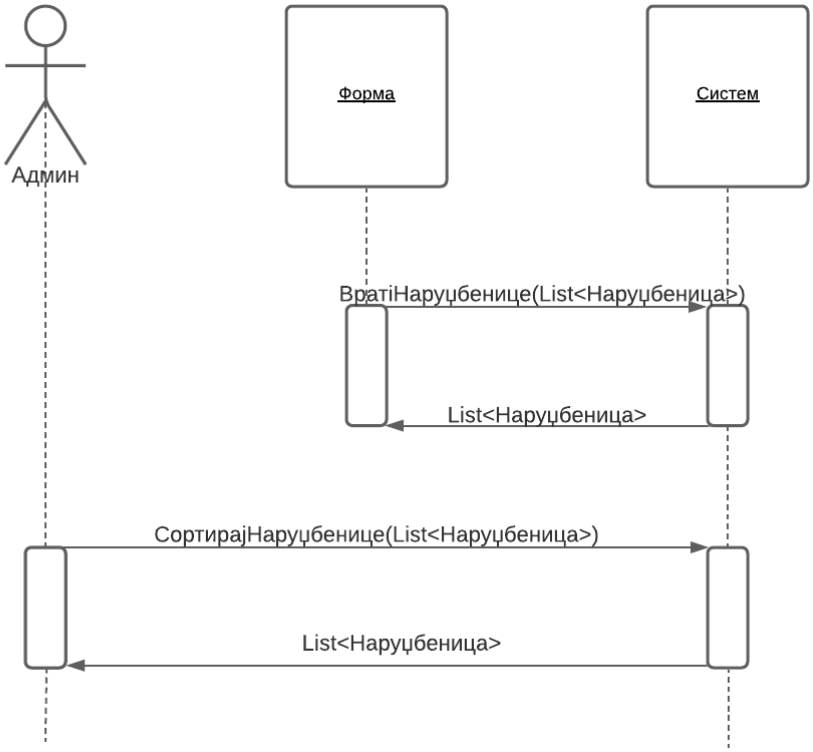
Слика 24 - ДС Ставке наруџбенице не постоје

Са наведених дијаграма секвенци уочава се 2 системске операције:

1. Сигнал **ВратиСтавкеНаруџбенице**(List<СтавкаНаруџбенице>)
2. Сигнал **ВратиНаруџбенице**(List<Наруџбеница>)

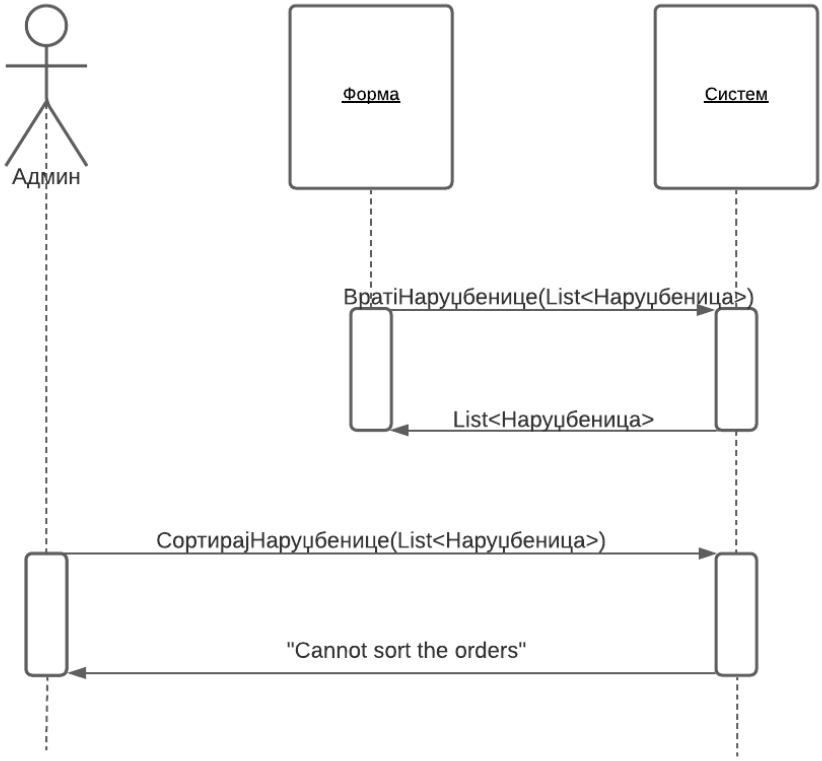
12.1.6 **Дијаграм секвенци случајева коришћења – Сортирање наруџбеница**

1. Форма позива систем да учита све наруџбенице. (АПСО)
2. Систем враћа све наруџбенице. (ИА)
3. Админ позива систем да сортира наруџбенице по неком критеријуму
4. Систем приказује админу сортиране наруџбенице



Слика 25 - ДС Сортирање наруџбеница

4.1 Уколико систем не може да сортира наруџбенице, приказује админупоруку „Cannot sort the orders“



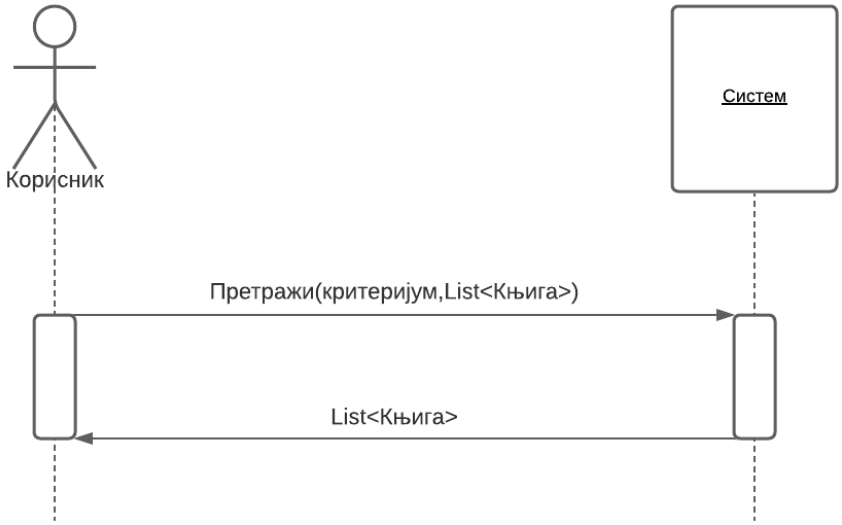
Слика 26 - ДС Наруџбенице се не могу сортирати

Са наведених дијаграма секвенци уочава се 2 системске операције:

1. Сигнал **СортирајНаруџбенице**(List<Наруџбеница>)
2. Сигнал **ВратиНаруџбенице**(List<Наруџбеница>)

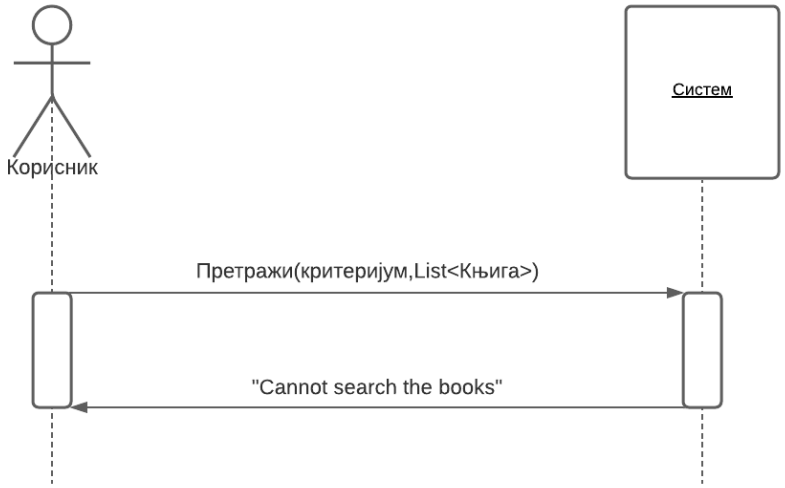
12.1.7 **Дијаграм секвенци случајева коришћења – Претраживање књига**

1. Корисник позива систем да претражи књиге по неком критеријуму (АПСО)
2. Систем приказује кориснику листу књига (ИА)



Слика 27 - ДС Претраживање књига

4.1 Уколико систем не може да нађе књиге по задатом критеријуму, приказује кориснику поруку„Cannot search the books“



Слика 28 - ДС Књиге се не могу претражити

Са наведених дијаграма секвенци уочава се 1 системска операција:

1. Сигнал **Претражи**(критеријум,List<Књига>)

Као резултат анализе сценарија добијено је укупно 9 системских операција које треба пројектовати:

1. ПрикажиКњигу(Књига)
2. ВратиКњиге(List<Књига>)
3. ВратиЛистуКњига(критеријум,List<Књига>)
4. ВратиНаруџбенице(List<Наруџбеница>)
5. АжурирајНаруџбенице(List<Наруџбеница>)
6. СачувајКњиге(List<Књига>)
7. ВратиСтавкеНаруџбенице(List<СтавкаНаруџбенице>)
8. СортирајНаруџбенице(List<Наруџбеница>)
9. Претражи(критеријум,List<Књига>)

2.2. Понашање софтверског система – Дефинисање уговора о системским операцијама

1. Уговор УГ1 : **ПрикажиКњигу**
   Операција: ПрикажиКњигу(Књига)
   Веза са СК: СК1
   Предуслови: /
   Постуслови: /

2. Уговор УГ2 : **ВратиКњиге**
   Операција: ВратиКњиге(List<Књига>)
   Веза са СК: СК1, СК4
   Предуслови: /
   Постуслови: /

3. Уговор УГ3 : **ВратиЛистуКњига**
   Операција: ВратиЛистуКњига(критеријум,List<Књига>)
   Веза са СК: СК4
   Предуслови:/
   Постуслови: /

4. Уговор УГ5 : **ВратиНаруџбенице**
   Операција: ВратиНаруџбенице(List<Наруџбеница>)
   Веза са СК: СК2, СК5, СК6
   Предуслови: /
   Постуслови: /

5. Уговор УГ6 : **АжурирајНаруџбенице**
   Операција:АжурирајНаруџбенице(List<Наруџбеница>)
   Веза са СК: СК3
   Предуслови: Вредносна и структурна ограничења над објектом Наруџбеница морају бити задовољена.
   Постуслови: Подаци о наруџбеницама су ажурирани.

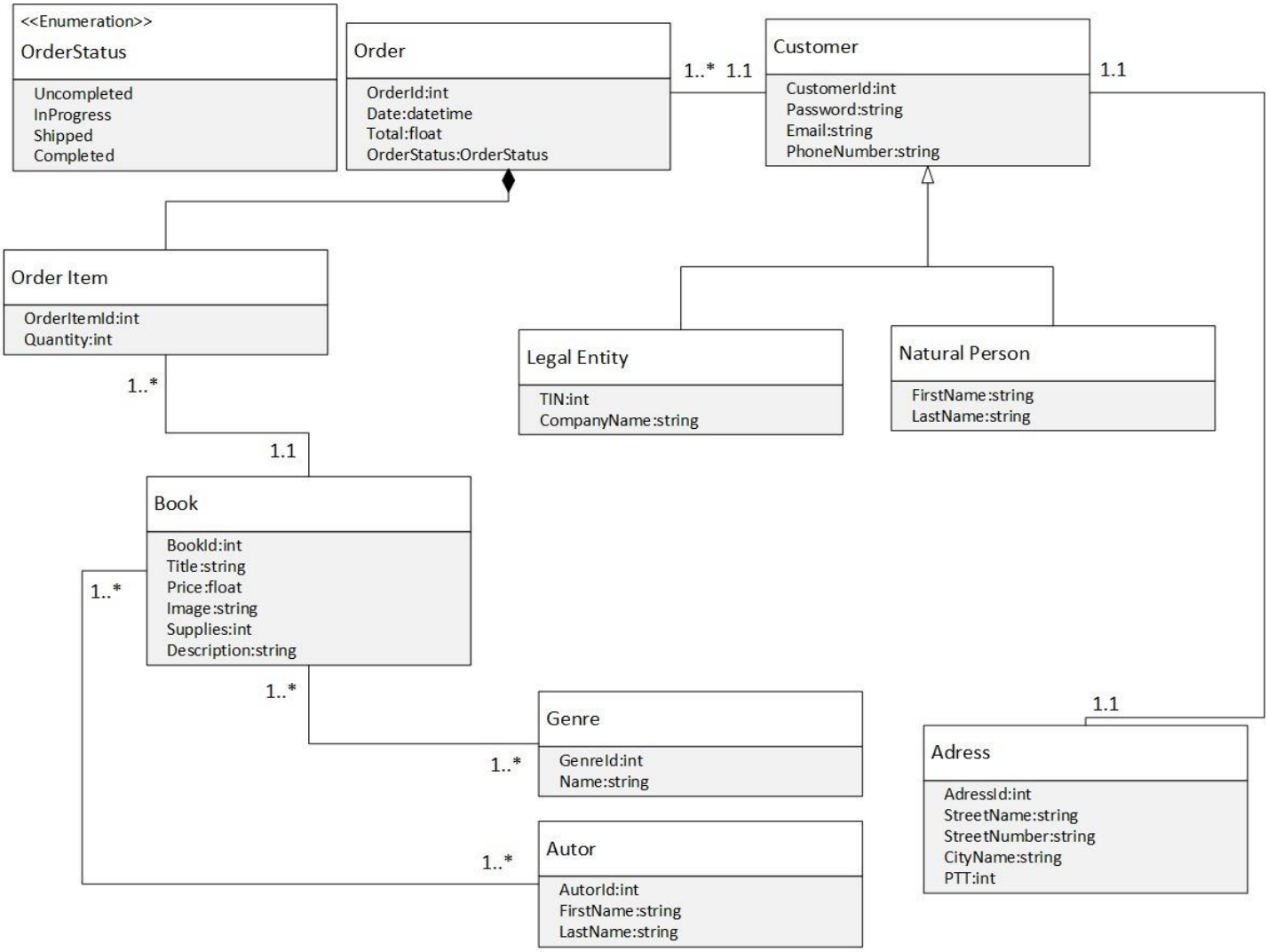
6. Уговор УГ7 : **СачувајКњиге**
   Операција: СачувајКњиге(List<Књига>)
   Веза са СК: СК4
   Предуслови: Вредносна и структурна ограничења над објектом Књига, Аутор и Жанр морају бити задовољена.
   Постуслови: Подаци о књизи, ауторима и занровима су сачувани.

7. Уговор УГ8 : **ВратиСтавкеНаруџбенице**
   Операција: ВратиСтавкеНаруџбенице(Наруџбеница)
   Веза са СК: СК5
   Предуслови: /
   Постуслови: /

8. Уговор УГ9 : **СортирајНаруџбенице**
   Операција: СортирајНаруџбенице(List<Наруџбеница>)
   Веза са СК: СК6
   Предуслови: /
   Постуслови: /

9. Уговор УГ10 : **Претражи**
   Операција: Претражи(критеријум,List<Књига>)
   Веза са СК: СК7
   Предуслови: /
   Постуслови: /

2.3 Структура софтверског система – Концептуални доменски модел



Слика 29 - Концептуални доменски модел

13 Пројектовање

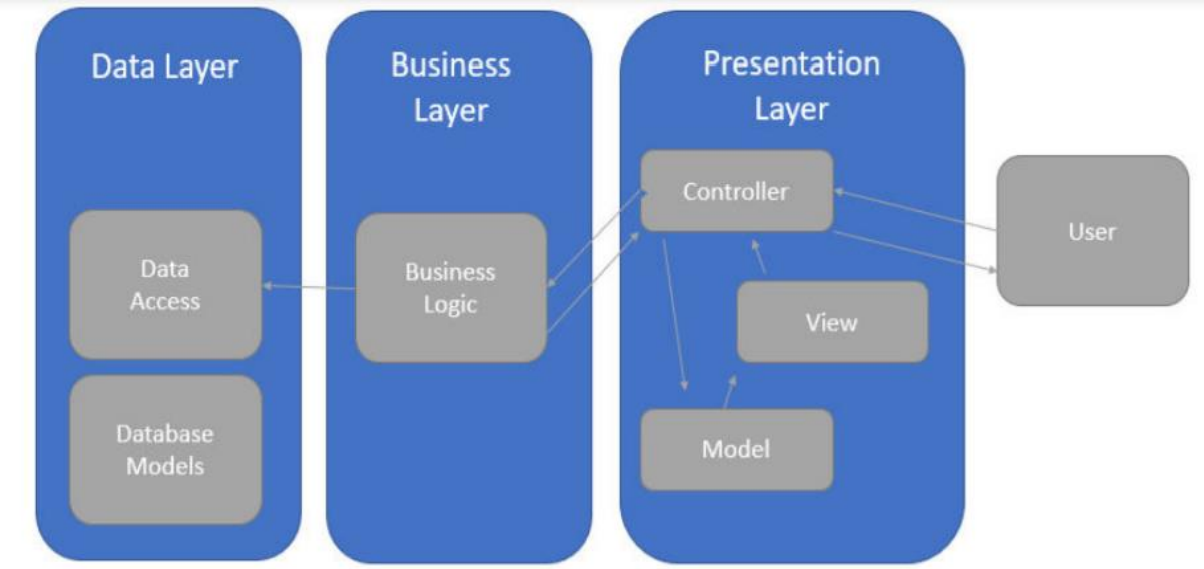
Фаза пројектовања описује физичку структуру и понашање софтверског система (архитектуру софтверског система). Најчешће коришћена архитектура је тронивојска архитектура која се састоји из следећих нивоа:

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

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

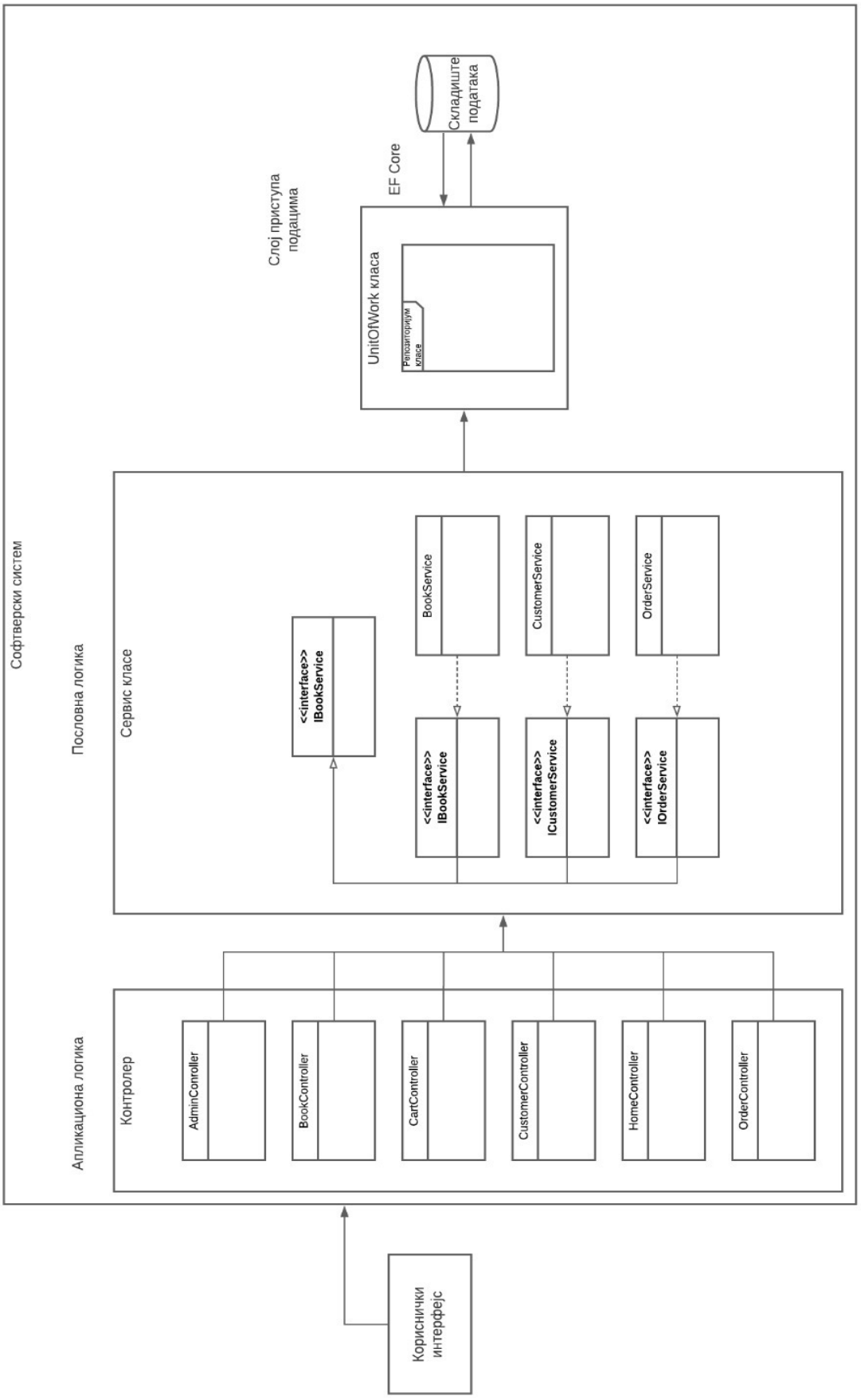
13.1 Архитектура софтверског система

Приликом пројектовања ASP .NET CORE MVC веб апликације, кориснички интерфејс представља део презентационог слоја у оквиру MVC патерна, апликациону логику чини пословна логика и слој приступа подацима остварен преко Entity Framework Core-а. Складиште података представља у овом случају SQL базу података.



Слика 30 - Архитектура ASP .NET Core MVC апликације.[12]

[12] [Digital image]. https://qph.fs.quoracdn.net/main-qimgc1bf4bd17f11d65cdfba817587ca52ea



Слика 31 - Архитектура софтверског система

До почетка фазе пројектовања, већ је дефинисана пословна логика софтверског система, односно његова логичка структура и понашање. У наставку ћемо пројектовати сваки од наведених елемената тронивојске архитектуре.

2.4 Пројектовање складишта података

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

Аутор (<u>идАутор</u>, Име, Презиме)
Књига (<u>идКњига</u>, Наслов, Слика, Цена, Залихе, Опис)
АуторКњига (<u>идАутор</u>, <u>идКњига</u>)
Жанр (<u>идЖанр</u>, Назив)
ЖанрКњига (<u>*идКњига*</u>, <u>*идЖанр*</u>)
Купац(<u>идКупац,</u> Шифра, Емаил, БројТелефона, Адреса_НазивУлице, Адреса_БројУлице, Адреса_НазивГрада, Адреса_ПТТ, ВерификациониКод, Статус, Админ)
ПравноЛице(<u>*идКупац,*</u> Тин, НазивКомпаније)
ФизичкоЛице(<u>*идКупац,*</u> Име, Презиме)
Наруџбеница(<u>идНаруџбенице</u>, Датум, Укупно, СтатусНаруџбенице, *идКупац*)
СтавкаНаруџбенице(<u>*идНаруџбенице*</u>, <u>идСтавка</u>, Количина, *идКњига*)

Табела Аутор		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Назив атрибута	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT / UPDATE CASCADES АуторКњига DELETE RESTRICTED АуторКњига
	идАутор	Integer	not null			
	Име	String	not null			
	Презиме	String	not null			

Табела 1 - Табела Аутор

Табела Књига		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Назив атрибута	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT/ UPDATE CASCADES АуторКњига ЖанрКњига UPDATE RESTRICT СтавкаНаруџбенице DELETE CASCADE АуторКњига ЖанрКњига DELETE RESTRICTED СтавкаНаруџбенице
	идКњига	Integer	not null			
	Наслов	String	not null			
	Слика	String	not null			
	Цена	Float	not null			
	Залихе	Integer	not null			
	Опис	String	not null			

Табела 2 - Табела Књига

Табела Жанр		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Назив атрибута	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT / UPDATE CASCADES ЖанрКњига DELETE RESTRICTED ЖанрКњига
	идЖанр	Integer	not null			
	Назив	String	not null			

Табела 3 - Табела Жанр

Табела АуторКњига		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Назив атрибута	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT RESTRICTED Аутор Књига UPDATE RESTRICTED Аутор Књига DELETE /
	идАутор	Integer	not null			
	идКњига	Integer	not null			

Табела 4 - Табела АуторКњига

Табела ЖанрКњига		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Назив атрибута	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT RESTRICTED Жанр Књига UPDATE RESTRICTED Жанр Књига DELETE /
	идКњига	Integer	not null			
	идЖанр	Integer	not null			

Табела 5 - Табела ЖанрКњига

Табела Купац		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Назив атрибута	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT / UPDATE CASCADE Наруџбеница ПравноЛице ФизичкоЛице DELETE CASCADE ПравноЛице ФизичкоЛице DELETE RESTRICTED Наруџбеница
	идКупац	Integer	not null			
	Шифра	String	not null			
	Емаил	String	not null			
	БројТелефона	String	not null			
	Адреса_НазивУлице	String	not null			
	Адреса_БројУлице	String	not null			
	Адреса_НазивГрада	String	not null			
	Адреса_ПТТ	Integer	not null			
	ВерификациониКод	BigInteger	not null			
	Статус	Bit	not null			
	Админ	Bit	not null			

Табела 6 - Табела Купац

Табела ПравноЛице		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Назив атрибута	Тип атрибута	Вредност атрибута	Међузав атрибута једне табеле	Међузав атрибута више табела	INSERT RESTRICTED Купац UPDATE RESTRICTED Купац DELETE CASCADE Купац
	идКупац	Integer	not null			
	ТИН	Integer	not null			
	НазивКомпаније	String	not null			

Табела 7 - Табела ПравноЛице

Табела ФизичкоЛице		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Назив атрибута	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT RESTRICTED Купац UPDATE RESTRICTED Купац DELETE CASCADE Купац
	идКупац	Integer	not null			
	Име	String	not null			
	Презиме	String	not null			

Табела 8 - Табела ФизичкоЛице

Табела Наруџбеница		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Назив атрибута	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT RESTRICTED Купац UPDATE CASCADE СтавкаНаруџбенице UPDATE RESTRICT Купац DELETE CASCADE СтавкаНаруџбенице
	идНаруџбеница	Integer	not null			
	Датум	Date	not null			
	Укупно	Float	not null			
	СтатусНаруџбенице	Integer	not null			
	идКупац	Integer				

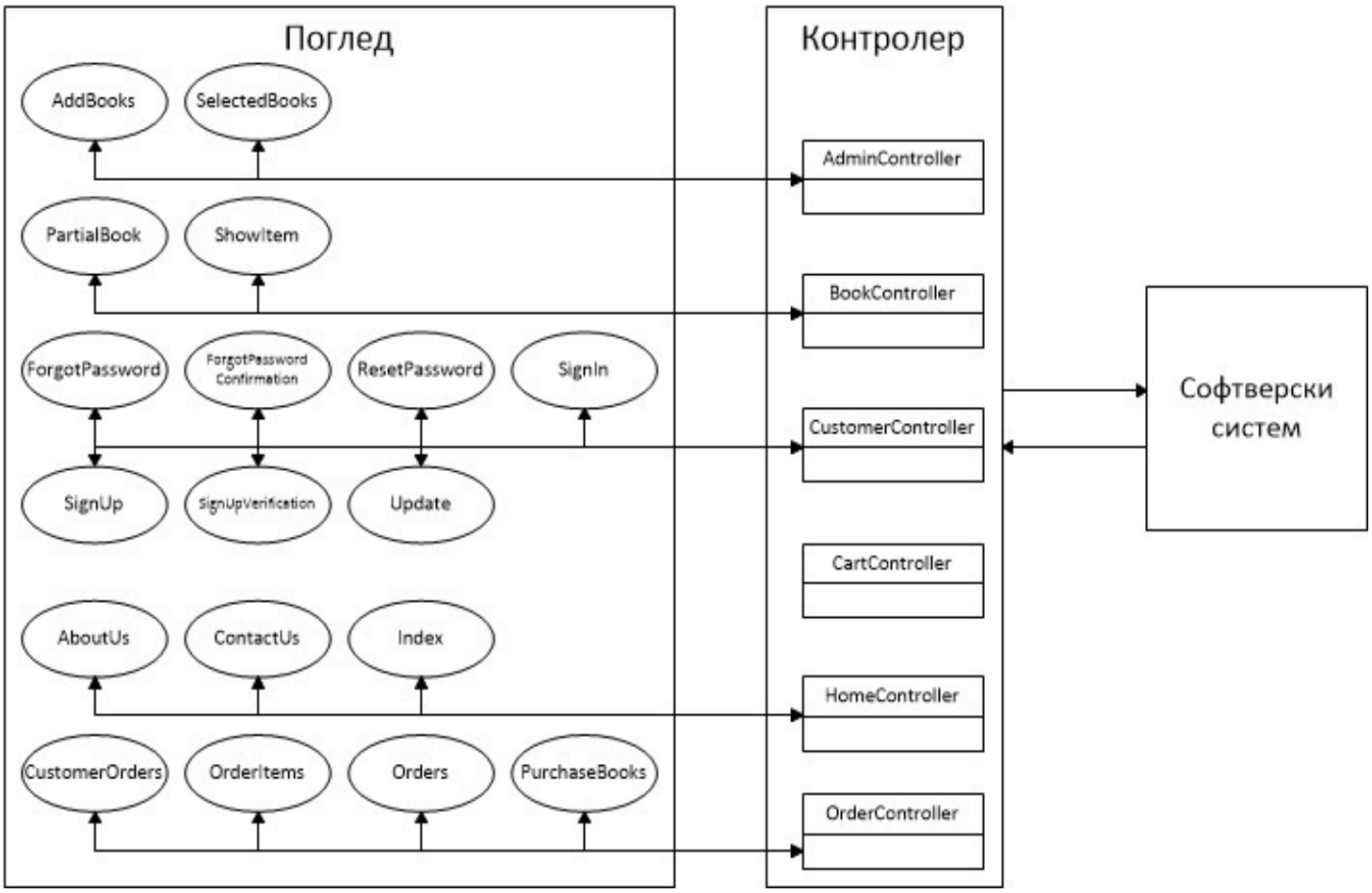
Табела 9 - Табела Наруџбеница

Табела СтавкаНаруџбенице		Просто вредносно ограничење		Сложено вредносно ограничење		Структурно ограничење
Атрибути	Назив атрибута	Тип атрибута	Вредност атрибута	Међузависност атрибута једне табеле	Међузависност атрибута више табела	INSERT RESTRICTED Наруџбеница UPDATE RESTRICTED Наруџбеница DELETE /
	идНаруџбеница	Integer	not null			
	идСтавка	Date	not null			
	Количина	Float	not null			
	идКњига	Integer	not null			

Табела 10 - Табела СтавкаНаруџбенице

13.2 Пројектовање екранских форми

Кориснички интерфејс представља реализацију улаза и/или излаза софтверског система. У овом раду структуру корисничког интерфејса чине веб странице које прихватају податке које корисник уноси, прослеђују их до одговарајућег контролера, а након што систем обради податке, приказује их кориснику.



Слика 32 - Повезивање контролера са корисничким интерфејсом у архитектури

13.2.1 **СК1 : Случај коришћења – Приказивање детаља књиге**

Назив СК

Приказивање детаља књиге

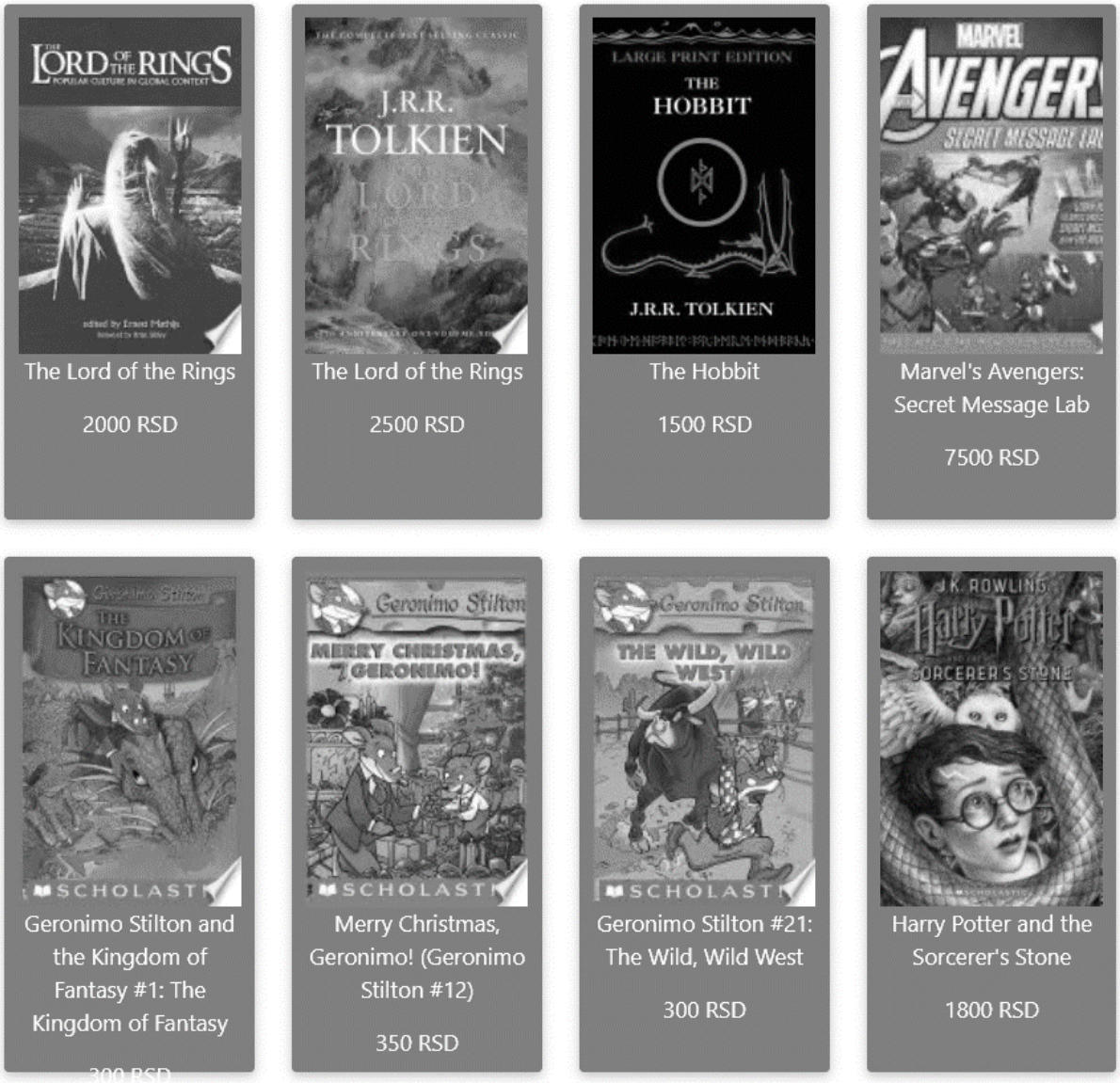
Актери СК

Корисник

Учесници СК

Корисник и систем

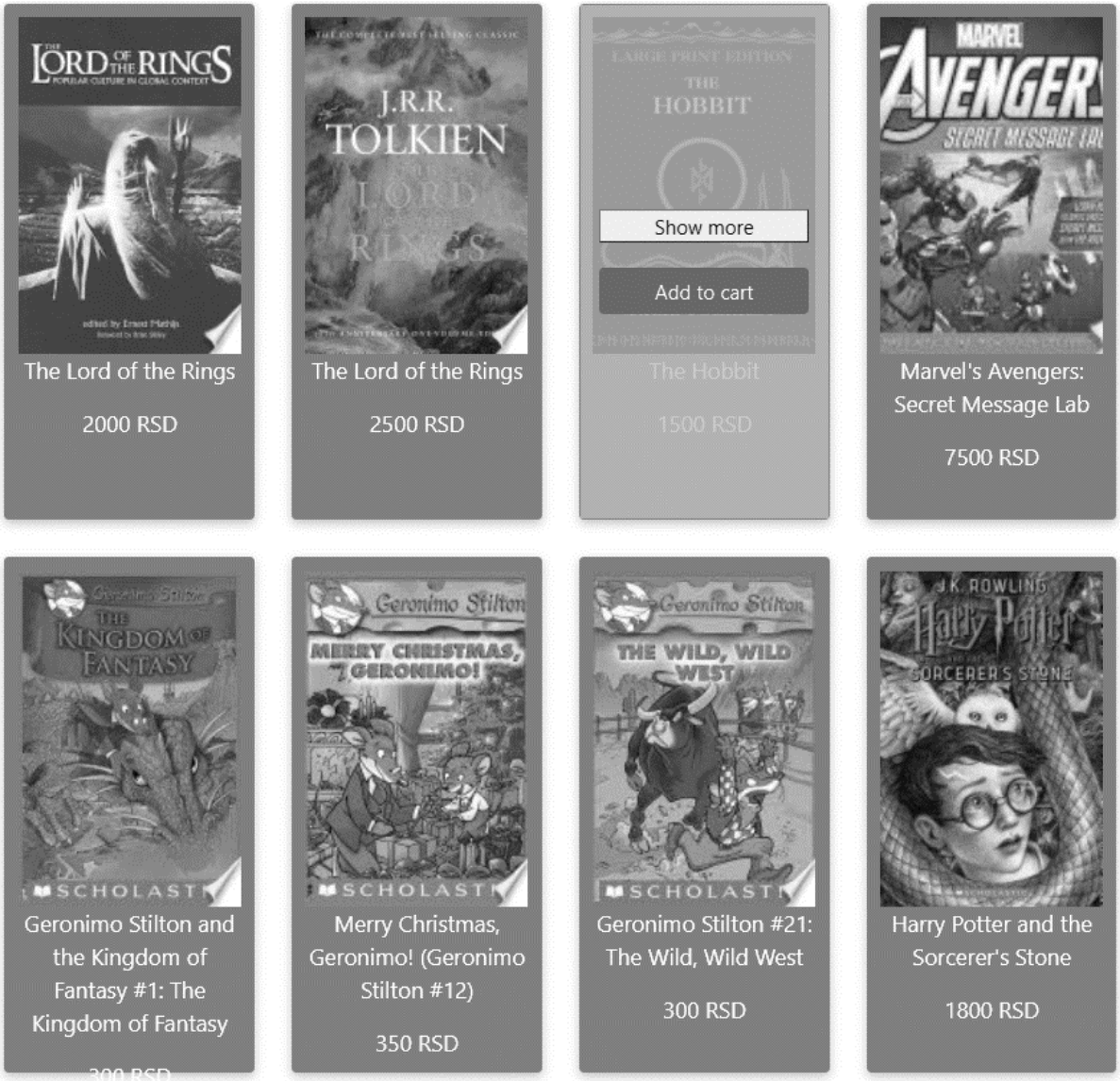
Предуслов: Кориснику је приказана листа књига.



Слика 33 - Листа књига

Основни сценарио СК

1. Корисник **уноси** књигу коју жели да му се прикаже. (АПУСО)



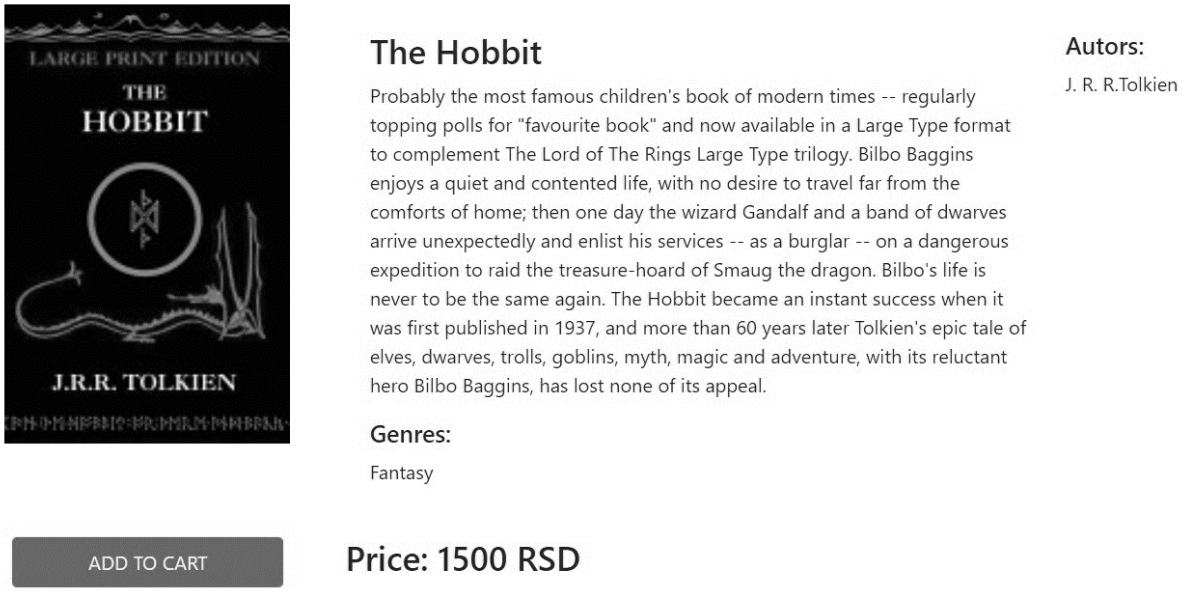
Слика 34 - Одабир књиге

2. Корисник **позива** систем да му прикаже изабрану књигу. (АПСО)

Опис акције: Корисник кликом на дугме „Show more“ позива системску операцију **ПрикажиКњигу(Књига)**

3. Систем **тражи** књигу по задатој вредности. (СО)

4. Систем **приказује** кориснику књигу. (ИА)



Слика 35 - Приказ књиге

Алтернативна сценарија

4.1. Уколико систем не може да пронађе књигу, кориснику се приказује порука : „Book doesn’t exist“ (ИА)

Слика 36 - Књига не постоји

13.2.2 СК2 : Случај коришћења – Приказивање наруџбеница

Назив СК

Приказивање наруџбеница

Актери СК

Админ

Учесници СК

Корисник и систем

Основни сценарио СК

1. Админ **позива** систем да прикаже све поружбине. (АПСО)

Слика 37 - Дугме за приступ страници за приказ поруџбина

2. Систем **тражи** поруџбине. (СО)
   Опис акције: Админ кликом на дугме менија „Orders“ позива системску операцију **ВратиНаруџбенице(List<Наруџбеница>)**
3. Систем **приказује** све поруџбине. (ИА)

Sort by

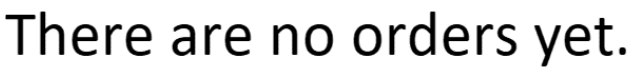
OrderNumber	Purchase date	Total	Customer	Order status	
1	24-Mar-21 1:23:37 PM	11500	Aleksa Miletić	Shipped	Show order items
2	01-Apr-21 4:21:02 PM	5183	Microsoft	Completed	Show order items
3	01-Apr-21 4:21:09 PM	6000	Microsoft	Shipped	Show order items
4	02-Apr-21 6:44:35 PM	11500	Marko Babovic	InProgress	Show order items
5	02-Apr-21 6:44:46 PM	12500	Marko Babovic	Completed	Show order items

Save changes

Слика 38 - Поруџбине

Алтернативна сценарија

3.1. Уколико систем не може да нађе поруџбине, админу се приказује порука : „There are no orders yet“ (ИА)



Слика 39 - Још увек нема поруџбина

13.2.3 СКЗ : Случај коришћења – Промена статуса наруџбеница

Назив СК

Промена статуса наруџбеница

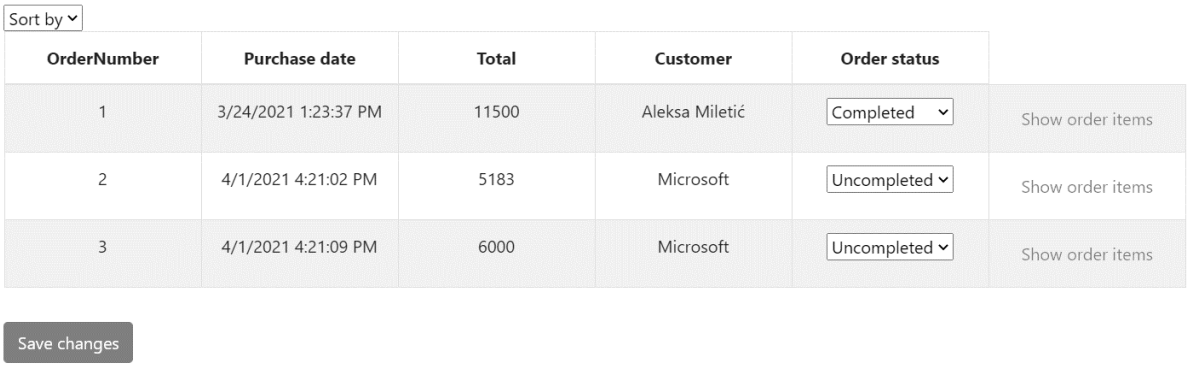
Актери СК

Админ

Учесници СК

Админ и систем

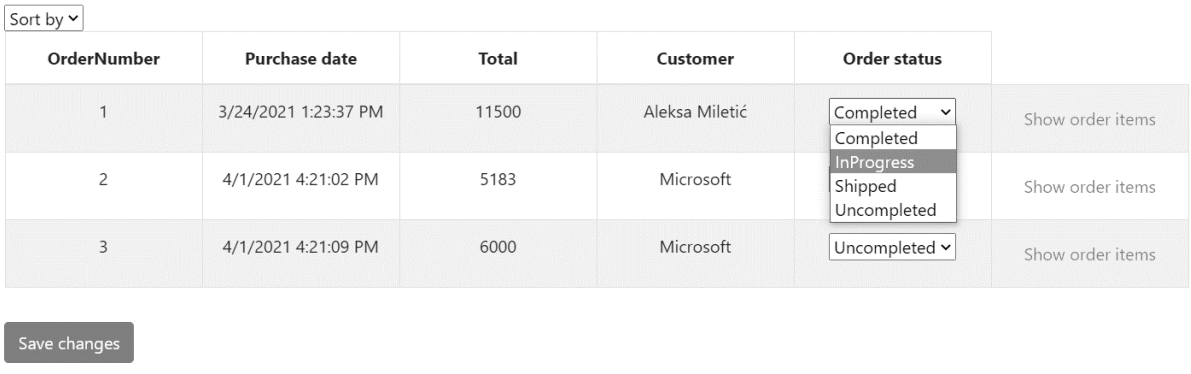
Предуслов: Админ је регистрован на свој налог и приказана је листа свих поруџбина.



Слика 40 - Табела наруџбеница

Основни сценарио СК

1. Админ **мења** статусе поруџбина. (АПУСО)



Слика 41 - Промена статуса поруџбине

2. Админ **проверава** да ли је добро променио статусе поруџбина. (АНСО)
3. Админ **позива** систем да сачува статусе поруџбина. (АПСО)
   Опис акције: Админ кликом на дугме „Save changes“ позива системску операцију **АжурирајНаруџбенице(List<Наруџбеница>)**
4. Систем **памти** поруџбине. (СО)
5. Систем **приказује** админу поруку „Successfuly updated orders“. (ИА)

Алтернативна сценарија

5.1. Уколико систем не може да запамти поруџбине, админу се приказује порука: „Cannot update orders“ (ИА)

Слика 42 - Поруџбина се не може ажурирати

13.2.4 **СК4 : Случај коришћења – Додавање књига**

Назив СК

Додавање књига

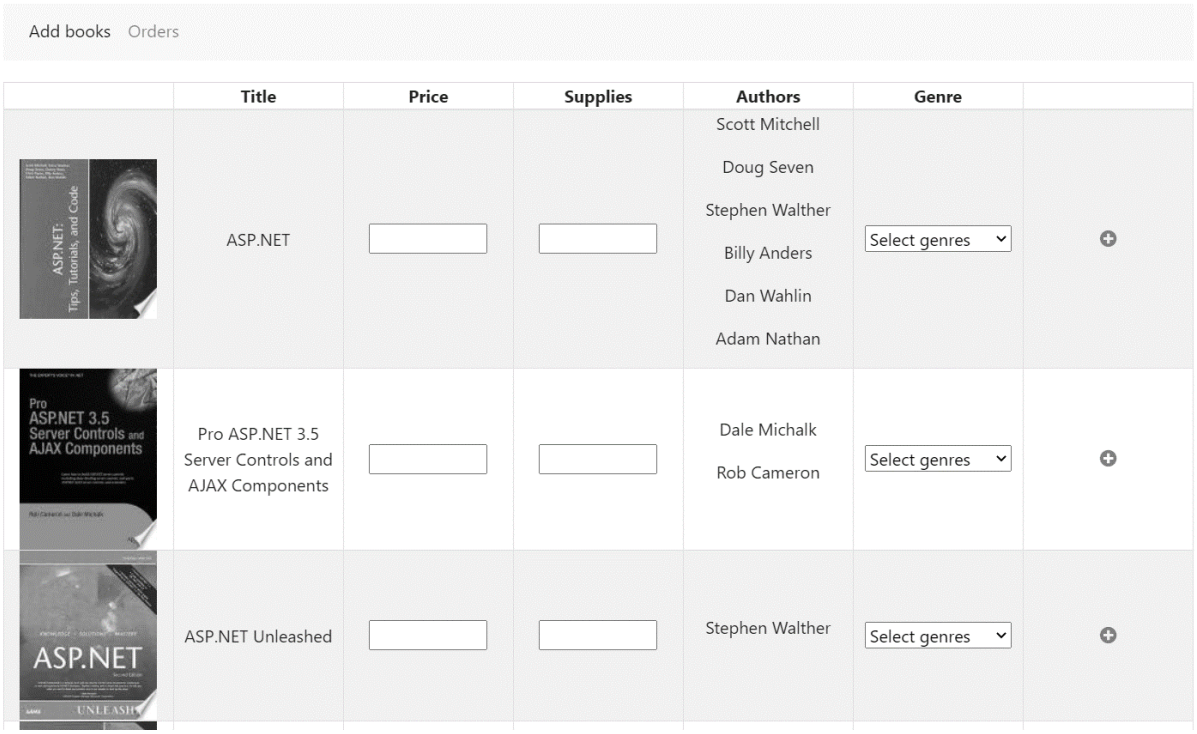
Актери СК

Администратор

Учесници СК

Администратор и систем

Предуслов: Администратор је регистрован на свој налог. Учитана је листа књига.



Слика 43 - Табела са приказаним књигама за поручивање

Основни сценарио СК

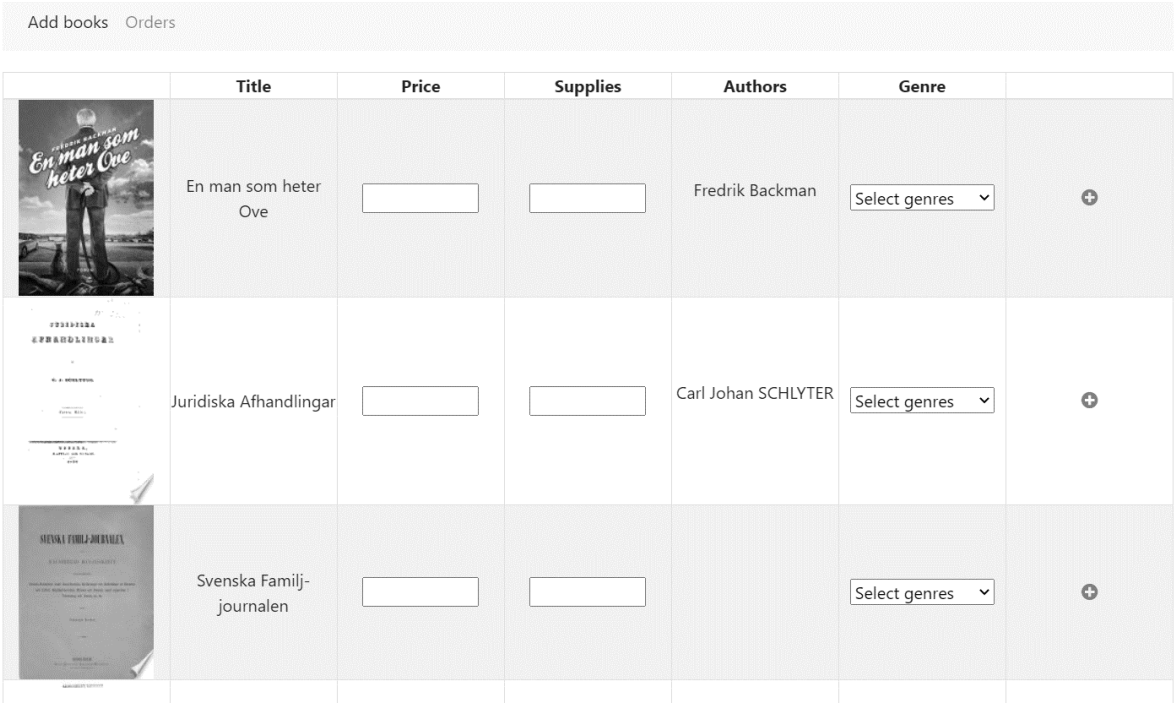
1. Администратор **уноси** критеријум по коме претражује књиге. (АПУСО)

Слика 44 - Форма за унос наслова књиге

2. Администратор **проверава** да ли је добро унео критеријум претраге. (АНСО)
3. Администратор **позива** систем да претражи књиге по задатом критеријуму. (АПСО)

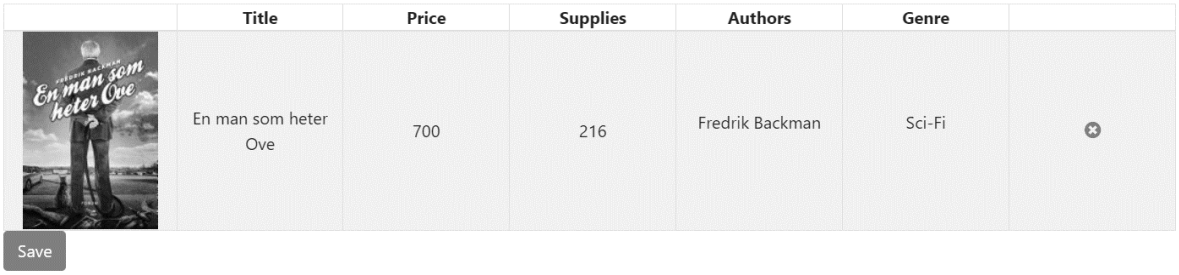
Опис акције: Админ кликом на дугме „Search“ позива системску операцију **ВратиЛистуКњига(критеријум,List<Књига>)**

4. Систем **претражује** књиге по задатом критеријуму. (СО)
5. Систем **приказује** админу листу књига по задатом критеријуму. (ИА)



Слика 45 - Приказане књиге након претраге

6. Админ **бира** књиге које жели да сачува. (АПУСО)



Слика 46 - Одабир књиге

7. Админ **проверава** да ли је добро одабрао књиге. (АНСО)
8. Админ **позива** систем да сачува књиге. (АПСО)
   Опис акције: Админ кликом на дугме „Save“ позива системску операцију **СачувајКњиге(List<Књига>)**
9. Систем **памти** књиге. (СО)
10. Систем **приказује** админу поруку „There are no selected books“. (ИА)

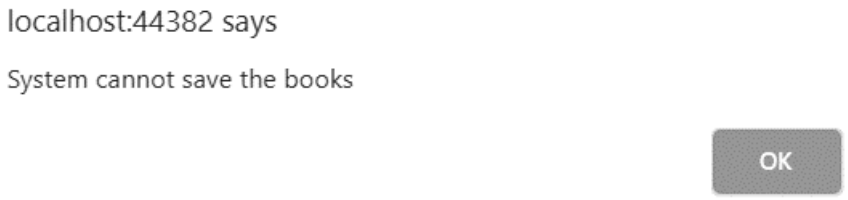
Слика 47 - Успешно унета књига

Алтернативна сценарија

5.1. Уколико систем не може да нађе књиге по задатом критеријуму, администратору се приказује порука : „System cannot find books“ (ИА)

Слика 48 - Систем не може пронаћи књиге

10.1. Уколико систем не може да запамти књиге, администратору се приказује порука : „System cannot save the books“ (ИА)



Слика 49 - Систем не може сачувати књиге

13.2.5 СК5 : Случај коришћења – Приказивање наруџбенице

Назив СК

Приказивање наруџбенице

Актери СК

Админ или Корисник

Учесници СК

Админ(Корисник) и систем

Предуслов: Админ(Корисник) је регистрован на свој налог. Учитана је листа наруџбеница.

OrderNumber	Purchase date	Total	Order status	
4	02-Apr-21 6:44:35 PM	11500	**Uncompleted**	Show order items
5	02-Apr-21 6:44:46 PM	12500	**Uncompleted**	Show order items

Слика 50 - Табела наруџбина

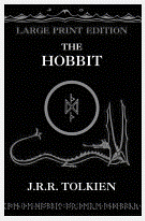
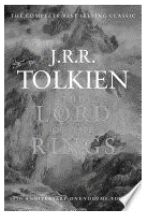
Основни сценарио СК

1. Админ(Корисник) **бира** наруџбеницу коју жели да му се прикаже. (АПУСО)

OrderNumber	Purchase date	Total	Order status	
4	02-Apr-21 6:44:35 PM	11500	**Uncompleted**	Show order items
5	02-Apr-21 6:44:46 PM	12500	**Uncompleted**	Show order items

Слика 51 - Дугме за приказ наруџбине

2. Админ(Корисник) **позива** систем да прикаже ставке наруџбенице. (АПСО)
   Опис акције: Админ кликом на линк „Show order items“ позива системску операцију **ВратиСтавкеНаруџбенице(Наруџбеница)**
3. Систем **претражује** одабрану наруџбеницу. (СО)
4. Систем **приказује** админу(кориснику) ставке наруџбенице. (ИА)

Image	Title	Price	Quantity	Total for this product
	The Hobbit	1500	1	1500
	The Lord of the Rings	2500	1	2500
	Marvel's Avengers: Secret Message Lab	7500	1	7500

Total price 11500 RSD

Слика 52 - Наруџбина

Алтернативна сценарија

4.1. Уколико систем не може да нађе ставке наруџбенице, админу(кориснику) се приказује порука : „“ (ИА)

Слика 53 - Ставке наруџбенице не постоје

13.2.6 **СК6 : Случај коришћења – Сортирање наруџбеница**

Назив СК

Сортирање наруџбеница

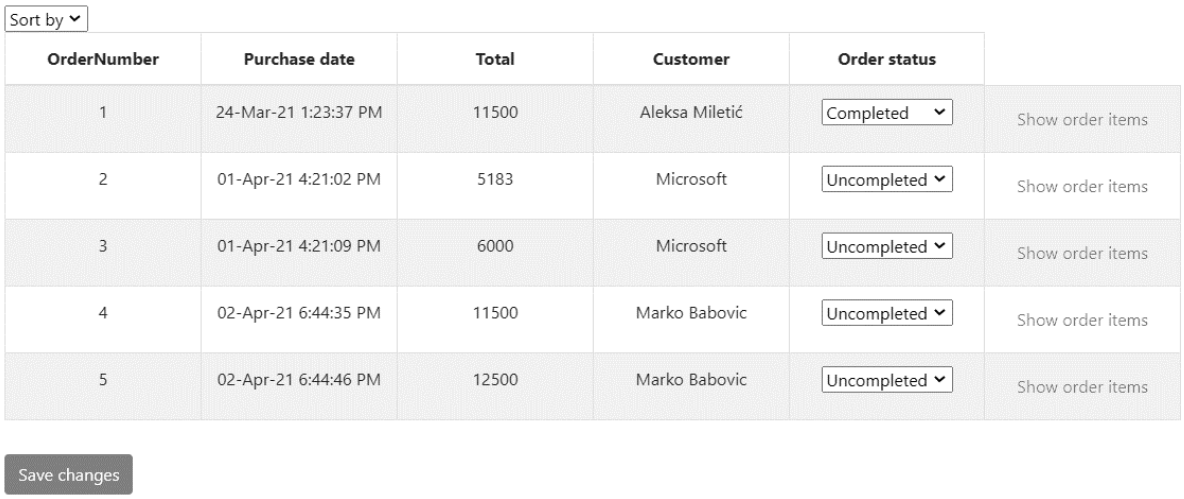
Актери СК

Админ

Учесници СК

Админ(Корисник) и систем

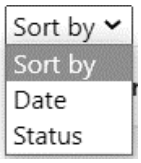
Предуслов: Админ је регистрован на свој налог. Учитана је листа наруџбеница.



Слика 54 - Табела наруџбеница

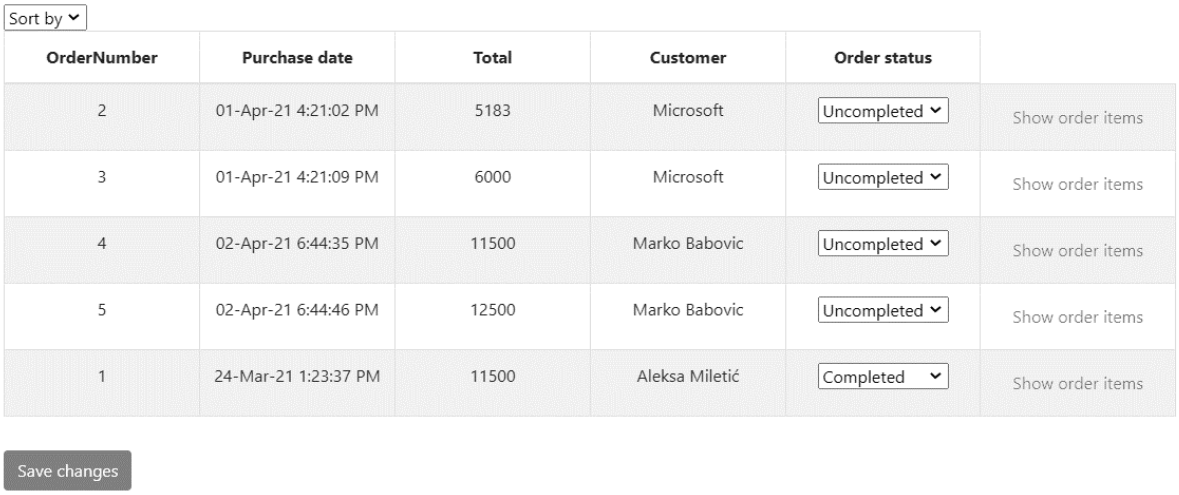
Основни сценарио СК

1. Админ(Корисник) **уноси** критеријум сортирања наруџбеница. (АПУСО)



Слика 55 - Мени за сортирање

2. Админ(Корисник) **позива** систем да сортира наруџбенице. (АПСО)
   Опис акције: Админ(Корисник) избором елемента из падајуће листе позива системску операцију **СортирајНаруџбенице(List<Наруџбеница>)**
3. Систем **сортира** наруџбенице. (СО)
4. Систем **приказује** админу наруџбенице. (ИА)



Слика 56 - Табела за приказ наруџбеница након сортирања

Алтернативна сценарија

4.1. Уколико систем не може да сортира наруџбенице, админу се приказује порука : „Cannot sort the orders“ (ИА)

Слика 57 - Наруџбенице се не могу сортирати

13.2.7 СК7: Случај коришћења – Претраживање књига

Назив СК

Претраживање књига

Актери СК

Корисник

Учесници СК

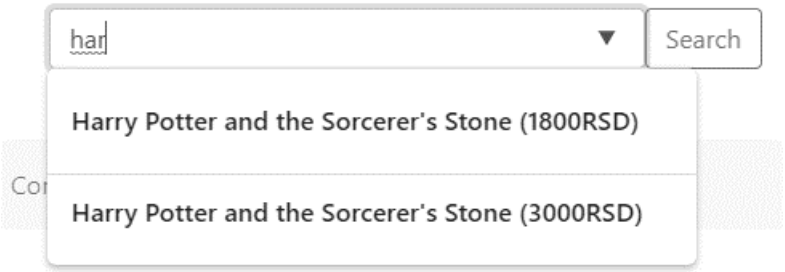
Корисник и систем

Основни сценарио СК

1. Корисник **уноси** критеријум претраживања књига. (АПУСО)

Слика 58 - Форма за унос наслова књиге

1. Корисник **позива** систем да претражи књиге по неком критеријуму. (АПСО)
   Опис акције: Кориснику се приликом куцања назива књиге позива ситемска операција **Претражи(критеријум, List<Књига>)**
2. Систем **претражује** књиге. (СО)
3. Систем **приказује** кориснику књиге. (ИА)



Слика 59 - Приказ пронађених књига

Алтернативна сценарија

4.1. Уколико систем не може да претражи књиге, кориснику се приказује порука: „Cannot search the books“ (ИА)

Слика 60 - Не постоји ни једна таква књига

13.3 Пројектовање апликационе логике

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

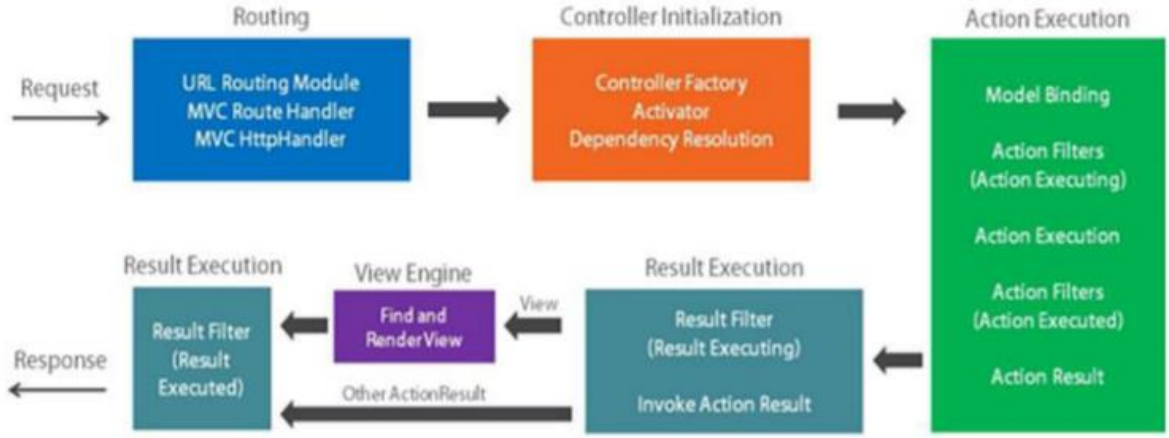
- **Контролери** – прихватају захтеве и прослеђују их сервисима на обраду
- **Сервиси** – обрађући захтеве који су пристигли, трансформише пристигле објекте у облик потребан за комуникацију са базом

Слој приступа подацима чине:

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

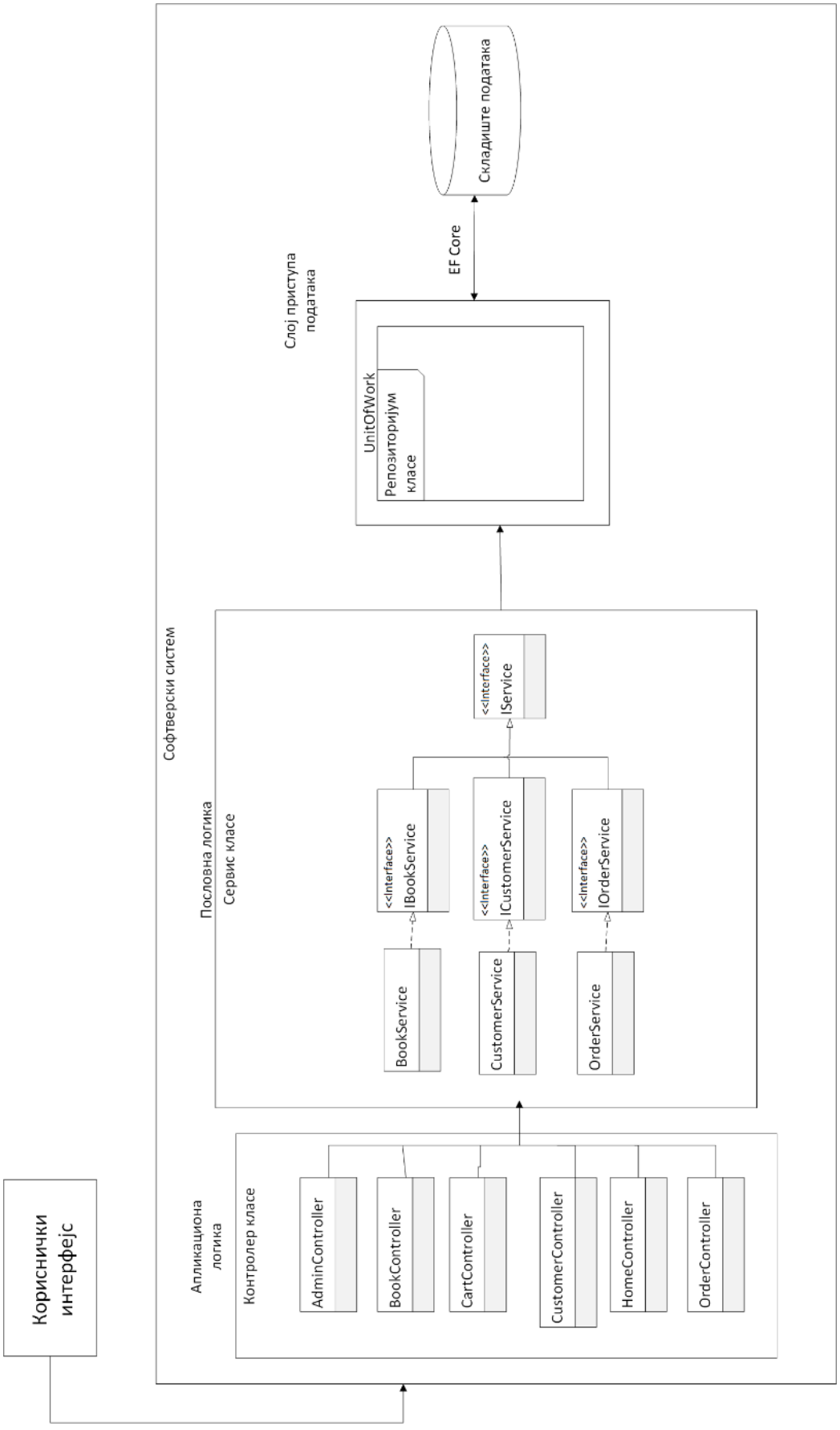
13.3.1 Контролер апликационе логике

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



Слика 61 - Ток корисничког захтева[13]

[13] [Digital image]. https://gwb.blob.core.windows.net/chetan/Windows-Live-Writer/1f4f73242d05_F000/image_thumb.png

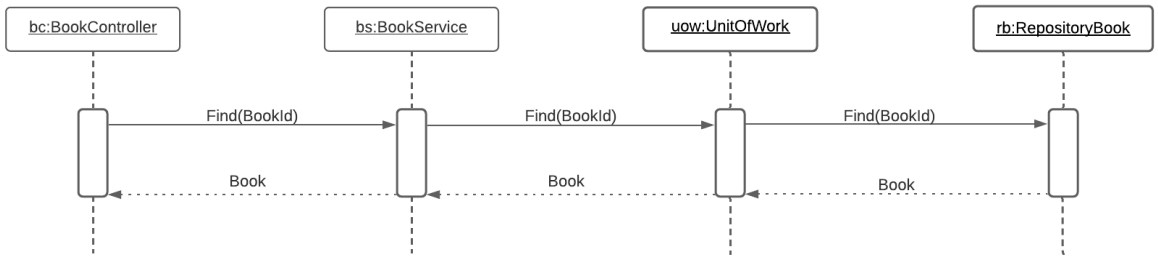


Слика 62 - Архитектура софтверског система након пројектовања контролера и класа које чине апликациону логику

13.3.2 **Пословна логика**

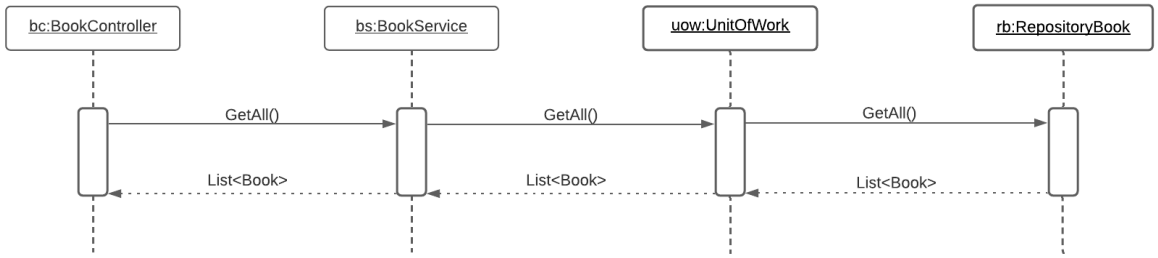
Пословна логика је описана структуром (доменским класама) и понашањем (системским операцијама). За сваки од уговора системских операција дефинисаних у фази анализе пројектује се концептуално решење.

1. Уговор УГ1 : **ПрикажиКњигу**
   Операција: ПрикажиКњигу(Књига)
   Веза са СК: СК1
   Предуслови: /
   Постуслови: /



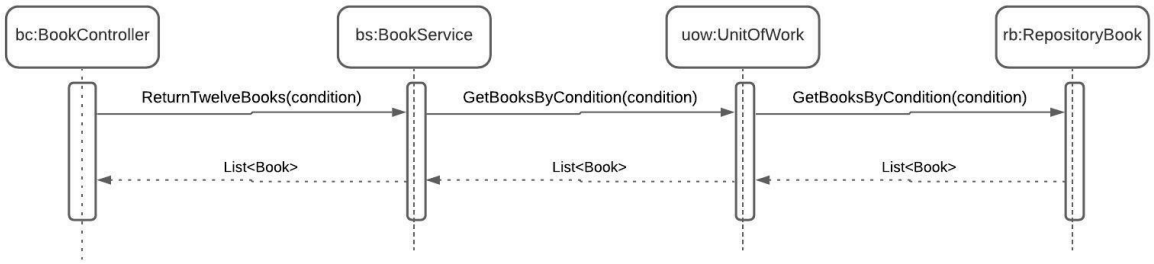
Слика 63 - Дијаграм секвенци: Уговор - ПрикажиКњигу

2. Уговор УГ2 : **ВратиКњиге**
   Операција: ВратиКњиге(List<Књига>)
   Веза са СК: СК1, СК4
   Предуслови: /
   Постуслови: /



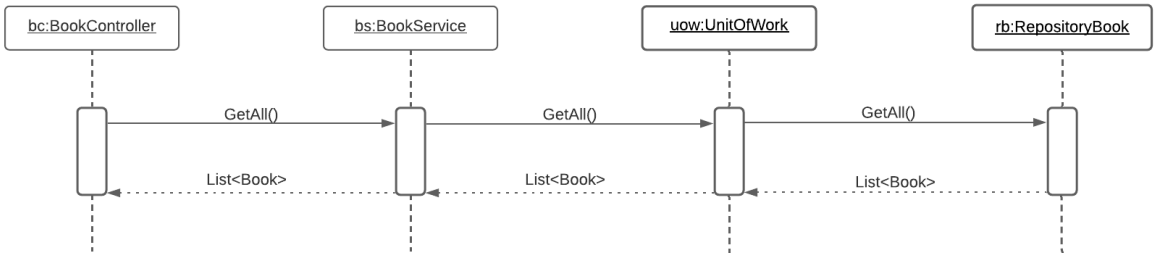
Слика 64 - Дијаграм секвенци: Уговор - ВратиКњиге

3. Уговор УГ3 : **ВратиЛистуКњига**
   Операција: ВратиЛистуКњига(критеријум, List<Књига>)
   Веза са СК: СК4
   Предуслови:/
   Постуслови: /



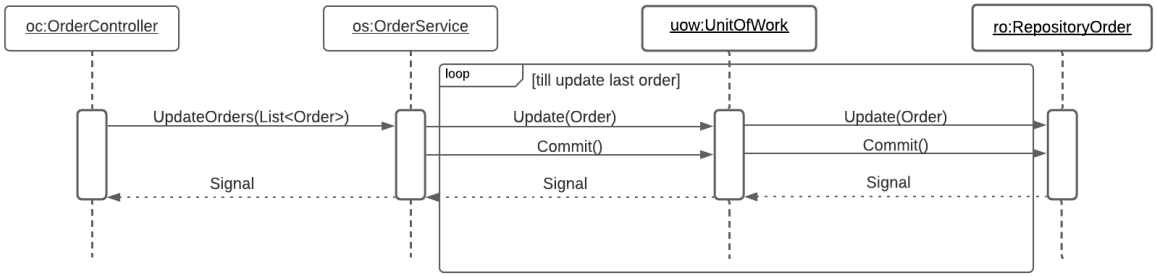
Слика 65 - Дијаграм секвенци: Уговор - ВратиЛистуКњига

4. Уговор УГ4 : **ВратиНаруџбенице**
   Операција: ВратиНаруџбенице(List<Наруџбеница>)
   Веза са СК: СК2, СК5, СК6
   Предуслови: /
   Постуслови: /



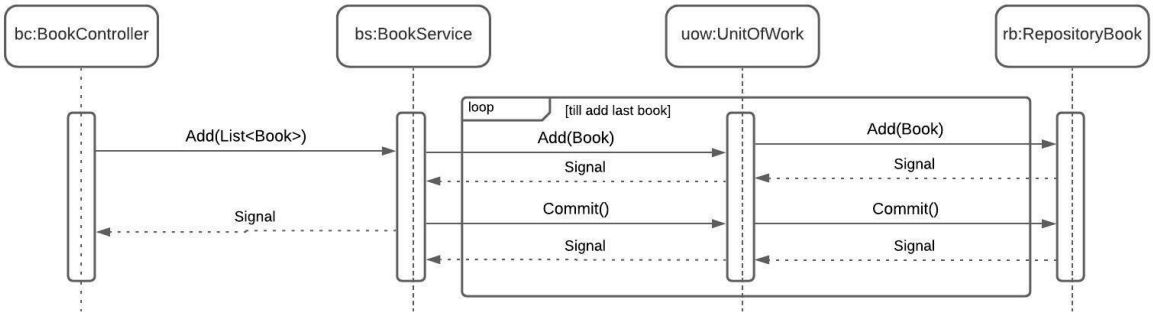
Слика 66 - Дијаграм секвенци: Уговор - ВратиНаруџбенице

5. Уговор УГ5 : **АжурирајНаруџбенице**
   Операција:АжурирајНаруџбенице(List<Наруџбеница>)
   Веза са СК: СК3
   Предуслови: Вредносна и структурна ограничења над објектом Наруџбеница морају бити задовољена.
   Постуслови: Подаци о наруџбеницама су ажурирани.



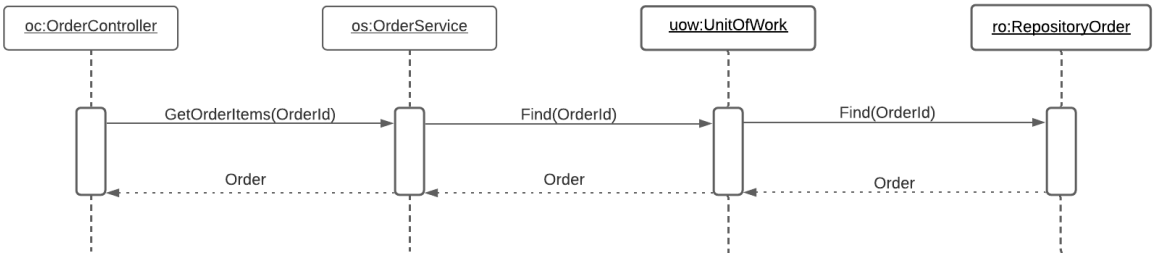
Слика 67 - Дијаграм секвенци: Уговор - АжурирајНаруџбенице

6. Уговор УГ6 : **СачувајКњиге**
   Операција: СачувајКњиге(List<Књига>)
   Веза са СК: СК4
   Предуслови: Вредносна и структурна ограничења над објектом Књига, Аутор и Жанр морају бити задовољена.
   Постуслови: Подаци о књизи, ауторима и занровима су сачувани.



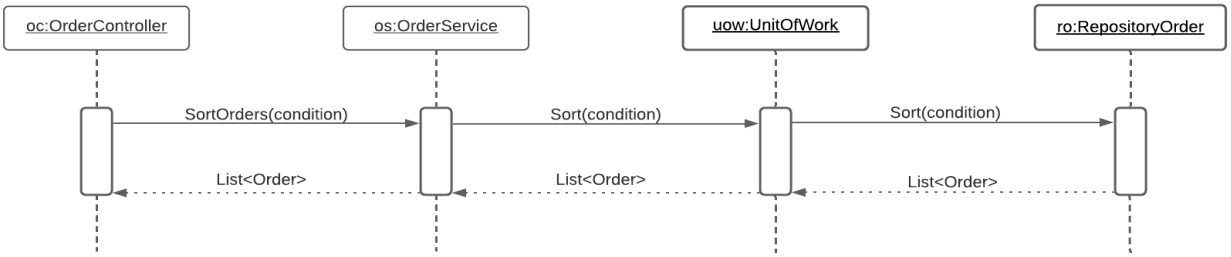
Слика 68 - Дијаграм секвенци: Уговор - СачувајКњиге

7. Уговор УГ7 : **ВратиСтавкеНаруџбенице**
   Операција: ВратиСтавкеНаруџбенице(Наруџбеница)
   Веза са СК: СК5
   Предуслови: /
   Постуслови: /



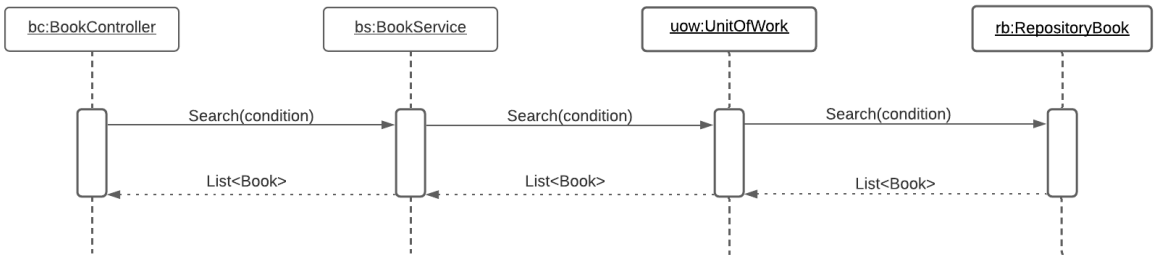
Слика 69 - Дијаграм секвенци: Уговор - ВратиСтавкеНаруџбенице

8. Уговор УГ8 : **СортирајНаруџбенице**
   Операција: СортирајНаруџбенице(List<Наруџбеница>)
   Веза са СК: СК6
   Предуслови: /
   Постуслови: /



Слика 70 - Дијаграм секвенци: Уговор - СортирајНаруџбенице

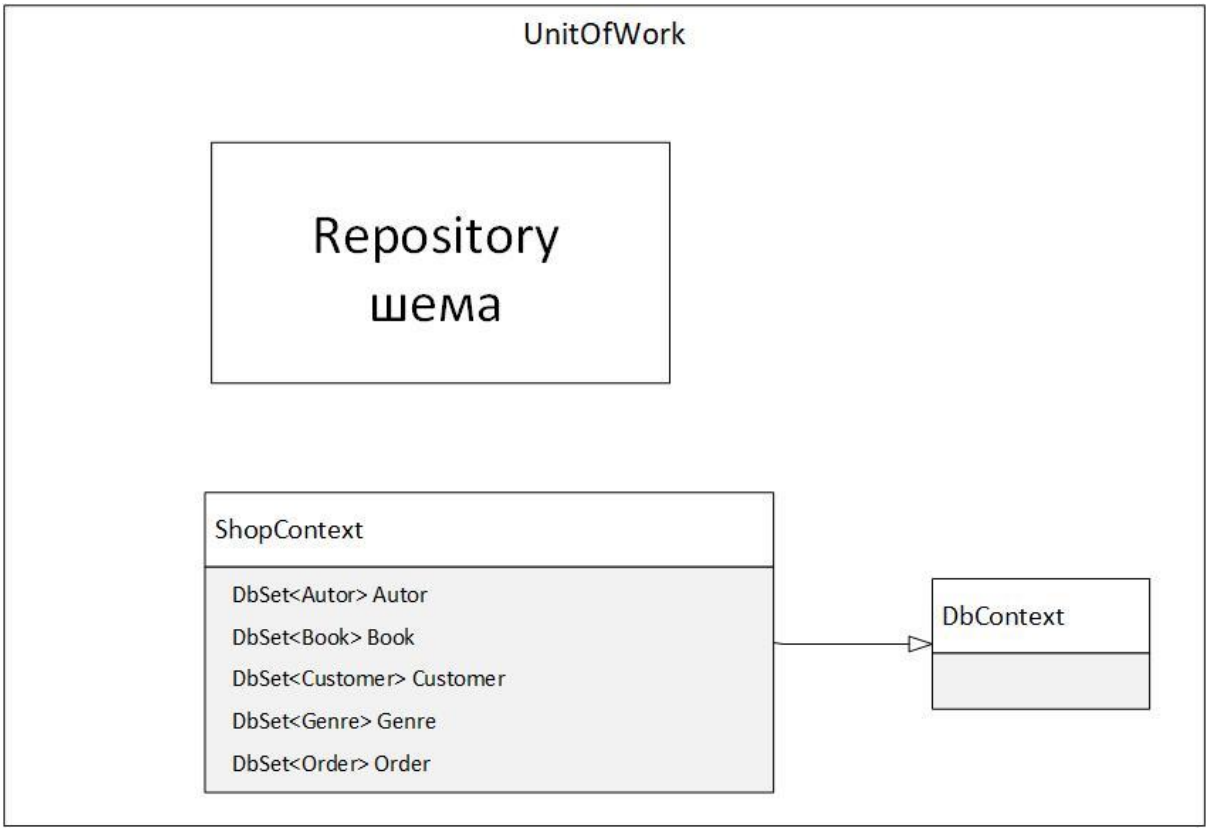
9. Уговор УГ9 : **Претражи**
   Операција: Претражи(критеријум,List<Књига>)
   Веза са СК: СК
   Предуслови: /
   Постуслови: /



Слика 71 - Дијаграм секвенци: Уговор - Претражи

Unit Of Work патерн и Repository патерн креирају апстрактни слој између слоја приступа података и пословне логике апликације. Примена ових шема може помоћи у изолацији апликације од промена у складишту података и може олакшати *аутоматско тестирање јединица (unit testing)*. (Microsoft, The Repository and Unit of Work Patterns, 2013)

Unit Of Work патерн и Repository патерн



Слика 72 - Unit Of Work шема

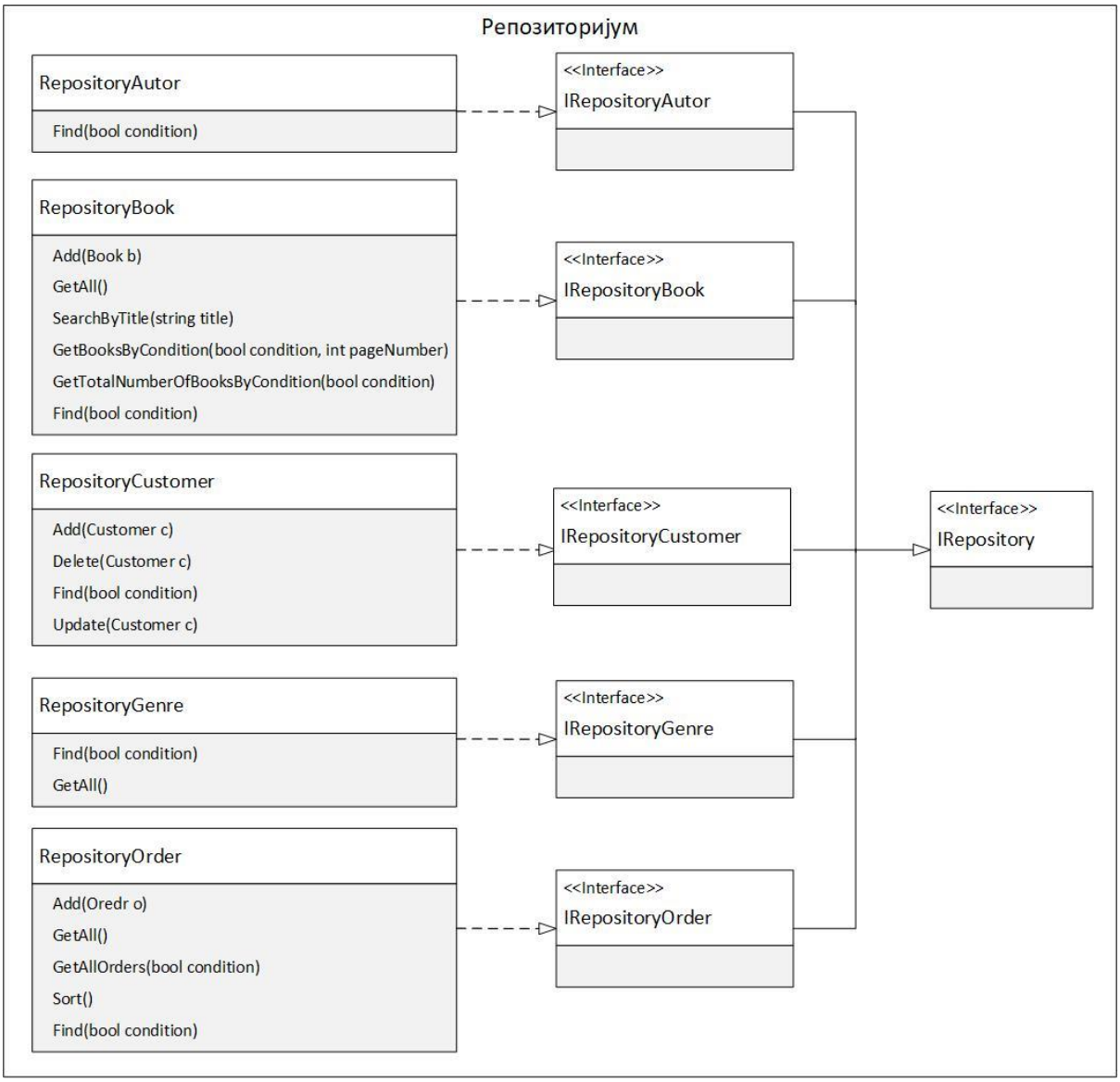
У слоју за приступ бази није пожељно обраћати се бази за сваку промену у подацима апликације и одговарајућем објектом моделу јер би такав приступ допринео стварању веома великог саобраћаја ка бази података и око сваког позива базе као што је отварање конекције, припрема пакета и слично. UnitOfWork као јединица рада представља логичку трансакцију која групише већи број позива ка бази. Када UnitOfWork јединица рада заврши са својим радом, може се позвати метода Commit() којом се потврђују промене у бази.

На овај начин повећава се ниво апстракције и одваја пословна логика од директног приступа подацима. UnitOfWork класа садржи референце ка Repository класама. У наставку је дат код којим се приказује имплементација овог патерна.

```
public class EShopUnitOfWork : IUnitOfWork
```

```
{
        private ShopContext context;
        public EShopUnitOfWork(ShopContext context)
        {
            this.context = context;
            RepostiryCustomer = new RepositoryCustomer(context);
            RepositoryBook = new RepositoryBook(context);
            RepositoryOrder = new RepositoryOrder(context);
            RepositoryGenre = new RepositoryGenre(context);
            RepositoryAutor = new RepositoryAutor(context);
        }
        public IRepositoryBook RepositoryBook { get ; set ; }
        public IRepositoryCustomer RepostiryCustomer { get; set; }
        public IRepositoryOrder RepositoryOrder { get; set; }
        public IRepositoryGenre RepositoryGenre { get; set; }
        public IRepositoryAutor RepositoryAutor { get; set; }
        public void Commit()
        {
            context.SaveChanges();
        }
}
```

Repository патерн



Слика 73 - Repository шема

За сваки тип у објектом моделу креирана је мапирајућа класа која имплементира интерфејс који приказује све операције над базом података које се могу извршити над типом. Овакве класе зову се репозиторијуми. Репозиторијум посредује између пословне логике и слоја приступа подацима, енкапсулира скуп објеката сачуваних на неком медијуму и дозвољене операције над њима, на објектно-оријентисани начин, такође одвајајући пословну логику од директног приступа подацима. На пример, пословна логика захтева методу GetAll() и очекује да добије све доступне објекте. Са друге стране, за више слојеве апликације није неопходно да имају информацију о томе да ли су ови објекти учитани из базе података или веб сервиса.

Основни интерфејс репозиторијума је генерички, чиме је омогућена једноставна и лака употреба у осталим деловима апликације. Конкретни интерфејси репозиторијума наслеђују базни интерфејс и допуњују га својим методама. Пракса је да се креира генеричка класа базна класа која имплементира заједничку функционалност за све

конкретне репозиторијуме. На крају, конкретна имплементациона класа наслеђује базну класу имплементације и имплементира конкретни интерфејс репозиторијума.

У наставку је дат код путем кога је имплементиран интерфејс репозиторијума:

```
public interface IRepository<T> where T : class
{
    void Add(T entity);
    List<T> GetAll();
    T Find(Predicate<T> p);
}
```

Интерфејс пружа стандардне методе за рад са ентитема користећи предности које доноси LINQ (Language Integrated Query). Главни задатак базне класе репозиторијума је да имплементира заједничку функционалност за све класе репозиторијума. За сваки ентитет имплементирана је класа репозиторијума. У наставку дат је код путем кога су имплементиране репозиторијум класе и одговарајући интерфејс за доменску класу Order. Аналогно су имплементиране и за све остале доменске класе. Класа RepositoryOrder имплементира интерфејс IRepositoryOrder. Поред имплементираних метода, конструктор класе прима контекст класу као параметар. Инстанца ове класе омогућава основне операције над жељеним ентитетом.

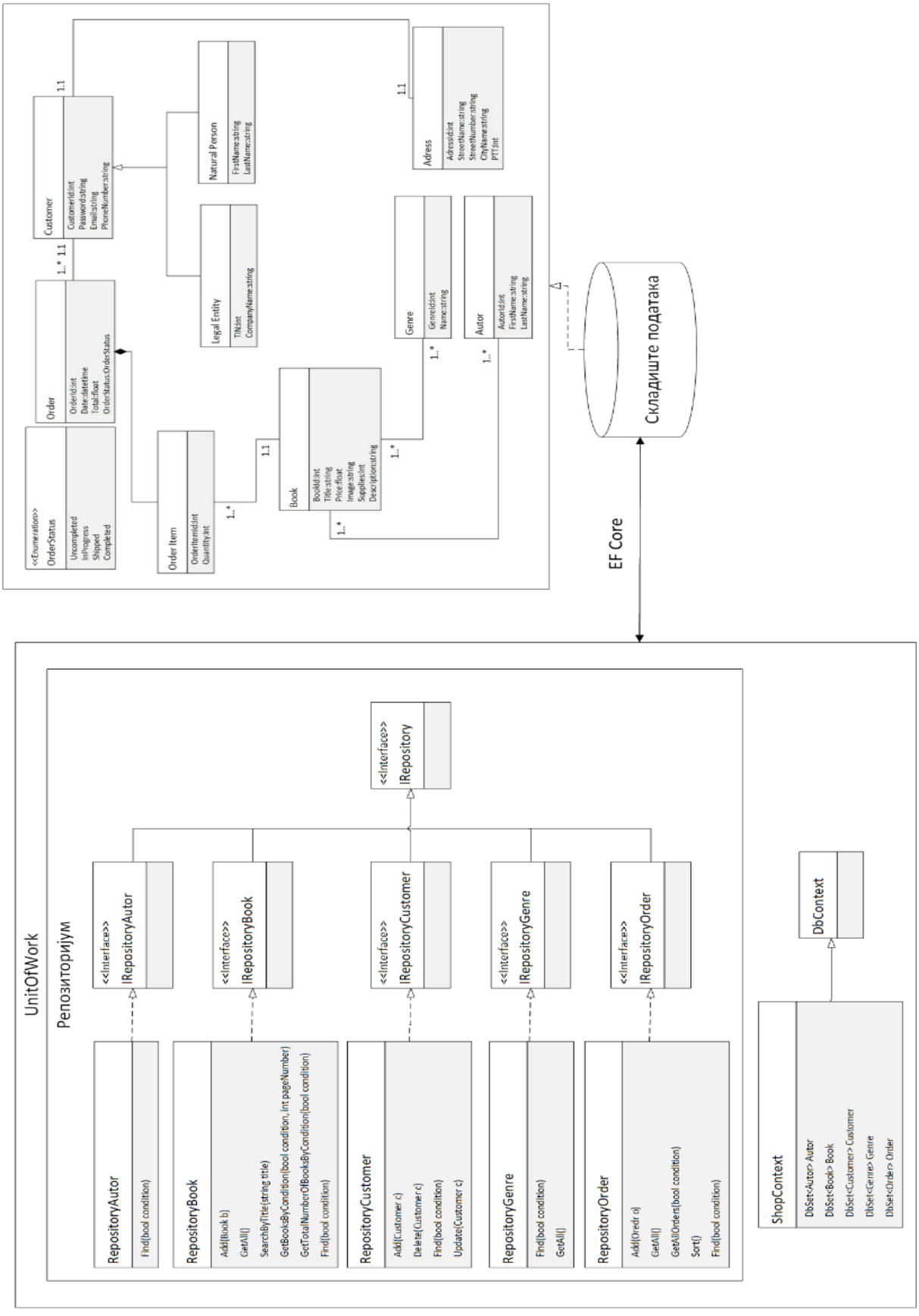
```
public interface IRepositoryOrder : IRepository<Order>
    {
        List<Order> GetAllOrders(Predicate<Order> condition);

        List<Order> Sort();
    }

public class RepositoryOrder : IRepositoryOrder
    {
        private readonly ShopContext context;

        public RepositoryOrder(ShopContext context) => this.context = context;
        public void Add(Order entity) => context.Add(entity);
        public Order FindWithoutInclude(Predicate<Order> condition) =>
context.Order.ToList().Find(condition);
        public List<Order> GetAll() => context.Order.Include(o =>
o.Customer).ToList();
        public List<Order> GetAllOrders(Predicate<Order> condition) =>
context.Order.Include(o => o.Customer).Include(o => o.OrderItems).ThenInclude(oi
=> oi.Book).ToList().FindAll(condition);
        public Order FindWithInclude(Predicate<Order> condition) =>
context.Order.Include(o => o.OrderItems).ThenInclude(oi =>
oi.Book).ToList().Find(condition);
        public List<Order> Sort() => context.Order.Include(order =>
order.Customer).OrderBy(o => o.OrderStatus).ToList();
        public Order Find(Predicate<Order> p) => context.Order.Include(o =>
o.OrderItems).ThenInclude(oi => oi.Book).ToList().Find(p);
    }
```

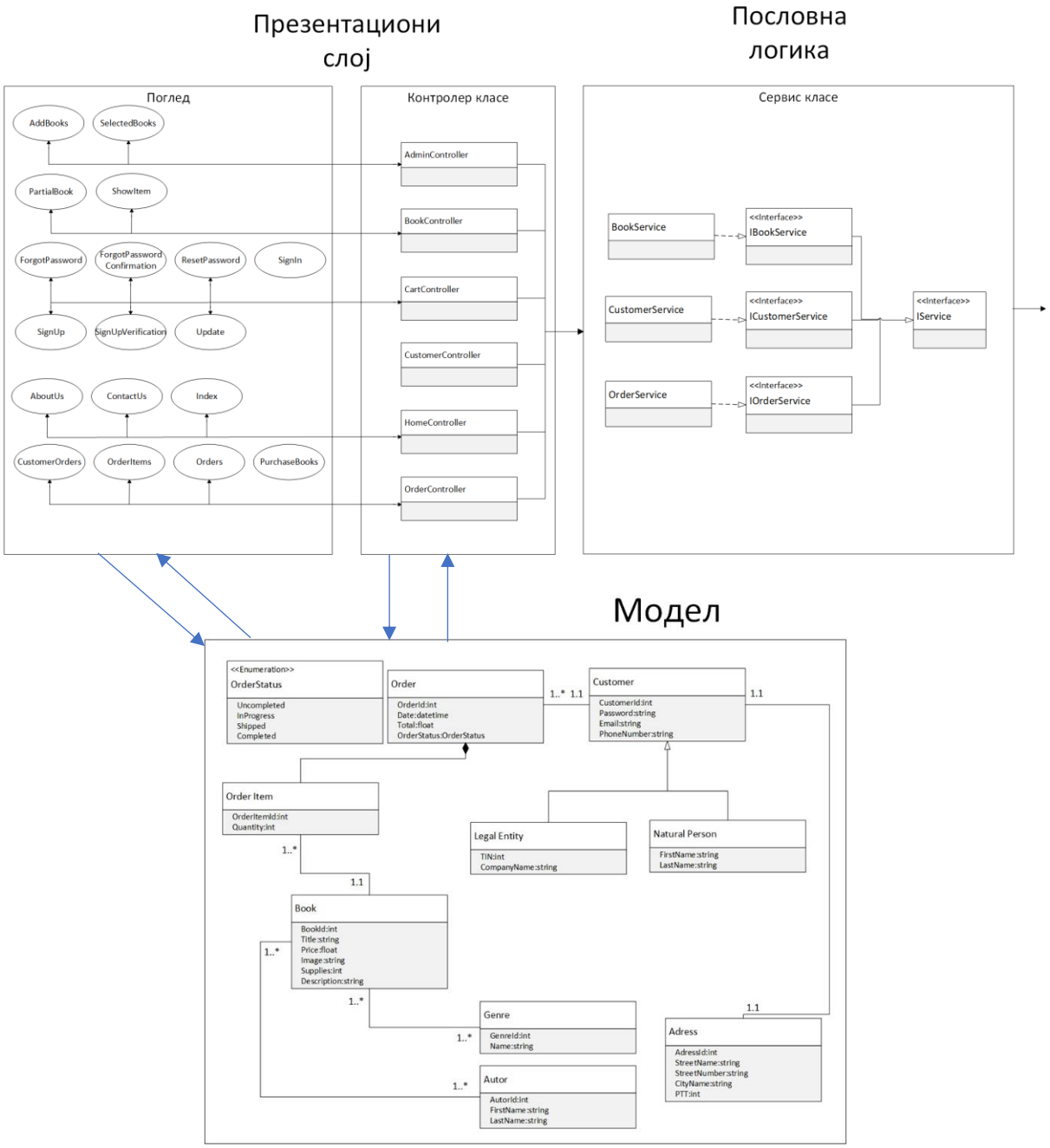
13.3.3 **Комуникација између пословне логике и складишта података**

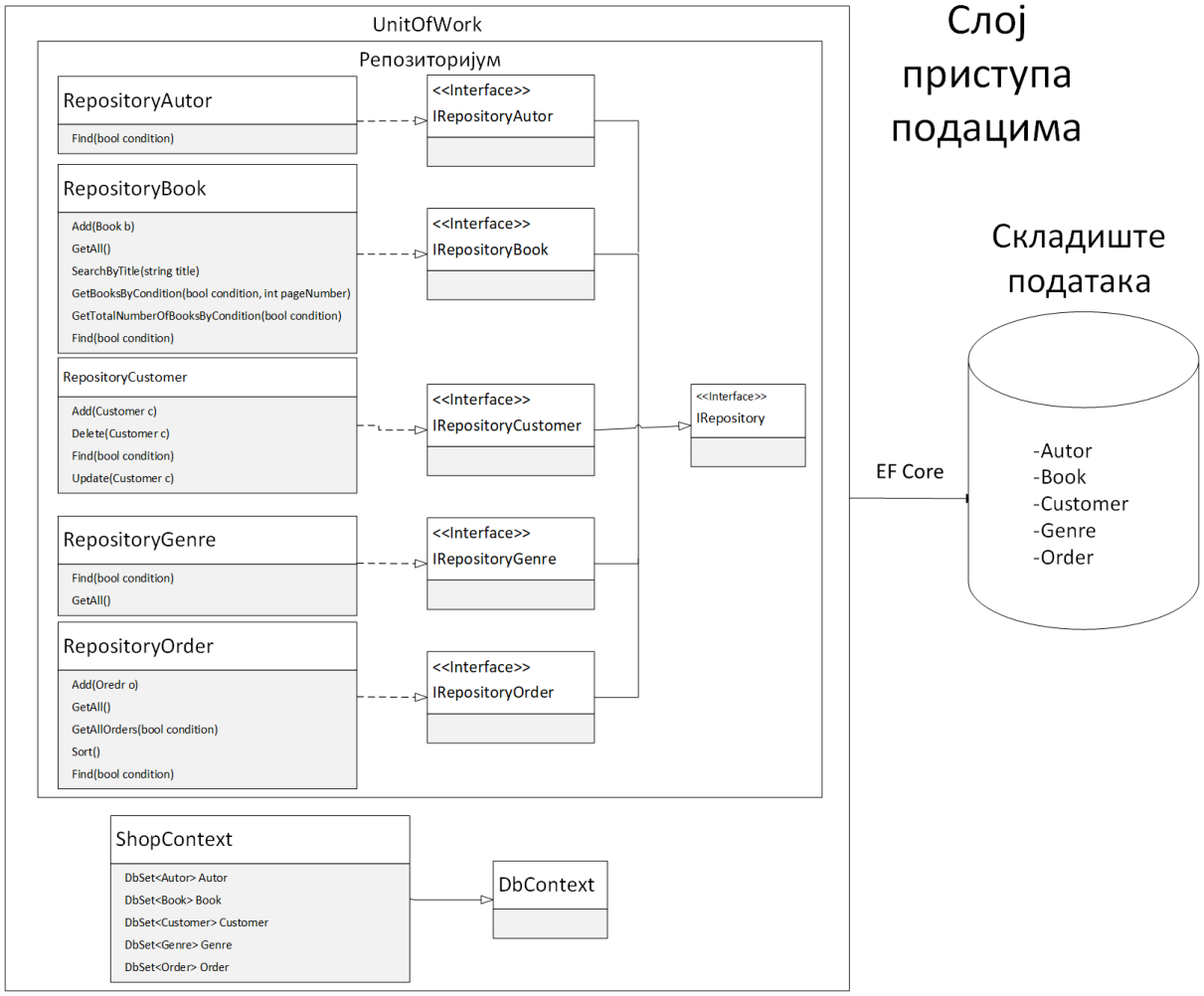


Слика 74 - Пројектовање апстрактног слоја између пословне логике и складишта података

13.4 Коначан изглед архитектуре софтверског система

Као резултат фазе пројектовања добија се коначна архитектура софтверског система. Погледи су одговорни за представљање садржаја путем корисничког интерфејса, контролери управљају интеракцијом корисника, раде с моделом и на крају бирају поглед који ће се приказивати. Све захтеве који стижу од корисника са корисничког интерфејса обрађује одговарајући контролер. Контролери имају референце на потребне сервисе. Примљене захтеве контролери шаљу на обраду одговарајућим сервисима. Сервиси имају референце на *UnitOfWork* класу која представља јединицу рада и која има референце на *Repository* класе. Repository класе служе за комуникацију са базом података. Табеле у бази података су креиране на основу концептуалних класа, као резултат објектно-релационог мапирања.





Слика 75 - Коначна архитектура софтверског система

14 Имплементација

Веб апликација која је настала као резултат овог рада развијена је у програмском језику у C#, у .NET окружењу уз помоћ MVC оквира. Као развојно окружење коришћен је Microsoft Visual Studio 2019. Систем за управљање базом података је SQL сервер, док је клијентска страна изграђена уз помоћ JavaScript, jQuery, AJAX и Bootstrap технологија.

14.1 Имплементација екранских форми

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

14.1.1 **СК1 : Случај коришћења – Приказивање детаља књиге**

Назив СК

Приказивање детаља књиге

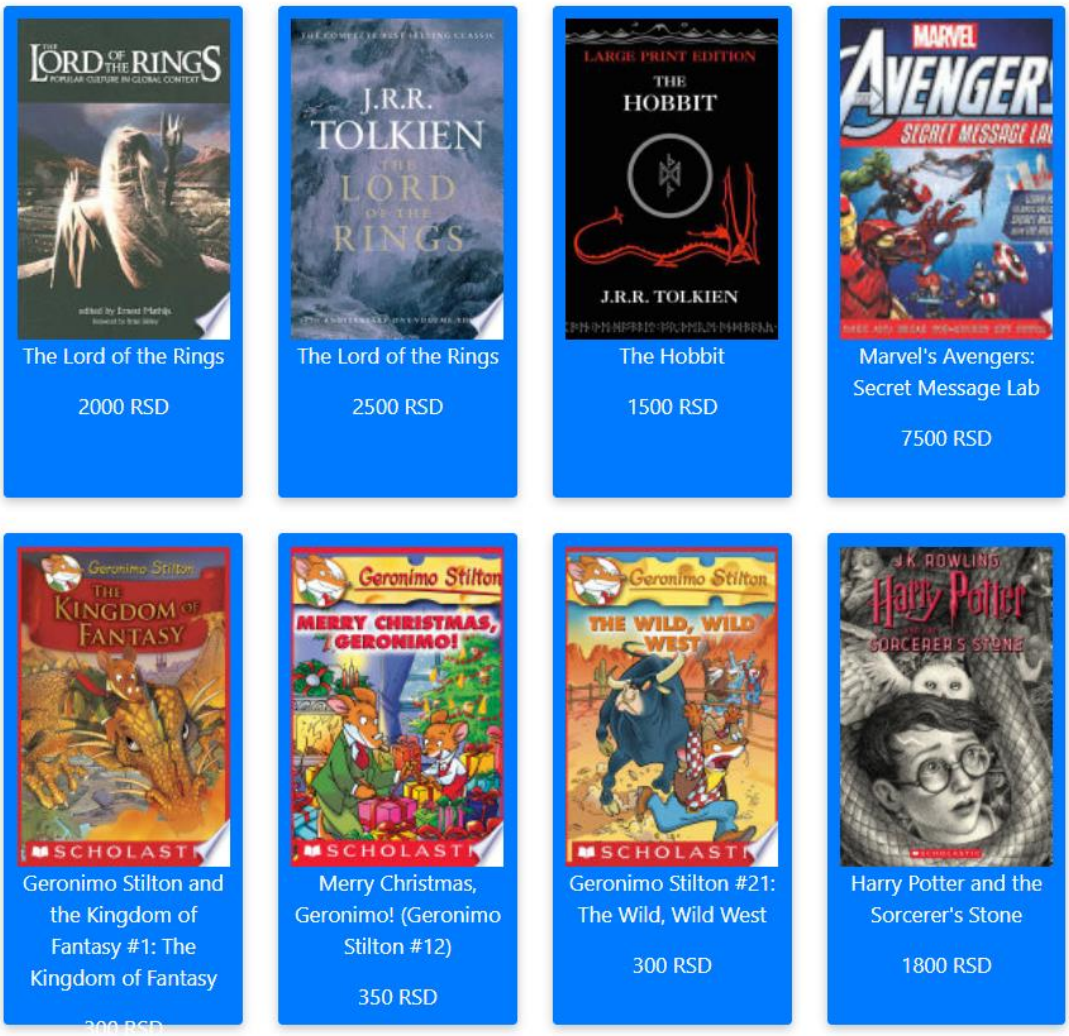
Актери СК

Корисник

Учесници СК

Корисник и систем

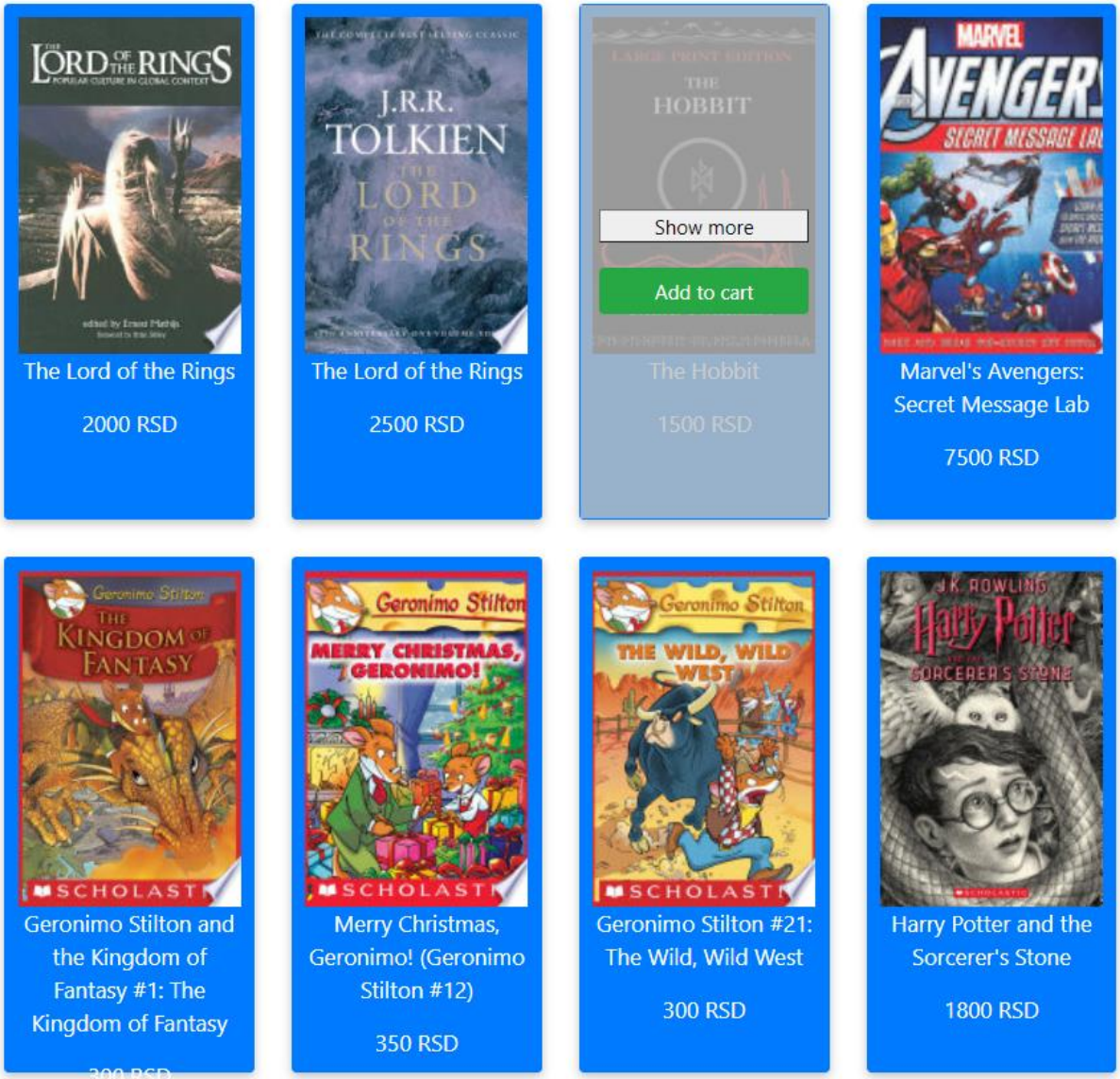
Предуслов: Кориснику је приказана листа књига.



Слика 76 - Листа књига

Основни сценарио СК

5. Корисник **уноси** књигу коју жели да му се прикаже. (АПУСО)



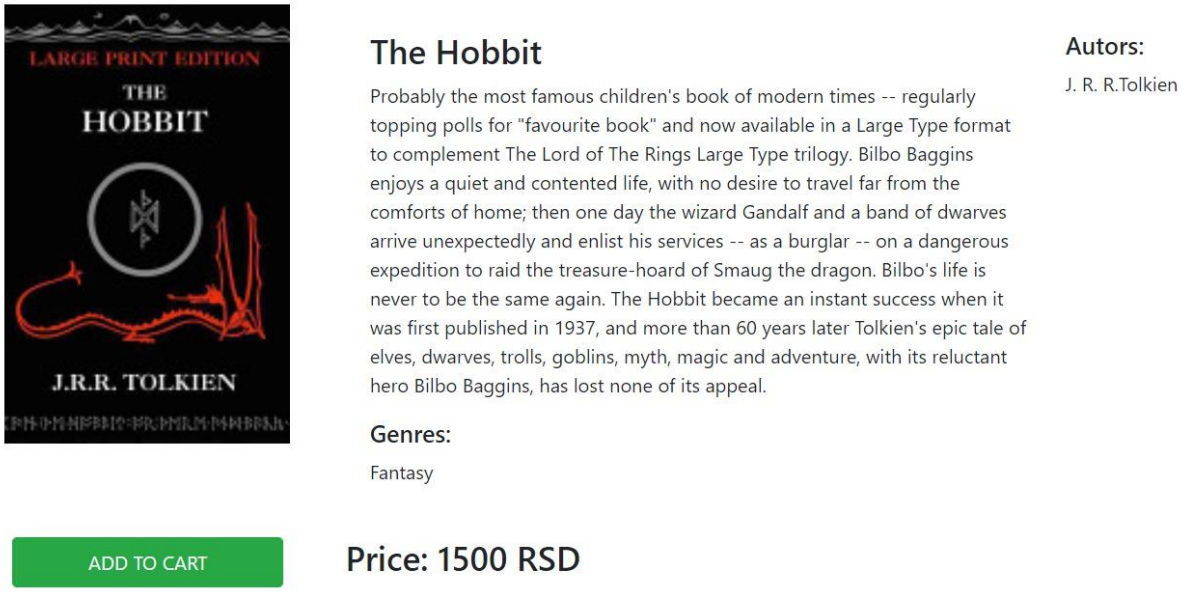
Слика 77 - Одабир књиге

6. Корисник **позива** систем да му прикаже изабрану књигу. (АПСО)

Опис акције: Корисник кликом на дугме „Show more“ позива системску операцију **ПрикажиКњигу(Књига)**

7. Систем **тражи** књигу по задатој вредности. (СО)

8. Систем **приказује** кориснику књигу. (ИА)



Слика 78 - Приказ књиге

Алтернативна сценарија

4.1. Уколико систем не може да пронађе књигу, кориснику се приказује порука : „Book doesn’t exist“ (ИА)

Слика 79 - Књига не постоји

14.1.2 СК2 : Случај коришћења – Приказивање наруџбеница

Назив СК

Приказивање наруџбеница

Актери СК

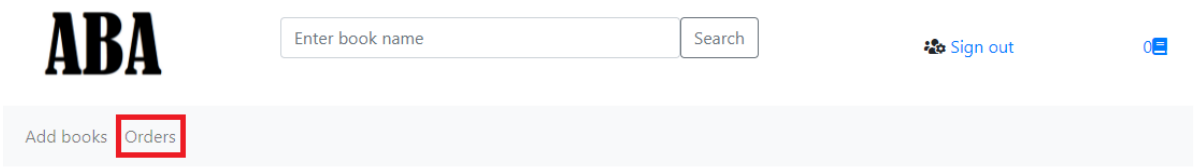
Админ

Учесници СК

Корисник и систем

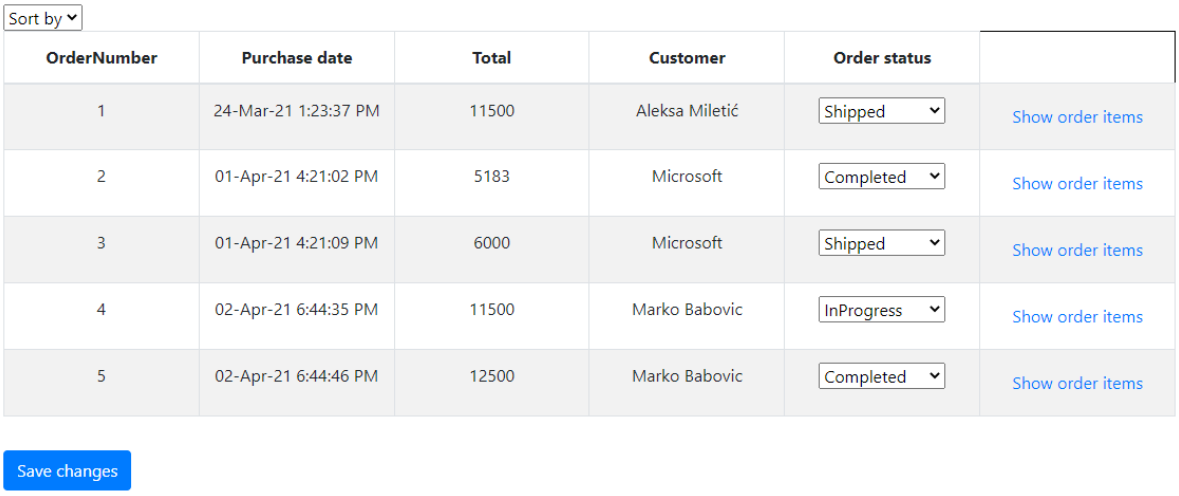
Основни сценарио СК

4. Админ **позива** систем да прикаже све поружбине. (АПСО)



Слика 80 - Дугме за приступ страници за приказ поруџбина

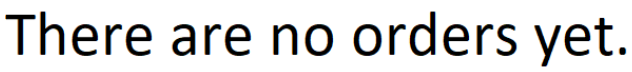
5. Систем **тражи** поруџбине. (СО)
   Опис акције: Админ кликом на дугме менија „Orders“ позива системску операцију **ВратиНаруџбенице(List<Наруџбеница>)**
6. Систем **приказује** све поруџбине. (ИА)



Слика 81 - Поруџбине

Алтернативна сценарија

3.1. Уколико систем не може да нађе поруџбине, админу се приказује порука : „There are no orders yet“ (ИА)



Слика 82 - Још увек нема поруџбина

14.1.3 **СК3 : Случај коришћења – Промена статуса наруџбеница**

Назив СК

Промена статуса наруџбеница

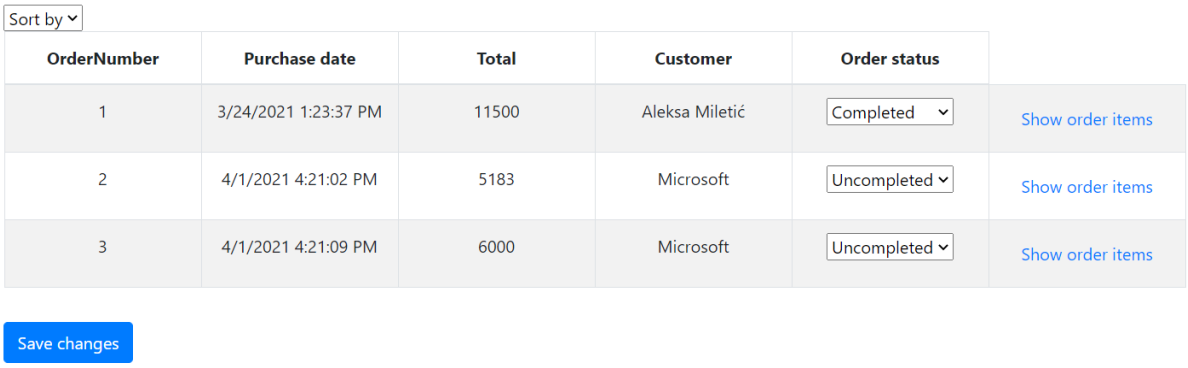
Актери СК

Админ

Учесници СК

Админ и систем

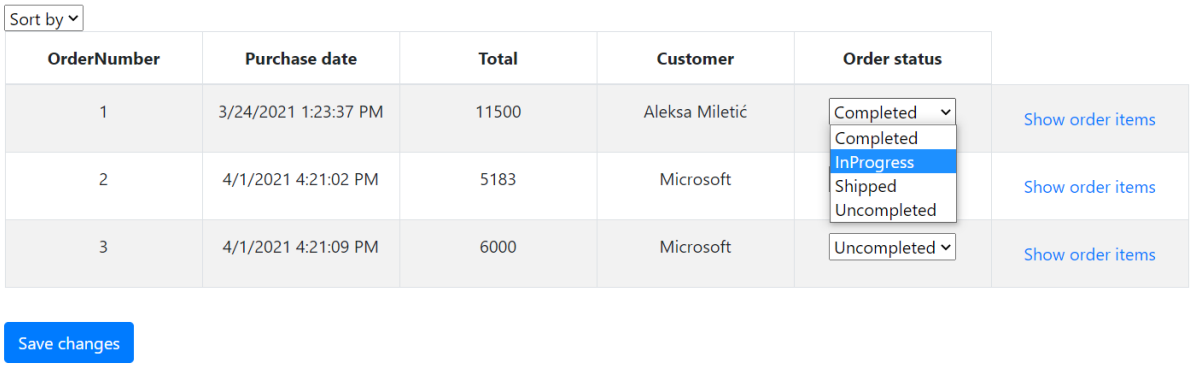
Предуслов: Админ је регистрован на свој налог и приказана је листа свих поруџбина.



Слика 83 - Табела наруџбеница

Основни сценарио СК

6. Админ **мења** статусе поруџбина. (АПУСО)



Слика 84 - Промена статуса поруџбине

7. Админ **проверава** да ли је добро променио статусе поруџбина. (АНСО)
8. Админ **позива** систем да сачува статусе поруџбина. (АПСО)
   Опис акције: Админ кликом на дугме „Save changes“ позива системску операцију **АжурирајНаруџбенице(List<Наруџбеница>)**
9. Систем **памти** поруџбине. (СО)
10. Систем **приказује** админу поруку „Successfuly updated orders“. (ИА)

Алтернативна сценарија

5.1. Уколико систем не може да запамти поруџбине, админу се приказује порука: „Cannot update orders“ (ИА)

Слика 85 - Поруџбина се не може ажурирати

14.1.4 **СК4 : Случај коришћења – Додавање књига**

Назив СК

Додавање књига

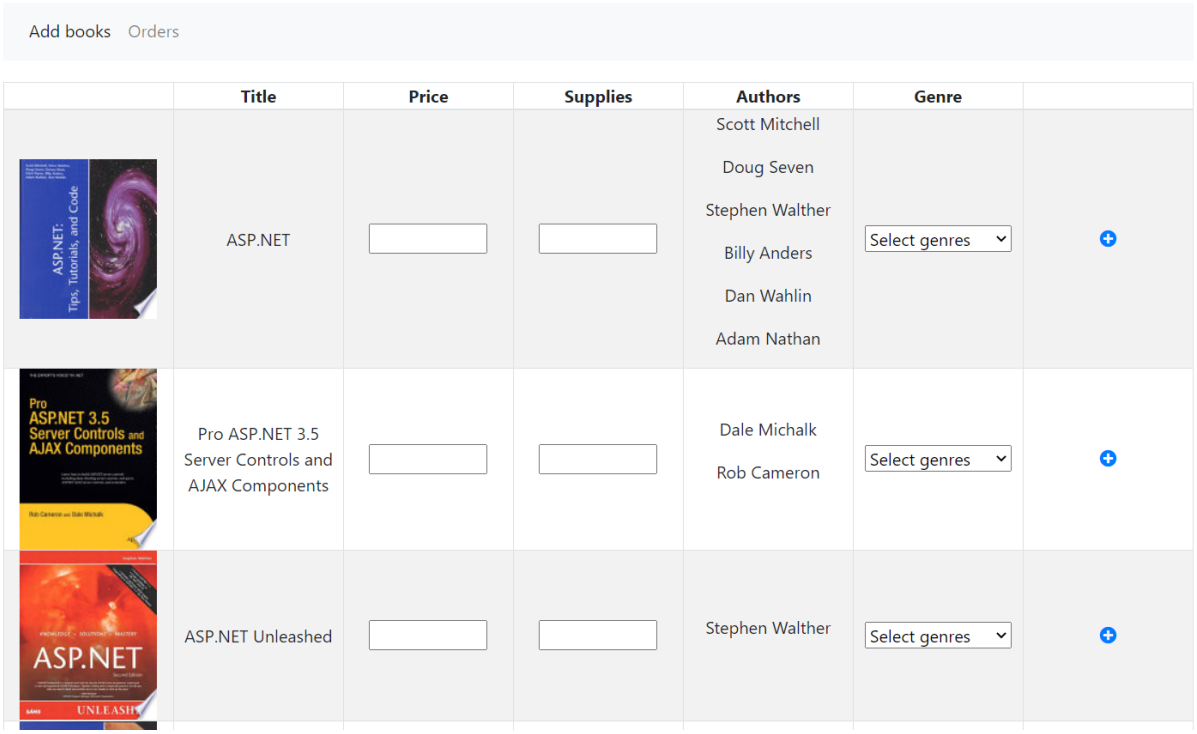
Актери СК

Админ

Учесници СК

Админ и систем

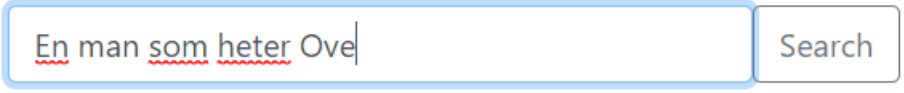
Предуслов: Админу је регистрован на свој налог. Учитана је листа књига.



Слика 86 - Табела са приказаним књигама за поручивање

Основни сценарио СК

11. Админ **уноси** критеријум по коме претражује књиге. (АПУСО)

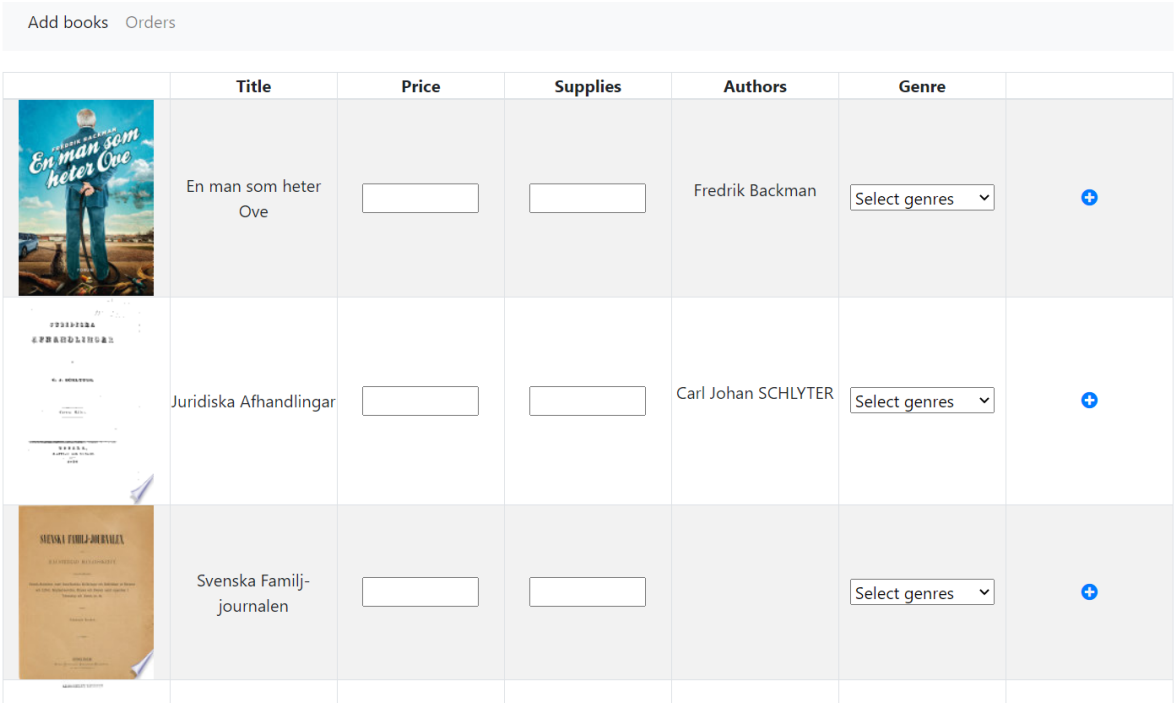


Слика 87 - Форма за унос наслова књиге

12. Админ **проверава** да ли је добро унео критеријум претраге. (АНСО)
13. Админ **позива** систем да претражи књиге по задатом критеријуму. (АПСО)

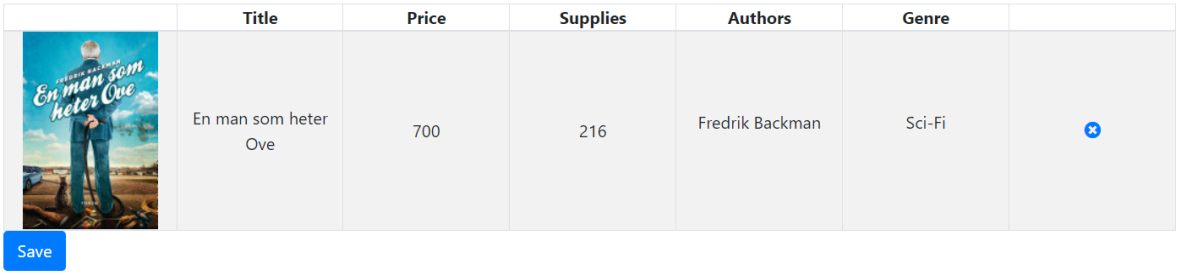
Опис акције: Администратор кликом на дугме „Search“ позива системску операцију **ВратиЛистуКњига(критеријум,List<Књига>)**

14. Систем **претражује** књиге по задатом критеријуму. (СО)
15. Систем **приказује** администратору листу књига по задатом критеријуму. (ИА)



Слика 88 - Приказане књиге након претраге

16. Администратор **бира** књиге које жели да сачува. (АПУСО)



Слика 89 - Одабир књиге

17. Администратор **проверава** да ли је добро одабрао књиге. (АНСО)
18. Администратор **позива** систем да сачува књиге. (АПСО)
    Опис акције: Администратор кликом на дугме „Save“ позива системску операцију **СачувајКњиге(List<Књига>)**
19. Систем **памти** књиге. (СО)
20. Систем **приказује** администратору поруку „There are no selected books“. (ИА)

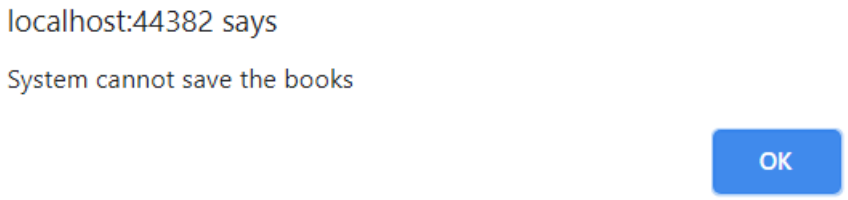
Слика 90 - Успешно унета књига

Алтернативна сценарија

5.1. Уколико систем не може да нађе књиге по задатом критеријуму, администи се приказује порука : „System cannot find books“ (ИА)

Слика 91 - Систем не може пронаћи књиге

10.1. Уколико систем не може да запамти књиге, администи се приказује порука : „System cannot save the books“ (ИА)



Слика 92 - Систем не може сачувати књиге

14.1.5 СК5 : Случај коришћења – Приказивање наруџбенице

Назив СК

Приказивање наруџбенице

Актери СК

Админ или Корисник

Учесници СК

Админ(Корисник) и систем

Предуслов: Админ(Корисник) је регистрован на свој налог. Учитана је листа наруџбеница.

OrderNumber	Purchase date	Total	Order status	
4	02-Apr-21 6:44:35 PM	11500	**Uncompleted**	Show order items
5	02-Apr-21 6:44:46 PM	12500	**Uncompleted**	Show order items

Слика 93 - Табела наруџбина

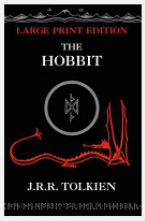
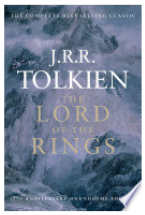
Основни сценарио СК

5. Админ(Корисник) **бира** наруџбеницу коју жели да му се прикаже. (АПУСО)

OrderNumber	Purchase date	Total	Order status	
4	02-Apr-21 6:44:35 PM	11500	**Uncompleted**	Show order items
5	02-Apr-21 6:44:46 PM	12500	**Uncompleted**	Show order items

Слика 94 - Дугме за приказ наруџбине

6. Админ(Корисник) **позива** систем да прикаже ставке наруџбенице. (АПСО)
   Опис акције: Админ кликом на линк „Show order items“ позива системску операцију **ВратиСтавкеНаруџбенице(Наруџбеница)**
7. Систем **претражује** одабрану наруџбеницу. (СО)
8. Систем **приказује** админу(кориснику) ставке наруџбенице. (ИА)

Image	Title	Price	Quantity	Total for this product
	The Hobbit	1500	1	1500
	The Lord of the Rings	2500	1	2500
	Marvel's Avengers: Secret Message Lab	7500	1	7500

Total price 11500 RSD

Слика 95 - Наруџбина

Алтернативна сценарија

4.1. Уколико систем не може да нађе ставке наруџбенице, админу(кориснику) се приказује порука : „“ (ИА)

Слика 96 - Ставке наруџбенице не постоје

14.1.6 **СК6 : Случај коришћења – Сортирање наруџбеница**

Назив СК

Сортирање наруџбеница

Актери СК

Админ

Учесници СК

Админ(Корисник) и систем

Предуслов: Админ је регистрован на свој налог. Учитана је листа наруџбеница.

Sort by

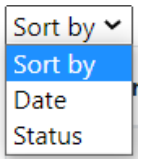
OrderNumber	Purchase date	Total	Customer	Order status	
1	24-Mar-21 1:23:37 PM	11500	Aleksa Miletić	Completed	Show order items
2	01-Apr-21 4:21:02 PM	5183	Microsoft	Uncompleted	Show order items
3	01-Apr-21 4:21:09 PM	6000	Microsoft	Uncompleted	Show order items
4	02-Apr-21 6:44:35 PM	11500	Marko Babovic	Uncompleted	Show order items
5	02-Apr-21 6:44:46 PM	12500	Marko Babovic	Uncompleted	Show order items

Save changes

Слика 97 - Табела наруџбеница

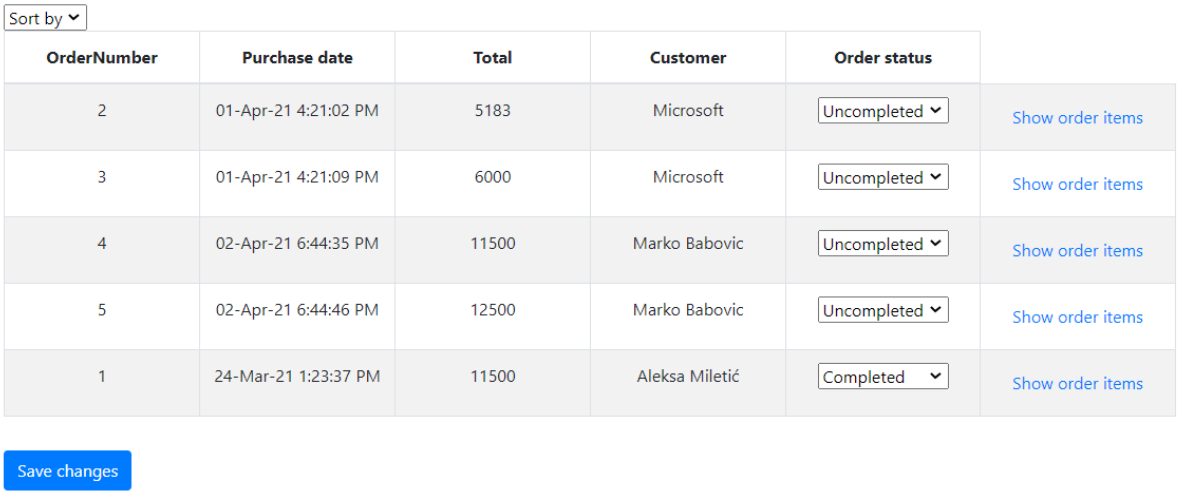
Основни сценарио СК

5. Админ(Корисник) **уноси** критеријум сортирања наруџбеница. (АПУСО)



Слика 98 - Мени за сортирање

6. Админ(Корисник) **позива** систем да сортира наруџбенице. (АПСО)
   Опис акције: Админ(Корисник) избором елемента из падајуће листе позива системску операцију **СортирајНаруџбенице(List<Наруџбеница>)**
7. Систем **сортира** наруџбенице. (СО)
8. Систем **приказује** админу наруџбенице. (ИА)



Слика 99 - Табела за приказ наруџбеница након сортирања

Алтернативна сценарија

4.1. Уколико систем не може да сортира наруџбенице, админу се приказује порука : „Cannot sort the orders“ (ИА)

Слика 100 - Наруџбенице се не могу сортирати

14.1.7 СК7: Случај коришћења – Претраживање књига

Назив СК

Претраживање књига

Актери СК

Корисник

Учесници СК

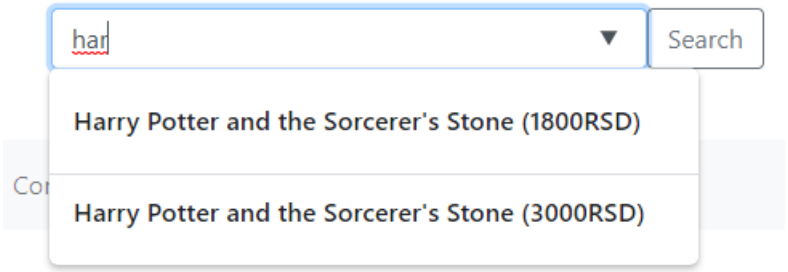
Корисник и систем

Основни сценарио СК

1. Корисник **уноси** критеријум претраживања књига. (АПУСО)

Слика 101 - Форма за унос наслова књиге

1. Корисник **позива** систем да претражи књиге по неком критеријуму. (АПСО)
   Опис акције: Кориснику се приликом куцања назива књиге позива ситемска операција **Претражи(критеријум, List<Књига>)**
2. Систем **претражује** књиге. (СО)
3. Систем **приказује** кориснику књиге. (ИА)



Слика 102 - Приказ пронађених књига

Алтернативна сценарија

4.1. Уколико систем не може да претражи књиге, кориснику се приказује порука: „Cannot search the books“ (ИА)

Слика 103 - Не постоји ни једна таква књига

14.2 Структура софтверског решења

На серверској страни имплементиране су следеће класе:

1. **Model**:
    a. **Domain**: Address.cs, Autor.cs, Book.cs, Customer.cs, Genre.cs, LegalEntity.cs, NaturalPerson.cs, Order.cs, OrderItem.cs, OrderStatus.cs
2. **Data**:
    a. **Implementation**:
        i. **Interfaces**: IRepository.cs, IRepositoryAutor.cs, IRepositoryBook.cs, IRepositoryCustomer.cs, IRepositoryGenre.cs, IRepositoryOrder.cs
        ii. **RepositoryClasses**: RepositoryAutor.cs, RepositoryBook.cs, RepositoryCustomer.cs, RepositoryGenre.cs, RepositoryOrder.cs
    b. **UnitOfWorkFolder**: EShopUnitOfWork.cs, IUnitOfWork.cs
3. **BusinessLogic**: IService.cs
    a. **Classes**: BookService.cs, CustomerService.cs, OrderService.cs
    b. **Exception**: CustomerNullException.cs, OrderException.cs
    c. **Interfaces**: IBookService.cs, ICustomerService.cs, IOrderService.cs
    d. **Models**: CustomerType.cs, ForgotPasswordViewModel.cs, SignInViewModel.cs, SignUpViewModel.cs, UpdateCustomerViewModel.cs
4. **EShop.WebApp:**
    a. **Controllers:** AdminController.cs, BookController.cs, CartController.cs, CustomerController.cs, HomeController.cs, OrderController.cs
    b. **Views:** _ViewImports.cshtml, _ViewStart.cshtml
        i. **Admin:** AddBooks.cshtml, SelectedBooks.cshtml
        ii. **Book:** ShowItem.cshtml
        iii. **Customer:** ForgotPassword.cshtml, ForgotPasswordConfirmation.cshtml, ResetPassword.cshtml, SignIn.cshtml, SignUp.cshtml, SignUpVerification.cshtml, Update.cshtml
        iv. **Home:** AboutUs.cshtml, ContactUs.cshtml, Index.cshtml
        v. **Order:** CustomerOrders.cshtml, OrderItem.cshtml, Orders.cshtml, PurchaseBooks.cshtml
        vi. **Shared:** _Layout.cshtml, _LayoutAdmin.cshtml, _ValidationScriptsPartial.cshtml, PartialAdmin.cshtml, PartialBooks.cshtml

14.3 Имплементација пословне логике

На основу пословне логике и дијаграма секвенци, за сваки уговор се креира конкретно имплементационо решење.

1. Уговор УГ1 : **ПрикажиКњигу**
   Операција: ПрикажиКњигу(Књига)
   Веза са СК: СК1
   Предуслови: /
   Постуслови: /

```
public Book Find(int? bookId) => uow.RepositoryBook.Find(b => b.BookId == bookId);
```

2. Уговор УГ2 : **ВратиКњиге**
   Операција: ВратиКњиге(List<Књига>)
   Веза са СК: СК1, СК4
   Предуслови: /
   Постуслови: /

```
public List<Book> GetAllBooks()=> uow.RepositoryBook.GetAll();
```

3. Уговор УГ3 : **ВратиЛистуКњига**
   Операција: ВратиЛистуКњига(критеријум, List<Књига>)
   Веза са СК: СК4
   Предуслови:/
   Постуслови: /

```
public List<Book> GetBooksByCondition(int pageNumber, string price,
List<string> genres)
 {

     Func<Book, bool> func;
     var FirstSecondPrice = Price(price);
     if (genres.Count > 0)
     {
         func = (b => b.Supplies != 0 &&
         b.Price >= FirstSecondPrice[0] && b.Price <= FirstSecondPrice[1] &&
         b.Genres.Any(g => genres.Any(genre => genre == g.Name)));
     }
     else
     {
       func = (b => b.Supplies != 0 && b.Price >= FirstSecondPrice[0] &&
       b.Price <= FirstSecondPrice[1]);
     }

     return uow.RepositoryBook.GetBooksByCondition(func, pageNumber);
  }
```

4. Уговор УГ4 : **ВратиНаруџбенице**
   Операција: ВратиНаруџбенице(List<Наруџбеница>)

Веза са СК: СК2, СК5, СК6
Предуслови: /
Постуслови: /

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

5. Уговор УГ5 : **АжурирајНаруџбенице**
Операција:АжурирајНаруџбенице(List<Наруџбеница>)
Веза са СК: СК3
Предуслови: Вредносна и структурна ограничења над објектом Наруџбеница морају бити задовољена.
Постуслови: Подаци о наруџбеницама су ажурирани.

```
public void UpdateOrders(List<Order> orders)
 {
     orders.ForEach(o =>
     {
         Order order = uow.RepositoryOrder.Find(or => or.OrderId == o.OrderId);
         order.OrderStatus = o.OrderStatus;
         uow.Commit();
     });
 }
```

6. Уговор УГ6 : **СачувајКњиге**
Операција: СачувајКњиге(List<Књига>)
Веза са СК: СК4
Предуслови: Вредносна и структурна ограничења над објектом Књига, Аутор и Жанр морају бити задовољена.
Постуслови: Подаци о књизи, ауторима и занровима су сачувани.

```
public void Add(List<Book> books)
 {
     foreach (var item in books)
     {
       Book b = uow.RepositoryBook.Find(b => b.Title == item.Title &&
       b.Description == item.Description);
         if (b == null)
         {
             for (int i = 0; i < item.Genres.Count; i++)
             {
              item.Genres[i] = uow.RepositoryGenre.Find(g => g.Name ==
              item.Genres[i].Name);
             }
             for (int i = 0; i < item.Autors.Count; i++)
             {
                    Autor a = uow.RepositoryAutor.Find(a => a.FirstName ==
                    item.Autors[i].FirstName && a.LastName ==
                    item.Autors[i].LastName);
                if (a != null)
                {
                    item.Autors[i] = a;
                }
             }
```

```
                uow.RepositoryBook.Add(item);
            }
            else
            {
                b.Supplies += item.Supplies;
            }
            uow.Commit();
        }
           }
```

7. Уговор УГ7 : **ВратиСтавкеНаруџбенице**
   Операција: ВратиСтавкеНаруџбенице(Наруџбеница)
   Веза са СК: СК5
   Предуслови: /
   Постуслови: /

```
public Order GetOrderItems(int orderId)
 {
     return  uow.RepositoryOrder.Find(o => o.OrderId == orderId);
 }
```

8. Уговор УГ8 : **СортирајНаруџбенице**
   Операција: СортирајНаруџбенице(List<Наруџбеница>)
   Веза са СК: СК6
   Предуслови: /
   Постуслови: /

```
public List<Order> SortOrders(bool condition)
 {
     List<Order> orders;
     if (condition)
         orders = uow.RepositoryOrder.Sort();
     else
         orders = uow.RepositoryOrder.GetAll();

     return orders;
 }
```

9. Уговор УГ9 : **Претражи**
   Операција: Претражи(критеријум,List<Књига>)
   Веза са СК: СК
   Предуслови: /
   Постуслови: /

```
public          List<Book>          Search(string          title)          =>
uow.RepositoryBook.SearchByTitle(title);
```

14.4 Имплементација складишта података

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

	Name	Data Type	Allow Nulls
🔑	AutorId	int	☐
	FirstName	nvarchar(MAX)	☐
	LastName	nvarchar(MAX)	☐

Слика 104 - Аутор

	Name	Data Type	Allow Nulls
🔑	BookId	int	☐
	Title	nvarchar(MAX)	☐
	Image	nvarchar(MAX)	☐
	Price	float	☐
	Supplies	int	☐
	Description	nvarchar(MAX)	☑

Слика 105 - Књига

	Name	Data Type	Allow Nulls
🔑	AutorsAutorId	int	☐
🔑	BooksBookId	int	☐

Слика 106 - АуторКњига

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

Слика 107 - Жанр

	Name	Data Type	Allow Nulls
🔑	BooksBookId	int	☐
🔑	GenresGenreId	int	☐

Слика 108 - ЖанрКњига

	Name	Data Type	Allow Nulls
🔑	CustomerId	int	☐
	Password	nvarchar(MAX)	☐
	Email	nvarchar(MAX)	☐
	PhoneNumber	nvarchar(MAX)	☐
	Address_StreetName	nvarchar(MAX)	☐
	Address_StreetNumber	nvarchar(MAX)	☐
	Address_CityName	nvarchar(MAX)	☐
	Address_PTT	int	☐
	VerificationCode	bigint	☐
	Status	bit	☐
	IsAdmin	bit	☐

Слика 109 - Купац

	Name	Data Type	Allow Nulls
🔑	CustomerId	int	☐
	TIN	int	☐
	CompanyName	nvarchar(MAX)	☐

Слика 110 - ПравноЛице

	Name	Data Type	Allow Nulls
🔑	CustomerId	int	☐
	FirstName	nvarchar(MAX)	☐
	LastName	nvarchar(MAX)	☐

Слика 111 - ФизичкоЛице

	Name	Data Type	Allow Nulls
🔑	OrderId	int	☐
	Date	datetime2(7)	☐
	Total	float	☐
	OrderStatus	int	☐
	CustomerId	int	☑

Слика 112 - Нарудбеница

🔑	OrderItemId	int	☐
🔑	OrderId	int	☐
	Quantity	int	☐
	BookId	int	☐

Слика 113 - СтавкаНаруџбенице

15 Тестирање

Покретањем апликације и уношењем неисправних података вршено је тестирање апликације. Такође су уношени и исправни подаци да би се показала и исправност апликације. Уколико су се појавиле грешаке приликом тестирања, те грешке су уклоњене.

16 Закључак

Приликом развоја једног софтверског система веома је битно посветити пажњу свакој од фаза. Уколико се прескоче неки захтеви или исти нису лепо дефинисани може доћи до великих проблема у току развоја апликације. На крају, један од кључних фактора је осмислити кориснички интерфејс, који треба бити једноставан и лак за коришћење.

У овом завршном раду приказан је компелтан поступак за креирање веб апликације, конкретно за веб апликације за набавку књига у књижари. Иако је креиран систем за коришћење у некој књижари за поручивање књига и даље има места за унапређење, како би се максимално олакшао посао за запослене који раде у набавци. Неке од функционалности које би додатно побољшале апликацију су: онлине плаћање, бољa заштита у случају хакерских напада и убацивање листе жеља за сваког користника понаособ.

Фаза тестирања је веома значајна фаза у току развоја једне апликације. Тестирање кода ми је доста помогло у отклањању грешака и оптимизовању саме апликације.

Једна од кључних ствари која може да се унапреди јесте коришћење JavaScript развојног оквира, уместно чистог JavaScript-а, као на пример *Angular*. То би додатно додатно убразало учитавање страница и било би лакше за одржавање. Одлучио сам се ипак за ове технологије које сам описао у раду зато што су исте представљене на предмету *Напредне .NET технологија* на коме сам стекао неопходно знање за даљи рад. Веб технологије и софтверско инжењерство је су области у којима бих волео да наставим своје усавршавање.

17 Литература

Antani, V., & Stefanov, S. (2017). Object-Oriented JavaScript - Third Edition. Packt Publishing.

Cohen, Y. (2021, April 21). *Understanding the Execution Context in JavaScript*. Преузето са Coralogix: https://coralogix.com/blog/understanding-the-execution-context-in-javascript/

Duckett, J. (2014). *JAVASCRIPT & JQUERY Interactive Front-End Web Development.* Indianapolis: John Wiley & Sons, Inc.

Freeman, A. (2018). *Pro Entity Framework Core 2 for ASP.NET Core MVC.* London: Apress.

guru99. (2021). *Relational Data Model in DBMS: Concepts, Constraints, Example*. Преузето са guru99: https://www.guru99.com/relational-data-model-dbms.html

Jain, S. (2017, 9 28). *Overview Of Common Language Infrastructure*. Преузето са C# Corner: https://www.c-sharpcorner.com/blogs/overview-of-common-language-infrastructure

Jin, B., Sahni, S., & Shevat, A. (2018). *Designing Web APIs.* O'Reilly.

Joshi, B. (2019). *Beginning Database Programming Using ASP.NET Core 3 With MVC, Razor Pages, Web API, jQuery, Angular, SQL Server, and NoSQL.* Apress.

Microsoft. (2013, 7 30). *The Repository and Unit of Work Patterns*. Преузето са Microsoft: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application

Microsoft. (2020, 7 13). *Understand the .NET Compiler Platform SDK model*. Преузето са Microsoft: https://docs.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/compiler-api-model

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

Microsoft. (2021, 1 28). *A tour of the C# language*. Преузето са Microsoft: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

Patel, P. (2019, May 6). *How Does JavaScript Really Work? (Part 1)*. Преузето са Bits and Pieces: https://blog.bitsrc.io/how-does-javascript-really-work-part-1-7681dd54a36d

TutorialTeacher. (2020). *.NET Core Command-Line Interface*. Преузето са TutorialTeacher: https://www.tutorialsteacher.com/core/net-core-command-line-interface

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