ASP.Net web aplikacija za raspored smjena zaposlenika

Grgić, Martina

Undergraduate thesis / Završni rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:755713

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

ASP.NET WEB APLIKACIJA ZA RASPORED SMJENA ZAPOSLENIKA

Završni rad

Martina Grgić

Osijek, 2017. godina.

SADRŽAJ

1. UVOD 1
2. ASP.NET CORE FRAMEWORK 2
2.1. Osnove 2
2.2. MVC pristup 5
2.3. Razor 7
3. WEB APLIKACIJA ZA RASPORED SMJENA 9
3.1. Dizajn i funkcionalnost 9
3.2. MVC obrazac 16
3.2.1. Upraviteljske metode 16
3.2.2. Entity Framework Core 20
3.2.3. Model 22
3.2.4. Dependency Injection 24
3.2.5. Detaljan opis MVC obrasca kroz primjer 26
4. ZAKLJUČAK 30
LITERATURA 31
SAŽETAK 32
ABSTRACT 32
ŽIVOTOPIS 33

1. UVOD

U ovom radu opisana je izrada web aplikacije korištenjem ASP.NET Core MVC tehnologije te Visual Studio 2017 kao integrirano razvojno okruženje tvrtke Microsoft. Pomoću izrađene aplikacije moguće je osmisliti raspored smjena zaposlenika za pojedine dane u tjednu. Informacije o zaposlenicima i njihovom rasporedu pohranjeni su u bazu podataka. Ova aplikacija može služiti za bilo koju vrstu poslovanja gdje voditelj smjene treba osmišljavati raspored smjena. Voditelj smjene može dodavati, uređivati i brisati zaposlenike te njihove smjene.

Navodi se nekoliko tehnologija pomoću kojih je aplikacija izrađena. Za izradu korisničkog sučelja korišteni su HTML, CSS te Bootstrap. Za kreiranje dinamičkih web stranica sa C#-om korišten je Razor pogonski mehanizam za poglede (engl. *view engine*).

U radu je naglasak na pozadinske tehnologije. Korišten je ASP.NET Core framework kroz MVC obrazac. Registracija i prijava voditelja smjene omogućena je primjenom *Identity* sustava. Korištena je lokalna MSSQL baza podataka za spremanje podataka voditelja smjene, zaposlenika te njihovih smjena. Za pristup i manipuliranje bazom podataka korišten je *Entity Framework Core*.

2. ASP.NET CORE FRAMEWORK

ASP.NET Core (engl. *Active Server Pages*) je framework otvorenog koda (engl. *open source*) što znači da je njegov izvorni kod dostupan svima na uvid, korištenje te izmjene uz poštivanje Apache licence, točnije ALv2. Također, ASP.NET Core je višeplatformski (engl. *cross platform*) što znači da se aplikacije koje koriste ASP.NET Core mogu razvijati i pokretati na Windowsu, Macu i Linuxu. [1]

2.1. Osnove

ASP.NET Core aplikacija je konzolna aplikacija koja kreira web poslužitelj u *Main* metodi koja je prikazana na slici 2.1.

```
public class Program
{
    public static void Main(string[] args)
    {
        var host = new WebHostBuilder()
            .UseKestrel()
            .UseStartup<Startup>()
            .Build();

        host.Run();
    }
}
```

Sl. 2.1. Prikaz *Main* metode

Main metoda koristi *WebHostBuilder* koji omogućuje kreiranje domaćina web aplikacije (engl. *web application host*). *WebHostBuilder* sadrži metode koji definiraju web poslužitelj npr. *UseKestrel* te *UseStartup*. Prema slici 2.1. korišten je Kestrel web poslužitelj, no moguće je koristiti i druge. Uz navedene metode postoji i nekoliko neobaveznih metoda kao što je *UseIISIntegration* za usluge IIS i IIS Express poslužitelja, te *UseContentRoot* za specificiranje korijenskog direktorija. *Build* i *Run* metode služe za stvaranje *IWebHost* objekta koji će posluživati aplikaciju i slušati nadolazeće HTTP zahtjeve (engl. *HyperText Transfer Protocol requests*).

U *Startup* klasi definira se zahtjev za rukovanje *pipelineom* te se konfiguriraju servisi potrebni za aplikaciju.

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
    }

    public void Configure(IApplicationBuilder app)
    {
    }
}
```

Sl. 2.2. Prikaz *Startup* klase

Servisi korišteni u aplikaciji definiraju se u metodi *ConfigureServices*. Neki od najčešćih servisa su *ASP.NET MVC Core framework*, *Entity Framework Core* te *Identity*.

U metodi *Configure* definira se *middleware*, softvere koji se ponaša kao most između baze podataka i aplikacije, u zahtijevanom *pipelineu*. ASP.NET Core dolazi sa ugrađenim *middlewareom* za statičke datoteke, usmjeravanje te autentikaciju. Moguće je koristiti bilo koji *middleware* zasnovan na OWIN (engl. *Open Web Interface for .NET*) *middlewareu*, te je moguće napisati vlastiti *middleware*. [2]

Web aplikacije mogu biti odvojene od web poslužitelja pomoću OWIN-a. OWIN je standardizirani oblik *middlewarea*. ASP.NET Core aplikacije i *middleware* mogu funkcionirati zajedno sa aplikacijama, poslužiteljima i *middlewareom* zasnovanim na OWIN-u. OWIN pruža odvojeni sloj koji omogućuje da dva različita frameworka sa različitim modelima budu korišteni zajedno. Paket *Microsoft.AspNetCore.Owin* pruža dvije implementacije: ASP.NET Core prema OWIN-u, te OWIN prema ASP.NET Core. [3]

ASP.NET Core dopušta korištenje samo onih NuGet paketa koji su potrebni za aplikaciji. Microsoftova razvojna platforma, uključujući .NET framework, koristi NuGet upravitelj podataka. NuGet klijentski alati pružaju mogućnost korištenja i stvaranja paketa. [4]

Uvođenjem .NET Core frameworka i Visual Studio IDE-a 2017 Microsoft je promijenio način na koji se NuGet paketi dodaju u projekt. Koristi se .csproj datoteka koja je XML formata. Na slici 2.3. nalazi se primjer sadržaja .csproj datoteke s nekoliko osnovnih paketa koji su potrebni za razvoj MVC web aplikacije.

```
<Project Sdk="Microsoft.NET.Sdk.Web">
<PropertyGroup>
	<TargetFramework>netcoreapp1.1</TargetFramework>
</PropertyGroup>
<ItemGroup>
	<Folder Include="wwwroot\" />
</ItemGroup>
<ItemGroup>
	<PackageReference Include="Microsoft.ApplicationInsights.AspNetCore"
		Version="2.0.0" />
	<PackageReference Include="Microsoft.AspNetCore" Version="1.1.1" />
	<PackageReference Include="Microsoft.AspNetCore.Mvc" Version="1.1.2" />
	<PackageReference Include="Microsoft.AspNetCore.StaticFiles" Version="1.1.1" />
	<PackageReference Include="Microsoft.VisualStudio.Web.BrowserLink"
		Version="1.1.0" />
</ItemGroup>

<ItemGroup>
	<DotNetCliToolReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Tools"
	Version="1.0.0"/>
</ItemGroup>

</Project>
```

Sl. 2.3 Prikaz *project.csproj* datoteke

Pomoću *PackageReference* dodaju se paketi, sa *Include* atributom specificira se ime paketa i sa *Version* atributom verzija koja je potrebna. Paketi koji se koriste za postavljanje i rad s alatima postavljeni su korištenjem *DotNetCliToolReference* elementima. U tablici 2.1. objašnjeni su navedeni paketi.

Tab. 2.1. Opis osnovnih NuGet paketa za razvoj MVC aplikacije

Naziv	Opis
Microsoft.AspNetCore.Mvc	Ovaj paket sadrži ASP.NET Core MVC i pruža pristup bitnim značajkama kao što su upravljači te Razor pogledi.
Microsoft.AspNetCore.StaticFiles	Ovaj paket pruža podršku za pružanje statičkih datoteka kao što su: slike, JavaScript i CSS iz wwwroot mape.
Microsoft.VisualStudio.Web.BrowserLink	Ovaj paket pruža podršku za automatsko ponovno pokretanje preglednika kada se datoteke u projektu promjene, što je korisno tijekom razvoja aplikacije.

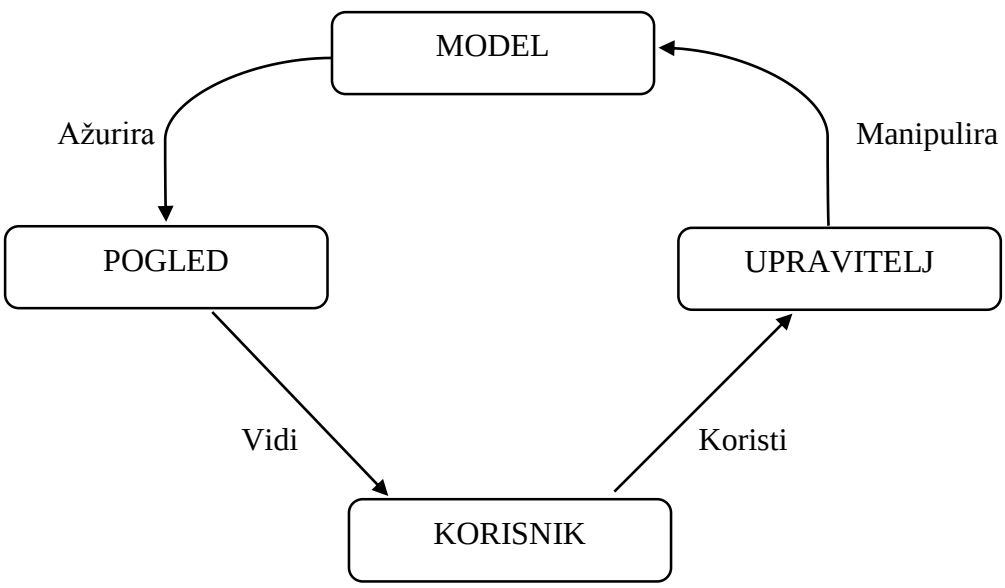
2.2. MVC pristup

Model-Pogled-Upravitelj (engl. *Model-View-Controller*) je obrazac koji odvaja pojedine dijelove aplikacije u komponente ovisno o njihovoj namjeni, dopuštajući efikasno ponovno korištenje koda i paralelni razvoj aplikacije. [5]

Trygve Reenskaug, norveški računalni znanstvenik i profesor, formulirao je MVC obrazac za GUI (engl. *Graphical User Interface*) 1979 dok je bio u posjetu Xerox Palo Alto istraživačkom centru. 1980-ih Jim Althoff, sa suradnicima, je implementirao verziju MVC-a za Smalltalk-80 biblioteku. 1988. objavljen je članak u JOT-u (engl. *The Journal of Object Technology*) u kojem su MVC prihvatili kao općepriznati koncept. [6]

Model se sastoji od podataka, poslovnih pravila, logike i funkcija ugrađenih u programsku logiku (engl. *business logic*). *View* ili pogled je bilo kakav prikaz podataka kao što je obrazac, tablica ili dijagram. Moguć je prikaz podataka (modela) kroz više različitih pogleda. *Controller* ili upravitelj prihvaća ulazne podatke (engl. *input*) i pretvara ih u smjernicu za model ili pogled.

MVC obrazac je najčešće korišten za razvoj web aplikacija, gdje je pogled stvarna HTML (engl. *HyperText Markup Languange*) stranica, upravitelj je kod koji dinamički prikuplja podatke i generira sadržaj unutar HTML-a. Model je prikazan kao stvarni sadržaj, najčešće je spremljen u bazi podataka ili XML (engl. *eXtensible Markup Language*) datoteci te sadržava programsku logiku koja mijenja taj sadržaj prema unosu korisnika. Slikovni prikaz vidljiv je na slici 2.4. [7]



Sl. 2.4 Prikaz MVC obrasca

Kako bi MVC obrazac bio točan postoji nekoliko pravila koja govore o tome što bi trebalo biti a što ne sadržano u svakom dijelu. Model bi trebao sadržavati podatke, trebao bi sadržavati logiku za kreiranje, upravljanje i mijenjanje tih podataka te pružati API koji otkriva podatke modela i operacije na njemu. Model ne bi trebao otkrivati upravitelju i pogledu detalje o načinu spremanja podataka, sadržavati logiku koja mijenja model u odnosu na korisničke unose te sadržavati logiku za prikazivanje podataka korisniku. Upravitelj bi trebamo imati akcije potrebne za mijenjanje modela prema korisničkim unosima, ne bi trebao sadržavati logiku koja upravlja izgledom podataka i logiku koja upravlja dosljednošću podataka. Pogled bi trebao imati logiku i izgled potreban da se podaci prikažu korisniku, ne bi trebao imati kompleksnu logiku te ni na koji način manipulirati modelom. [8]

ASP.NET Core MVC implementira MVC obrazac. Upravitelji su klase pisane C# programskim jezikom, izvedene iz *Microsoft.AspNetCore.Mvc.Controller* klase. Svaka javna metoda u klasi izvedena iz *Controller*-a je akcijska metoda i ona je povezana URL-om (engl. *Uniform Resource Locator*). Kada je zahtjev poslan URL-om koji je povezan sa akcijskom metodom, izjave u toj akcijskoj metodi se izvršavaju u modelu i prikazuje se pogled korisniku. Model se može kreirati koristeći obične C# objekte, te se njihova dosljednost može implementirati koristeći bilo koju bazu podataka, ORM (engl. *Object-Relational Mapping*) framework ili neki drugi oblik podatkovnih alata koji podržava .NET.

Prilikom kreiranja novog ASP.NET Core MVC projekta, Visual Studio nudi nekoliko različitih predložaka: prazan predložak (engl. *Empty*), *Web API* (engl. *Application Programming Interface*) ili *Web Application* predložak. Prazan predložak sadržava nekoliko osnovnih datoteka za razvoj ASP.NET Core aplikacije, ali ne dolazi sa bibliotekama i konfiguracijom potrebnom za MVC aplikaciju. Uz *Web API* i *Web Application* predložak mogu biti konfigurirani sa različitim shemama za autentikaciju i autorizaciju korisnika. Bilo koja funkcionalnost može biti naknadno dodana u prazan projekt, jedina razlika između predložaka je početni skup biblioteka, konfiguracijskih datoteka, koda i sadržaja koje Visual Studio dodaje kada kreira novi projekt.

Postoje dvije vrste konvencija u MVC projektu. Prva je prijedlog o tome kako bi trebala izgledati struktura projekta. Na primjer, konvencija da se *JavaScript* i *CSS* paketi postavljaju u *wwwroot/lib* mapu iako je moguće izmijeniti naziv *lib* mape ili ga čak obrisati i takve pakete postaviti negdje drugdje. Druga vrsta konvencija polazi iz principa konvencija prije konfiguracije (engl. *convention over configuration*). Princip govori o tome da nije potrebno izričito konfigurirati asocijacije između upravitelja i pogleda, potrebno je samo pratiti određenu konvenciju imenovanja datoteka.

Upraviteljske klase imaju imena koja završavaju sa *Controller*, npr. *HomeController*. Koristeći takvu konvenciju MVC automatski povezuje ime upravitelja (*Home*) sa upraviteljskom klasom. Poglede treba spremiti u mapu */Views/ImeUpravitelja*, tj. */Views/Home* mapu, npr. ako se zadani pogled povezan sa akcijskom metodom u upraviteljskoj klasi koji se zove Indeks nazove *Index.cshtm*l u upraviteljskoj metodi nije potrebno navoditi ime pogleda. Primjer takve metode prikazan je na slici 2.5. [8]

```
public class HomeController : Controller
{
    public IActionResult Index()
    {
        return View();

    }
}
```

Sl. 2.5 Prikaz *Index* metode u *Home* upravitelj

ASP.NET Core MVC koristi *Routing middleware* kako bi spojio URL-ove nadolazećih zahtjeva sa upraviteljskim akcijama. Rute se mogu definirati u *Startup* klasi prema konvenciji (kod je vidljiv na slici 2.6) ili kao atributi postavljanjem rute na upravitelju ili upraviteljskoj akciji.

```
app.UseMvc(routes =>
{
   routes.MapRoute("default", "{controller=Home}/{action=Index}/{id?}");
});
```

Sl.2.6 Dio koda unutar *Configure* metode u *Startup* klasi

Prema slici 2.6. kao zadani upravitelj definira se *Home*, te kao zadana akcija *Index*, gdje je id neobavezno polje. Koristeći ovu rutu *Index* akcija koja se nalazi u *Home* upravitelju bila bi izvršena za bilo koji od ovih URL ruta: */Home/Index/3*, */Home/Index*, */Home* ili /.

2.3. Razor

Razor je markup jezik sa poslužiteljske strane (engl. *server side*). Na web stranici koja koristi Razor sintaksu postoje dvije vrste sadržaja: klijentski sadržaj (engl. *client content*) i poslužiteljski kod (engl. *server code*). U *client code* ulaze: HTML markup elementi, informacije o stilu kao što je CSS, JavaScript i obični tekst. Razor sintaksa dopušta dodavanje poslužiteljskog koda klijentskom sadržaju. Ako postoji poslužiteljski kod na stranici, poslužitelj prvo izvršava taj kod i nakon toga šalje stranicu pregledniku. Korištenjem poslužiteljskog koda moguće je izvršiti puno

kompleksnije zadatke nego korištenjem samo klijentskog koda, kao što je na primjer pristupanje bazi podataka. Poslužiteljski kod može dinamički kreirati klijentski sadržaj, može generirati HTML markup ili drugi sadržaj i slati ga pregledniku zajedno sa bilo kojim statičkim HTML-om koji stranica sadržava. Iz perspektive preglednika klijentski sadržaj generiran na ovakav način nije ni na kakav način drugačiji od klijentskog sadržaja.

ASP.NET web stranice koji uključuju Razor sintaksu imaju posebni nastavak: *.cshtml*. Pomoću te ekstenzije poslužitelj prepoznaje da se na web stranici nalazi Razor sintaksa te prvo izvršava Razor naredbe i zatim šalje stranicu pregledniku. [9]

Razor sintaksa je izdana 2011. godine kao dio ASP.NET MVC3 frameworka i koristila je HTML pomoćne elemente (engl. *helpers*). Izdavanjem ASP.NET Core uvedeni su pomoćni elementi za označnike (engl. *Tag helpers*). Korištenjem *Tag Helpera* Razor sintaksa izgleda kao standardni HTML i zbog toga *front-end* dizajneri mogu uređivati Razor bez učenja C# Razor sintakse. Microsoftov *IntelliSense*, alat za popunjavanje koda, dostupan je i u Razor sintaksi. Na slici 2.7. prikazan je dio koda koji koristi *Tag Helpere*, te kako izgleda pripadajući HTML. [10]

```
@* Korištenje TagHelpera:  *@

<label asp-for="Email"></label>

@* Generirani HTML:  *@

<label for="Email">Email</label>
```

Sl.2.7 Prikaz Razor sintakse

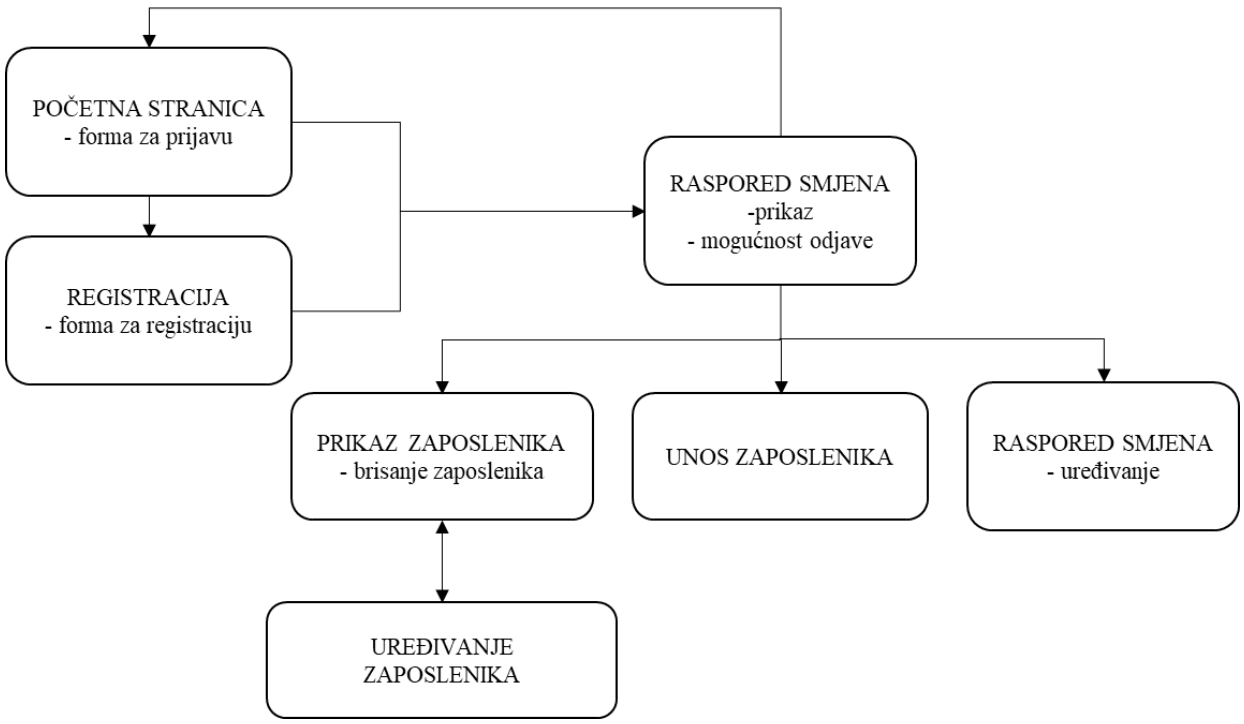
Pomoću *asp-for* atributa u *input* elementu izvodi se ime svojstva specificiranog modela u HTML-u. *Asp-for* atribut je omogućen zbog *for* svojstva u *LabelTagHelper*-u.

Za korištenje *Tag Helpera* potrebno je dodati *@addTagHelper* direktivu u *_ViewImports.cshtml* datoteku unutar *View* foldera u MVC projektu.

3. WEB APLIKACIJA ZA RASPORED SMJENA

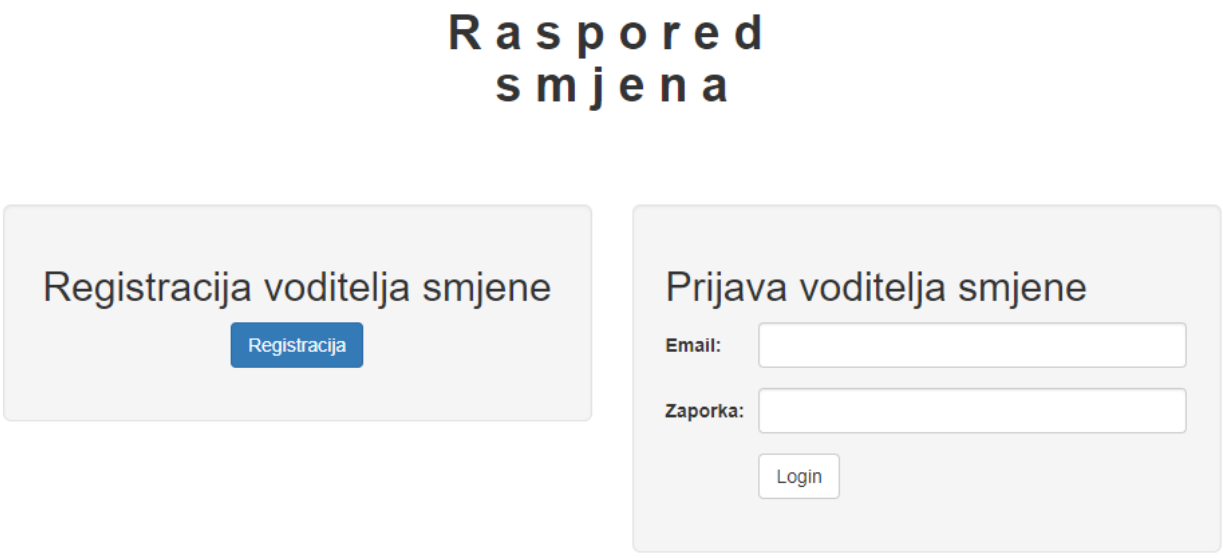
3.1. Dizajn i funkcionalnost

Sučelje web aplikacije sastoji se od šest različitih stranica. Sažet prikaz stranica i njihovih odnosa prikazan je na slici 3.1.

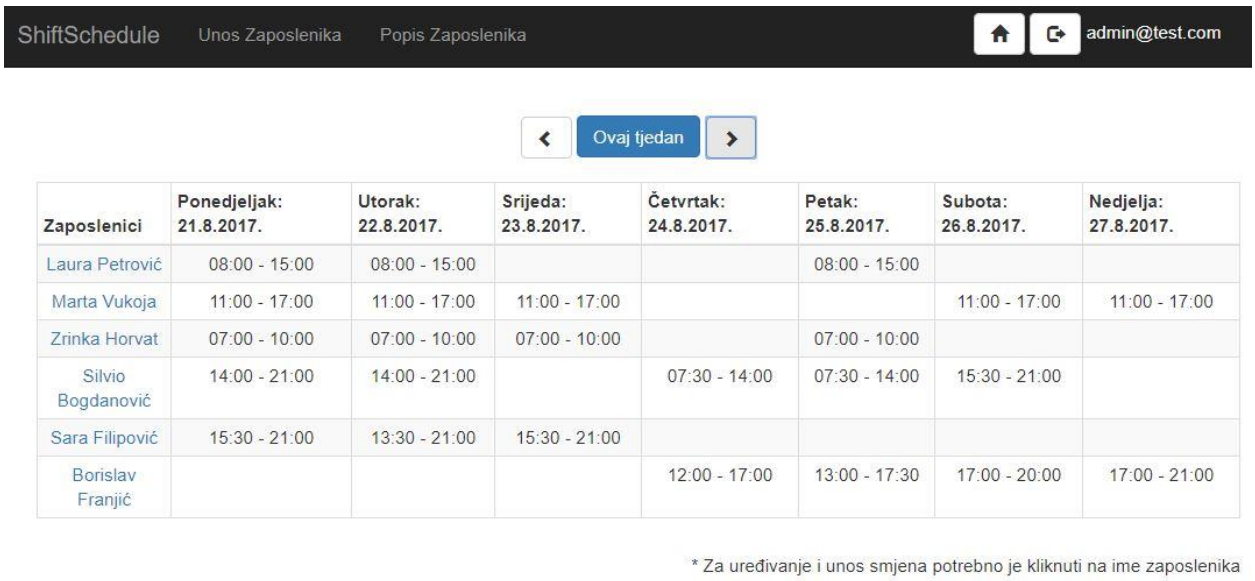


Sl. 3.1 Pregled stranica i njihovih odnosa

Početna stranica sadržava formu za prijavu, korisnik se može prijaviti elektronskom poštom i lozinkom. Nakon uspješne prijave korisnik je preusmjeren na stranicu koja sadržava raspored smjena. Korisnik se također može registrirati pritiskom na gumb registracija koja ga vodi na stranicu registracije. Izgled početne stranice vidljiv je na slici 3.2. Kako bi se voditelj smjene registrirao potrebno je navesti ispravnu elektronsku poštu, zaporku te naziv poduzeća. Nakon uspješne registracije preusmjeren je na stranicu koja sadržava prikaz rasporeda smjena čije je sučelje prikazano na slici 3.3. Klikom na ime zaposlenika moguće je urediti i unijeti smjene tog zaposlenika za određeni tjedan.



Sl.3.2 Sučelje početne stranice

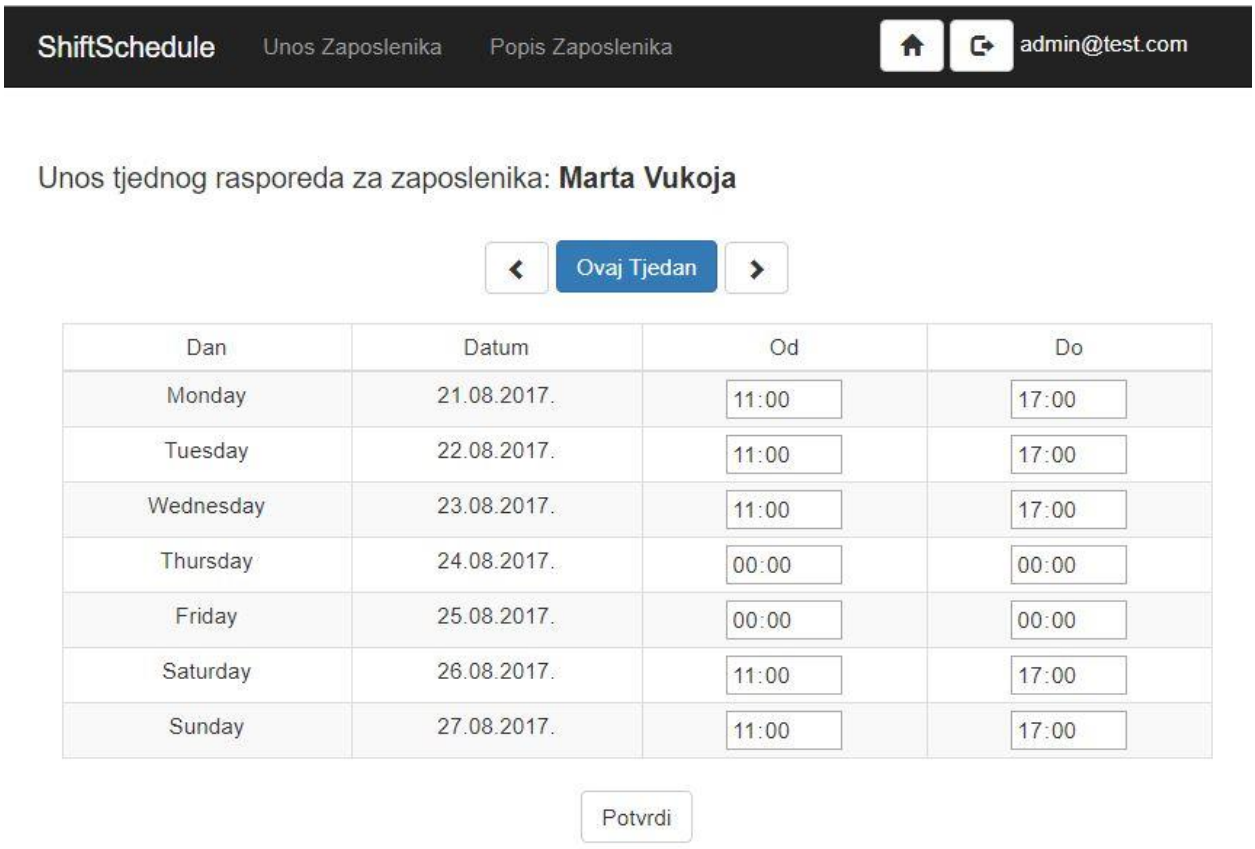


Zaposlenici	Ponedjeljak: 21.8.2017.	Utorak: 22.8.2017.	Srijeda: 23.8.2017.	Četvrtak: 24.8.2017.	Petak: 25.8.2017.	Subota: 26.8.2017.	Nedjelja: 27.8.2017.
Laura Petrović	08:00 - 15:00	08:00 - 15:00			08:00 - 15:00		
Marta Vukoja	11:00 - 17:00	11:00 - 17:00	11:00 - 17:00			11:00 - 17:00	11:00 - 17:00
Zrinka Horvat	07:00 - 10:00	07:00 - 10:00	07:00 - 10:00		07:00 - 10:00		
Silvio Bogdanović	14:00 - 21:00	14:00 - 21:00		07:30 - 14:00	07:30 - 14:00	15:30 - 21:00	
Sara Filipović	15:30 - 21:00	13:30 - 21:00	15:30 - 21:00				
Borislav Franjić				12:00 - 17:00	13:00 - 17:30	17:00 - 20:00	17:00 - 21:00

Sl.3.3 Sučelje glavne stranice

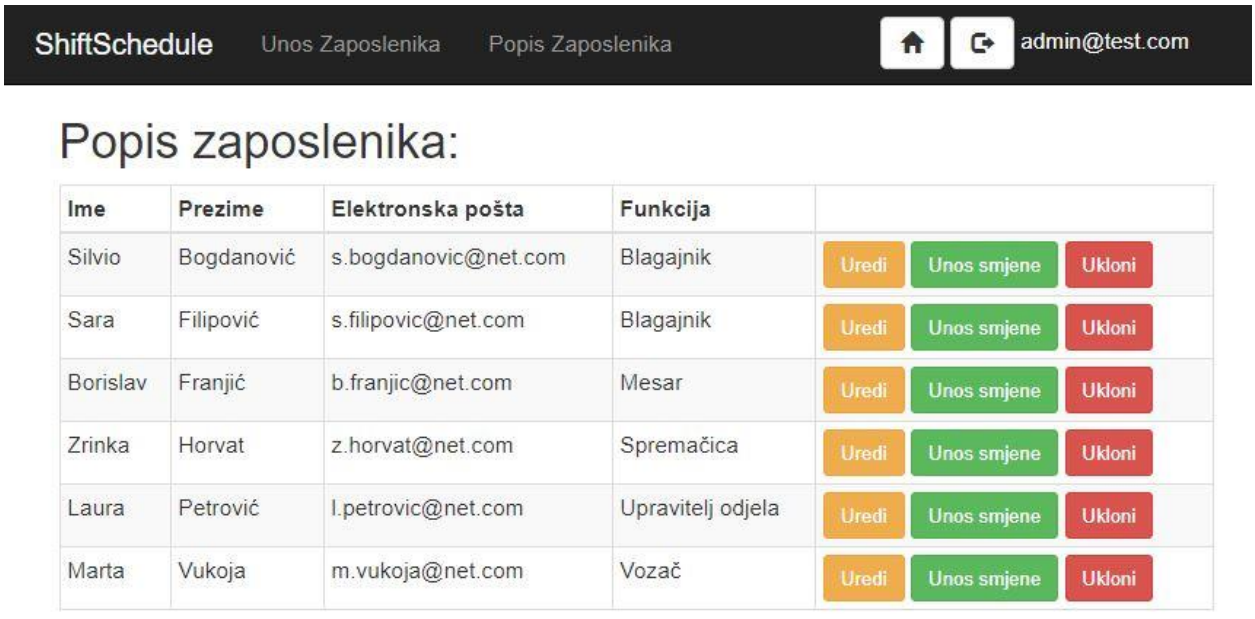
Na stranici sa rasporedom smjena, prikazom zaposlenika, unosom novog zaposlenika te na stranici za uređivanje zaposlenika nalazi se navigacijska traka na kojoj se nalaze gumbi i poveznice za odjavu, povratak na stranicu sa rasporedom smjena, popisom zaposlenika te stranicu za unos novog zaposlenika.

Na stranici za uređivanje tjednog rasporeda smjena pojedinog zaposlenika nalazi se tablica koja sadrži postojeće smjene, koje je moguće urediti kao i dodati nove. Sučelje stranice vidljivo je na slici 3.4.



Sl. 3.4 Sučelje stranice za uređivanje smjena

Na stranici sa prikazom zaposlenika tablično su prikazani svi uneseni zaposlenici, gdje je moguće svakog zaposlenika urediti, obrisati te unijeti smjene. Sučelje ove stranice nalazi se na slici 3.5. Imena zaposlenika korištena u primjeru odabrana su *Random Name Generatorom*. [11]



Sl.3.5 Sučelje stranice sa prikazom zaposlenika

Na stranici za unos zaposlenika nalazi se forma u kojoj je potrebno unijeti ime i prezime zaposlenika, elektronsku poštu te funkciju koju obavlja u poduzeću. Stranica za uređivanje zaposlenika sadrži istu tu formu samo sa podacima koji su specifični za određenog zaposlenika i svaku stavku moguće je promijeniti što je vidljivo na slici 3.6.



Sl. 3.6 Izgled forme za uređivanje postojećeg zaposlenika i dodavanje novog

Za stiliziranje web stranica korišten je *Bootstrap, open source front-end* web framework. *Bootstrap* sadrži HTML i CSS dizajnerske obrasce za tipografiju, forme, gumbe, navigacijske trake i ostale dijelove sučelja. Mark Otto i Jacob Thornton razvili su *Bootstrap* za *Twitter* 2011. godine. Zadnja stabilna verzija je *Bootstrap* 3, točnije 3.3.7. Svi najčešće korišteni preglednici uključujući Google Chrome, Firefox, Internet Explorer, Operu i Safari podržavaju *Bootstrap*. *Bootstrap* ima modularni dizajn i sastoji se od niza *less* stilskih predložaka. Ti stilski predlošci se najčešće prevode u paketu i onda se dodaju u web stranicu, ali se i pojedinačni paketi mogu dodavati i brisati. Svaki sastavni dio *Bootstrapa* se sastoji od HTML strukture, CSS deklaracije i u nekim slučajevima *JavaScript* koda. [12]

Jedna od prednosti korištenja *Bootstrapa* je njegov sustav rešetke. Na slici 3.7 prikazan je dio koda korišten za sučelje početne stranice.

```
<div class="col-md-6">
    <div class="panel panel-primary well well-lg">
        <div class="form-group text-center">
            <h2>Registracija voditelja smjene</h2>
            @* preskočeni dio koda *@
        </div>
    </div>
    <br>
</div>
<div class="col-md-6">
    <div class="panel panel-default well well-lg">
        <form class="form-horizontal" asp-controller="Account" asp-action="Login"
                                                                    method="post">
            <h2>Prijava voditelja smjene</h2>
            @* preskočeni dio koda *@
        </form>
    </div>
</div>
```

Sl. 3.7 Dio HTML koda početne stranice

Prema slici 3.4 vidljivo je korištenje *Bootstrap* rešetkastog sustava. Kako bi forme koje sadrže prijavu i registraciju bile jedna do druge potrebno je obje forme omotati *div* elementom koji sadrži klasu *col-md-6* koja dijeli stranicu na dva jednaka dijela, *Bootstrapov* rešetkasti sustav dijeli stranicu na 12 jednakih dijelova.

Kako bi se mogao koristiti *Bootstrap* potrebno je kreirati *bower.json* datoteku i dodati paket u *dependencies* odjeljak. To je potrebno napraviti i za JavaScript biblioteke, jQuery i moment.js koji su također korišteni u projektu. Sadržaj datoteke *bower.json* prikazan je na slici 3.8.

```
{
  "name": "asp.net",
  "private": true,
  "dependencies": {
    "bootstrap": "3.3.7",
    "jquery": "3.2.1",
    "moment": "2.18.1"
  }
}
```

Sl.3.8. Sadržaj *bower.json* datoteke

Bower upravitelj paketa instalira CSS i JavaSript pakete u */wwwroot/lib* mapu prema konvenciji.

Views ili pogledi su dio MVC obrasca koji se koriste za prikaz sadržaja korisniku. U ASP.NET Core MVC aplikaciji, *view* je datoteka koja sadržava HTML dokumente i C# kod, koji je obrađen kako bi generirao odgovor.

U projektu se nalazi mapa *Views* koja sadržava nekoliko podmapa i *view* datoteka. Svaka podmapa unutar *Views* mape ima naziv prema *Controlleru* ili upravitelju koji ju koristi. Tako se na primjer početna stranica koja sadrži formu za prijavu zove *Login.cshtml* i nalazi se u mapi *Account* prema

AccountController.cs koji sadrži *Login* upraviteljsku metodu. Osim podmapa koji su nazvani prema odgovarajućem upravitelju postoji podmapa *Shared* u kojoj se nalazi *_Layout.cshtml* i *Nav.cshtml* datoteka. U ovoj podmapi nalaze se datoteke koje nisu vezane specifično za jednog upravitelja. Slikoviti prikaz sadržaja mape *Views* ovog projekta, sa svim podmapama i datotekama nalazi se na slici 3.9.

```
Views
├── Account
│   ├── Login.cshtml
│   └── Register.cshtml
├── Employee
│   ├── Edit.cshtml
│   └── Index.cshtml
├── Shared
│   ├── _Layout.cshtml
│   └── Nav.cshtml
├── Shift
│   ├── Edit.cshtml
│   └── Index.cshtml
├── _ViewImports.cshtml
└── _ViewStart.cshtml
```

Sl. 3.9 Struktura *Views* mape

Datoteka *_Layout.cshtml* sadrži sve <*link*> i <*script*> elemente koji su potrebni za svaki *view*. Svi pogledi koji nasljeđuju ovaj izgled u sebi već sadržavaju navedene elemente i zbog *razorovog* atributa *@RenderBody()* koji se nalazi u *layoutu* potrebno je samo pisati onaj dio koda koji bi se inače nalazio u <*body*> elementu.

Datoteka *Nav.cshtml* predstavlja navigacijsku traku, koju koriste više pogleda. Navigacijska traka u pogled se uključuje pomoću *@Html.Partial("Nav").*

_ViewImports.cshtml datoteka se koristi za specificiranje prostora imena (engl. *namespace*) koji se koriste u *view* datotekama. Koristi se i za postavljanje *tag helpera.* Na slici 3.10 vidi se sadržaj ove datoteke za projekt.

```
@using Project.Models
@using Project.ViewModels
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Sl. 3.10 *_ViewImports.cshtml* datoteka

Datoteka *_ViewStart.cshtml* se koristi za specificiranje zadanog *layouta* za sve poglede.

```
@model IEnumerable<Employee>

@Html.Partial("Nav")

<div class="container">
@if (!Model.Any())
{
    <h2>Nema podataka u bazi</h2>
}
else
{
    <h2>Popis zaposlenika</h2>
        <table class="table table-striped table-bordered table-condensed">
            <thead>
                <tr>
                    <th>Ime</th>
                    <th>Prezime</th>
                    <th>Email</th>
                    <th>Funkcija</th>
                    <th> </th>
                </tr>
            </thead>
            <tbody>
             @foreach (var e in Model)
              {
              <tr>
              <td>@e.Name</td>
              <td>@e.Surname</td>
              <td>@e.Email</td>
              <td>@e.JobDescription</td>
              <td>
              <form asp-action="Delete" asp-controller="Employee" method="post">
              <a asp-action="Edit" asp-controller="Employee" class="btn btn-sm btn-
              warning" asp-route-EmployeeId="@employee.Id">Uredi</a>
              <input type="hidden" name="EmployeeId" value="@employee.Id" />
              <a asp-action="Edit" asp-controller="Shift" class="btn btn-sm btn-
              success" asp-route-EmployeeId="@employee.Id">Unos smjene</a>
              <button type="submit" class="btn btn-danger btn-sm">Ukloni</button>
              </form>
              </td>
              </tr>
              }
              </tbody>
        </table>
}
</div>
```

Sl. 3.11. *List.cshtml* datoteka

Na slici 3.11 nalazi se HTML kod potreban za prikaz stranice sa pregledom zaposlenika. Uz HTML kod korištena je *Razor* sintaksa za dohvaćanje podataka o pojedinom zaposleniku. Omogućen je pristup modelu zaposlenika, odnosno listi zaposlenika koji se trenutno nalaze u bazi podataka i to za trenutno prijavljenog voditelja smjene.

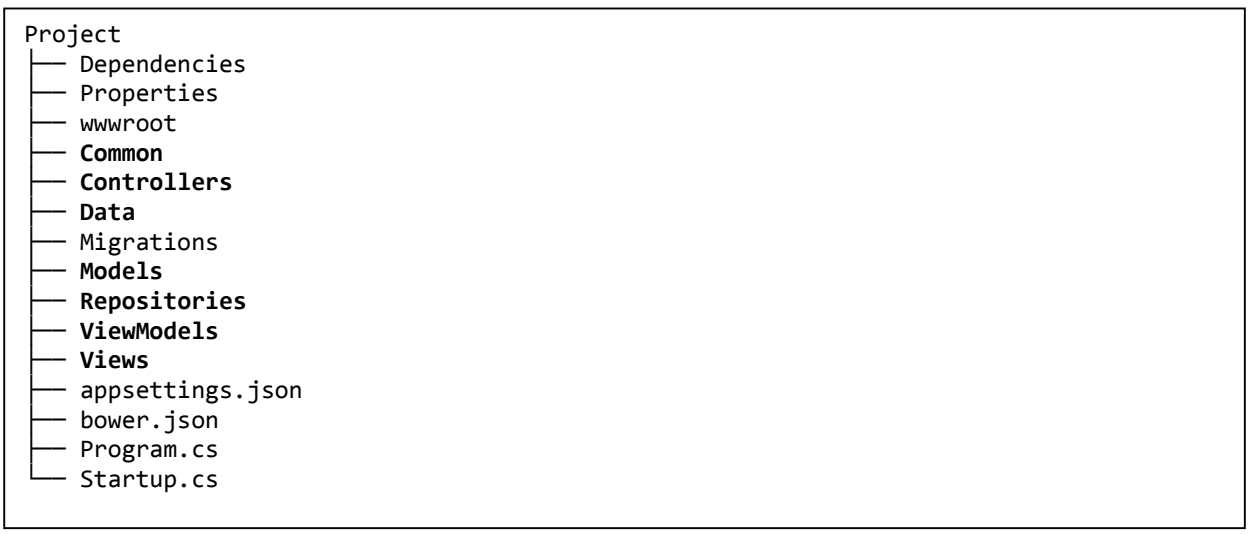
Pomoću *if-else* naredbe omogućen je prikaz dva različita pogleda ukoliko postoje zaposlenici u bazi odnosno ukoliko ih trenutno nema. Ako trenutno nema zaposlenika u bazi podataka prikazuje se poruka *Nema podataka u bazi*. Ako postoje zaposlenici u bazi stvara se tablica sa njihovim

informacijama: ime, prezime, elektronska pošta te funkcija, to je postignuto *foreach* petljom koja prolazi kroz cijelu listu zaposlenika.

U posljednjem stupcu tablice nalaze se tri gumba: gumb uredi, gumb obriši te gumb za unos smjene koji služe za prikaz stranice za uređivanje odabranog zaposlenika ili za brisanje istog. Gumb za unos smjene vodi na stranicu sa mogućnošću uređivana smjena. Pomoću *asp-action* i *asp-controller* atributa pristupa se određenom upravitelju odnosno upraviteljskoj akciji.

3.2. MVC obrazac

Mapa projekta sadrži nekoliko podmapa i datoteka kao što je prikazano na slici 3.12.



Sl. 3.12 Mapa projekta

Ovakva struktura odgovara pravilima korištenja MVC obrasca. Unutar mape *Controllers* nalaze se upravitelji, unutar mape *Models* nalaze se modeli, unutar mape *ViewModels* modeli potrebni za određeni pogled, te u mapi *Repositories* repozitoriji. Opis sadržaja *Views* mape dan je u prethodnom poglavlju.

3.2.1. Upraviteljske metode

U projektu se nalaze tri upravitelja. Struktura mape prikazana je na slici 3.13.



Sl. 3.13 Mapa *Controllers*

Svaki upravitelj prikazan na slici 3.13. ima određenu funkciju i sastoji se od nekoliko metoda. Svi upravitelji nasljeđuju *Microsoft.AspNetCore.Mvc.Controller* klasu.

Upravitelji sadrže akcije potrebne za mijenjanje modela prema unosu korisnika, a dohvaćanje i slanje podataka obavlja pomoću repozitorija. Oni ne ovise direktno o trenutno korištenom repozitoriju, poznaje samo ulazne parametre funkcija koje repozitorij sadrži i na temelju izlaznih vrijednosti upravitelj upravlja pogledima.

EmployeeController sadrži metodu za prikaz, uređivanje i brisanje zaposlenika. *ShiftController* upravlja podacima potrebnim za stranicu sa rasporedom smjena, za prikaz i unošenje smjena. *AccountController* koristi metode potrebne za upravljanje prijavom, odjavom i registracijom korisnika odnosno voditelja smjene koristeći ASP.NET Core *Identity* sustav.

Identity je sustav članstva koji omogućava dodavanje funkcionalnost prijave u aplikaciju. Korisnici mogu kreirati korisnički račun i prijaviti se sa korisničkim imenom i lozinkom ili mogu koristiti vanjske račune kao što su *Facebook*, *Google*, *Microsoft Account*, *Twiter* i drugi. U projektu ASP.NET Core *Identity* konfiguriran je tako da podatke o korisnicima sprema u *SQL Server* bazu podataka. [13]

Projekt sadržava paket *Microsoft.AspNetCore.Identity.EntityFrameworkCore* koji će spremati i manipulirati *identity* podacima spremljenim u *SQL Server* i to koristeći *Entity Framework Core*. Unutar *ConfigureServices* metode u *Startup* klasi dodani su *identity* servisi za model *ShiftManager* odnosno voditelja smjene prema slici 3.11.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
       options.UseSqlServer(Configuration["Data:BlueSunData:ConnectionString"]));
    services.AddIdentity<ShiftManager, IdentityRole>()
       .AddEntityFrameworkStores<ApplicationDbContext>();
    services.AddTransient<IEmployeeRepository, EmployeeRepository>();
    services.AddTransient<IShiftRepository, ShiftRepository>();
    services.AddMvc();
}
```

Sl. 3.11 *ConfigureServices* metoda *Startup* klase

AcountController sastoji se od pet metoda. Dvije HTTP GET metode te dvije HTTP POST metode, za registraciju i prijavu, te metode za odjavu. Sadržaj metoda potrebnih za registraciju nalazi se na slici 3.12.

```
//  GET: Register()
[HttpGet]
public IActionResult Register()
{
    return View();
}

//  POST: Register()
[HttpPost]
public async Task<IActionResult> Register(RegisterViewModel model)
{
    if (ModelState.IsValid)
    {
        ShiftManager user = new ShiftManager
        {
            UserName = model.Email,
            Email =
                    model.Email,
            CompanyName = model.CompanyName
        };
        IdentityResult result = await _userManager.CreateAsync(user, model.Password);
        if (result.Succeeded)
        {
            await _signInManager.SignInAsync(user, false);
            return RedirectToAction("Index", "Shift");
        }
        foreach (var error in result.Errors)
        {
            ModelState.AddModelError(string.Empty, error.Description);
        }

    }
    return View();
}
```

Sl. 3.12 *Register* metode u *AccountController* klasi

Get Register metoda služi samo za prikaz prazne registracijske forme koja se nalazi na lokaciji /*Views/Account/Register*.

Post Register metoda je asinkrona metoda koja dohvaća podatke o *RegisterViewModel* modelu i ako spremanje prođe korisnika preusmjerava na stranicu sa rasporedom smjena, u drugom slučaju korisnik ostaje na formi za registraciju uz opis zbog čega registracija nije uspješna.

Kreiranje korisnika odvija se pomoću asinkrone funkcije *CreateAsync*, koja se nalazi u *_userManager* objektu klase *UserManager* koja je dostupna zbog korištenja *Identity* sustava. *CreateAsync* kreira specificiranog korisnika i njegove lozinke unutar navedene baze podataka.

Nakon uspješno kreiranog korisnika pomoću *SignInAsync* funkcije prijavljuje se taj korisnik te se preusmjerava na stranicu sa rasporedom smjena, koja nije dostupna korisnicima koji nisu prošli prijavu.

Na slici 3.13 nalaze se metode *AccountController* klase koje omogućavaju prijavu korisnika odnosno voditelja smjene.

```
//     GET : Login()
[HttpGet]
public IActionResult Login()
{
    return View();
}

//     POST : Login()
[HttpPost]
public async Task<IActionResult> Login(LoginViewModel model)
{
    if (ModelState.IsValid)
    {
        var result = await _signInManager.PasswordSignInAsync(model.Email,
            model.Password, false, false);
        if (result.Succeeded)
        {
            return RedirectToAction("Index", "Shift");
        }
        else
        {
            ModelState.AddModelError(string.Empty, "Prijava neuspješna");
        }
    }
    return View();
}
```

Sl. 3.13 *Login* metode u *AccountController* klasi

Get Login metoda služi samo za prikaz početne stranice koja sadrži formu za prijavu, a nalazi se na lokaciji */Views/Account/Login*.

Post Login metoda je asinkrona metoda koja dohvaća podatke o *LoginViewModel* modelu te ako su podaci ispravni korisnik je preusmjeren na stranicu sa smjenama, ukoliko podaci nisu ispravni korisnik ostaje na trenutnoj stranici sa porukom o neuspješnoj prijavi.

Za prijavu koristi se *PasswordSignInAsync* funkcija koja kao parametre prima elektronsku poštu odnosno korisničko ime te lozinku korisnika. Podaci se provjeravaju s onima u bazi podataka te se ovisno u rezultatu korisnik preusmjerava.

Odjava korisnika odvija se na sličan način pomoću *SignOutAsync* funkcije, te se korisnik preusmjerava na početnu stranicu sa prijavom. Sadržaj *Logout* metode nalazi se na slici 3.14.

```
public async Task<RedirectResult> Logout(string returnUrl = "/")
{
    await _signInManager.SignOutAsync();
    return Redirect(returnUrl);
}
```

Sl. 3.14 *Logout* metoda u *AccountController* klasi

3.2.2. Entity Framework Core

Entity Framwork Core (EF Core) je objektno-relacijski framework za preslikavanje ili ORM (engl. *Object-relational mapping*). ORM framework tablice, stupce i retke relacijske baze podataka predstavlja kao C# objekte pomoću LINQ-a.

LINQ (engl. *Language-Integrated Query*) je dio .NET frameworka koji omogućava pisanje upita na bazu podataka pomoću sintakse sličnije C# programskom jeziku. EF pretvara LINQ u SQL (engl. *Structured Query Language*) koji se onda izvršava na bazi podataka. Kada se SQL izvrši, EF odgovor na upit pretvara u rezultate koji se mogu dohvaćati kroz modele.

Context klasa baze podataka je „most“ između aplikacije i EF Core. Ona omogućuje pristup podacima aplikacije i korištenje modela, možemo reći da je izvedeni *context* zapravo sesija sa bazom podataka. Prikaz *Context* klase nalazi se na slici 3.15.

```
public class ApplicationDbContext : IdentityDbContext<ShiftManager>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) :
                                        base(options){ }
    public DbSet<Employee> Employees { get; set; }
    public DbSet<Shift> Shifts { get; set; }
}
```

Sl. 3.15 *ApplicationDbContext* klasa

ConnectionString specificira lokaciju i ime baze podataka, nalazi se u *appsettings.json* datoteci. Za izradu projekta korištena je lokalna SQL baza podataka koju pruža *Visual Studio*. Za korištenje *ConnectionStringa* korišten je konstruktor u *Startup* klasi prikazan na slici 3.16.

```
IConfigurationRoot Configuration;

public Startup(IHostingEnvironment env)
{
    Configuration = new ConfigurationBuilder()
        .SetBasePath(env.ContentRootPath)
        .AddJsonFile("appsettings.json").Build();
}
```

Sl. 3.16 Konstruktor *Startup* klase

Untar *ConfigureServices* metode u *Startup* klasi nalazi se *AddDbContext* metoda koja postavlja servise EF Core za *ApplicationDbContext* klasu.

Pomoću modela je izvedeno dohvaćanje podataka. Model se sastoji od entitetskih klasa, u ovom projektu to su *ShiftManager*, *Employee* te *Shift* klase. Model je moguće generirati iz postojeće

baze podataka, ili kao u projektu gdje su prvo napisani kodovi za modele i zatim odrađene migracije u bazu podataka.

ShiftManager model, odnosno model voditelja smjene, prikazan na slici 3.17 nasljeđuje IdentityUser klasu koja sadržava nekoliko svojstava važnih za korisnika aplikacije kao što je elektronska pošta, lozinka zaštićena kriptografskom *hash* funkcijom i slično.

```
public class ShiftManager : IdentityUser
{
    public string CompanyName { get; set; }
    public virtual ICollection<Employee> Employees { get; set; }
}
```

Sl. 3.17 *ShiftManager* model

Svojstvo *Employees* unutar *ShiftManager* modela označava da voditelj smjene ima više zaposlenika, ta veza bit će prikazana kao jedan na više unutar SQL koda.

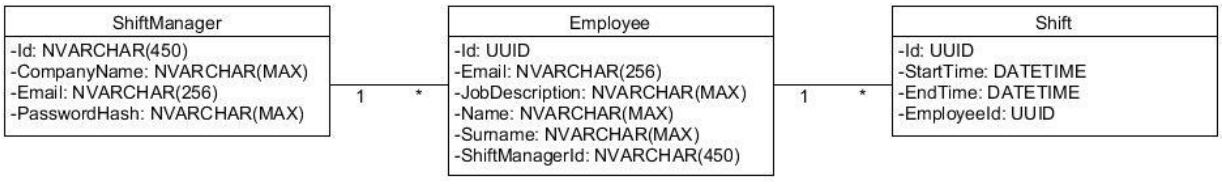
Employee model, odnosno model zaposlenika, prikazan je na slici 3.18.

```
public class Employee
{
        public Guid Id { get; set; }
        public string Email { get; set; }
        public string Name { get; set; }
        public string Surname { get; set; }
        public string JobDescription { get; set; }
        public string ShiftManagerId { get; set; }

        public virtual ShiftManager ShiftManager { get; set; }
        public virtual ICollection<Shift> Shifts { get; set; }
}
```

Sl. 3.18 *Employee* model

Uz svojstva specifična za zaposlenika nalazi se i svojstvo *ShiftManagerId* koji će biti povezan sa modelom *ShiftManager* kao strani ključ na strani *Employee*. Zaposlenik također ima više smjena i zato ima svojstvo *Shifts*, na sličan način kao što voditelj smjene ima više zaposlenika. Prikaz strukture tablica i njihovih odnosa prikazan je UML dijagramom na slici 3.19.



Sl. 3.19 UML dijagram tablica

Za izgradnju svih modela korištene su naredbe za migraciju u konzoli. Na primjer za migraciju *Employee* modela naredba izgleda ovako: *dotnet ef migrations add AddEmployees*. Nakon izvršavanja naredbe kreirana je datoteka unutar *Migrations* mape. U datoteci migracije nalazi se kod koji je automatski generiran pomoću EF, služi za kreiranje tablice, dodavanje ograničenja, brisanje tablice i slično. Dijelovi koda nalaze se na slici 3.20.

```
migrationBuilder.CreateTable(
                name: "Employee",
                columns: table => new
                {
                    Id = table.Column<string>(nullable: false),
                    Email = table.Column<string>(nullable: true),
                     // ...
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Employee", x => x.Id);
                    table.ForeignKey(
                        name: "FK_Employee_AspNetUsers_ShiftManagerId",
                        column: x => x.ShiftManagerId,
                        // ...
                        );
                });
```

Sl.3.20. Migracijska datoteka za *AddEmployees* migraciju

Nakon primjenjivanja migracije pomoću naredbe u konzoli: d*otnet ef database update.* EF prevodi migraciju u SQL jezik kako bi kreirao tablicu, prikaz koda za *Employee* model nalazi se na slici 3.21. EF prema konvenciji postavlja ime tablice u množinu, te također prema konvenciji prepoznaje u svojstvima modela *Id* koji postavlja kao primarni ključ.

```
CREATE TABLE [dbo].[Employees] (
    [Id]             NVARCHAR (450) NOT NULL,
    [Email]          NVARCHAR (MAX) NULL,
    [JobDescription] NVARCHAR (MAX) NULL,
    [Name]           NVARCHAR (MAX) NULL,
    [ShiftManagerId] NVARCHAR (450) NULL,
    [Surname]        NVARCHAR (MAX) NULL,
    CONSTRAINT [PK_Employees] PRIMARY KEY CLUSTERED ([Id] ASC),
    CONSTRAINT [FK_Employees_AspNetUsers_ShiftManagerId] FOREIGN KEY
([ShiftManagerId]) REFERENCES [dbo].[AspNetUsers] ([Id])
);
```

Sl.3.21 SQL kod za kreiranje *Employees* tablice

3.2.3. Model

Unutar *Models*, *ViewModels* i *Repositories* mapa nalaze se sve klase potrebne za kreiranje entiteta voditelja smjene, zaposlenika i njegovih smjena, te logiku za kreiranje, upravljanje i mijenjanje

tih podataka uz korištenje ubrizgavanja ovisnosti ili DI (engl. *Dependency Injection*). Sadržaj ovih mapa prikazan je na slici 3.22.

```
Common
├── IEmployeeRepository.cs
└── IShiftRepository.cs
Data
├── ApplicationDbContext.cs
└── MappingProfile.cs
Models
├── Employee.cs
├── Shift.cs
└── ShiftManager.cs
Repositories
├── EmployeeRepository.cs
└── ShiftRepository.cs
ViewModels
├── LoginViewModel.cs
├── RegisterViewModel.cs
└── ShiftViewModel.cs
```

Sl. 3.22 Sadržaj mapa *Common*, *Data*, *Models* i *Repositories*

Mapa *Models* sadrži definirane klase koje opisuju entitete koji se nalaze u bazi podataka, korišteni su za kreiranje tablica i veza između njih.

Mapa *ViewModels* sadrži modele koji se koriste za prihvaćanje ili slanje podataka određenom pogledu. Rijetko kad se koriste oni isti modeli koji su predložak za tablice nekoj bazi podataka. Na ovaj način u *ViewModel* moguće je dodati *DataAnotations* koji se nalazi u *namespaceu System.ComponentModel.DataAnnotations.* Primjer *ViewModela* dan je na slici 3.23.

```
public class RegisterViewModel
{
    [Required(ErrorMessage = "Obavezno polje")]
    [EmailAddress]
    public string Email { get; set; }

    [Required(ErrorMessage = "Obavezno polje")]
    [DataType(DataType.Password)]
    public string Password { get; set; }

    [Required(ErrorMessage = "Obavezno polje")]
    [DataType(DataType.Password)]
    [Compare("Password", ErrorMessage = "Unesene zaporke nisu jednake")]
    public string ConfirmPassword { get; set; }

    [Required(ErrorMessage = "Obavezno polje")]
    public string CompanyName { get; set; }
}
```

Sl. 3.23 Klasa *RegisterViewModel*

RegisterViewModel sadrži svojstva koja su potrebna za registracijsku formu, a to su elektronska pošta, lozinka, ponovljena lozinka i naziv poduzeća. Sva četiri polja u formi su obvezna što se

provjerava u modelu, na taj način nije potrebno to isto provjeravati u pogledu. Lozinka i ponovljena lozinka moraju biti jednake i to provjeravamo *Compare* naredbom.

ViewModeli su preneseni odnosno mapirani u *entity* modele pomoću neke tehnike mapiranja, u projektu korišten je *AutoMapper library*. Unutar klase MappingProfile.cs nalazi se dio programskog koda koji omogućuje prevođenje jednog modela u drugi. Sadržaj klase prikazan je na slici 3.24.

```
public class MappingProfile : Profile
{
    public MappingProfile()
    {
        CreateMap<Shift, ShiftViewModel>().ReverseMap();
    }
}
```

Sl. 3.24 Klasa *MappingProfile*

3.2.4. Dependency Injection

Dependency Injection je tehnika koja omogućava fleksibilnost aplikacijama. Kada je aplikacija dizajnirana da prati DI tehniku, ona ima nekoherentne komponente (engl. *loosely coupled*). Takav način prati *Dependency Injection* načelo koji kaže da moduli više razine ne bi trebali ovisiti o modulima niže razine nego bi trebali ovisiti o apstrakcijama. Umjesto korištenja specifične implementacije klase traže apstrakcije (najčešće *interface*) koji su omogućeni kada je klasa konstruirana. [14]

U *Startup* klasi nalaze se podaci o klasama koje su korištene za određene funkcionalnosti specificirane *interfaceima*. Prikaz potrebnog koda za projekt nalazi se na slici 3.25.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
       options.UseSqlServer(Configuration["Data:BlueSunData:ConnectionString"]));
    services.AddIdentity<ShiftManager, IdentityRole>()
       .AddEntityFrameworkStores<ApplicationDbContext>();
    services.AddTransient<IEmployeeRepository, EmployeeRepository>();
    services.AddTransient<IShiftRepository, ShiftRepository>();
    services.AddMvc();
}
```

Sl. 3.25. *ConfigureServices* metoda *Startup* klase

Trenutno u projektu *IEmployeeRepository* je *interface* odnosno nekakav ugovor, sa zadanim funkcionalnostima, prikazanim na slici 3.26, a te funkcionalnosti trenutno izvodi

EmployeeRepository klasa. Upravitelj *EmployeeController* koriste funkcije navedene u *interfaceu* ali nisu snažno povezani sa *EmployeeRepository* klasom.

```
public interface IEmployeeRepository
{
        IQueryable<Employee> GetEmployees(string shiftManagerId);
        Employee GetEmployeeById(Guid employeeId);
        void SaveEmployee(Employee employee);
        void DeleteEmployee(Guid employeeId);
}
```

Sl. 3.26 *IEmployeeRepository interface*

EFEmployeeRepository ima definirane metode koje su navedene u *IEmployeeRepository interfaceu.* Prikaz metode *SaveEmployee* korištene za spremanje zaposlenika nalazi se na slici 3.27.

```
public void SaveEmployee(Employee employee)
{
    if (employee.Id == null)
    {
        context.Employees.Add(employee);
    }
    else
    {
        Employee dbEntry = context.Employees.FirstOrDefault(p => p.Id == employee.Id);
        if (dbEntry != null)
        {
            dbEntry.Name = employee.Name;
            dbEntry.Surname = employee.Surname;
            dbEntry.JobDescription = employee.JobDescription;
        }
    }
    context.SaveChanges();
}
```

Sl. 3.27 Metoda za spremanje zaposlenika u *EmployeeRepository* klasi

Varijabla *context* sadržava sesiju sa bazom podataka. Prikazana funkcija sprema novi redak u tablicu *Employees* ukoliko je primila podatke o novom zaposleniku, a inače radi izmjene na postojećim recima u tablici.

Metoda koja dohvaća podatke o određenom zaposleniku zove se *GetEmployees* a prikazana je na slici 3.28.

```
public IQueryable<Employee> GetEmployees(string shiftManagerId)
{
       return context.Employees.Where(e => e.ShiftManagerId ==
       shiftManagerId).OrderBy(e => e.Surname);
}
```

Sl. 3.28 Metoda za dohvaćanje zaposlenika u *EmployeeRepository* klasi

GetEmployees metoda kao parametar prima *Id* trenutno prijavljenog voditelja smjene. Pomoću LINQ upita na bazu podataka dohvaćaju se svi zaposlenici zadanog voditelja smjene te se spremaju u listu koja se dalje koristi u *EmployeeControlleru*.

3.2.5. Detaljan opis MVC obrasca kroz primjer

Dohvaćanje, spremanje i uređivanje smjena zaposlenika odvija se na posebnom pogledu, upravitelju i repositoriju. *Entity* model koji se koristi za opis tablice koja se nalazi u bazi podataka spremljen je u *Shift.cs* datoteku u *Models* mapi i vidljiv je na slici 3.29.

```
public class Shift
{
    public Guid Id { get; set; }
    public DateTime StartTime { get; set; }
    public DateTime EndTime { get; set; }
    public Guid EmployeeId { get; set; }
    public virtual Employee Employee { get; set; }
}
```

Sl. 3.29 *Shift* model

ShiftRepository koristi *Shift* model za spremanje, dohvaćanje i mijenjanje smjena svakog zaposlenika. *ShiftRepository* nasljeđuje *IShiftRepository interface* i implementira zadane metode, sadržaj *IShiftRepository interfacea* nalazi se na slici 3.30.

```
public interface IShiftRepository
{
    IQueryable<Employee> GetEmployees(string shiftManagerId);
    Employee GetEmployeeById(Guid employeeId);
    Shift GetShift(Guid employeeId, DateTime date);
    void SaveShift(Shift shift);
}
```

Sl. 3.30 *IShiftRepository* interface

Kako bi *ShiftRepository* uspješno komunicirao sa bazom podataka, pomoću *ConstructorInjectiona* on ubrizgava instancu *ApplicationDbContext* klase kao što je vidljivo na slici 3.31.

```
private ApplicationDbContext context;

public ShiftRepository(ApplicationDbContext ctx)
{
        context = ctx;
}
```

Sl. 3.31 *ConstructorInjection* unutar *ShiftRepositorija*

GetShift metoda prikazana na slici 3.32. kao parametre prima Id zaposlenika i datum kako bi za određeni datum i određenog zaposlenika u bazi podataka pronašla traženu smjenu ukoliko postoji.

```
public Shift GetShift(Guid employeeId, DateTime date)
{
    Shift result = context.Shifts
        .Where(e => e.EmployeeId == employeeId)
        .Where(s => s.StartTime.Date == date.Date)
        .Include(e => e.Employee)
        .FirstOrDefault();
    if (result == null)
    {
        result = new Shift();
        result.StartTime = date;
        result.EndTime = date;
        result.EmployeeId = employeeId;
        result.Employee = context.Employees.Where(e => e.Id == employeeId)
       .FirstOrDefault();
    }
    return result;
}
```

Sl. 3.32 *GetShift* metoda

Pomoću spomenute instance baze podataka moguće je pretraživati bazu podataka gdje se datum i zaposlenikov id poklapaju s onima u bazi podataka, ukoliko ne postoji tražena smjena stvara se nova sa predloženim datumom. Na sličan način odvija se i spremanje smjene u bazu podataka *SaveShift* metodom.

Upravljač *ShiftController* uz *ShiftRepository* koristi i *ShiftViewModel* kako bi podatke obradio i poslao pogledu koji ih koristi. Izgled *ShiftViewModela* prikazan je na slici 3.33. *ShiftViewModel* za razliku od *Shift* modela sadrži nekoliko dodatnih svojstava kao što je *InDb* tipa *bool*, koji određuje nalazi li se model u bazi podataka ili ne, svojstva *st* i *et* tipa *TimeSpan* koriste se za prikaz početka i završetka smjene na pogledu.

```
public class ShiftViewModel
{
    public Guid Id { get; set; }
    public DateTime StartTime { get; set; }
    public DateTime EndTime { get; set; }
    public bool InDb { get; set; }
    public TimeSpan st { get; set; } = new TimeSpan(0, 0, 0);
    public TimeSpan et { get; set; } = new TimeSpan(0, 0, 0);
    public Guid EmployeeId { get; set; }
    public virtual Employee Employee { get; set; }

}
```

Sl. 3.33 *ShiftViewModel*

Pogled odnosno *View* koji se koristi za uređivanje smjena svakog pojedinog zaposlenika prihvaća listu *ShiftViewModela* pomoću koje generira sučelje. *Edit.cshtml* datoteka unutar *Views/Shift* mape prikazuje smjene unutar tablice, dio koda prikazan je na slici 3.34.

```
@for (int i = 0; i < Model.Count; i++)
{
    <tr>
        <td>
            @Model[i].StartTime.DayOfWeek
            <input type="hidden" asp-for="@Model[i].Id" />
            <input type="hidden" asp-for="@Model[i].EmployeeId" />
            <input type="hidden" asp-for="@Model[i].InDb" />
            <input type="hidden" asp-for="@Model[i].StartTime" />
            <input type="hidden" asp-for="@Model[i].EndTime" />
        </td>
        <td>@Model[i].StartTime.ToString("dd.MM.yyyy.")</td>
        <td><input asp-for="@Model[i].st" type="time" /></td>
        <td><input asp-for="@Model[i].et" type="time" /></td>
    </tr>
}
```

Sl. 3.34 Dio koda unutar *Edit.cshtml*

Za svaku pojedinu smjenu unutar liste, prikazuje se u svakom stupcu dan, datum, početno i završno vrijeme smjene za sedam dana u tjednu. Svaku smjenu moguće je izmijeniti i spremiti.

Unutar upravljačkih metoda odvija se pozivanje metoda na repositoriju te mapiranje *Shift* modela u *ShiftViewModel* i obrnuto, kao i spremanje u listu. Korištenje *mapera* prikazano je na slici 3.35. U varijablu *currentShift* koja je tipa *ShiftViewModel* sprema se varijabla *cS* koja je tipa *Shift*.

```
List<ShiftViewModel> shifts = new List<ShiftViewModel>();
for (int i = 0; i< 7; i++)
{
    Shift cS = repository.GetShift(employeeId, startOfWeek.AddDays(i));
    var currentShift = mapper.Map<Shift, ShiftViewModel>(cS);
    if (currentShift.Id == new Guid())
    {
        currentShift.InDb = false;
    }
    else
    {
    currentShift.st = currentShift.StartTime.TimeOfDay;
    currentShift.et = currentShift.EndTime.TimeOfDay;
    currentShift.InDb = true;
    }
    shifts.Add(currentShift);
}
```

Sl. 3.35 Dio koda unutar *ShiftControllera*

Na slici 3.34 prikazano je mapiranje *Shift* modela u *ShiftViewModel*, dohvaćanje podataka iz baze podataka pomoću metoda unutar repositorija, te definiranje podataka svojstvenih za *ShiftViewModel*. Ukoliko je određena smjena pronađena u repositoriju *InDb* svojstvo postavlja se na *true*, te *st* i *et* svojstva poprimaju vrijednosti stvarnog početka odnosno završetka smjene, u suprotnom se *InDb* svojstvo postavlja na *false*. Podaci se spremaju u listu koja se šalje na pogled.

Za spremanje podataka sa pogleda u bazu podataka koristi se *HttpPost Edit* metoda *ShiftControllera*, koja svaki *ShiftViewModel* iz liste mapira u *Shift* model te poziva *SaveEmployee* metodu *Shift* repozitorija što je prikazano na slici 3.6.

```
foreach (var shift in shifts)
{
    if (shift.st<shift.et)
    {
        shift.StartTime = shift.StartTime.Date + shift.st;
        shift.EndTime = shift.EndTime.Date + shift.et;
        repository.SaveShift(mapper.Map<ShiftViewModel, Shift>(shift));
    }
}
```

Sl. 3.36 Dio koda unutar *ShiftControllera*

4. ZAKLJUČAK

Izradu web aplikacije olakšao je MVC obrazac dostupan kroz *ASP.NET Core framework*. Podjelom zadataka na tri dijela (model, pogled i upravitelj) omogućena je slojevita dogradnja aplikacije. Zbog uporabe *Entity Framework Core* i lokalne baze podataka nije bilo potrebno ručno konfigurirati bazu podataka već je ona bila odmah dostupna za korištenje. Aplikacija je sigurna, nije moguće pristupiti korisnicima te podacima o zaposlenicima i njihovim smjenama bez prethodne prijave što je omogućeno korištenjem *Identity* sustava koji pruža ASP.NET Core. Korištenjem *Bootstrap frameworka* za izradu korisničkog sučelja aplikacije uspješno je i brzo realizirana mogućnost korištenja aplikacije i preko mobilnih uređaja koji imaju manji zaslon od monitora računala.

Prijedlog za dogradnju aplikacije kako bi se poboljšalo korisničko iskustvo je mogućnost prijave svakog zaposlenika kako bi imao uvid u svoj raspored, a to se može implementirati *Identity* sustavom. Korištenjem *ASP.NET Core frameworka* na jednostavan način moguće je implementirati autentikaciju korištenjem vanjskih korisničkih računa što bi pojednostavilo postupak registracije i prijave. Kako bi se dodatno poboljšalo korisničko iskustvo potrebno je još uvesti validaciju za sve forme gdje korisnik unosi podatke, također korištenjem *javascripta*, *ajax* poziva ili *AngularJS*a moguće je poboljšati i pristupačnost aplikaciji.

LITERATURA

[1] Rick-Anderson, „Introduction to ASP.NET Core“. [Na internetu]. Dostupno na: https://docs.microsoft.com/en-us/aspnet/core/. [Pristupljeno: 07-lip-2017].

[2] Rick-Anderson, „ASP.NET Core fundamentals“. [Na internetu]. Dostupno na: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/. [Pristupljeno: 11-lip-2017].

[3] „Open Web Interface for .NET (OWIN)“. [Na internetu]. Dostupno na: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/owin. [Pristupljeno: 11-lip-2017].

[4] „NuGet Gallery | Home“. [Na internetu]. Dostupno na: https://www.nuget.org/. [Pristupljeno: 11-lip-2017].

[5] E. Gamma, R. Helm, R. Johnson, i J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994.

[6] „Trygve/MVC“. [Na internetu]. Dostupno na: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html. [Pristupljeno: 12-lip-2017].

[7] „Simple Example of MVC (Model View Controller) Design Pattern for Abstraction - CodeProject“. [Na internetu]. Dostupno na: https://www.codeproject.com/Articles/25057/Simple-Example-of-MVC-Model-View-Controller-Design. [Pristupljeno: 12-lip-2017].

[8] A. FREEMAN, *Pro ASP.NET Core MVC*. Apress, 2016.

[9] „Introduction to ASP.NET Web Programming Using the Razor Syntax (C#)“. [Na internetu]. Dostupno na: https://docs.microsoft.com/en-us/aspnet/web-pages/overview/getting-started/introducing-razor-syntax-c. [Pristupljeno: 13-lip-2017].

[10] Rick-Anderson, „Tag Helpers in ASP.NET Core“. [Na internetu]. Dostupno na: https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro. [Pristupljeno: 13-lip-2017].

[11] M. Campbell, „Random Name Generator“, *Behind the Name*. [Na internetu]. Dostupno na: https://www.behindthename.com/random/?number=1&gender=f&surname=&randomsurname=yes&all=no&usage_cro=1. [Pristupljeno: 14-lip-2017].

[12] J. Spurlock, *Bootstrap: Responsive Web Development*. O'Reilly Media, Inc., 2013.

[13] Rick-Anderson, „Introduction to Identity“. [Na internetu]. Dostupno na: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity. [Pristupljeno: 16-lip-2017].

[14] „Dependency Injection in ASP.NET Core“. [Na internetu]. Dostupno na: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection. [Pristupljeno: 24-svi-2017].

SAŽETAK

Cilj ovog rada bio je napraviti web aplikaciju koja omogućuje izradu rasporeda smjena. Korisnici aplikacije su voditelji smjene koji imaju mogućnost prijave odnosno registracije. Omogućeno je dodavanje, mijenjanje i brisanje zaposlenika kao i njihovih smjena. Informacije se pohranjuju u bazu podataka. Korisnik aplikacije može kontinuirano mijenjati i pregledavati unesene smjene. Web aplikacija izrađena je prema MVC arhitekturi na ASP.NET Core *frameworku*, te je napisana C# programskim jezikom, a podaci se pohranjuju u MS SQL bazu podataka.

Ključne riječi: ASP.NET, MVC, C#, web aplikacija, Entity Framework Core, Razor, SQL.

ABSTRACT

ASP.NET Web Application for Shift Scheduling

The aim of this paper was to create a web application that allows creating a shift schedule. Application users are shift managers who have the option of signing in or registering. It is possible to add, modify or delete employees as well as their shifts. The information is stored in a database. The application user can continually modify and browse the entered shifts. The web application was created according to the MVC architecture on the ASP.NET Core Framework and was written in C# programming language, the data is stored in an MS SQL database.

Keywords: ASP.NET, MVC, C#, web application, Entity Framework Core, Razor, SQL.

ŽIVOTOPIS

Martina Grgić rođena je 15. prosinca 1995. godine u Osijeku, Hrvatska. Srednju školu, III. Gimnaziju, završava 2014. godine u Osijeku. Iste godine upisuje sveučilišni preddiplomski studij Računarstvo na Elektrotehničkom Fakultetu u Osijeku.

Martina Grgić