|  |  |
| --- | --- |
|  | **Универзитет у Београду**  **Електротехнички Факултет** |

**Дизајн и имплементација веб продавнице**

**Дипломски рад**

|  |  |
| --- | --- |
| Кандидат | Професор |
| Илија Стевановић | Др. Бошко Николић |

Београд,

Септембар 2017.

**САДРЖАЈ**

[1. Увод 3](#_Toc481580181)

[2.](#_Toc481580182) [Кориснички захтев и преглед сличних алата 6](#_Toc481580182)

[3. Имплементација система 9](#_Toc481580188)

[3.1 Општа архитектура 9](#_Toc481580182)

[3.2 База података 11](#_Toc481580183)

[3.3 Back end апликација](#_Toc481580184) **[12](#_Toc481580184)**

[3.4 Front end aпликација](#_Toc481580184) **[20](#_Toc481580184)**

[4. Oпис софтверског система 26](#_Toc481580188)

[4.1 Корисничка апликација 26](#_Toc481580182)  
[4.2 Опис опција за рад администратора 30](#_Toc481580182)

[5. Закључак 34](#_Toc481580188)

[6. Литература 36](#_Toc481580204)

1. **Увод**

Електронскo пословање (*Е-commerce*, електронска трговина) је процес куповине или продаје различитих врста робе и услуга путем интернета. Овакав тип пословања је, у врло кратком временском периоду, преузео примат као основни вид сваког успешног пословног модела, пословања и трговине. Ово се десило као последица нагле доступности и популаризације интернета, као платформе за повезивање људи и размeну информација. Електронска трговина се може дефинисати и као спрега између четири главна трговинска односа:

 • Потрошач – Продавац;  
 • Продавац - Потрошач;  
 • Потрошач – Потрошач;  
 • Продавац - Продавац;

Електронско пословање омогућује продавцу да успостави присуство на тржисту и унапреди своју тренутну позицију на њему. То омогућава јефтинији и ефикаснији начин дистрибуције и оглашавања својих производа, услуга и сервиса. Један пример тога је, рецимо, ланац - продавница одеће која, поред физичких објеката - продавница, има и онлајн продавницу где корисници - потрошачи могу наручивати и куповати одећу директно са својих личних рачунара, паметних телефона и других уређаја које користе интернет. Такође, веб-продавница би својом имплеметацијом могла значајно да олакша евиденцију, документацију и праћење интерне статистике, како својих запослених, тако и стања својих продајних услуга/артикала. Вођење статистике и прикупљање података се последњих година показало као најбољи извор информација, као и најбољи алат за истраживање тржишта, извлачењем релевантних закључака из бројних података које се могу добити анализом велике количине података – *Big data analysis*.

Електронско пословање омогућује купцу (потрошачу, кориснику услуга и сервиса) да пронађе на најлакши могући начин услуге и добра, по цени која му највише одговара. Такође, на основу интеракције са продавцем или са међусобном интеракцијом других посетиоца - корисника система (на пример, форум купаца-корисника на веб-порталу продавнице) купац може да стекне увид у квалитет артикла или услуге коју жели да купи. Једна инстанца процеса електронске куповине, од свог почетка (корисник налази производ који жели да купи) до свог краја (купац је платио производ и продавац доставља продати производ купцу) се назива трансакција.

Oнлајн куповина је вид електронског пословања који омогућује купцу да купи продукт или сервисе диркетно од продавца преко интернета, користећи интернет претраживач. Потрошач налази производ од интереса тако што приступи веб – страници продавнице која продаје тај производ, или тако што претражује најбољу понуду за производ који жели, користећи неки од алата за претраживање продавница и њихових понуда. Потрошач је, у последњих пар година, у могућности да извршава онлајн куповину са низа уређаја који могу да успоставе интернет конекцију, укључујући десктоп рачунаре, преносиве рачунаре, таблете, паметне телефоне, ... Потрошачи, у најједноставнијем случају, морају да поседују барем две ствари: интернет конекцију, да би се транскација успоставила и обавила, као и валидан метод плаћања, да би се та транскација успешно завршила. Постоје разни методи плаћања којима корисник може да заврши трансакцију, као што су плаћање кредитном картицом, плаћање поузећем (давањем новца лицу које је задужено за доставу производа купцу), плаћање коришћењем сервиса за онлајн плаћање (PayPal)...

Почетни концепт електронског пословања установљен је још пре 40ак година, када су успотављени први једноставни системи, који су користили веома популарне технологије за то време, које се називају *electronic funds transfer* (EFT – електронски трансфер новца), и *electronic data exchange* (EDI – elektronska razmena podataka). Oве системе и технологије практиковао је и даље релативно мали број људи.

У периоду између 1970 – 1980, популаризацијом АТМ машина коришћених од стране банкарских система, као и кредитних картица, успоставила се јака основа за растући свет електронске трговине. Овде се већ појављују прве популарне е-комерц платформе, као што су *Boston Computer Exchange* i *Mintel*.

У периоду између 1990 – 2000, World Wide Web је отворио врата бројним е-комерц сервисима да функционишу на глобалном нивоу. Најпознатији гиганти у свету електронске трговине, попут портала *Amazon.com* и портала *EBay* настали су у овом периоду. Meђутим, развој интернета није био довољан услов да елекронска трговина напредује. Морало се сачекати још пар година, док се нису развили сигурносни протоколи (*Http* на пример).

У периоду између 2000 па до данас, настало је на стотине сервиса електронског пословања који нуде најразличитики спектар производа и услуга које се могу купити онлајн, као што су сервиси за наручивање хране, сервиси за онлајн оглашавање, сервиси за стримовање мултимедијалног садржаја, онлајн продавнице, системи за електронско плаћање, као и бројни други

Крајњи резултат овог рада је функционална веб–апликација која ће бити коришћена од стране потрошача и продаваца . Потрошачи тј. купци су корисници који приступају веб-апликцији са намером да се упознају са понуђеним производима односно услугама, изаберу жељени производ, наруче га и плате.

Продавци су корисници, који излажу своје производе тј, услуге, и обављају администрирају над корисницима система, предметима који се продају, као и категоријама којима ти предмети припадају;

Захтави корисника које веб-апликација подржава су описани су у наредној глави. Ту се налази и листа неких већ остварених алата који решавају представљени проблем.

Коришћене технологије, упоређиване са алтернативама, као и карактеристичне ситуације у којима се аутор налазио у кључним тренуцима израде рада су описане у глави 3. Она садржи и опис имплементације и клијентске и серверске стране апликације.

У четвртој глави је приказан приказан графички изглед веб-продавнице као и начин њеног коришћења. Она још садржи и администраторско упутство за управљање веб-продавницом.

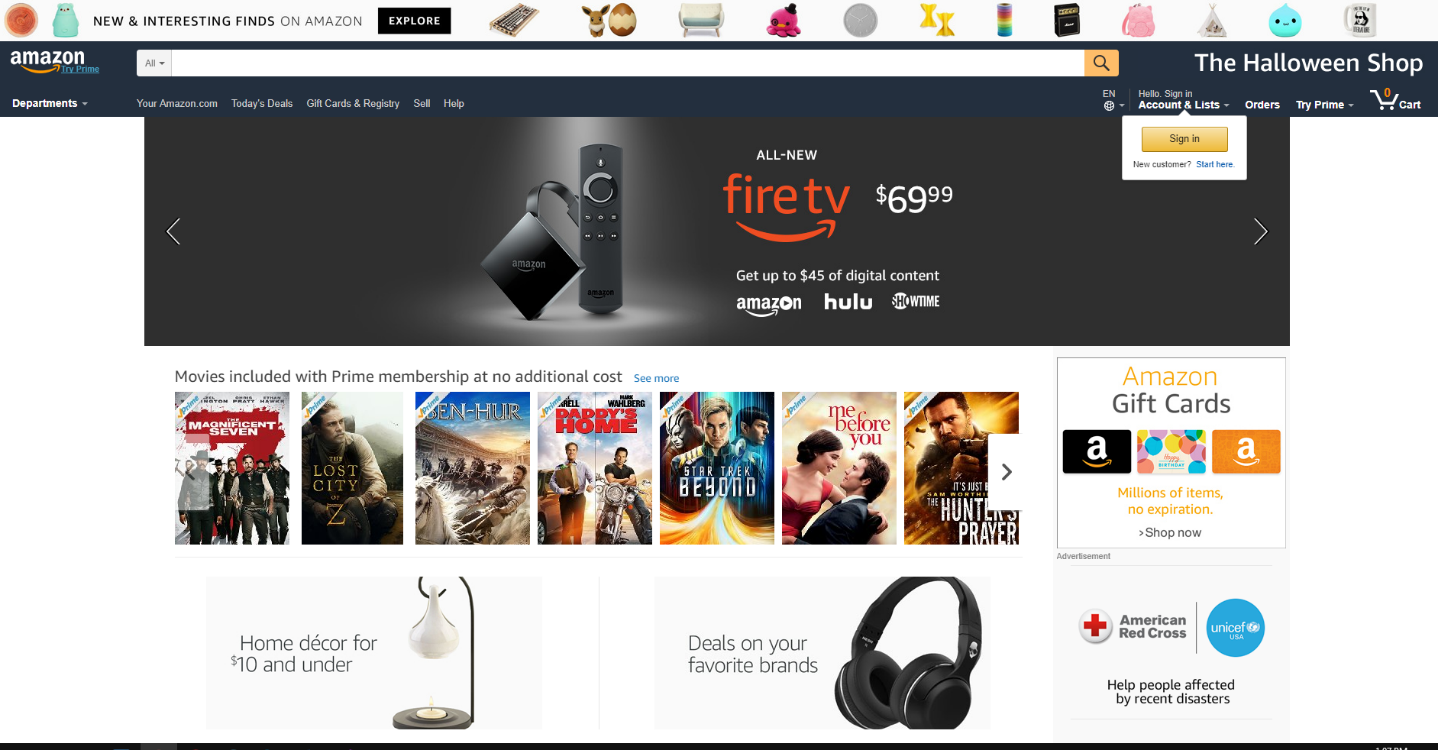
На крају рада дат је критички осврт на урађено, предности и мане у поређењу са већ постојећим алатима, као и могућности на које се рад може евенуално проширити или унапредити.

1. **Кориснички захтев и преглед сличних алата**

Почетна фаза израде сваког пројекта оваквог типа, пре него што се уопште почне са имплементационим и дијзајнерским одлукама, заснива сe на истраживању тржишта и прикупљању информација. На основу разултата ових активности, пракса је показала да је за успешан *online* продајни систем неопходно имплементирати већину (ако не и све) од следећих захтева:

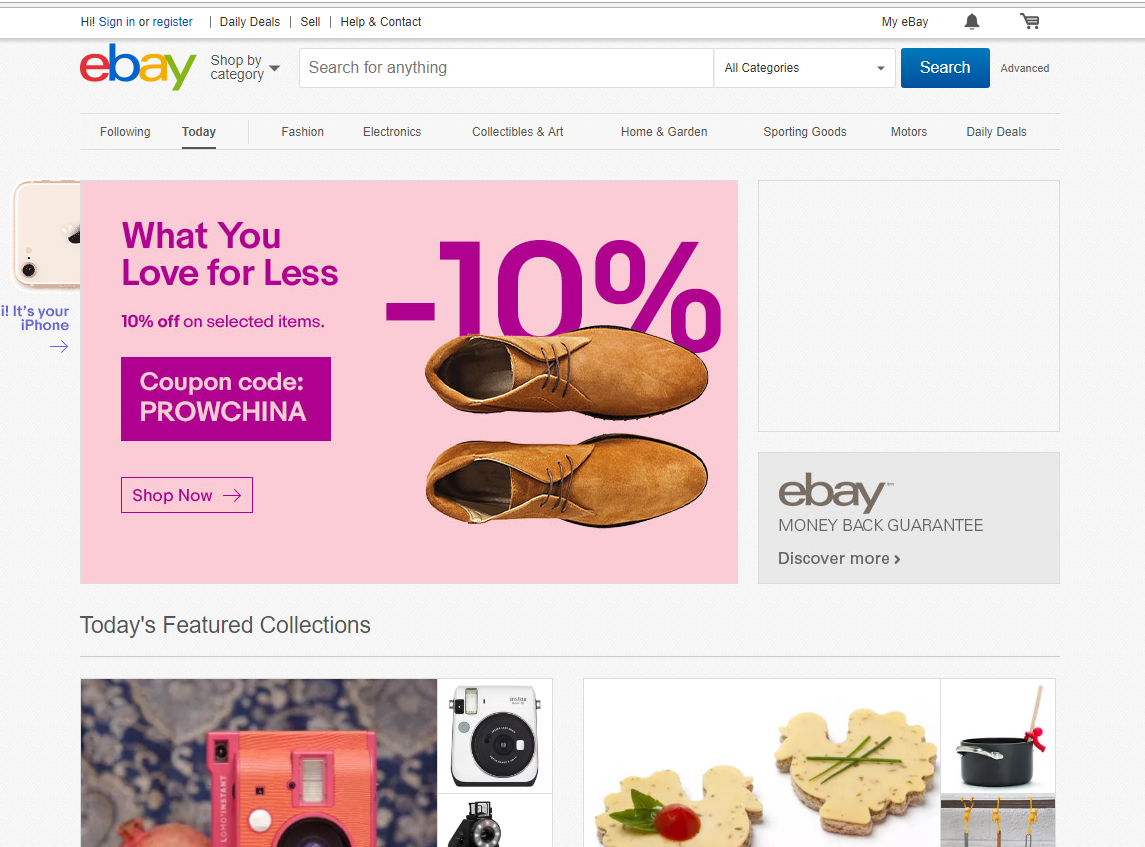
* *Online* презентација онога што се продаје, мора бити и текстуална и визуелна. Предмети који се продају, уколико има основа за то, требају бити распоређени у категорије. На тај начин би се олакшало евидентирање продајних предмета од стране продавца, уз значајно поједностављење њихове претраге од стране купца. Неопходно је да сваки предмет који се продаје буде описан ca релевантним информацијама (назив, цена, произвођач, техничке катактеристике,..).
* Функционалност виртуелне корпе треба да омогући:
* додавање предмета који се продају у виртуелну корпу;
* уклањање предмета из виртуелне корпе од којих је купац одустао;
* могућност бројања артикала који се тренутно налазе у виртуелној корпи, као и њихову укупну цену;
* преглед артикала који се налазе у виртуелној корпи;
* формирање наруџбенице, као и њено и плаћање кроз неки једноставан и сигуран портал за плаћање (*payment gateway*).
* Купац се на одређен начин обавештава да је његова трансакција успешно завршена (порука на екрану, email или sms порука...)
* заштићен систем за администрирање продавнице, тј. за кориснике система који ће водити евиденцију о свим корисницима система и предметима који се продају.
* сигурност, у смислу да сви осетљиви подаци које корисник оставља преко интернета морају да буду заштићени.
* једноставан кориснички интерфејс са евентуално вишејезичком подршком.

У случају трговином физичким предметима, компаније су задужене да на одређен начин доставе производ до купца, а у случају трговином мултимедијалних садржаја, софтвера, апликација или других виртуелних добара продавац је дужан да омогући преузимање купцу ових података. Најпознатије платформе за веб-куповину данашњице су дефинитвно Amazon.com i ЕBay. Amazon.com је највећа светска интернет трговина, настала 1994 као онлајн књижара. Ускоро је понуда проширена продајом DVD-а, видео касета, mp3 садржаја, па и продајом софтвера, видео-игара, електроником, одећом, па и храном, намештајем, играчкама и накитом. Веб сајт Амазона годишње посети преко 600 милиона корисника, па је компанија доста уложила у своје сервере да могу да поднесу велико оптерећење. Амазон је такође покренуо своје посебне странице у Америци, Европи, Азији и Океанији, и сваки од ових сајтова је посебан по ценама и понуди производа.



Слика 2.1 приказ странице Amazon.com

*eBay* je мултинационална е-комерц платформа, која је медијатор у трговниским односима између корисника система. Еbay.com је веб страница која функционише као продавница и онлајн аукцијска кућа, у којој људи и компаније могу да тргују разним добрима и сервисима. Услуге сајта су бесплатне за купце, али продавцима се наплаћује приказ њихових предмета, а такође и проценат од продатог предмета по обављеној трансакцији.



Слика 2.2 приказ странице еВау.com

Најпопуларније платформе код нас су Kupindo.com (веб сајт за аукцију и продају добара између корисника система), Polovniautomobili.rs (веб сајт за продају половних аутомобила), Donesi.com (платформа за онлајн наручивање хране...)

**3. Имплементација система**

У овој глави приказана је општа архитектура система, детаљан приказ модела базе података као и карактеристични контролни дијаграми. Такође у овој глави биће објашњена и мотивација приликом избора технологија и програмских модела који су коришћени у процесу развоја система.

**3.1 Oпшта архитектура**

*Web* продавница је замишљена као динамички интернет сајт, базиран на MVC (*Model View Controller*) шеми. *Front-end* апликација је урађена у *SPA* (*Single Page Application*) архитектури која обезбеђује корисничко искуство слично оном код *desktop* апликацијa. Интеракција корисника са *SPA*, подразимева честу коминикацију са *web* сервером иза сцене. *Back-end* страна је урађена у *Microsoft ASP.NET – Web API* технологији. Општа архитектура *web*-продавнице дата је на Слици 1.

|  |
| --- |
| WP04.bmp |
| Слика 3.1 Општа архитектура *web*-продавнице |

Имајући у виду да je реализација пројекта веб-продавнице замишљена у складу са горе наведеним концептом, аутор рада се одлучио за израду апликавије за веб-продавницу која ће бити специјализована за продају ручних сатова, при чему ће се користити следећи алати и технологије приликом развоја и имплементације пројекта:

***CA Erwin – Data Modeler 7.3,*** алат за моделовање података, прављење и одржавање релационих база. Основне карактеристике Erwin-а су:

* Брзо формирање базe података на основу направљеног модела. Може се спровести синхронизација између базе и модела, чиме се олакшава процес одржавања.
* Олакшан тимски рад на моделу.
* Омогућен је реверзни инжењеринг.
* Омогућено је дефинисање и одржавање конвенције о именима, атрибутима и типовима података.

***Microsoft SQL Server 2008*** као циљна релациона база података и *database* сервер. У њој се складиште сви релевантни подаци неопходни за функционисање система.

***Microsoft Visual Studio Enterprise 2015****,* као интегрисано развојно окружење намењено развоју рачунарских програма. У оквиру овог развојног окружења искоришћен је посебан додатак *Entity Framework (6.1.3)*, као архитектура за конекцију и комуникацију *API* контролера са базом података.

***C#***se koristi kao *back-end* programski jezik у оквиру *Microsoft ASP.NET – Web API* технологије.

***Microsoft Internet Infromation Services (IIS)*** се користи као web сервер. Протоколи које *IIS* подржава су: [*FTP*](https://sr.wikipedia.org/wiki/FTP)*,* [*FTPS*](https://sr.wikipedia.org/w/index.php?title=FTPS&action=edit&redlink=1)*,* [*SMTP*](https://sr.wikipedia.org/wiki/SMTP)*,* [*NNTP*](https://sr.wikipedia.org/w/index.php?title=NNTP&action=edit&redlink=1) *и* [*HTTP*](https://sr.wikipedia.org/wiki/HTTP)*/*[*HTTPS*](https://sr.wikipedia.org/wiki/HTTPS)*.*

***Angular JS***се користи као *front-end* радни оквир у циљу добијања динамичких веб страна система. Овај радни оквир прилагођава и проширује традиционални *HTML* како би представио динамички садржај кроз двосмерно везивање података.

|  |
| --- |
|  |
| Слика 3.2 Дијаграм разграничења клијентске стране са леве и десне серверске стране. |

Због овакве структуре Aнгулар омогућава аутоматску синхронизацију између *model*-a и *view*-а у *MVC* архитектури*.* Na *front-end* strani се jos користи *Bootstrap, jQuery, JS, CSS* и *HTML*. Графички приказ спајања две различите технологије у оквиру пројекта web продавнице дат је на Слици 2.

**3.2 База података**

Као што је претходно наведено, за потребе смештања и чувања података искоришћен је *Microsoft SQL Server 2008* database сервер. Модел релационе базе података „Prodavnica“ приказан је на Слици 3.

|  |
| --- |
|  |
| Слика 3.3 *ER* модел базе података |

Табела **korisnik** је намењена за регистрацију две групе корисника. Прву групу чине особе који су корисници услуга веб-продавнице (купци), док се у другој групи налазе особе за администрацију веб-продавнице. Диференцијација ове две групе корисника се реализује на нивоу атрибута „role“ који може имати вредност „user” или “admin”.

Табела **proizvod** намењена је за смештање података о производима који ће бити у понуди web- продавнице. Производи се међусобно диференцирају у на основу категорија, које су у нашем случају сведене на ручне сатове за мушкарце, жене или децу. Корисник из „admin“ роле има могућност додаваља нових категорија, брисања или ажурирања постојећих категорија, уз стриктно поштовање референцијалног интегритета када је реч о брисању одређених категорија (ако категорија садржи производе не може бити обрисана).

Дефиниције категорије производа се смештају у табелу **kategorija**.

Табела **narudzbenica** је намењена за евидентирање података коју су дефинисани коначним поступком куповине. У њој се налазе основни подаци о купцу, потребни за реализацију поруџбенице тј. доставу, док се сви производи из корпе, по основу једне наруџбенице смештају у табелу **narudzbenicaProizvod**.

Табела **slike** је табела-дете (*child)* табеле **proizvod**, у којој се смештају путање до слика које су неопходне за приказ и детаљан преглед одређеног производа.

Евидентно је да у *ER* моделу базе података „Продавница“ није дефинисана табела „Корпа“, као саставни део сваког модела базе података која је намењена *е*-трговини.

Аутор се одлучио да се табела Kopпa „виртуелизује“ на *front-end* страни, имајући у виду да *HTML5* стандард, (доступан још од 2008. год.) заједно са .*Angular JS framework*-ом пружа методе за креирање апликација које могу да сачувају одређену количину информација (кеширање, *localStorage*, *sessionStorage*...) на клијентској страни, тако да је апликација потпуно функционална и без приступа интернету, а када се веза успостави пружа могућност да се сви подаци обраде на правилан начин.

Примена оваквог чувања података не мора да се огледа само у случајевима прекида везе, већ се може користити за брже учитавање, односно извршавање саме апликације. Управо из овог разлога искоришћена је oва могућност, као једно квалитетно унапређење у области израде апликација намењених е-трговини.

**3.3 Back end апликација**

*Entity Framework* je компонента у оквиру .NET платформе која обезбеђује *ORM (object relational mapping)* приступ у програмирању, чиме значајно олакшава развој data-оријентисаних апликација. *Entity Framework* девелоперима омогућује да се подацима баве у форми објеката и одговарајућих карактеристика, уместо да директно баратају табелама и колонама база података. Реч је o *Microsoft*-овој технологији која омогућава приступање релационој бази података помоћу објектно-оријентисаних концепата. У архитектури *Entity Framework*-а разликујемо:

* Модел ентитета (енг. *Entity* *Data Model (EDM)*) који успоставља везу између

објектног модела и базе података;

* Објектне сервисе који представљају скуп класа које се генеришу и омогућавају

рад са базом података;

* Слој који користи синтаксу .*NET* програмских језика за постављање упита над

базом података.

На Слици 4 приказана је архитектура *Entity Framework*-а.

|  |
| --- |
|  |
| Слика 3.4 Архитектура *Entity Framework*-а. |

Да би било могуће користити *Entity Framework* у пројекту, потребно га је прво конфигурисати у *Visual Studio*-у укључивањем посебне компоненте *EntityFramework* у пројекат, затим следи дефинисанје *EDM*-а, додавање пројектних референци, дефинисање параметара за повезивање на базу (енг. connection string) и сл.

*Entity Framework* обезбеђује три начина за дефинисање *EDM*-а:

* *Database first* тј. дефинисање *EDM*-а на основу базе података;
* *Model first* тј. генерисање *EDM*–а, па на основу њега генерисање базе података;
* *Code only* тј .коришћење *Plain Old CLR Objects - POCO* ентитета ( класа која се

састоји искључиво од чланова класе и одговарајућих *get* и *set* метода).

За израду пројекта *web* продавнице, коришћенa је методa *Database first*. Коришћењем алата *Entity Data Model Wizard*  конекцијом на *database server* и *Sql bazu* „Продавница“, генерисан је *EDM* приказан на Слици 5.

|  |
| --- |
|  |
| Слика 3.5 Генерисани *EDM (Entity* *Data Model)* |

Параметри за повезивање на базу које ће апликација користити, се чувају у XML датотеци *web.config*. Конекциони стринг u конкретном случају изгледа овако:

|  |  |
| --- | --- |
| Параметри за повезивање на базу – *connection string* |  |
| <<connectionStrings>  <add name="prodavnicaDbContext" connectionString="metadata=res://\*/Models.ProdavnicaModel.csdl|res  ://\*/Models.ProdavnicaModel.ssdl|  res://\*/Models.ProdavnicaModel.msl;provider=System.Data.SqlClient;provider  connection string=&quot;data source=10.0.80.129;initial catalog=prodavnica;persist  security info=True;user id=sa;password=sa;MultipleActiveResultSets=True;  App=EntityFramework&quot;" providerName="System.Data.EntityClient" />  <add name="prodavnicaEntities" connectionString="metadata=res://\*/Models.ProdavnicaModel.csdl|res://\*/Models.ProdavnicaModel.ssdl|res  ://\*/Models.ProdavnicaModel.msl;provider=System.Data.SqlClient;provider  connection string=&quot;data source=10.0.80.129;initial catalog=prodavnica;persist  security info=True;user id=sa;password=sa;multipleactiveresultsets=True;application  name=EntityFramework&quot;" providerName="System.Data.EntityClient" />  </connectionStrings> | |

Генерисањем *EDM*–а аутоматски се, између осталог, генерише и датотека са изворним кодом за *C# класе*. У наставку, следи пример генерисане *C#* класе за ентитет „категорија“.

|  |
| --- |
| Генерисан класа за ентитет категорија добијена на основу *EDM* -а |
| //------------------------------------------------------------------------------  // <auto-generated>  // This code was generated from a template.  //  // Manual changes to this file may cause unexpected behavior in your application.  // Manual changes to this file will be overwritten if the code is regenerated.  namespace Prodavnica.Models  {  using System;  using System.Collections.Generic;    public partial class kategorija  {  [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]  public kategorija()  {  this.proizvod = new HashSet<proizvod>();  }    public int idKategorija { get; set; }  public string nazivKategorija { get; set; }    [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]  public virtual ICollection<proizvod> proizvod { get; set; }  }  } |

*ASP.NET Web API* (***Application Programing Interfaces)*** је радни оквир за изградњу HTTP сервиса намењеног широком спектру клијената, укључујући web-прегледаче, мобилне уређаје и сл. Веома је сличан *ASP.NET MVC* архитектури јер садржи *MVC* функције као што су рутирање, контролере, резултате акција и сл. Иако *ASP.NET Web API* није део *MVC* радног оквира, може се користити са *MVC* технологијом, као и са осталим врстама веб-апликација као што су *ASP.NET WebForms*. Такође се може користити као самостална апликација за *Web* сервисе, како је и у овом пројекту урађено.

Основне карактеристике *ASP.NET Web API* радног оквира:

* Подржава *CRUD* функције, које раде са HTTP сервисима *GET*, *POST*, *PUT* и *DELETE*

Одговори (Responses) имају *Accept* заглавље као и *HTTP* статусни код.

* Одговори (Responses) су форматирани од стране *Web API’s MediaTypeFormatter* у *JSON*, *XML* или било који други формат додат као *MediaTypeFormatter*.
* Може да се *host*-ује у оквиру апликација или на Web serveru (IIS), као што је случај у овом пројекту

Подржава архитектуру у сегменту рутирања, контролера, акцијске резултате које враћају контролери и сл.

|  |
| --- |
|  |
| Слика 3.6 Базни дизајн апликације која користи Web API технологију |

Структура API контролера proizvodController који се користи у пројекту је приказана у коду који следи. Контролер proizvodController у себи садржи сет *CRUD* функција. (CRUD је акроним за четри основна типа SQL команди : *Create*, *Read*, *Update*, *Delete)*. За остваривање везе са базом неопходно је направити нову инстанцу класе за приступ, која је добијена кроз поступак формирања *Entity* *Data Model-а.*

|  |
| --- |
| private prodavnicaEntities db = new prodavnicaEntities(); |

Објекат db у себи садржи конекциони стринг ка бази података, прочитан из XML датотеке web.config. Комуникација између апликационог слоја и базе података је остварена коришћењем *LINQ* технологије. *LINQ* је скраћеница за *Language Integrated Query* што означава да је у питању екстензија за .*NET* компатибилне програмске језике која омогућава извршење упита над локалним или удаљеним колекцијама података. Језици који подржавају *LINQ* могу да пруже комплетну проверу типова и валидацију упита већ у фази компајлирања.

Функција *Getkategorija*(), треба да врати клијентској страни листу свих категорија из табеле kategorija. Имајући у виду да класа kategorija има референцу и на табелу proizvod, оригинална функција би нам поред података из табеле kategorija вратила и одговарајуће референциране податке из табеле proizvod. То је знатно већи скуп података, од онога што је неопходно за приказ на клијентској страни. Из тог разлога функција *Getkategorija*() је модификована на тај начин што је оргинални скуп података који је упитом из баzе враћен у структури оргиналне класе из модела kategorija.cs, трансформисан у помоћни - *Data Transfer Object (DTO)* kategorijaDto. Упоређењем оргиналне класе из *EDM*-а kategorija и *DTO* класе kategorijaDto, долазимо до закључка да је класа kategorijaDto проста класа која само садржи тзв. get-ere и set-ere ослобођена свих осталих непотребних података које би се клијентској страни пренели употребом оригиналне класе kategorija

|  |  |
| --- | --- |
| Пример : *DTO* класа kategorijaDto |  |
| namespace Prodavnica.Models.dto  {  public class kategorijaDto  {  public int idKategorija { get; set; }  public string nazivKategorija { get; set; }  }  } | |

Начин на који је извршена трансформацијa добијених података у структури оргиналне класе kategorija у структуру класе kategorijaDto је приказан је у коду који следи.

|  |  |
| --- | --- |
| Пример *ASP.NET Web API* контролера proizvodController његове [HttpGet] функције Getkategorija () |  |
| namespace Prodavnica.Controllers  {  // [EnableCors("\*", "\*", "\*")]  public class kategorijaController : ApiController  {  private prodavnicaEntities db = new prodavnicaEntities();  //[Authorize(Roles = "admin")]  // GET: api/kategorija  // public IQueryable<kategorija> Getkategorija()  public IQueryable<kategorijaDto> Getkategorija()  {  **//\*\*\*\*\* LINQ упит \*\*\*\*\***  var kategorija = from kat in db.kategorija  **//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  select new kategorijaDto()  { idKategorija = kat.idKategorija,  nazivKategorija = kat.nazivKategorija };  return kategorija;  // return db.kategorija;  }  // GET: api/kategorija/5  [ResponseType(typeof(kategorija))]  public IHttpActionResult Getkategorija(int id)  {  kategorija kategorija = db.kategorija.Find(id);  if (kategorija == null)  {  return NotFound();  }  var kat=new kategorijaDto()  {  idKategorija = kategorija.idKategorija,  nazivKategorija = kategorija.nazivKategorija  };  return Ok(kat);  } putanjaSlika = s.putanjaSlika,  })  };  return proizvodi;  // return db.proizvod;  } | |

*Microsoft ASP.NET Web API*  контролер у суштини, клијентској страни враћа податке у *XML* структури. Пример података које је api контролер proizvod вратио клијентској страни у *XML* структури дат је на Слици 6.

|  |
| --- |
|  |
| Слика 3.7 HTTP Response API контрилера kategorijaController и функције Getkategorija () |

Имајући у виду да је данас ***JSON (JavaScript Object Notation)*** је један од лакших текстуалних отворених стандарда, дизајниран за читљиву размену података, као и то да **JSON** форматполако али сигурно замењује *XML* као стандард јер има неколико предности у односу на употребу *XML*-а, аутор је морао да прилагоди конфигурацију самог *VS* пројекта тако да *API* контролери у који се користе у пројекту пре враћања података изврше њихову серијализацију и уместо *XML* структуре врате податке у ***JSON* ф**ормату.

Предности размене података у ***JSON* формату леже у чињеници да се избегава**  коришћење тагова, па стога има краћи код који је лакши за писање и разумљивији за читање. Ипак, најбитнија предност ***JSON***-а у односу на *XML* је та што се ***JSON*** парсира кроз стандардну **JavaScript** функцију док се *XML* парсира кроз *XML* парсер.

Да би се у пројекту добила ова функционалност, неопходно је да се функција Register(HttpConfiguration config) у класи *WebApiConfig* измени на следећи начин.

|  |  |
| --- | --- |
| Пример: Ф-ција Register(HttpConfiguration config) у класи *WebApiConfig* |  |
| namespace Prodavnica  {  public static class WebApiConfig  {  public static void Register(HttpConfiguration config)  {  // Web API configuration and services  // config.EnableCors(new EnableCorsAttribute("\*", "\*", "\*"));  // Web API routes  config.MapHttpAttributeRoutes();  config.Routes.MapHttpRoute(  name: "DefaultApi",  routeTemplate: "api/{controller}/{id}",  defaults: new { id = RouteParameter.Optional }  );  **// Dodatak koji obezbedjuje vracanje podataka u JSON Formatu \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **var jsonFormatter = config.Formatters.OfType<JsonMediaTypeFormatter>().First();**  **jsonFormatter.SerializerSettings.ContractResolver = new CamelCasePropertyNamesContractResolver();**  **// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  jsonFormatter.SerializerSettings.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore;  }  }  } | |

За тестирање рада функција у оквиру контролeра у пројекту, аутор је користио алат *HttpRequester*. Алат је из скупа додатака уз *Mozilla FireFox web* претраживач.

Пример теста функције Getkategorija () са добијеним резултатима, кроз алат *HttpRequester* дат је на Слици 7. Можемо закључити да је резултат позива функције враћен у **JSON ф**ормату.

|  |
| --- |
|  |
| Слика 7. HttpRequester – позив функције Getkategorija () |

**3.4 Front end апликација**

*AngularJS* је коришћен као *front-end* радни оквир отвореног кода, који се заснива на *MVC (Model-View-Controller)* концепту. То подразумева да се *front-end* компонента апликације састоји из модела, апликационе логике и презентационог дела. Модели се користе за комуникацију са базом, док су у апликационој логици смештени контролери, посредници који уз помоћ *AngularJS* сервиса комуницирају између базе података и презентационог дела. *AngularJS* омогућава проширење *HTML* синтаксе у складу са потребама дате апликације. За форматирање и дефинисање изгледа *HTML* документа користе се *CSS* документа, док су *ЈavaScript* документа и *jQuery* библиотеке задужени за динамичко понашање страница.

*Front end* страну веб-продавнице у основи чине два *AngularJS* пројекта. Први пројекат је сама веб продавница, док је други намењен за њену администрацију.

Пројектна структура веб-продавнице дата је на Слици 8.

|  |  |
| --- | --- |
|  |  |
| AJS projekat Web Prodavnica | AJS projekat – Administarija web prodavnice |
| Слика 3.8 Пројектна организација front end стране web продавнице | |

*View* представља део *MVC* концепта који обезбеђује *interfejs* помоћу кога корисник уноси податке и позива одговарајуће операције које треба да се изврше над моделом. *View* приказује кориснику стање модела. Све *HTML* странице у пројекту су компоненте *view* -а. Агулар проширује *HTML* код и уводи двосмерно везивање података (приказано на Слици 9). Све акције контролера позивају се посредством Ангулара, Ангулар прима резултате и уграђује их у *view*.

|  |
| --- |
| Srodna slika |
| Слика 3.9 AngularJS двострано везивање података |

Први корак који је потребно направити како би се омогућило коришћење Ангулар радног оквира на wеб страници, је учитати Ангулар код. Затим, како бисмо повезали Ангулар апликацију са *HTML* документом, потребно је у *HTML* документу (најчешће) у оквиру сегмента *body* на документу, написати *ng-app="imeAplikacije".* На тај начин се дефинише која апликација ће имати контролу над *HTML* документом. На пример, почетак *HTML* документа Index.Html web-продавнице изгледа овако:

|  |  |
| --- | --- |
| Пример . Повезивање (HTML документа и Ангулар апликације: WebApiAuth) |  |
| <body >  <div ng-app="WebApiAuth" data-ng-controller="homeController">  <!--top-header-->  <div class="top-header"> | |

Ангулар *Module* је целина која се може састојати од неколико врста компонената: controller, directive, service, factory, constant, и тд. *module* креирамо на следећи начин:

*angular.module(’imeModula’, [zavisne komponente]).*

На тај начин се ствара апликацијски модул чији је назив ’imeModula’. У угластим заградама се наводе називи осталих модула од којих основни зависи.

|  |  |
| --- | --- |
| Пример . (Креирање и регистрација Ангулар модула) |  |
| var appweb = angular.module('WebApiAuth', ['WebApiAdmin','ngRoute', 'LocalStorageModule','ngCart']); | |

Контролер у Ангулару користимо за дефнисање понашања *HTML* кода. Креирамо га тако да методи .controller() проследимо назив контолера, сервисе од којих он зависи и на крају функцију за креирање контролер објекта. Сервис који је задужен за праћење свих променљивих унутар апликације је *$scope* модул. Све променљиве које требају да буду видљиве на HTML страни су заправо променљиве *$scope* модула.

|  |  |
| --- | --- |
| Пример . (Креирање и регистрација Ангулар контролера) |  |
| appweb.controller('kategorijeController', ['$scope', 'dataService', function ($scope, dataService) {  $scope.kategorije = [];  dataService.getAll('api/kategorija/').then(function (results) {  $scope.kategorije = results.data;  }, function (error) {  //alert(error.data.message);  });  }]); | |

Осим самог креирања контролера, потребно га је и везати на одређени *HTML* елемент. У ту сврху користимо директиву ng-controller="ctrl" ˇчиме заправо кажемо да ће тим *HTML* елементом и свим његовим *child* елементима управљати контролер "ctrl".

|  |  |
| --- | --- |
| Пример . (Везивање контролера на документ) |  |
| <div ng-controller="homeController">  <!--top-header-->  <div class="top-header">  <div class="container">  <div class="top-header-main">  <div class="col-md-8 top-header-left"> | |

Ангулар factory је функција за прављење сервиса који ће се касније користити у директивама и контролерима. Најчешћа намена *factory* функције је комуникација са *web*-сервером, у циљу добијања или слања података са/на *web*-сервер.

|  |  |
| --- | --- |
| Пример . Креирање и регистрација Ангулар factory сервиса |  |
| appweb.factory('dataService', ['$http', 'ngAppSettings', function ($http, ngAppSettings) {  var serviceBase = ngAppSettings.apiServiceBaseUri;  var url='';  var dataServiceFactory = {};  var \_getAll = function (url) {  return $http.get(serviceBase + url).then(function (results) {  // console.log("getall "+serviceBase + url);  return results;  });  };  dataServiceFactory.getAll = \_getAll;  return dataServiceFactory;  }]); | |

Модул $http је сервис који се користи за све *HTTP* захтеве којима контролер треба да рукује. Уз помоћ овог сервиса обезбеђени су позиви свих *HTTP* захтева као и управљање одговорима добијених са удаљене серверске стране. Подаци са удаљених сервера се добијају у текстуалном или JSON формату. *$http* сервис има следеће скраћене методе за позив $аjax сервиса:

* $http.get
* $http.post
* $http.head
* $http.put
* $http.delete
* $http.patch
* $http.jsonp

У наставку ће бити приказан пример из пројекта, у коме се позива *$http.post* метода који се односи на процес аутентикације корисника, који се пријављује на веб-продавницу. Аутор се одлучио да за аутентикацију користи JSON Web Token (JWT).

Приликом пријављиваља на веб-продавницу позива се функција factory сервиса authService.login. У овој функцији се на основу предатог параметра loginData, који у себи садржи корисничко име и лозинку позива *$http.post* методу за $аjax сервис.

|  |  |
| --- | --- |
| Пример |  |
| var login = function (loginData)  {  var data = "grant\_type=password&username=" + loginData.username + "&password=" +  loginData.password;    var deferred = $q.defer();    $http = $http || $injector.get('$http');  **$http.post**(serviceBase + 'token', data, { headers: { 'Content-Type': 'application/x-www-form-urlencoded' } }).then(function (response) {  localStorageService.set('authorizationData', {  token: response.data.access\_token,  username: loginData.username,  refreshToken: response.data.refresh\_token,  role: response.data.role,  id:response.data.id})    var authData = localStorageService.get('authorizationData');  //console.log(authData);  \_fillAuthData();  deferred.resolve(response);  },  function (err, status) {  \_logOut();  deferred.reject(err);  });    return deferred.promise;  }; | |

Уколико искористимо алат *HttpRequester* за позив API контролера token са тачним подацима за корисничко име и лозинку, које смо предали функцији *factory* сервиса *authService.login* као одговор на *HTTP-post* захтев уследиће потврдан одговор на исти, са следећим подацима у JSON формаату.

|  |
| --- |
|  |
| Слика 3.10 HttpRequester – позив функције Getkategorija () |

Параметри које прима $http.post (као и остали доступни методи овог модула) су следећи:

* Обавезан први параметар URL сервиса који се позива:

http://localhost:3827/token,

* Објекти који представљају параметре датог позива:

var data = "grant\_type=password&username=" + loginData.username + "&password=" + loginData.password;,

* Опциони конфигурациони објекти :

{headers:{'Content-Type':'application/x-www-form-urlencoded'}}

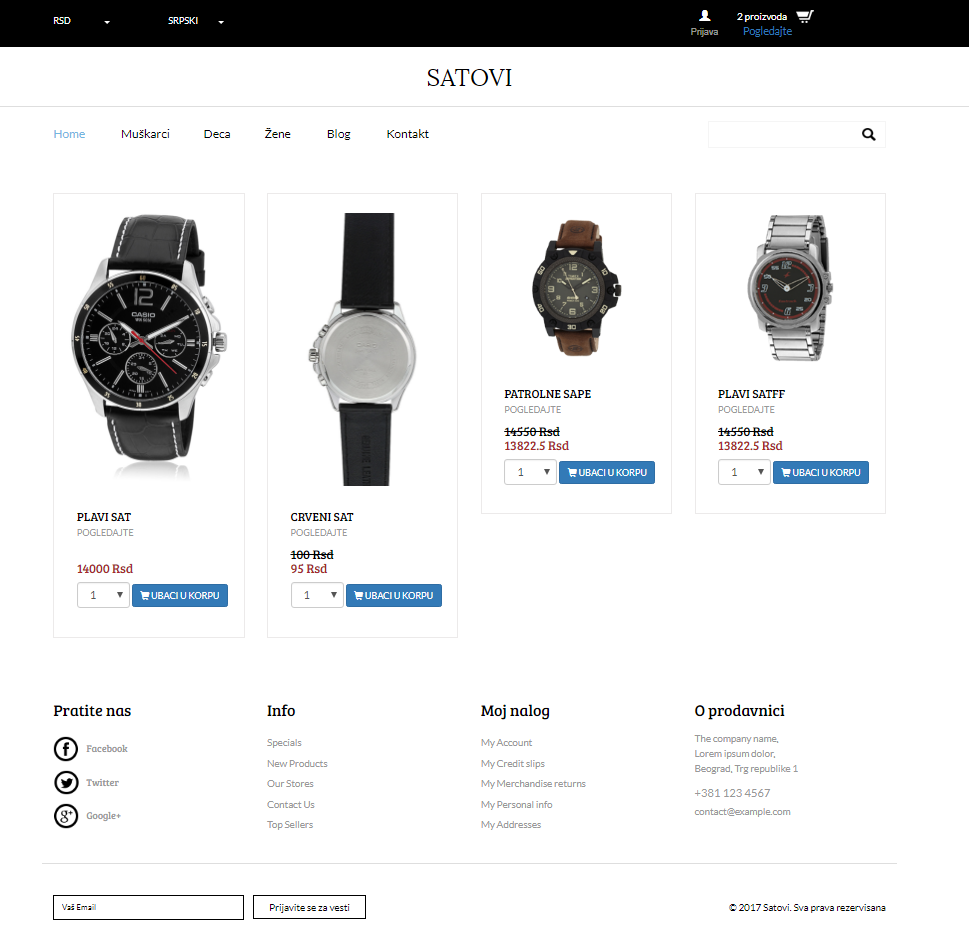
Прва наведена функција након тога (then(function)) представља функцију која ће бити извршена уколико са сервера као повратна вредност дође одговор који означава успех.

Уколико са сервера као повратна информација дође неки негативан одговор, за руковање том ситуацијом задужена је друга наведена функција (function (err, status)).

Обе функције примају један параметар *response*. Овај параметар представља ResponseEntity објекат који у себи поред статуса који је сервер вратио може да носи додатне податке везане за клијент-сервер комуникацију, као и саме податке које се очекују да буду враћени са сервера као резултат извршене акције.

1. **Опис софтверског система**
   1. **Корисничка апликација**

Отварањем почетне стране апликације добија се следећи приказ.

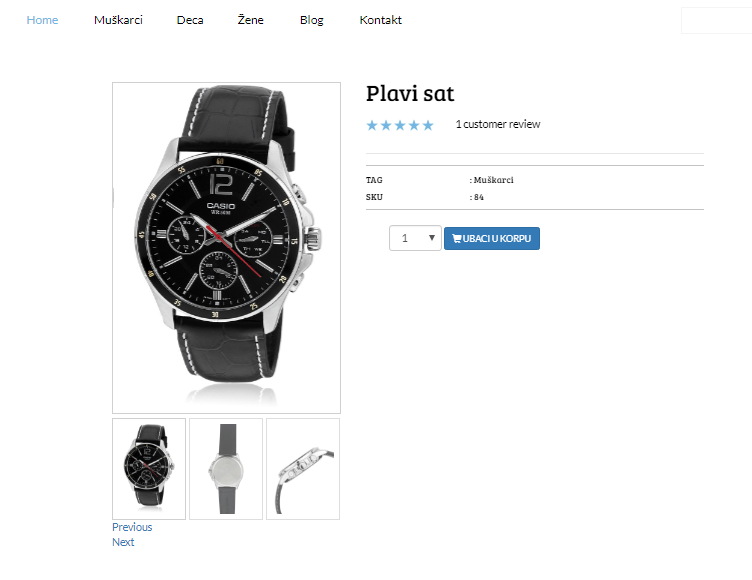


Слика 4.1. - почетна страница

У заглављу странице се налазе два падајућа менија, у којима се налазе опције за избор језика и жељене валуте. Такође, ту се налази и дугме за пријаву на систем, као и дугме које описује тренутно стање виртуелне корпе. У почетном стању, корпа је празна.

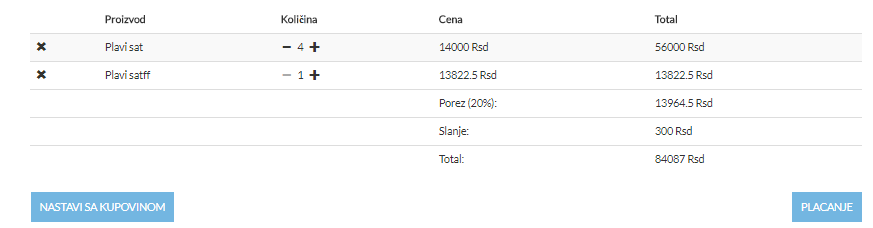
Испод заглавља, је сет дугмади којим се врши селекција по категоријама производа у понуди. У почетном стању, (*Home* дугме је активно), нема избора категорије и сви производи су приказани. Кликом на одређену категорију ( У овом примеру, Мушкарци, Жене, Деца...) производи се филтрирају по тој, изабраној категорији. Ту се налази и дугме које би упућивало на контакт страницу продавнице. Такође, обезбеђено је и поље за претраживње предмета по називу.

Избором одређеног предемта из понуде (кликом на њега), отвара се следећи приказ.



Слика 4.2 – преглед продајног предмета

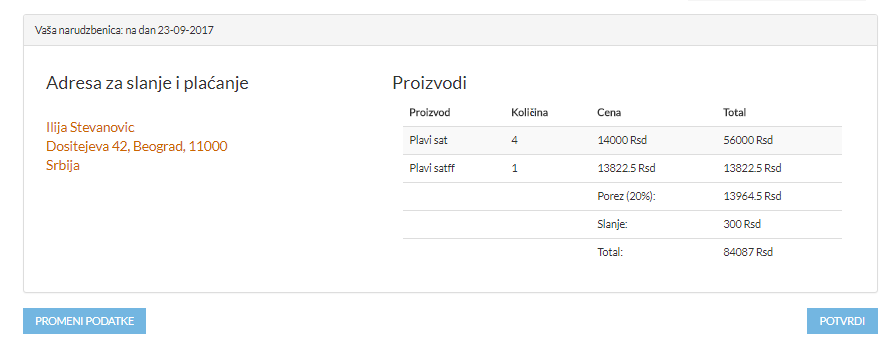
Поред прегледа слика које га описују, имплементирана је опција “Ubaci u korpu”, која, избором жељеног броја предемта, како и сам назив дугмета сугерише, додаје предмете у корпу. Корпа остаје попуњена предметима иако се кликне дугме refresh на претраживачу.



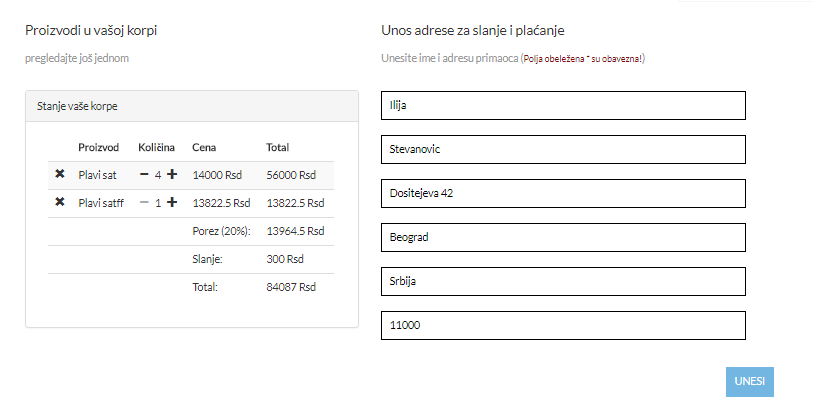
Слика 4.3 – приказ виртуелне корпе

Кликом на икону корпе у заглављу, отвара се приказ објашњен у слици 10. Ту се налазе 2 дугмета, “Nastavi sa kupovinom”, које враћа корисника на почетну страну немењајући тренутно стање корпе, и “plaćanje”. Takoђе, у редовима који описују предмете у корпи, налазе се дугмићи који омогућују уклањање одабраног предмета из корпе, као и дугмићи у колони за количину, + и –, који управљају количином наручених предмета датог производа у корпи.

Дугме за плаћање упућује на две разлиличите акције за два карактеристична случаја - када је корисник пријављен на систем, и када није. Уколико је корисник пријављен, отвара се следећи приказ.

 Слика 4.4 – приказ информација о кориснику

Поред основних информација о кориснику, (попут имена, презимена и адресе становања), налази се поновљен приказ садржаја корпе и њене вредности, да би корисник могао још једном да прегледа да ли је његова наруџбина прошла без грешке. Ту се такође налазе два дугмета, “Promeni podatke” I “Potvrdi”. Дугме за промену података отвара следећу форму.



Слика 4.5 – форма за промену података

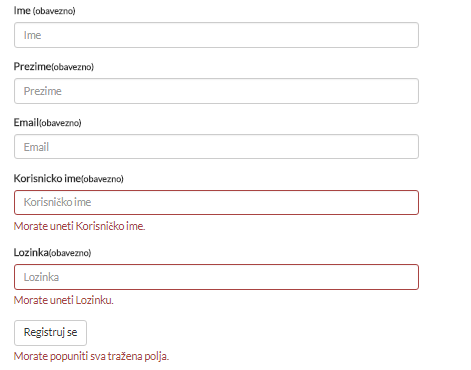
Уколико је дошло до погрешног наручивања, или неки од података везаних за плаћање морају бити промењени, (на пример име наручицоа или адреса за доставу), на овој форми се то може изменити и тиме освежити приказ тренутне наруџбенице.  
Кликом на дугме “Potvrdi”, трансакција се завршава, и шаље се *e-mail* потврда о успешности трансакције у сандуче корисника. У сваком тренутку, кликом на дугме “Odjava” које се налази у заглављу апликације, корисник може да се одјави из система.

Уколико се на приказу описаном на слици 10. кликне на дугме за плаћање, а корисник није пријављен на систем, отвара се следећи приказ.



Слика 4.6 – форма за пријаву и регистрацију

Кориснику су понуђене две могућности. Једна за пријаву корисника на систем, уколико корисник већ има свој кориснички порфил на систему, и друга, која упићује корисника на форму за формирање новог налога, која је приказана на следећој слици.

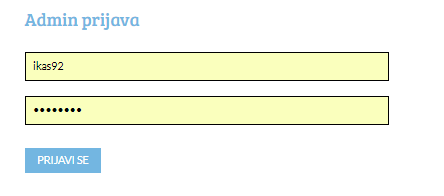


Слика 4.7 – регистрација новог корисника

Имплементирана је класична форма за регистрацију новог корисника. У placeholder text-u je описано шта се од корисника очекује да унесе, при чему се врше неке основне провере унетих података (нпр. формат email адресе). Дугме за регистрацију ће бити омогућено тек када су испуњени сви услови који задовољавају форму, то јест када су сва поља исправно попуњена. Дугме за регистрацију креира профил корисника у бази података, и враћа га поново на форму за пријаву (слика 13).

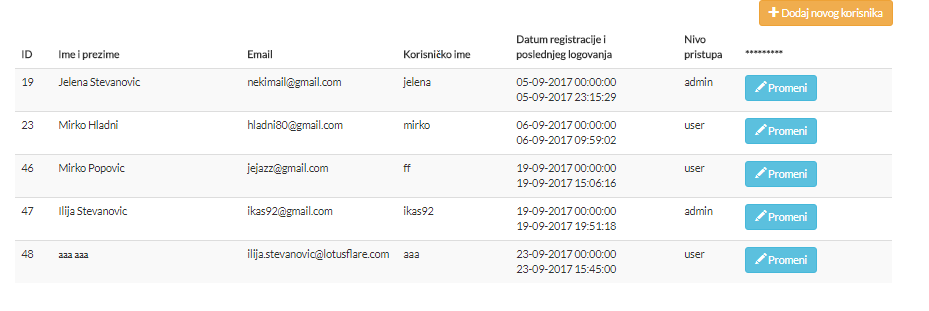
* 1. **Опис опција за рад администратора**

Како је и раније речено, админ апликација је развијана као засебна апликација. Администратори су у овом случају супервизори система, имају своју независну форму за пријављивање изван система. До ње се долази додавањем /admin на основној *url* адреси веб-продавнице, у претраживачу на почетној страни апликације. На следећој слици приказан је изглед форме за логовање администратора.

Слика 4.8 – форма за пријављивање администратора

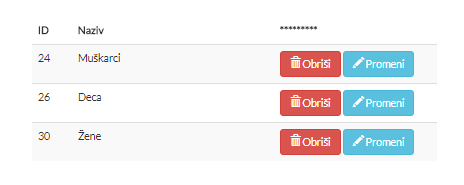
Пријавом администратора на систем, кориснику се отварају администраторске могућности система, које омогућују преглед укупног броја производа, укупан број наруџбеница, преглед корисника, производа, категорија и наруџбеница.

  
 Слика 16 – админ панел  
  
Избором опције “корисници”, администратору се отвара следећа форма

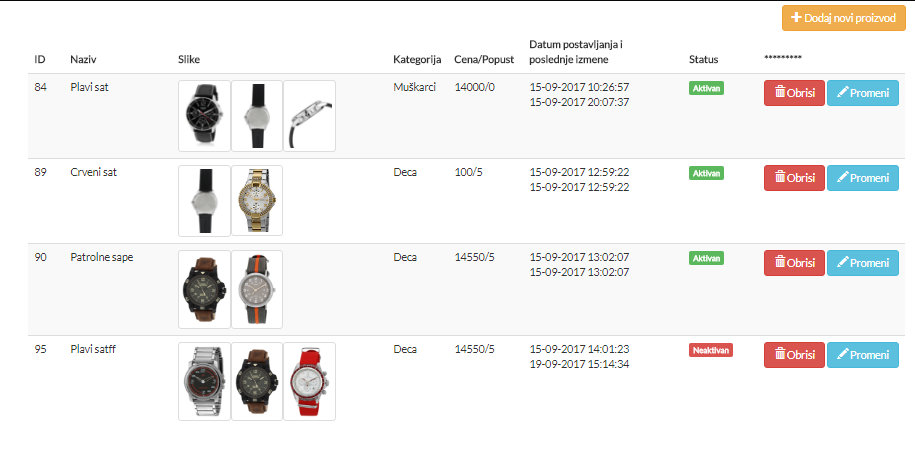
  
Слика 4.9 – преглед корисника

Приказ, поред прегледа основних информација о кориснику, даје и последњи датум приступа и датум регистрације корисника. Такође, њему се отварају разне могућности за манипулацију са корисничким налозима система, попут измене одређених поља, мењање типа корисника admin/user... Администратор има опцију додавања новог корисника. Притиском на дугме “Dodaj novog korisnika”, отвара се форма слична оној приликом регистрације новог корисника. Форма има додатно поље - тип корисника, чиме се администратору поред могућности додавања обичног корисника омогућава да креира и нове администраторе система.

Такође, администратор система има опцију да манипулише са производима и њиховим категоријама, бирањем опција “kategorije” ili “proizvod”. Треба напоменути да се поштује референцијални интегритет података при брисању категорија, то јест, није могуће обрисати категорију уколико она садржи производе који су у тој категорији.

  
Слика 4.10 – манипулација категоријама

Форме за манипулисање производима и категоријама које припадају су поједностављене за лако коришћење од стране администратора. Могућа је измена производа, уклањање производа са листе, као и могућност сетовање статуса (активан/неактиван) предмет које може имати разне практичне примене (на пример, није на стању).



Слика 4.11 – манипулација производима

Избором менија „наруџбенице“ на главној форми, отвара се историја свих остварених наруџбеница од почетка праћења евиденције.

  
Слика 4.12 – преглед остварених наруџбеница

Излистане су разне информације од значаја, попут информација о кориснику који је остварио наруџбину, његовој адреси, датуму и времену остваривања наруџбенице. Овде је дато и дугме “Proizvodi” које приказује наручене производе за изабрану наруџбеницу.  
Администратор има и опцију “Vratite se na sajt”, чиме прелази у класичан кориснички режим, и том приликом му се отвара преглед корисничког сајта са корисничким функционалностима.

1. **Закључак**

Резултат овог пројекта је динамичка веб-апликација која служи за интернет презнтацију веб продавнице. Веб продавница у раду је демонстрирана као продавница која је специјализована за продају сатове путем интернета. На основу начина на који је имплементирана, може може се лако прерадити за било коју продају. Имајући у виду да продавница која је имплементирана у пројекту задовољава већину корисничких захтева који су раније наведени (визуелна презентација продајних предмета са могућношћу њиховог детаљног описа, постојање виртуелне корпе за куповину која се лако формира и са којом се лако управља, енкапсулиран кориснички систем и систем администрације, униформан кориснички интерфејс који је лак за коришћење), закључује се да апликација испуњава основне функционалности које веб продавница треба да оствари.

Технологије коришћене у изради серверске стране апликације су олакшале и убразале сам процес израде пројекта. Коришћењем *Entity framework-*апроцес интегрисања базе података са коришћеним развојним окружењем је изузетно олакшан. Искоришћена је могућност аутоматског генерисања већег дела кода, који се касније, ипак морао у одређеном сегменту кориговати и прилагодити за оставривање жељених функционалности.

Остаје питање да ли је ово решење боље од сценарија да се формира сопствена класа за комуникацију са базом података. То би подразумевало писање додатног кода у C#, при чему би се користила ADO.Net технологија. Такође, неопходно би било искодирати све класе које репрезентују податке у табелама дате базе података, које би *WEB Api* контролери користили.

На страни SQL сервера би морале бити исписане све (CRUD) процедуре и функције за манипулацију са подацима, што опет има и одређене предности у виду добијања кода на страни SQL сервера који је компајлиран, oчишћен од евентуалних грешака и спреман за извођење.

Алат коришћен у пројектовању саме базе података (Erwin) је убрзао процес дизајнирања и креирања базе.

Клијентска страна је урађена у *MVC* архитектури, где је је Ангулар био задужен за пружање модела и контролера, док је *View* био *HTML* приказ. Показало се да је Англуар прави избор за израду *SPA* пројекта попут овог. Пружа униформне перформансе и стиче се осећај рада као у десктоп апликацији. Значајно је унапредио апликацију тако што је омогућио “виртуелизацију” корпе. Сви подаци везани за корпу се нису чували у бази, већ на нивоу веб претраживача. То је знатно умањило време за комуникацију између клијента и веб-сервера у овом сегменту. Такође, комуникација података ка серверској страни је остварена у JSON формату, за који се показало да је супериорнији од стандардног XML формата. Има једноставнију синтаксу, не користи тагове и читљивији је. Захваљујући *Bootstrap*-у апликација је *responsive* и на мобилним уређајима и таблетима.

Укључивањем неког безбедног метода плаћања ( коришћењем неког *payment gateway*-a или банковног gateway-a) пројекат веб-продавнице може се проширити и унапредити што би био додатни квалитет.

1. **Литература**
2. https://www.asp.net/web-api
3. <https://msdn.microsoft.com/en-us/library/ee712907(v=vs.113).aspx>
4. Ali Uurlu, Alexander Zeitler, Ali Kheyrollahi, Pro ASP.NET Web API, HTTP Web Services in ASP.NET, 2013.
5. Peter Bacon Darwin, Pawel Kozlowski, Mastering Web Application Development with AngluarJS, 2013.
6. <https://www.amazon.com/p/feature/rzekmvyjojcp6uc>
7. <http://www.json.org/>
8. Boško Nikolić, Predavanja iz predmeta Programiranje internet aplikacija, Elektrotehnički fakultet, Beograd, 2014.
9. Ari Lerner, ng-book, fulstack.io 2013
10. <https://www.wikipedia.org/>
11. <https://jwt.io/>
12. Kenneth Schaefer, Jeff Cochran, Scott Forsyth, Professional Microsoft IIS 8, 2013
13. <https://www.iis.net/>
14. <https://stackoverflow.com/>
15. Anthony T. Holdener III, Ajax: The Definitive Guide, 2008
16. Julia Lerman, Programming Entity Framework, 2010