**BOJKOVIĆ Nikola**

**VISOKA ŠKOLA ELEKTROTEHNIKE I RAČUNARSTVA STRUKOVNIH STUDIJA**

**IZRADA INTERNET APLIKACIJE POMOĆU ASP MVC TEHNOLOGIJE**

**-diplomski rad-**

**Beograd 2016**

Kandidat: **Bojković Nikola**

Broj indeksa: **NRT-16/12**

Smer: **Nove računarske tehnologije**

Tema: **IZRADA INTERNET APLIKACIJE POMOĆU ASP MVC TEHNOLOGIJE**

Osnovni zadaci:

1. Izrada modela, kontrolera i korisničkog interfejsa
2. Povezivanje vizualnih objekata sa kodom
3. Konstrukcija baze podataka, povezivanje aplikacije na bazu i razmena podataka aplikacije sa bazom

Hardver: **0%** Softver: 7**0%** Teorija:3**0%**

Beograd 2016

Mentor:

Dr Zoran Ćirović, prof. VISER

**IZVOD**

U diplomskom radu opisan je celokupni proces kreiranja web aplikacije pomoću ASP.NET tehnologije. Bisnis logikom upravlja MVC Framework dok su modeli pisani u C# -pu mapiranji Entity Framework-om(koji upravlja razmenom podataka) u MS SQL bazu podataka. Korisnički interfejs je povezan sa bisnis logikom pomoću ASP.NET HTML helper-a i TAG helper-a Razor-om. Za implementaciju dizajna korisničkog interfejsa korisćen je bootstrap CSS framework i JQuery framework.

**ABSTRACT**

In this work web application implementation process that uses ASP.NET technology is described. MVC framework operates with business logic while models have been codded in C# and they are mapped into MS SQL Database with Entity Framework. User interface is connected with business logic with ASP.NET HTML helpers and TAG helpers and Razor. User Interface is designed with Bootstrap Framework and JQuery Framework.

Sadržaj

[1 UVOD 1](#_Toc461971257)

[2 POVEZIVANJE DELOVA APLIKACIJA 4](#_Toc461971258)

[2.1 Konfigurisanje aplikacije 4](#_Toc461971259)

[2.2 Setovanje dbContext -ta 5](#_Toc461971260)

[3 KREIRANJE MODELA 6](#_Toc461971261)

[3.1 Model 6](#_Toc461971262)

[3.2 ViewModel 6](#_Toc461971263)

[4 KREIRANJE KONTROLERA 8](#_Toc461971264)

[5 KREIRANJE KORISNIČKOG INTERFEJSA 13](#_Toc461971265)

[5.1 Layout 13](#_Toc461971266)

[5.2 ViewImport 15](#_Toc461971267)

[5.3 ViewStart 15](#_Toc461971268)

[5.4 Views 15](#_Toc461971269)

[5.4.1 Index View 15](#_Toc461971270)

[5.4.2 Details View 16](#_Toc461971271)

[5.4.3 Edit View 19](#_Toc461971272)

[5.4.4 Create View 20](#_Toc461971273)

[6 AUTORIZACIJA KORISNIKA 21](#_Toc461971274)

[7 KREIRANJE BAZE I RAZMENA PODATAKA 23](#_Toc461971275)

[7.1 Mapiranje modela 23](#_Toc461971276)

[7.2 Kreiranje migracija 24](#_Toc461971277)

[7.3 Update baze 27](#_Toc461971278)

[7.4 Razmena podataka izmedju aplikacije i baze podataka 28](#_Toc461971279)

[8 ZAKLJUČAK 30](#_Toc461971280)

[9 INDEKS POJMOVA 31](#_Toc461971281)

[10 LITERATURA 32](#_Toc461971282)

[11 DODATAK 33](#_Toc461971283)

[12 ZAHVALA 37](#_Toc461971284)

# UVOD

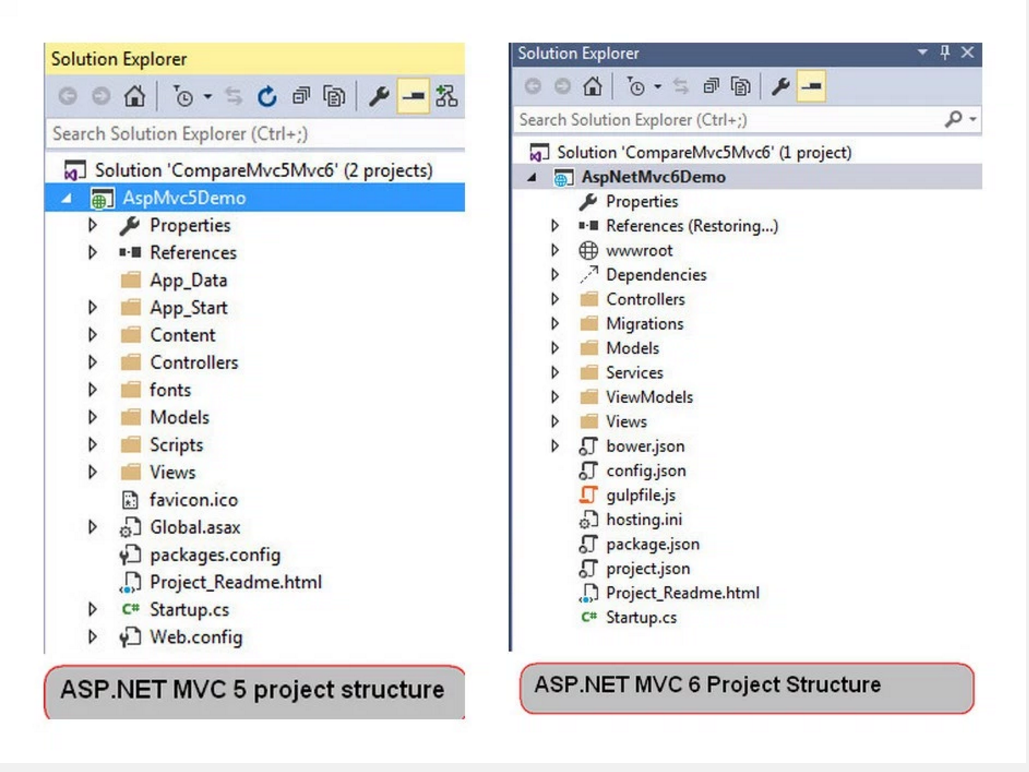
.NET tehnologija pruža mogućnost razvoja i održavanja web aplikacije. ASP.NET CORE je open-source cross-platform framework za konstrukciju modernih cloud based internet connected aplikacija kao što su web aplikacije, IoT aplikacije i mobile backends. ASP.NET CORE je najnovija verzija ASP.NET platforme koja donosi poboljšanja u odnosu na predhodne verzije, što je samim tim čini bolji izbor za implementaciju aplikacije u ovom projketu. MVC Framework daje odličnu kotrolu nad bisnis logikom, entitetima i korisničkim interfejsom dok Entity Framework upravlja bazom i posreduje u komunikaciji izmedju aplikacije i baze podataka. Samim tim .NET tehnologija pruža mogućnost da se na jako jednostavan načim kroz implementaciju web aplikacije prikaže razmena podataka izmedju celina i modula, unos i prikaz podataka kroz grafičke kontrole, rukovanje dogadjajima kao i grafičnim kontrolama, korisćenje klasa i interfejsa, projektovanje nove aplikacije, izrada kompletnog grafičnog korisničkog interfejsa kao i povezivanje vizualnih objekata sa kodom.

Korisćenjem svih navedenih Framework-a skraćuje se vreme potrebno za implementaciju i održavanje koda. ASP.NET CORE je konzolna aplikacija koja kreira web server i koristi WebHostBuilder koji prati builder pattern da kreira web aplikaciju. Sastoji se od nekoliko komponenti koje omogućavaju konfigurisanje, upravljanje rekvestima i servisima koji su dostupni kroz dependancy injection, upravljanje putanjama koje su dostupne u aplikaciji, upravljanje lokacijama odakle se HTTP zahtevi obradjuju i komponentama za kofiguraciju projektong sistema. ASP.NET core MVC razdvaja biznis logiku, korisnički interfejs i entitete sa podacima, upravlja entitetima, korisničkim interfejsom, biznis logikom, organizuje postojeće celine i kao takav pretstavlja MVC pattern prilagodjen za implementaciju u ASP.NET Core tehnologiji. Entity Framework omogućava mapiranje entiteta u bazu podataka i podataka iz baze u modele.

Razlike ASP.NET Core platforme u odnosu na predhodnu verziju ASP.NET arhitekture bitne za ovaj projekat su:

* Izmene u strukturi projekta(Solution)

U strukturi projekta više ne postoje Web.config, Global.asax fajlovi ali podešavanje konfiguracije, autentifikacija i izvršavanje specifičnih delova koda aplikacije se definiše pomoću novih fajlova Project.json i Config.json fajlova. Zapravo postoji jos mnogo promena u strukturi projekta koje možemo uočiti ako uporedimo sledeća dva projekta prikazana na slici na kojoj su prikazane najnovija i prethodna verzija ASP.NET MVC artitehture



*Slika 1.1.1: Razlika u strukturi porojekta dve verzije*

* ASP.NET Core referencira Full .NET and .NET Core

.NET Core Framework je cross-platform implementacija .NET – a što znači da je moguće kreirati aplikaciju na Windows koju je moguće pokrenuti na Windows, Linux i Mac ali je takodje moguće i kreirati aplikaciju na Linux i Mac koristeći Visual Studio Code IDE ili bilo koji drugi IDE.

* ASP.NET Core ne zahteva IIS za hosting

Cilj ASP.NET 5 je da bude cross-platform koristeći ASP.NET Core framework, zbog toga je Microsoft odlučio ne samo da hostuje ASP.NET 5 aplikacije na IIS već i da mogu biti self-hosted ili da koriste kestrel web server na Linux.

* wwwroot je sada mesto za statičke fajlove

wwwroot pretstavlja actual root web aplikacije kada je pokrenuta na web serveru. Statički fajlovi kao što je config.json nikada neće biti dostupni i zato nema potrebe kreirati posebna pravila za blokiranje pristupa osetljivim fajlovima

* Inbuilt Dependency Injection (DI) podrška za ASP.NET Core

Dependency Injection (DI) postiže loosely coupled, more testable code i veoma je važnan zato sto je postao vrsta standarda kodiranja. U prethodnim verzijama ASP.NET-a postojao je poseban DI kontejner korišćen kao Unity, AutoFac, StructureMap i tako dalje, tako da je za podizanje projekta koji je koristio DI bilo potrebno uložiti dodatan napor. U ASP.NET 5 or MVC 6(Core) aplikacijama DI je inbuild tako da nema potrebe za posebnim podešavanjima za DI. Samo je potrebno dodati odredjene servise i DI je spreman za upotrebu.

* Ability to see changes without re-building the project
* A unified story for building web UI and web APIs
* Integration of [modern client-side frameworks](https://docs.asp.net/en/latest/client-side/index.html) and development workflows
* A cloud-ready environment-based [configuration system](https://docs.asp.net/en/latest/fundamentals/configuration.html)
* New light-weight and modular HTTP request pipeline
* Built on [.NET Core](https://microsoft.com/net/core), which supports true side-by-side app versioning
* Ships entirely as [NuGet](https://nuget.org/) packages
* New tooling that simplifies modern web development
* Open source and community focused

Razlike izmedju ASP.NET Core MVC i standardnog MVC je u tome sto je MVC integrisan u ASP.NET i to je sve svedeno pod jednim imenom ASP.NET Core.

U nastavku biće prikazano korisćenje svih navedenih komponenti radi sklapanja funkcionalne celine koja odgovara biznis logici nekog sistema.

Osnovno polazište u teoriskom pogledu su komponente koje su same po sebi funkcionalne ali nedovoljne za rad sistema.

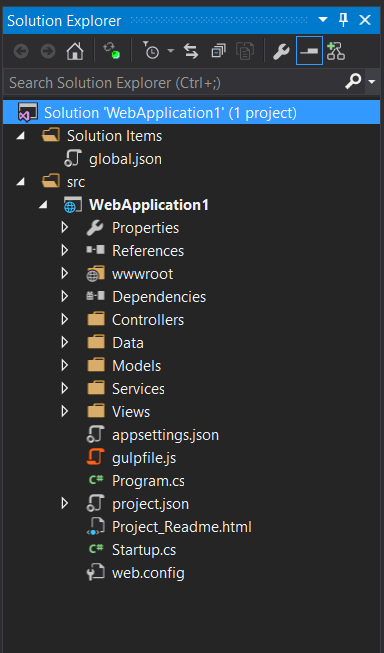
Potrebno je isprojektovati sistem klasa i fajlova koji čine funkcionalnu celinu. Takodje povezati različite delove jedne kao i delove različitih .NET komponenti. Aplikaciju prilagoditi zahtevima korisnika i uprostiti interakciju korisnika sa sistemom u cilju lakseg korisćenja aplikacije.

Da bi ASP.NET Core bio dostupan potrebno je instalirati .NET Core SDK.

# POVEZIVANJE DELOVA APLIKACIJA

## Konfigurisanje aplikacije

Konfigurisanje aplikacije tj. celog projekta se vrši u Startup fajlu, project.json i appsettings.json fajlu. Ovi fajlovi se generišu tokom kreiranja projekta, potrebno je prilikom kreiranja projekta izabrati ASP.NET CORE web application ponudjeni šablon.

Po šablonu se generiše sledeća struktura foldera i fajlova.

*Slika 2.1.1: Struktura projekta*

ASP.NET Core referencija NuGet pakete i potrebno je odraditi restore NuGet paketa ukoliko fale reference u projektu. Nakon restora projekat se mora bildovati uspesno.

U StartUp fajlu unutar ConfigureServices metode potrebno je dodati servise kako bi željeni kontroleri mogli pristupiti dbContex-tu. Dodavanjem servisa zapravo se omogućava DI koji obezbedjuje povezivanje i prenos instance objekata u sistemu.

services.AddTransient<IRepository, BlogRepository>();

services.AddTransient<CommentRepository, CommentRepository>();

U Configure megodi u podrazumevanom stanju se dodaje definicija rute i na koji način aplikacija raspoznaje koji deo url-a je ime kontolera, akcije ili vrednost parametra, na ovaj način MVC upravlja kontrolerima, akcijama i View komponentama kao i prosledjivanjem parametara.

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

template: "{controller=Home}/{action=Index}/{id?}");

});

project.json fajl sadrži reference ka komponentama koje su potrebne projektu.

Setovanje konekcije na bazu podataka se podešava u appsettings.json fajlu tako što se definiše objekat Connection setrings i unutar njega DefaultConnection. Potrebno je navesti Server, Databese i setovati indikatore Trusted\_Connection i MultipleActiveResultStatus na true.

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-DiplomskiCore1-v2;Trusted\_Connection=True;MultipleActiveResultSets=true"

},

## Setovanje dbContext -ta

U StartUp fajlu potrebno dodati u servise DbContext za predhodno definisan connection string iz json fajla i na taj način se povezuje aplikacija na bazu podataka.

services.AddDbContext<ApplicationDbContext>(options => options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

# KREIRANJE MODELA

## Model

Model predstavlja jedan entitet koji se entity framework-om preslikava u relacionu bazu podataka. Model sadrži propertije koji predstavljaju polja u tabeli definisane tipom i spoljne ključeve ka povezanim tabelama. Model takodje sadrži potreban broj ICollection propertija koje omogućavaju Entity framework-u da napravi veze izmedju tabela i koji predstavljaju kolekcije modela tipa drugog modela koji je u direktnoj vezi sa modelom koji sadrži ICollection property. U nastavku je prikazan Blog model.

public class Blog : Repository.Data

{

[Key]

public int Id { get; set; }

public string Title { get; set; }

public string Text { get; set; }

public int NumberOfLike { get; set; }

public int NumberOfDisike { get; set; }

public int? Vote { get; set; }

public int AuthorId { get; set; }

[ForeignKey("AuthorId")]

public virtual User Author { get; set; }

public virtual ICollection<Comment> Comments { get; set; }

}

Primarni i spoljni ključevi se označavaju flag atributima. Key atribut odredjuje primari ključ, a ForeighKey(“AuthorId”) odredjuje spoljni ključ i predstavlja ime polja koje se mapira u bazi.

Public virtual User Author je properti koji predstavlja vezu posmatranog modela sa modelom tipa User i odredjuje vezu jedan prema više u kojoj jedan autor može imati više blogova.

Public virtual ICollection<Comment> Comments properti predstavlja vezu posmatranog modela sa modelom tipa Comments i odredjuje vezu jedan prema više u kojoj jedan Blog tj. posmatrani model može imati više komentara tj. kolekciju modela tipa Comments. Sekundarno svojstvo ovog propertija je da se iz baze pročitani komentari koji pripadaju odgovarajućem blogu mogu smestiti u ovaj property kao i pročitati iz njega.

## ViewModel

ViewModel klase predstavljanju most izmedju grafičkog korisničkog interfejsa (u dalje tekstu View) i jednog ili vise modela. Svaki View ima odgovarajući ViewModel koji sadrži podatke potrebne za taj View ili podatke koje View salje kontroleru. ViewModel takodje može sadržati attribute koji detektuju validnost nekog polja.

U nastavku prikazan je ViewModel koji sadrži listu modela potrebnih za prikaz na Index View strani.

public class BlogViewModel

{

public IEnumerable<Blog> Blogs { get; set; }

}

U nastavku prikazan je ViewModel koji sadrži podatke potrebne za editovanje odredjenog bloga.

public class BlogEditViewModel

{

public string Title { get; set; }

public string Text { get; set; }

}

U nastavku prikazan je ViewModel koji povezuje dva modela Blog i Comments model sa View-om i sadrži sva potrebna polja za prikazivanje detalja jednog bloga i prikazivanje svih komentara koje taj blog sadrži.

public class BlogCommentsViewModel

{

public int BlogId { get; set; }

public string Title { get; set; }

public string Text { get; set; }

public bool Like { get; set; }

public bool Dislike { get; set; }

public int NumberOfLike { get; set; }

public int NumberOfDisike { get; set; }

[Display(Name = "New Comment")]

public string NewCommentText { get; set; }

public IEnumerable<Comment> Comments { get; set; }

}

*Slika 2.2.3: BlogCommentsViewModel*

# KREIRANJE KONTROLERA

Kontroler predstavlja komponentu koja sadrži upravljačku logiku i posreduje izmedju modela i View-a. U kontroleru se mogu nalaziti medote označene atributima koje se izvršavaju u zavistnosti od tipa zahteva (GET ili POST). Kontroler takodje sadrži instancu koja upravlja dbContext- om, instance klase zadužene za komunikaciju sa dbContext-om se dobija kroz konstrukor. MVC u saradnji sa ASP.NET CORE upravlja ostatkom prenosa instance kroz sistem, jedino sto je potrebno je dodati servis za definisanu klasu u Startup.cs. Kontroler takodje sardži Model i ViewModel u zavisnosti sta je potrebno, a View se kao povratna vrednost metode obično odredjuje samo eksplicitnim imenom ili imenom metode.

Home kontroler se poziva prvi kao podrazumevani, a ostali kontroleri preko neke od akcije iz View-a.

public class HomeController : Controller

{

public IActionResult Index()

{

return View();

}

public IActionResult About()

{

ViewData["Message"] = "Your application description page.";

return View();

}

public IActionResult Contact()

{

ViewData["Message"] = "Your contact page.";

return View();

}

public IActionResult Error()

{

return View();

}

public IActionResult Portfolio()

{

return View();

}

public IActionResult Services()

{

return View();

}

}

Podrazumevani View koji kotnroler na početku renderuje ima naziv Index, ali se mogu odrediti View komponente sa drigim imenima ali se moraju nazivati po imenima metoda koje tim View komponentama manipulišu, jer u drugom slučaju MVC neće znati da poveže kontroler i View.

Pored podrazumevanog mogu se definisati i custom kontroleri koje obično smeštamo u Controllers folder. Kontroler koji ćemo malo bliže posmatrati je BlogController.

BlogController nasledjuje MVC clasu Controller koje obezbedjuje osnovne funkcionalnosti kontrolleru. Kroz konstruktor kontroler dobija instancu klase koja ga povezuje sa dbContext-om.

BlogController u sebi sadrži biznis logiku za prikazivanje svih blogova, kreiranje novog bloga, editovanje postojećeg, kao i brisanje izabranog bloga. Takodje kontroler upravlja preusmeravanjem na drugi View koji ne mora striktno pirapadati metodi koja je trenutno u kontroleru pozvana.

private IRepository \_repository;

public BlogController(IRepository repository)

{

\_repository = repository;

}

Prva akcija blog kontrolera je Index koja pravi instancu BlogViewModela popunjava podacima i prosledjuje je u View sa imenom Index.

public ViewResult Index()

{

var model = new BlogViewModel();

model.Blogs = \_repository.GetAll() as IEnumerable<Blog>;

return View(model);

}

Druga akcija blog kontrolera je Details koja kao parameter prima id bloga koji treba biti pročitan iz baze, a zatim prikazan u View. Blog u kontroler stiže kao Blog model, a zatim se podaci iz modela prebacuju u BlogCommentsViewModel koji predstavlja ViewModel.

ViewModel je potreban da bi se u Details View prikazale inforamcije za izabran model kao i komentari koji pripadaju tom modelu i još neke informacije koje nisu potrebne Blog modelu a koje jesu View. Te informcije će kasnije biti prosledjene CommentControlleru koji će te informacije kontkretno za kreiranje komentara proslediti u Comment model i kasnije preko dbContexta i klase koja služi za manipulaciju te vrste podataka sačuvati u bazu podataka. ViewModel kontretno BlogCommentsViewModel se prosledjuje u View kao parameter metode View() koja se vraća kao povratna vrednost tipa ViewResult metode details.

Na taj način se poziva Details View koji sadrži BlogCommentViewModel sa podacima i koji kasnije može te podatke da prikaže korisniku u odgovarajućem grafičkom korisničkom interfejsu.

public ViewResult Details(int id)

{

var model = \_repository.Get(id) as Blog;

if (model == null)

return View();

var viewModel = new BlogCommentsViewModel();

viewModel.Title = model.Title;

viewModel.Text = model.Text;

viewModel.BlogId = model.Id;

viewModel.Comments = model.Comments;

return View(viewModel);

}

Create akcija je metoda koja odgovara na Http Get i Http Post zahtev u zavisnosti koji od zahteva je poslat akciji. Jedan od zahteva u ovom slučaju Get služi za pozivanje akcije Create koja je označena HttpGet atributom i koja služi za prikazivanje View-a koji ne sadrži podatke.

[HttpGet]

public ViewResult Create()

{

return View();

}

Create akcija koja je označena atributom HttpPost služi za prihvatanje podataka iz View-a poslatih iz forme Post zahtevom i obradu tih podataka. Podaci se obradjuju na takav način sto se krerira blog model i podaci smeste u njega, a zatim se model prosledjuje repository-ju koji pomoću dbContext-a čuva podatke u bazi podataka. Nakon toga se radi preusmerenja na Index akciju koja iz baze cita i u View prikazuje sve već postojeće blogove kao i novi blog koji je dodat u sistem.

[HttpPost]

public ActionResult Create(BlogEditViewModel model)

{

if (model.Title == null || model.Text == null)

return View();

var blog = new Blog();

blog.Title = model.Title;

blog.Text = model.Text;

User tempUser = \_userRepository.GetByAspNetUserId(\_userManager.GetUserId(User)) as User;

blog.AuthorId = tempUser.Id;

\_repository.Add(blog);

return RedirectToAction("Index");

}

Edit akcija takodje ima Get i Post verziju. Akcija označena atributom HttpGet kao parametar prihvata id bloga koji treba editovati, zatim pravi instance BlogEditViewModel-a popunjava podacima i ViewModel šalje u View kako bi informacije za editovanje bile dostupne korisniku.

[HttpGet]

public ViewResult Edit(int id)

{

var blog = \_repository.Get(id) as Blog;

if (blog == null)

return View("Index");

var model = new BlogEditViewModel();

model.Title = blog.Title;

model.Text = blog.Text;

return View(model);

}

Edit Akcija označena atributom HttpPost služi da prihvati ViewModel popunjen podacima, da te podatke konvertuje u Blog model i da pošalje zahtev repository-ju za čuvanje izmena u bazi. Nakon toga se pravi instanca i popunjava BlogCommentsViewModel podacima i salje u Details View kako bi korisnik mogao da detaljno pregleda sve promene koje je napravio. U slučaju da blog nije pronadjen u bazi ili ako je u medjuvremenu došlo do neke greške prikazuje se Index View i preskače se dalji tok izvršenja da bi se izbegle moguće greške. Poželjna je bolja zaštita od neočekivanih grešaka.

[HttpPost]

public ViewResult Edit(int id, BlogEditViewModel model)

{

var blog = \_repository.Get(id) as Blog;

if (blog == null)

return View("Index");

blog.Title = model.Title;

blog.Text = model.Text;

\_repository.Edit(blog);

BlogCommentsViewModel blogViewModel = new BlogCommentsViewModel();

blogViewModel.Title = blog.Title;

blogViewModel.Text = blog.Text;

blogViewModel.Comments = blog.Comments;

return View("Details", blogViewModel);

}

Akcija koja služi za brisanje bloga je motoda koja ne vraća nikakav View i nema atributa za odredjivanje tipa zahteva. Preusmerenje se vrši na Index akciju zato što se korisniku zeli prikazati osvežen prikaz blogova nakon što se u Delete metodi iz repository-ja pozove Get metoda koja preko dbContex-a pronalazi izabran blog a zati se briše izabrani blog iz baze podataka.

public ActionResult Delete(int id)

{

\_repository.Delete(\_repository.Get(id));

return RedirectToAction("Index");

}

# KREIRANJE KORISNIČKOG INTERFEJSA

## Layout

Layout je deo View tj. korisničkog interfejsa koji sadrži komponente koje su zajedničke za svaki View tj. za sve html strane. Layout sadrži html link tagove koji referenciraju ka css fajlovima i html script tagove koji referenciraju ka java script fajlovima. Link tagovi se smeštaju u header-u zajedno sa title tagom i favicon-om, dok se script tagovi smeštaju u body tagu. Takodje u body tagu je potrebno dodati i Razov View tag @RenderBody() koje renderuje izabranu html stranu kao deo Layout-a. Osim tagova za povezivanje dokumenata u Layout stoji i bottom sekcija i footer koje je potrebno prikazivati na svakoj Html strani.

Html title tag sadrži @ViewData[“Title”] rezor View tag koji se setuje u odredjenom kontroleru.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>@ViewData["Title"] - DiplomskiCore1</title>

<**environment** **names**="Development">

<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />

<link rel="stylesheet" href="~/css/site.css" />

</**environment**>

<**environment** **names**="Staging,Production">

<**link** rel="stylesheet" **href**="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.6/css/bootstrap.min.css"

**asp-fallback-href**="~/lib/bootstrap/dist/css/bootstrap.min.css"

**asp-fallback-test-class**="sr-only" **asp-fallback-test-property**="position" **asp-fallback-test-value**="absolute" />

<**link** rel="stylesheet" **href**="~/css/site.min.css" **asp-append-version**="true" />

</**environment**>

@Html.ApplicationInsightsJavaScript(TelemetryConfiguration)

<!-- ------ CSS ------ -->

<link href="~/css/font-awesome.min.css" rel="stylesheet" type="text/css" />

<link href="~/css/animate.min.css" rel="stylesheet" type="text/css" />

<link href="~/css/prettyPhoto.css" rel="stylesheet" type="text/css" />

<link href="~/css/main.css" rel="stylesheet" type="text/css" />

<link href="~/css/responsive.css" rel="stylesheet" type="text/css" />

<link href="http://maxcdn.bootstrapcdn.com/font-awesome/4.2.0/css/font-awesome.min.css"

rel="stylesheet" type="text/css" />

<!--[if lt IE 9]>

<script src="js/html5shiv.js"></script>

<script src="js/respond.min.js"></script>

<![endif]-->

<link rel="shortcut icon" href="~/favicon.ico" />

</head>

<body>

<header id="header"></header>

@RenderBody()

<section id="bottom">

</section>

<!--/#bottom-->

<footer id="footer" class="midnight-blue"></footer>

<!--/#footer-->

<!-- Back To Top -->

<**environment** **names**="Development">

<script src="~/lib/jquery/dist/jquery.js"></script>

<script src="~/lib/bootstrap/dist/js/bootstrap.js"></script>

<**script** **src**="~/js/site.js" **asp-append-version**="true"></**script**>

</**environment**>

<**environment** **names**="Staging,Production">

<**script** **src**="https://ajax.aspnetcdn.com/ajax/jquery/jquery-2.2.0.min.js"

**asp-fallback-src**="~/lib/jquery/dist/jquery.min.js"

**asp-fallback-test**="window.jQuery">

</**script**>

<**script** **src**="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.6/bootstrap.min.js"

**asp-fallback-src**="~/lib/bootstrap/dist/js/bootstrap.min.js"

**asp-fallback-test**="window.jQuery && window.jQuery.fn && window.jQuery.fn.modal">

</**script**>

<**script** **src**="~/js/site.min.js" **asp-append-version**="true"></**script**>

</**environment**>

<script type="text/javascript">

jQuery(document).ready(function() {

var offset = 300;

var duration = 500;

jQuery(window).scroll(function() {

if (jQuery(this).scrollTop() > offset) {

jQuery('.back-to-top').fadeIn(duration);

} else {

jQuery('.back-to-top').fadeOut(duration);

}

});

jQuery('.back-to-top').click(function (event) {

event.preventDefault();

jQuery('html, body').animate({ scrollTop: 0 }, duration);

return false;

});

});

</script>

<!-- /top-link-block -->

<!-- Jscript -->

<script src="~/js/jquery.js" type="text/javascript"></script>

<script src="~/js/bootstrap.min.js" type="text/javascript"></script>

<script src="~/js/jquery.prettyPhoto.js" type="text/javascript"></script>

<script src="~/js/jquery.isotope.min.js" type="text/javascript"></script>

<script src="~/js/main.js" type="text/javascript"></script>

<script src="~/js/wow.min.js" type="text/javascript"></script>

@RenderSection("scripts", required: false)

</body>

</html>

## ViewImport

ViewImports je fajl koji importuje modele i tag helpere u View-ove.

@using DiplomskiCore1

@using DiplomskiCore1.Models

@using DiplomskiCore1.Models.AccountViewModels

@using DiplomskiCore1.Models.ManageViewModels

@using Microsoft.AspNetCore.Identity

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

@inject Microsoft.ApplicationInsights.Extensibility.TelemetryConfiguration TelemetryConfiguration

## ViewStart

ViewStart fajl koji izvršava zajednički kod za sve View komponente i izvršava se pre rederovanja bolo kog View-a. U ovom projektu ViewStart setuje Layout aplikacije na \_Layout file koji je specificirana kao zajednička komponenta svakog View-a.

@{

Layout = "\_Layout";

}

## Views

Svaka akcija u kotroleru koja vraća kao rezultat View mora imati definicuju za isti. Za blog kontroler potrebna su četiri View-a. Index koji prikazuje sve blogove, Create koji služi za kreiranje bloga, Edit koji služi za editovaje bloga i Details koji sadrži detalje o izabranom blogu.

### Index View

Index View na početku sadrži kod koji setuje title dokumenta koji se renderuje u Layout-u. Posle H2 taga za naslov stranice u paragrafu je smešten link ka stranici koja služi za kreiranje novog bloga. Link se kreira pomoću Html helper-a koji kao prvi parameter prima tekst linka, a drugi parametar je akcija koja se treba izvršiti na serverskoj strani.

Html kod se koristi u kombinaciji sa rezor View elementima, nakon znaka @ piše se razor View kod. BlogViewModel stiže iz kontrolera i kroz foreach petrlju se popunjavaju Html tagovi sa informacijama o blogu na osnovu čega se obrazuje lista blogova koji postoje u bazi podataka. Pored nalsova i teksta blogova stoje i linkovi u obliku dugmeta koji vode ka stranicama za prikazivanje detalja, editovanje i brisanje odredjenog bloga. U linkovima se kao treći parametar salje i objekat sa blog id-jem koji je potreban radi indentifikacije traženog bloga, a kao četvrti parametar stoji objekat koji definiše css klasu koja odredjuje izgled linka. Klase koje odredjuju izgled View komponenti su definisane Bootstrap framework-om koji se može naknadno izmeniti.

@using DiplomskiCore1.ViewModels

@using Microsoft.EntityFrameworkCore.Metadata.Internal

@model DiplomskiCore1.ViewModels.BlogViewModel

@{

ViewBag.Title = "Blog List";

}

<div class="container">

<h2>This is a blog page</h2>

<p>

@Html.ActionLink("Create New", "Create")

</p>

@foreach (var blog in Model.Blogs)

{

<div class="panel panel-default">

<div class="panel-heading">@blog.Title</div>

<div class="panel-body">

@blog.Text

</div>

<div class="panel-footer">

<span>@Html.ActionLink("Details", "Details",

new { id = blog.Id },

new { @class = "btn btn-default"})</span>

<span>@Html.ActionLink("Edit", "Edit",

new { id = blog.Id},

new { @class = "btn btn-primary" })</span>

<span>@Html.ActionLink("Delete", "Delete",

new {id = blog.Id},

new { @class = "btn btn-danger" })</span>

</div>

</div>

}

</div>

### Details View

Details View takodje setuje ViewBag.Title na zahtevanu vrednost koja predstavlja ime stranice i vidi se u tabu na vrhu pretraživača. Na vrhu se nalazi link za povratak na listu blogova. Nakon sto se prikažu naslov i tekst izabranog bloga prikazuju se i svi komentari koji pripadaju tom blogu odamah ispod teksta bloga. Foreach petljom se prolazi kroz sve komantare iz BlogCommentsViewModel-a koji u sebi sadrži potrebne infomacije o blogu i njegovim komentarima.

Odredjeni komentar se može obrisati klikom na dugme delete koje se nalazi odmah ispod njega,

Dugme je odredjeno Html helperom Html.ActionLink koji kao prva tri parametara prima tekst koji se treba prikazati u linku, kontroler na koji link vodi i akciju koju je potrebno izvršiti u kontroleru. Četvrti parametar su podaci koji se salju kontroleru i peti je objekat koji cini css class atribut za formatiranje linka – u ovom slučaju izgled dugmeta. Klikom na dugme vrši se redirekcija ka CommentController-u i akciji za brisanje komentara koja na osnovu poslatog id komentara briše željeni komentar. Akcija je sinhrona i osvežava se cela html stranica nakon brisanja odredjenog komentara.

Editovanje komantara se odvija asinhrono, Komentar se bira tako što se klikne na tekst, zatim java script briše tekst is paragrafa na koji je kliknuto i u paragraf dodaje html tag textarea i popunjava ga prethodno obrisanim tekstom komentara. Ispod textarea taga dodaje dugme za pokretanje AJAX koda koji asinhrono šalje zahtev i podatke za editovanje komentara CommentControlleru kontoleru i Edit akciji. Nakon izvršene akcije editovanja AJAX dobija povratne informacije i ukoliko je uspešno izvršeno editovanje textarea tag se menja novim editovanim tekstom i tako se odvija modifikovanje samo dela stranice bez osvežavanja celokupnog sadrzaja html stranice. Java script kod koji obavlja asinhrono editovanje komentara nalazi se u fajlu main.js.

var currentComment = 0;

var newComment = 0;

var textComment;

var canEdit = true;

// replace text with textarea

$(".comment").on('click', function () {

newComment = $(this).attr("id");

if (newComment === currentComment && !canEdit)

return;

if (currentComment !== 0) {

$("p[id =" + currentComment + "]").text(textComment);

}

currentComment = newComment;

canEdit = false;

textComment = $(this).text();

$(this).html("<textarea id=\"commentPlaceholder\" class=\"form-control\">" +

textComment + "</textarea><br/>" +

"<button id=\"commentEdit\" class=\"btn btn-primary\">Save</button>");

});

// asynch post - send data to server side and display callback information

$(document).on('click', '#commentEdit', function () {

$.post("../../Comment/Edit",

{

id: currentComment,

text: $("#commentPlaceholder").val()

},

function (data) {

$("p[id =" + currentComment + "]").text(data);

textComment = data;

canEdit = true;

});

});

Ispod komentara nalazi se forma sačinjena od tag helpera koja služi za prikupljanje i slanje informacija o novom komentaru koji je vezan za izabrani blog.

Prvi tag helper se nalazi u form html tagu - **asp-controller**="Comment" i to je tag helper koji služi da se odredi kontroler koji prihvata informacije sa forme, drugi tag odredjuje akciju koja obradjuje onformacije i to je tag **asp-action**="Create". Unutar forme se nalazi label tag koji odredjen tag helperom **type**="hidden"/> koji se koristi ukoliko je potrebno sakriti od korisnika id koji je potrebno poslati kontroleru. **asp-for**="BlogId" služi kako bi label tag povezao na odredjeno polje iz ViewModel-a.

@using Microsoft.EntityFrameworkCore.Metadata.Internal

@model DiplomskiCore1.ViewModels.BlogCommentsViewModel

@{

ViewBag.Title = "Blog Details";

}

<div class="container">

@Html.ActionLink("Back to List", "Index")

<div class="panel panel-default">

<div class="panel-heading">@Model.Title</div>

<div class="panel-body">

@Model.Text

</div>

</div>

<h3>Comments</h3>

<div id="comments">

@foreach (var comment in Model.Comments)

{

<div class="panel panel-default">

<div class="panel-body">

<p class="comment" id="@comment.Id"> @comment.Text</p>

</div>

<div class="panel-footer">

<span>@Html.ActionLink("Delete", "Delete", "Comment",

new {id = comment.Id, blogId = Model.BlogId},

new {

@class = "btn btn-danger"

})</span>

</div>

</div>

}

</div>

<**form** **asp-controller**="Comment" **asp-action**="Create" method="post"

class="form-horizontal">

<**input** **asp-for**="BlogId" **type**="hidden"/>

<div class="form-group">

<**label** **asp-for**="NewCommentText"></**label**>

<**textarea** **asp-for**="NewCommentText" class="form-control"

placeholder="Enter your comments here..."></**textarea**>

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

</div>

</**form**>

</div>

### Edit View

Edit View za razliku od prethodnih View stranica koristi formu koja je sačinjena od Html helpera Html.BeginForm helper renderuje form tag. Podrazumevane vrednosti su Post zahtev i akcija koja obradjuje podatke je akcija zadužena za trenutni View. Html.LabelFor je helper koji pravi label, a kao parametar prihvata lambda izraz. Html.TextBoxFor renderuje input polje i takodje kao prvi parametar prihvata lambda izraz a kao drugi html atribut za formatiranje elementa.

Edit View sadrži BlogEditViewModel koji pri renderovanju View-a sadrži podatke koje stranica prikazuje i koje korisnik edituje, Nakon sto korisnik pošalje zahtev sistemu za editovanje podataka klikom na dugme Save forma kao parametar pored kontrolera i akcije kjoj šalje podatke šalje I BlogEditViewModel koje sistem dalje obradjuje.

@using Microsoft.AspNetCore.Mvc.ViewFeatures

@model DiplomskiCore1.ViewModels.BlogEditViewModel

@{

ViewBag.Title = "Blog Edit";

}

<div class="container">

<h2>Edit Blog</h2>

@Html.ActionLink("Back to List", "Index")

@using (Html.BeginForm())

{

<div class="form-group">

@Html.LabelFor(m => m.Title)

@Html.TextBoxFor(m => m.Title, new { @class = "form-control" })

</div>

<div class="form-group">

@Html.LabelFor(m => m.Text)

@Html.TextAreaFor(m => m.Text, new { @class = "form-control" })

</div>

<div class="form-group">

<input type="submit" value="Save" class="btn btn-primary" />

</div>

}

</div>

### Create View

Create View takodje sadrži BlogEditViewModel ali je on na početku prazan. Jos jedna razlika u odnosu na Edit View je sto unutar helpera kao parametar se šalje objekat sa dva html atributa, prvi je style atribut koji setuje css klasu, a drugi je placeholder koje je postavljen na neku vrednost i predstavlja tekst koji je vidljiv u input polju dok se ne unese tekst.

@using Microsoft.AspNetCore.Mvc.ViewFeatures

@model DiplomskiCore1.ViewModels.BlogEditViewModel

@{

ViewBag.Title = "Blog Create";

}

<div class="container">

<h2>Create Blog</h2>

@Html.ActionLink("Back to List", "Index")

@using (Html.BeginForm())

{

<div class="form-group">

@Html.LabelFor(m => m.Title)

@Html.TextBoxFor( m => m.Title, new { @class = "form-control",

@placeholder = "Enter your title here..."

})

</div>

<div class="form-group">

@Html.LabelFor(m => m.Text)

@Html.TextAreaFor(m => m.Text, new { @class = "form-control",

@placeholder = "Enter your text here..."

})

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

</div>

}

</div>

# AUTORIZACIJA KORISNIKA

Autorizacija je realizovana Identity framework-om. Prilikom kreiranja web site aplikacije ASP.NET CORE kreira potrebne kontrolere, modele i view klase za autorizaciju. Izmenom pojedinih delova vrši se prilagodjavanje autorizacije zeljenim pootrebama

Potrebno je izmeniti AccountController klasu koja sadrži za ovaj projekat bitne metode register, login, logout i dodati sledeći kod u register metodu koja je označena kao HttpPost zahtev, dodeliti vrednost za UserName istu kao za Email kako bi omogućili da registracija korisnika radi na zeljeni način.

var user = new ApplicationUser { UserName = model.Email, Email = model.Email };

Provera da li je korisnik ulogovan se obavlja na sledeci način. Prvo se u kontroleru kroz konstruktor prosleduju klase UserManager i SignInManager, zatim se instance objekata čuvaju u odredjenim poljima.

private readonly UserManager<ApplicationUser> \_userManager;

private readonly SignInManager<ApplicationUser> \_signInManager;

public BlogController(

UserManager<ApplicationUser> userManager,

SignInManager<ApplicationUser> signInManager,

BlogRepository repository,

UserRpository userRepository)

{

\_userManager = userManager;

\_signInManager = signInManager;

\_repository = repository;

\_userRepository = userRepository;

}

Ukoliko je potrebno pristupiti UserManager ili SignInManager objektu u nekom View tada se klase dodaju preko Dependency Injection metode

@inject SignInManager<ApplicationUser> SignInManager

@inject UserManager<ApplicationUser> UserManager

Provera da li je korisnik ulogovan na stistem se obavlja pomoću metode SignInManager.IsSignedIn(User) koja vraća bool vrednost, true ukoliko je korisnik ulogovan. Pomocu UserManager objekta može se pristupiti podacima ulogovanog korisnika. Primer je izmena koda koja je napravljena nad Index View bloga.

User currentUser = userRepository.GetByAspNetUserId(UserManager.GetUserId(User)) as User;

@if (SignInManager.IsSignedIn(User) && (currentUser.Id == blog.AuthorId || currentUser.IsAdmin))

{

<span>@Html.ActionLink("Edit", "Edit",

new {id = blog.Id, blogViewModel = new BlogEditViewModel()},

new {@class = "btn btn-primary"})</span>

<span>@Html.ActionLink("Delete", "Delete",

new {id = blog.Id},

new {@class = "btn btn-danger"})</span>

}

# KREIRANJE BAZE I RAZMENA PODATAKA

## Mapiranje modela

Baza podataka se kreira na osnovu Code First tehnike, gde se baza krerira na osnovu definisanih entiteta u kodu. Ukoliko baza već postiji onda se samo update-uje. Kreiranje i upravljanje bazom podataka se vrši pomoću komande dotnet ef - EF command-line tools for .NET Core Command Line Interface (CLI). Ukoliko je već instaliran Visual Studio 2015 onda su i DNVM(Dot Net Version Manager) i dotnet ef comande instalirane.

Entity Framework vrši mapiranje objekta u relacionu bazu podataka na osnovu polja i atrubuta definisanih u modelu.

public class Blog : Repository.Data

{

[Key]

public int Id { get; set; }

public string Title { get; set; }

public string Text { get; set; }

public int NumberOfLike { get; set; }

public int NumberOfDisike { get; set; }

public int? Vote { get; set; }

public int AuthorId { get; set; }

[ForeignKey("AuthorId")]

public virtual User Author { get; set; }

public virtual ICollection<Comment> Comments { get; set; }

}

Na osnovu atrubuta Key koji definiše koje polje je primarni ključ i atrubuta ForeignKey koje definiše koje polje predstavlja spoljni ključ Entity Framework vrši povezivanje tabela koje su u relaciji. ICollection property definiše vezu izmedju tabela. Poslednji korak je dodavanje dataSeta propertija u DbContext klasu.

public DbSet<Blog> Blog { get; set; }

Nakon mapiranja koje se radi dodavanjem migracija i updatovanjem baze dobija se sledeći rezultat. U nastavku je prikazana definicija blog tabele.

CREATE TABLE [dbo].[Blog] (

[Id] INT IDENTITY (1, 1) NOT NULL,

[AuthorId] INT NOT NULL,

[Text] NVARCHAR (MAX) NULL,

[Title] NVARCHAR (MAX) NULL,

[Vote] INT NULL,

[NumberOfDisike] INT DEFAULT ((0)) NOT NULL,

[NumberOfLike] INT DEFAULT ((0)) NOT NULL,

CONSTRAINT [PK\_Blog] PRIMARY KEY CLUSTERED ([Id] ASC),

CONSTRAINT [FK\_Blog\_User\_AuthorId] FOREIGN KEY ([AuthorId]) REFERENCES [dbo].[User] ([Id]) ON DELETE CASCADE

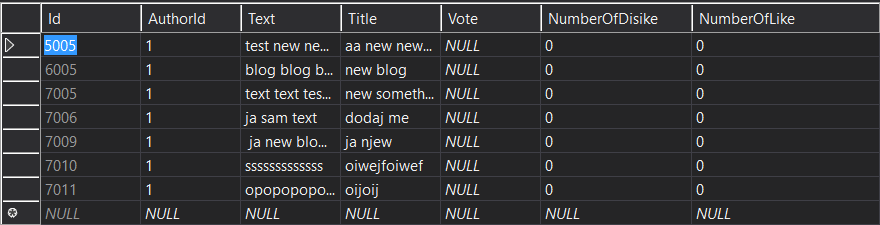
);

GO

CREATE NONCLUSTERED INDEX [IX\_Blog\_AuthorId]

ON [dbo].[Blog]([AuthorId] ASC);

Primer tabele sa test podacima.

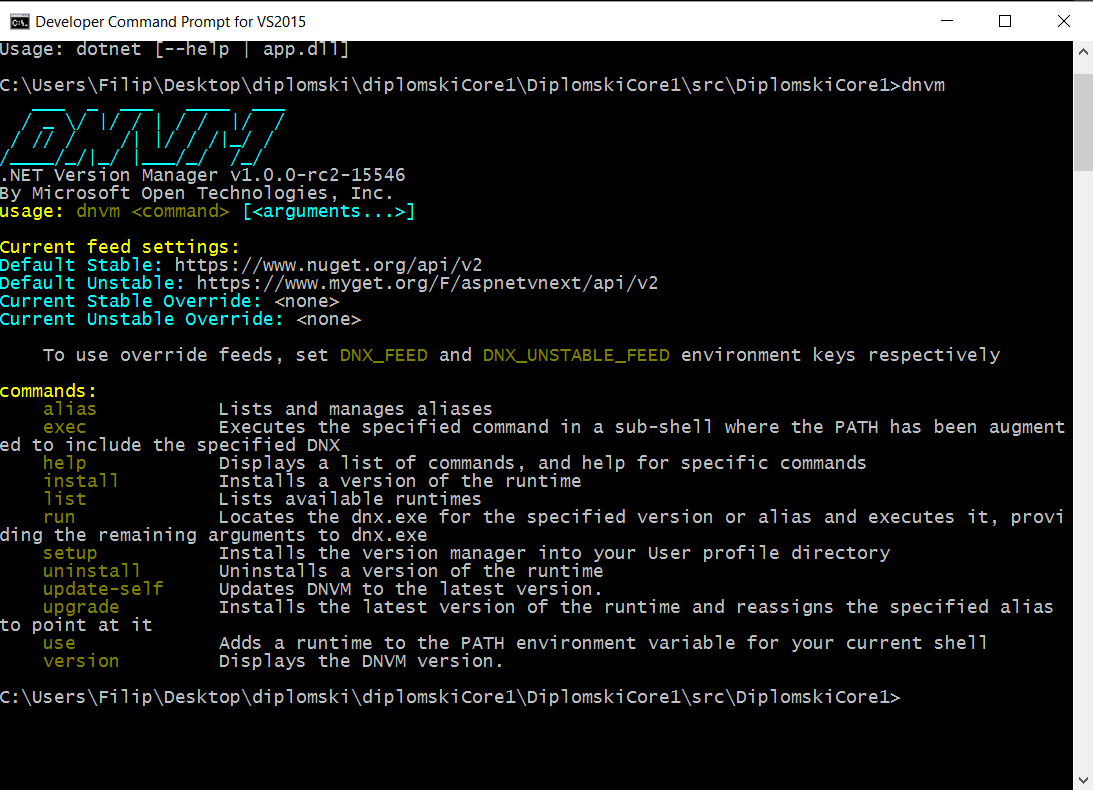


*Slika 7.1.1: Struktura blog tabele*

## Kreiranje migracija

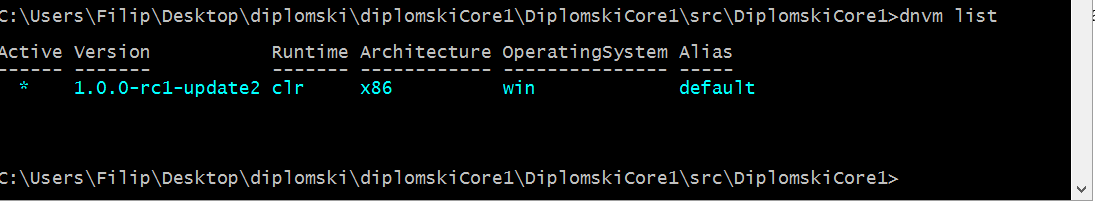
Kreiranje migracija se vrši pomoću dnvm-a i potrebno je dodati migraciju i primeniti je(applay migration). Migracija je c# code koji se smešta u poseban folder migrations. Putanja gde se migracije smestaju se može promeniti, u ovom projektu koristiće se podrazumevani folder.

Kreiranje migracija se postiže pomoću dotnet komande koja se pokreće tako što se otvori Developer command for VS2015 i pokrene se komanda dnvm, zatim je potrebno odabrati run time environment.



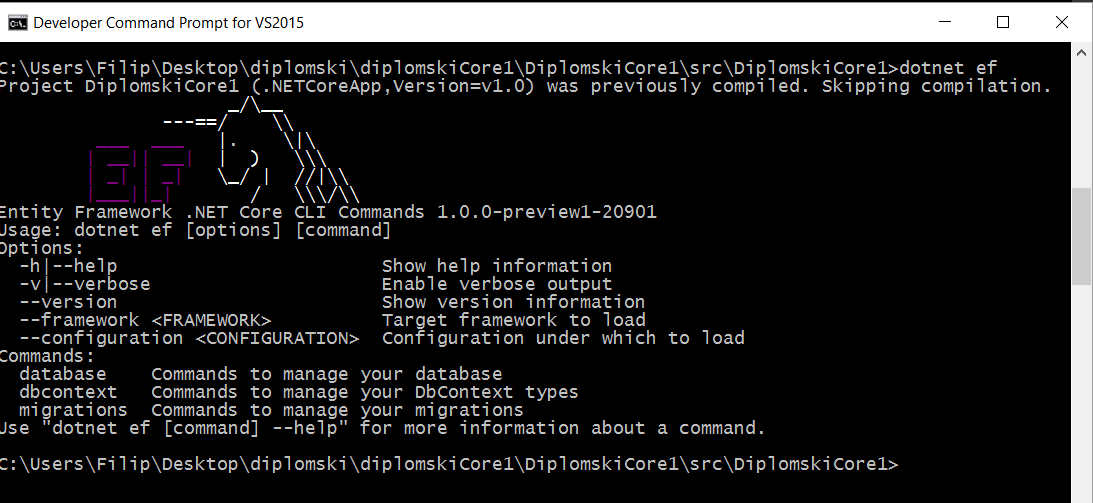
*Slika 7.2.1: DNVM komande*

Komanda dnvm list prikazuje listu dostupnih environment-a. Trenutno postoji samo jedan environment i on je podrazumevani. U slučaju da postoji više environment-a potrebno je komandom dnvm-use [name] -p, izabrati jedan od ponudjenih koji sadrži dnx ili dotnet komande za kreiranje migracija i updatovanje baze.



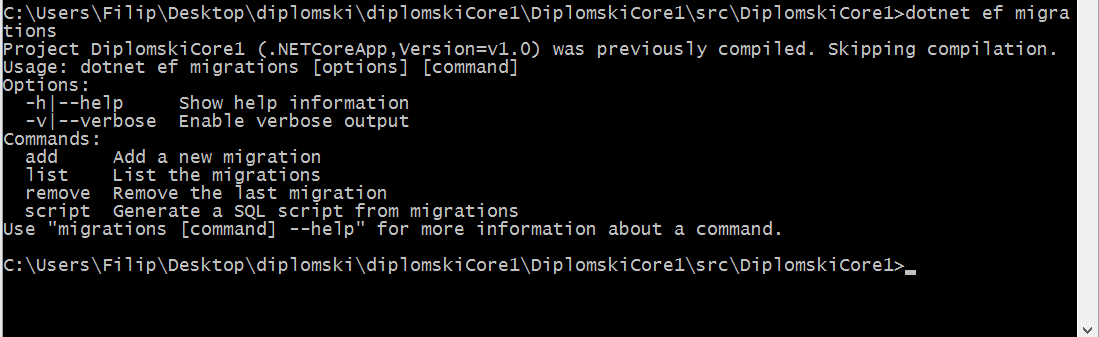
*Slika 7.2.2: Lista environment-a*

Komandom dotnet ef dobijamo spisak komandi koje se mogu izvršiti nad migracijama i bazom medju kojim se nalaze opcije prikazane na sledećoj slici.



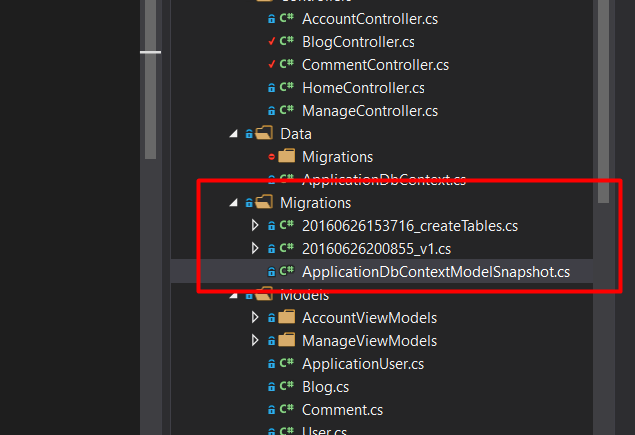
*Slika 7.2.3: Lista opcija dobijena EF komandom*

Komandom - dotnet ef migrations - dnvm prikazuje listu opcija vezanih za migrations



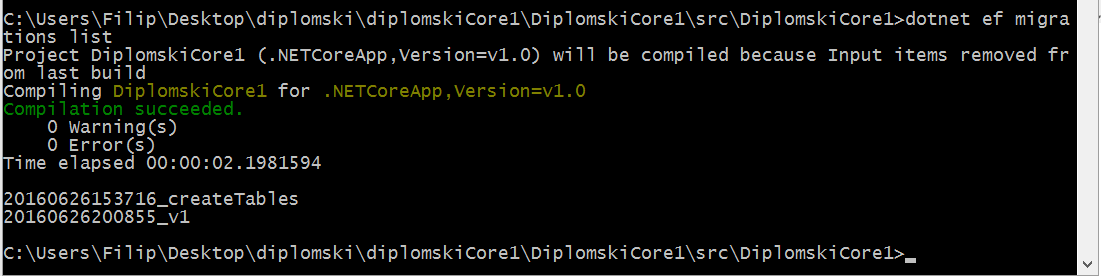
*Slika 7.2.4: Lista dostupnih opcija za upravljanje migracijama*

Migracija se dodaje tako što se pokrene komanda dotnet ef migrations add [name] i tada se migracija pojavljuje u migrations folderu u projektu.



*Slika 7.2.5: Folder sa migracijama*

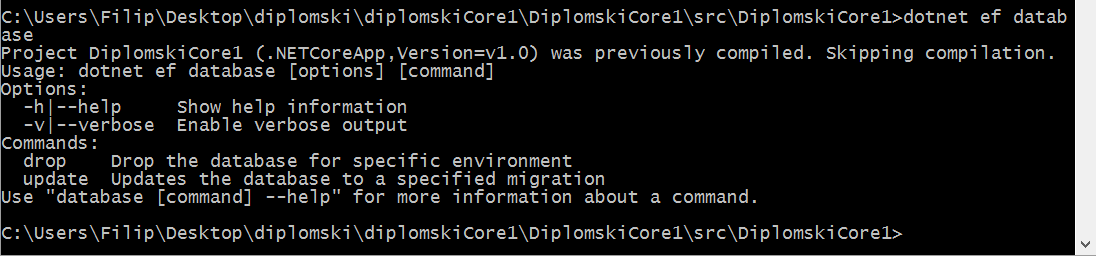
Pokretanjem komande dotnet ef migrations list prikazuju se sve migracije koje su dodate.



*Slika 7.2.6: Lista dodatih migracija*

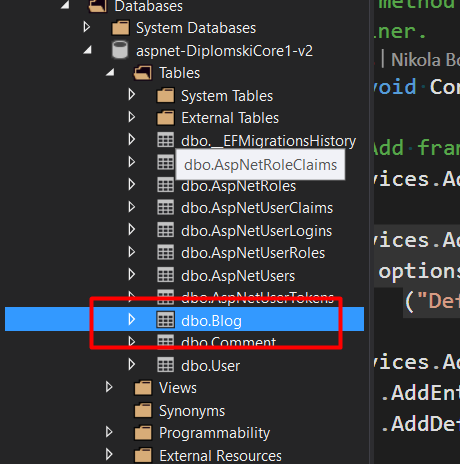
## Update baze

Komandom dotnet ef database prikazuje se lista opcija dostupnih za upravljanje bazom podataka.



*Slika 7.3.1: Lista dosputnih opcija za upravljanje bazom podataka*

Pokretanje komande – dotnet ef database update - primenjuju se dodate migracije i update-uje se baza. Kao rezultat blog tabela je dodata u bazu i povezana sa potrebnim tabelama.



*Slika 7.3.2: Lokacija blog tabele*

Prilikom dodavanja novog polja u model ili izmene postojećeg tj. izmene postojeće strukture modela, potrebno je dodati novu migraciju i pokrenuti update baze u dnvm-u kako bi baza podataka bila izmenjena.

Postoji mogućnost da se baza menja pomoću Ms Sql Server-a, potrebno je samo ucitati mdf fajl u Ms Sql Server. Mdf fajl je ne dostupan i ne vidljiv ukoliko se nalazi u folderu za koji korisnik nema full permission ili ako Ms Sql Server iz nekog drugog razloga ne može da pristupi tom folderu, tako da je potrebno korisniku dodati full permission nad folderom da bi bio u mogućnosti da pristupi i učita mdf fajl.

## Razmena podataka izmedju aplikacije i baze podataka

Razmena podataka izmedju baze i applikacije se obavlja preko Entity Frameworka. Pomoću instance DbContext klase pristupa se objektima koji su pročitani iz baze i takodje je moguće pokrenuti slq upit i manipulisati dobijenim podacima. Klasa koja sadrži dbContext prati repository patern kako bi proces bio pojednostavljen.

public class BlogRepository : IRepository

{

private ApplicationDbContext \_dbContext;

public BlogRepository(ApplicationDbContext dbContext)

{

\_dbContext = dbContext;

}

public IEnumerable<Data> GetAll()

{

return \_dbContext.Blog;

}

Get metoda čita na dva načina blogove i komentare koje taj blog sadrži. Prvi način čitanja i željenog bloga i njegovih komentara je pomoću dbContext-a. Drugi način citanja Bloga za poslati id je pomoću upita, koristi se FromSql kome se kao parametar prosledi upit kao string. Drugi način čitanja komntara je pomoću LINQ izraza.

public Data Get(int id)

{

// find blog

// first approach

//var blog = \_dbContext.Blog.FirstOrDefault(item => item.Id == id);

// second approach

string query = "SELECT \*"

+ " FROM blog "

+ " WHERE id = " + id;

IEnumerable<Blog> blogs = \_dbContext.Blog.FromSql(query);

Blog blog = blogs.FirstOrDefault( x=> x.Id == id);

//find comments for blog

// first approach

// var comments = \_dbContext.Comment.Where(item => item.BlogId == id);

// second approach

var commentsLoaded = \_dbContext.Comment.Include(x=> x.Author);

var comments = from comment in commentsLoaded

where comment.BlogId == id

select comment;

blog.Comments = comments.ToArray();

return blog;

}

public void Add(Data item)

{

\_dbContext.Add(item);

\_dbContext.SaveChanges();

}

public Data Edit(Data item)

{

\_dbContext.Update(item);

\_dbContext.SaveChanges();

return item;

}

U delete metodi prikazana su dva načina brisanja bloga. Prvi je preko dbContexta, a drugi izvršavanjem upita.

public void Delete(Data item)

{

//\_dbContext.Remove(item);

string query = "DELETE FROM blog"

+ " WHERE id = " + ((Blog)item).Id;

\_dbContext.Database.ExecuteSqlCommand(query);

foreach (var blogActivityItem in \_dbContext.BlogActivity.Where(x => x.BlogId == ((Blog)item).Id))

{

\_dbContext.Remove(blogActivityItem);

}

\_dbContext.SaveChanges();

}

}

DbContext objekat sadrži objekte mapirane iz baze tako da je potrebno samo vratiti željeni objekat ili listu objekata iz dbContext objekta kao povratnu vrednost metode. Add metoda dbContext klase dodaje novi record u tabelu, Edit edituje postojeći record i Remove briše odabrani record. Potrebno je eksplicitno pozvati SaveChanges metodu posle svake od akcija kako bi izmene bile sačuvane u bazi podataka.

# ZAKLJUČAK

U odnosu na postavljene zahteve izrada internet aplikacije pomoću ASP MVC tehnologije je jednostavna i brza. Kreiranjem projekta pomoću sablona ponudjenog u visual studio-u dobija se gotova struktura foldera i fajlova čije delove treba izmeniti i tako se uštedi dosta vremena. Konfigurisanje projekta u odredjenim fajlovima pruža brže povezivanje delova oplikacije i daje mogućnost lakog održavanja. Konfiguraciju je moguće menjati bez promena ne zavisnih fajlova. Definisanje ruta i stranica koje prikazuju poruku ukoliko je došlo do greške u sistemu takodje je ne zavisno od ostatka projekta i vrši se u konfiguracionim fajlovima.

Korišćenjem deljenih fajlova skraćuje se vreme i količina koda potrebna za implementaciju aplikacije. Povezivanje ne zavisnih i deljenih fajlova je jednostavno i brzo kao i povezivanje sa bazom podataka preko dbContext-a. Pristup i manipulacija objektima koje Entity Framework čita iz baze podataka je jednostavno kao i upis.

Kontroleri mogu raditi ne zavisno jedan od drugog ili u kombinaciji u zavisnosti od potrebe aplikacije. Prenos objekata i parametara kroz aplikaciju kontroliše MVC i dosta pojednostavljuje taj proces. Samim tim jendu celinu čine najmanje 3 komponente, po jedan model, kontroler i view koje su povezane na strogo definisan način. Opciona komponenta je ViewModel

View čita podatke preko ViewModel-a samim tim ViewModel može sadržati zajedničke podatke iz više modela. View može sadržati i java script kod koji bez problema vrši interakciju sa serverskim delom aplikacije.

U ovom projektu je prikazano izvodjenje aplikacije iz dostupnog sablona, njeno konfigurisanje, prilagodjavanje potrebama korisnika, povezivanje različitih komponenti kao i prenos podataka kroz sistem i čuvanje istih u bazu podataka. Akcenat je upravo na povezivanju komponenti kako komponenti korisničkog interfejsa sa biznis logikom tako i prenos podataka iz kontrolera u bazu podataka i obrnuto pomoću novije Microsoft tehnologije.

# INDEKS POJMOVA

**A**

asp.net

aplikacija

autorizacija

atribut

akcija

**D**

dbContext

**B**

blog

**H**

html helper

**E**

entity framework

**I**

instanca

**K**

kontroler

komponenta

klasa

**M**

migracija

mvc

**O**

objekat

**P**

prenos

podaci

parametar

povezuje

properti

**S**

service

sitem

**T**

tag helper

tabela

**U**

update

user Interface

**V**

view

viewModel

# LITERATURA

<https://app.pluralsight.com/library/courses/aspdotnet-core-1-0-fundamentals/table-of-contents>

<https://docs.asp.net/en/latest/tutorials/first-mvc-app/start-mvc.html>

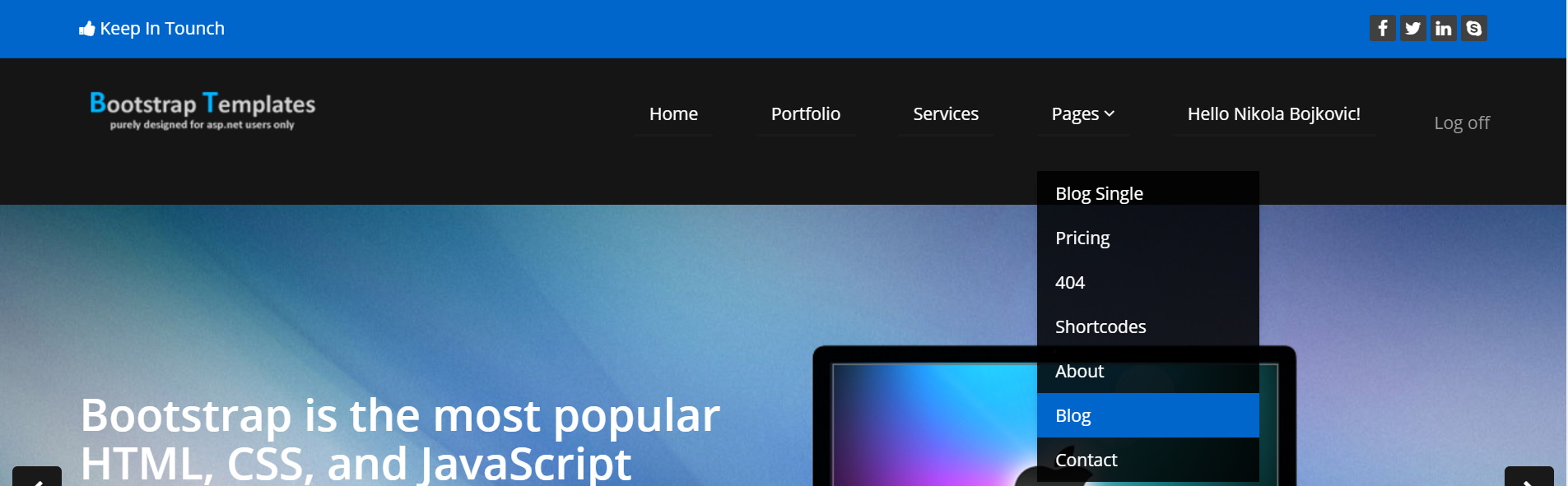
<https://msdn.microsoft.com/en-us/data/ef.aspx>

<https://docs.asp.net/en/latest/mvc/views/tag-helpers/intro.html>

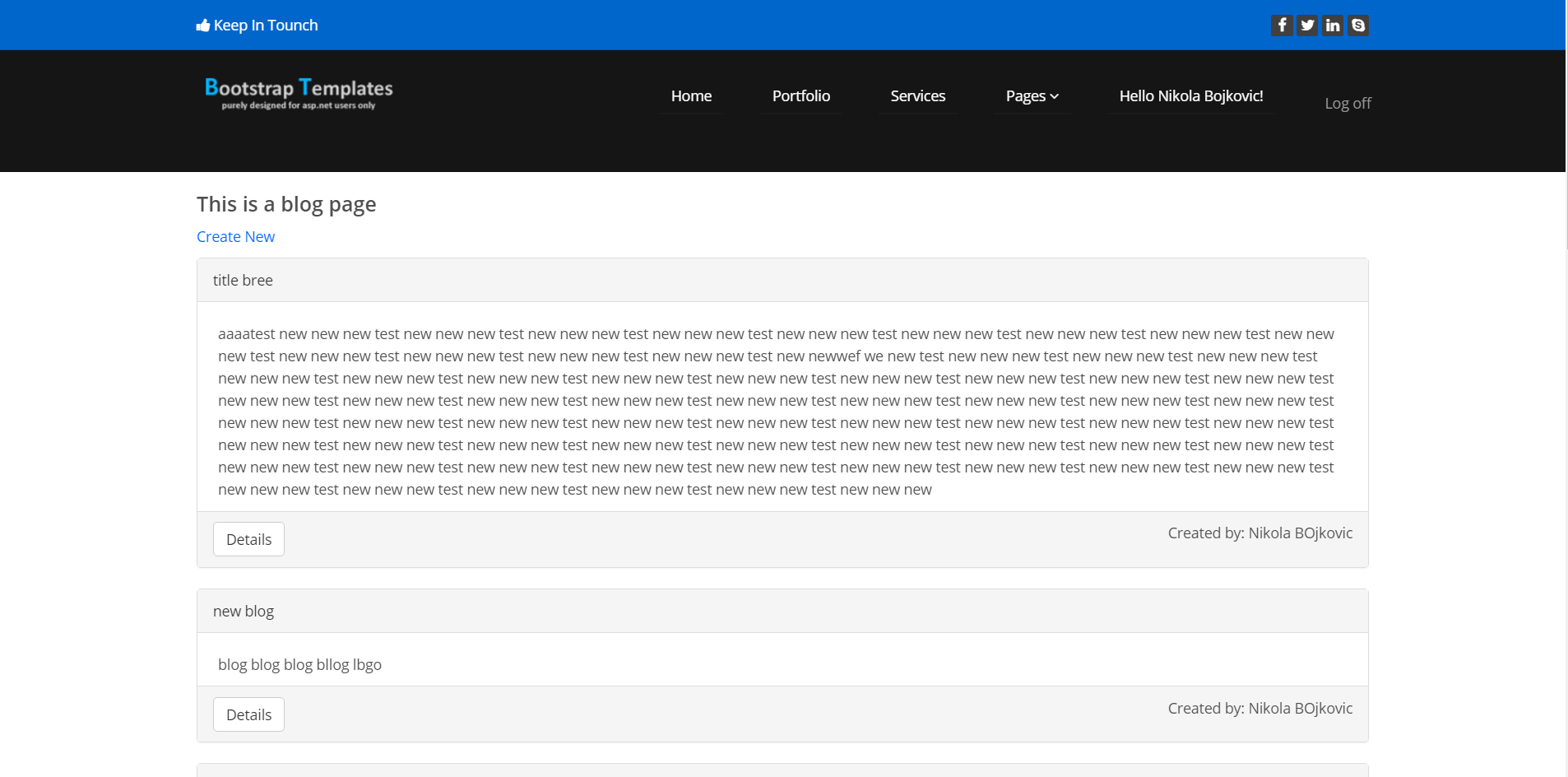
<https://docs.asp.net/en/latest/mvc/views/index.html>

<https://docs.asp.net/en/latest/mvc/views/view-components.html>

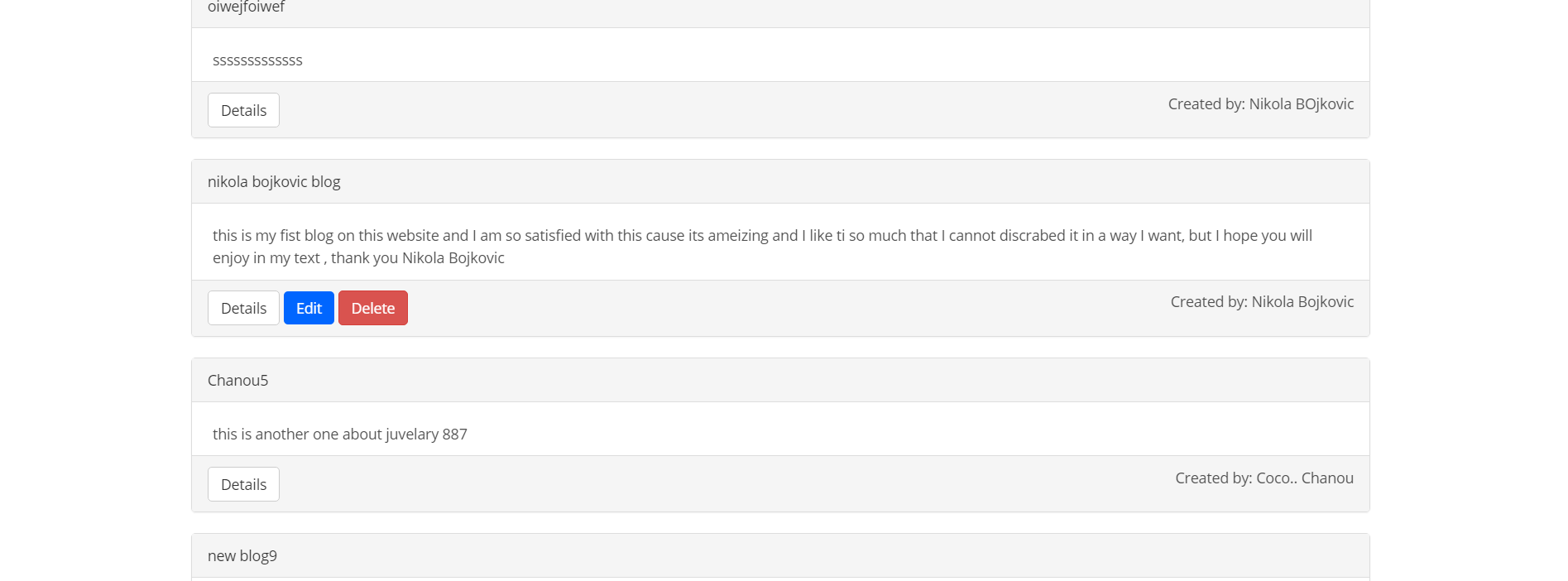
# DODATAK



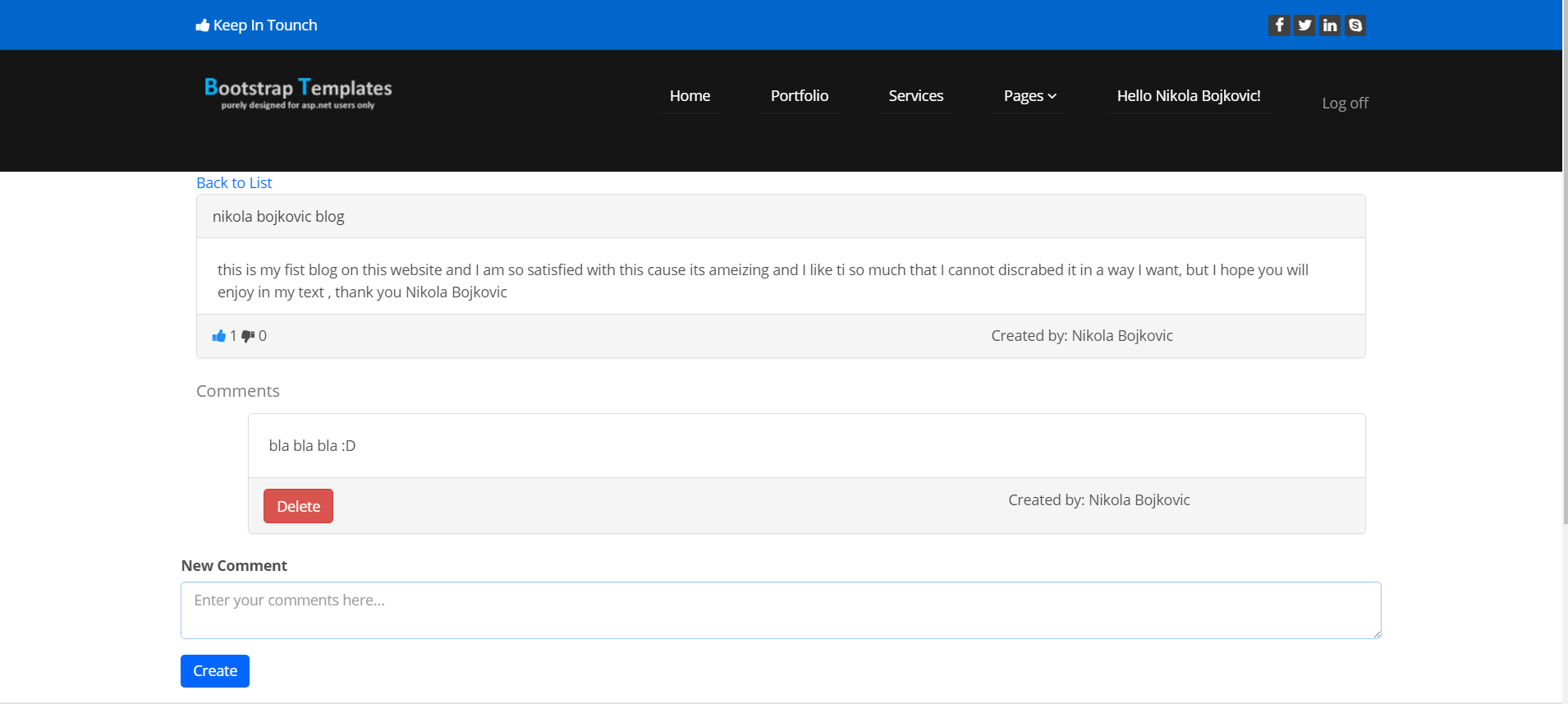
*Slika 11.1.1: Navigacioni bar sa drop down menijem*



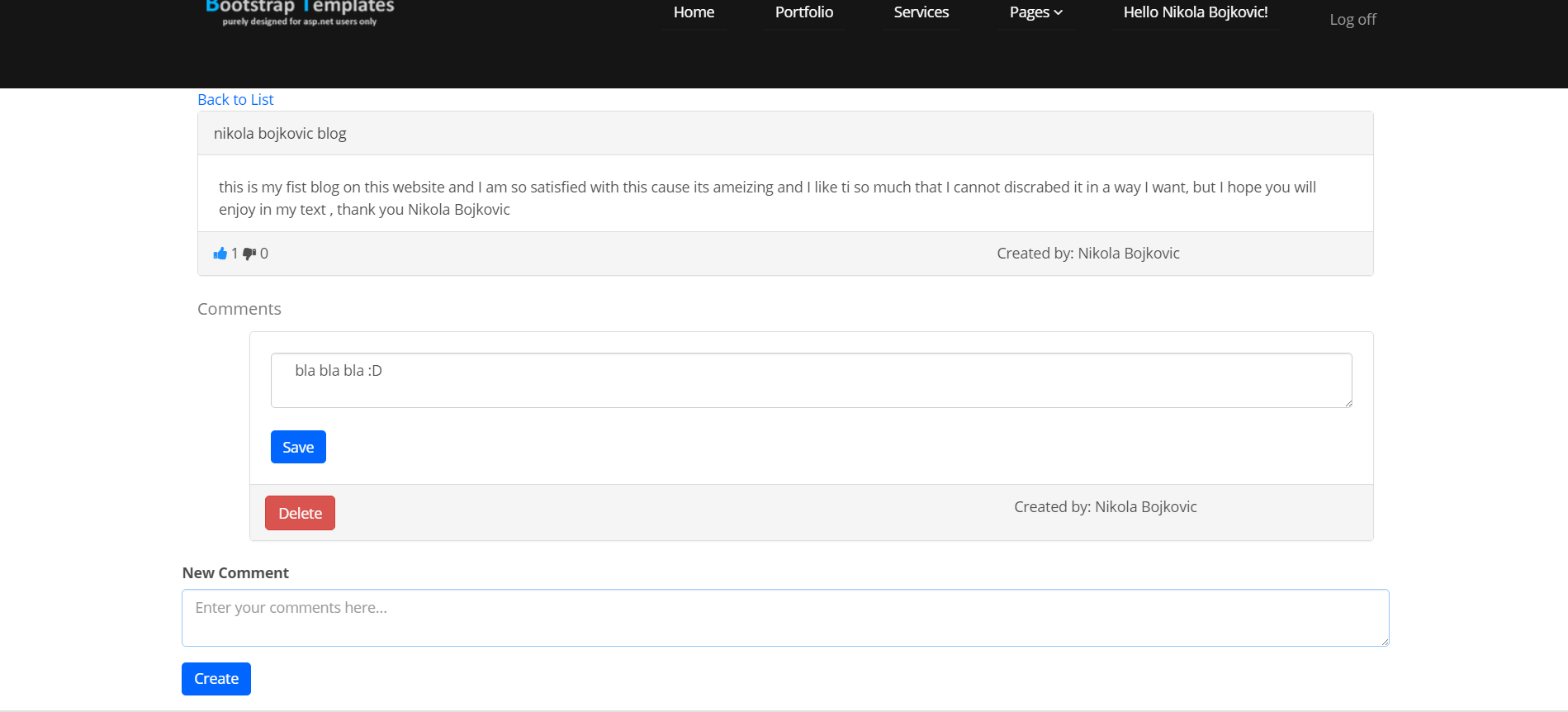
*Slika 11.1.2: Lista blogova*



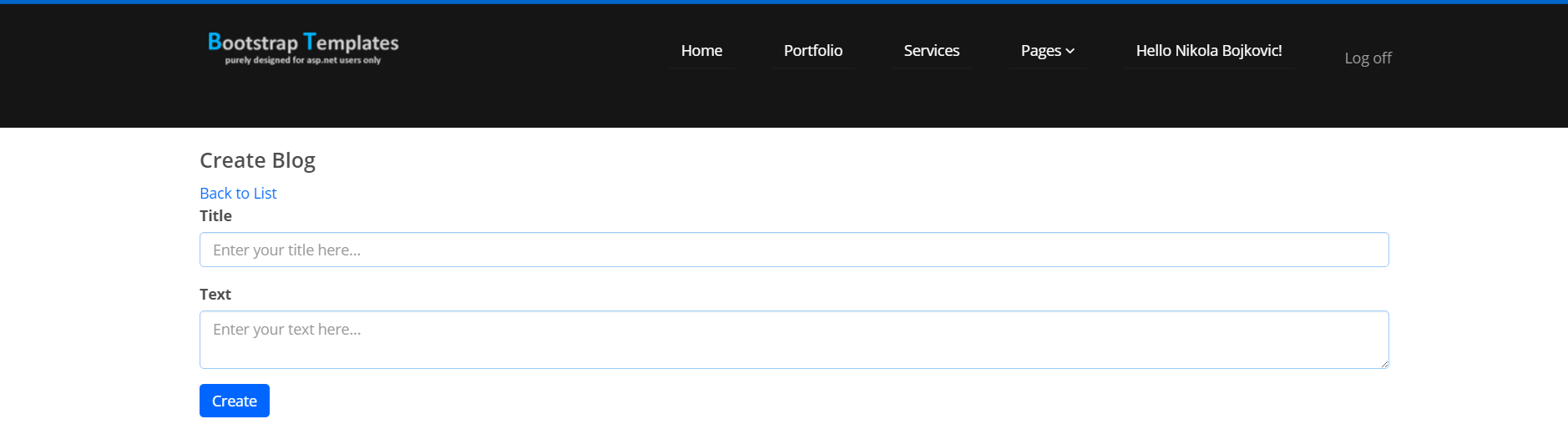
*Slika 11.1.3: Dostupne opcije ulogovanom korisniku*



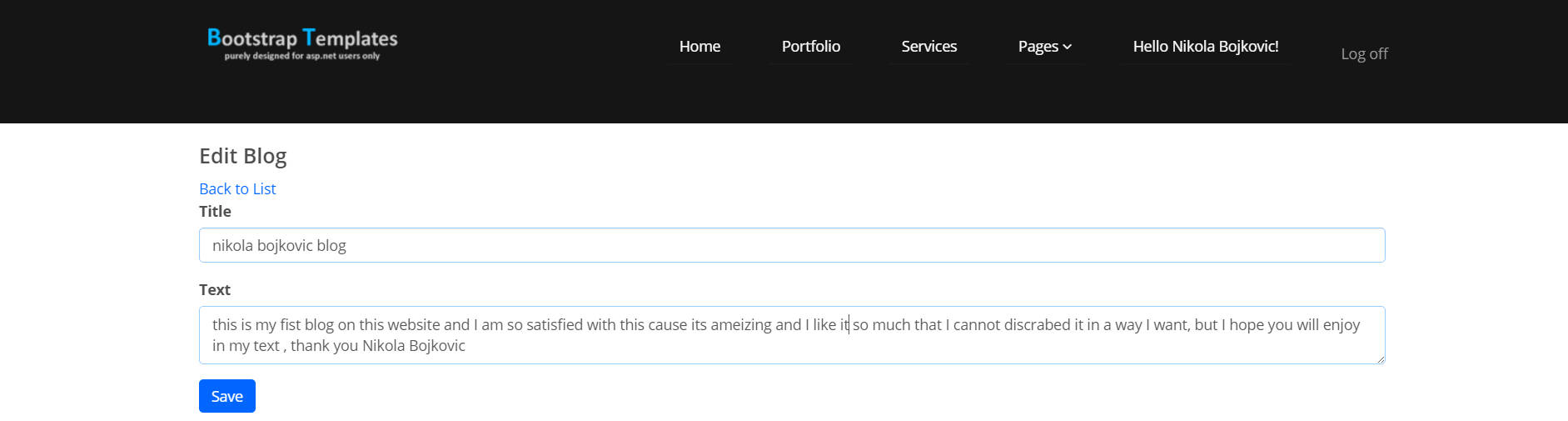
*Slika 11.1.4: Prikaz detalja bloga i njegovih komentari*



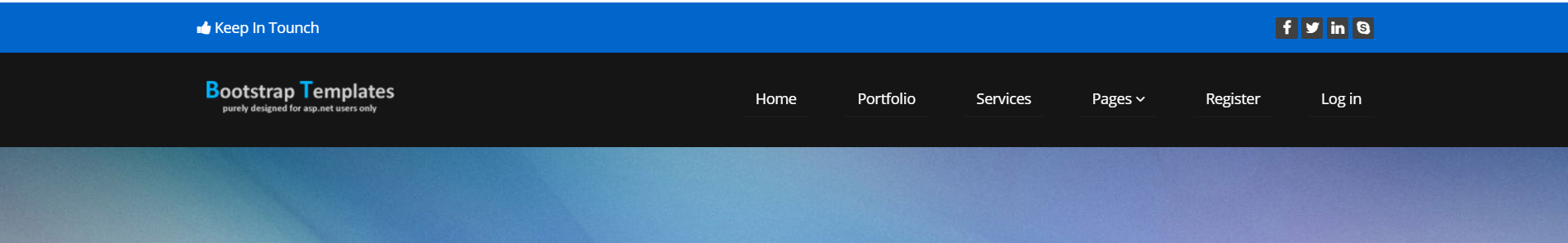
*Slika 11.1.5: Editovanje komentara*



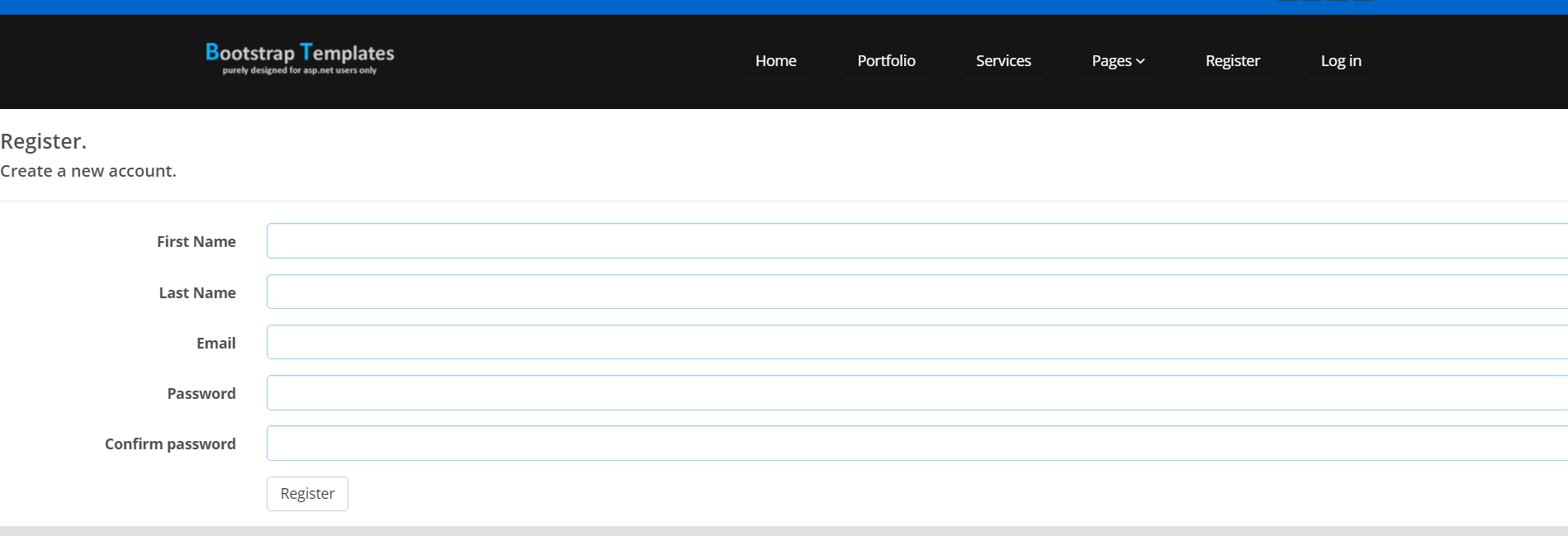
*Slika 11.1.6: Kreiranje bloga*



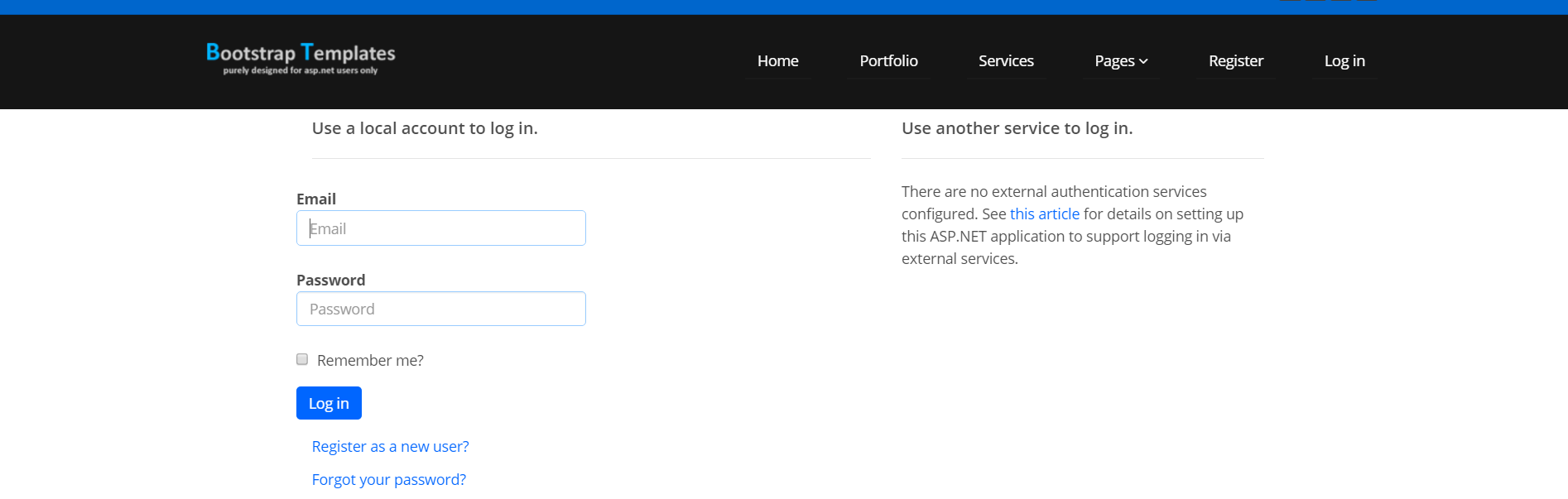
*Slika 11.1.7: Editovanje bloga*



*Slika 11.1.8: Navigacioni bar sa delom za autorizaciju*



*Slika 11.1.9: Forma za registraciju*



*Slika 11.1.10: Forma za logovanje*

# ZAHVALA