Izrada web-aplikacija na platformi ASP.NET

Žinić, Ana Maria

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Ana Maria Žinić

IZRADA WEB-APLIKACIJA NA PLATFORMI ASP.NET

Diplomski rad

Voditelj rada:
doc. dr. sc. Zvonimir Bujanović

Zagreb, rujan, 2017

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Mojoj mami, najvećoj potpori i inspiraciji

Sadržaj

Uvod

U sklopu Microsoftove platforme .NET važno mjesto imaju i web-aplikacije. Njima je namijenjen razvojni okvir ASP.NET. Ovaj okvir omogućava izgradnju skalabilnih web-aplikacija, koristeći pritom industrijske standarde poput arhitekturalnog uzorka Model-View-Controller unutar .NET okruženja.

Cilj ovog diplomskog rada je napraviti pregled platforme ASP.NET MVC 5, te opisati proceduru izgradnje web-aplikacije pomoću nje, kako s klijentske, tako i sa serverske strane. Sukladno tome bit će prikazan proces izrade složenije web-aplikacije uz pomoć navedene platforme.

U prvom poglavlju objasnit ćemo pojmove usko vezane uz temu rada, te proći kroz korake koji objašnjavaju kako postići osnove funkcionalnosti jednostavne aplikacije banke krvi. U drugom poglavlju bit će dan detaljniji uvid u razvijenu složeniju aplikaciju za donacije. Aplikacija za donacije daje mogućnost korisnicima da pokrenu kampanju za skupljanje novaca za određenu svrhu (za ljude, životinje ili zajednicu kojoj su donacije potrebne). Istovremeno je svaki korisnik u mogućnosti podržati kampanje donacijom vlastitih novčanih sredstava.

Poglavlje 1

Razvojni okvir ASP.NET MVC 5

ASP.NET MVC je razvojni okvir za izradu web-aplikacija koji primjenjuje generalni Model-View-Controller uzorak na ASP.NET okvir.

1.1 MVC

Model-View-Controller (MVC) važan je arhitekturalni uzorak u računarstvu već duži broj godina. Nastao je 1979. godine pod izvornim nazivom *Thing-Model-View-Editor* i kasnije je pojednostavljen na *Model-View-Controller*.

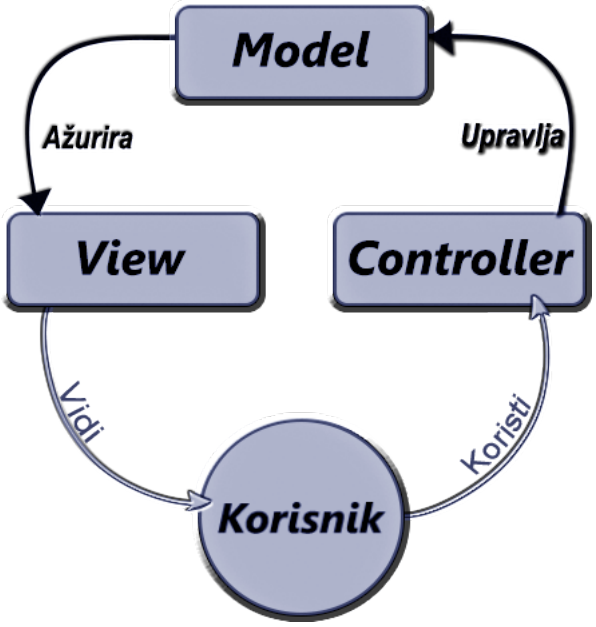
MVC predstavlja moćno i elegantno sredstvo za razdvajanje problema unutar aplikacije (primjerice, razdvaja logiku pristupanja podacima od logike pogleda), što se pokazuje izuzetno primjenjivo na web aplikacije. Njegovo eksplicitno razdvajanje sredstava povećava složenost dizajna aplikacije, no izvanredne prednosti nadjačavaju dodatni trud. MVC se može pronaći u *Javi* i *C++*, na *Mac-u* i na *Windowsima*, kao i unutar stotine drugih okvira. Prvenstveno se primjenjuje kod izrade grafičkih korisničkih sučelja (engl. *Graphical User Interface - GUI*) u klasičnim desktop aplikacijama.

MVC razdvaja korisničko sučelje (engl. *User Interface - UI*) aplikacije na 3 glavna aspekta:

1. **Model**: Skup klasa koje opisuju podatke s kojima radimo, kao i aplikacijska logika kako vršiti izmjenu i manipulaciju podacima.

2. **Pogled** (engl. *View*): Definira kako će UI aplikacije biti prikazan.

3. **Upravitelj** (engl. *Controller*): Skup klasa koji se brine o komunikaciji s korisnikom, sveukupnom toku aplikacije i specifičnoj logici aplikacije.



Slika 1.1: MVC shema

Puno više detalja o uzorku MVC i načinu njegove implementacije unutar okvira ASP.NET dat ćemo u narednim cjelinama.

1.2 ASP.NET

ASP.NET je okvir otvorenog koda (engl. *open source*) za izradu web-aplikacija sa serverske strane, dizajniran za razvoj dinamičkih web-stranica. Razvio ga je *Microsoft*, te je prva verzija *.NET Frameworka* objavljena 2002. godine. To je bio nasljednik *Active Server Pages* (ASP) tehnologije. ASP.NET je kreiran prema *Common Language Runtime* (CLR), dozvoljavajući programerima da pišu ASP.NET kod koristeći bilo koji podržan .NET jezik.

Neke od glavnih značajki razvijenih u ASP.NET-u čine UI pomoćnici (engl. *UI helpers*) s automatskim generiranjem koda za manipulaciju bazom podataka (engl. *scaffolding*) i prilagodljivim predlošcima, HTML pomoćnici (sučelja za olakšanu izradu UI), podrška za razdvajanje velikih aplikacija u manje dijelove, podrška za asinkrone upravitelje, *Razor engine* za pogled. Bitne značajke dodane s izdanjem ASP.NET MVC 5 su *ASP.NET Identity*, *One ASP.NET*, predlošci *Bootstrap*, ASP.NET *Scaffolding* te filteri za autentifikaciju.

One ASP.NET

Od MVC verzije 5 postoji samo jedna vrsta ASP.NET projekta. Pri samom kreiranju projekta više nije bitno donositi ključne odluke. Ranije je slučaj bio da se pri stvaranju projekta moralo opredijeliti za stvaranje MVC aplikacije, *Web Forms* aplikacije ili nekog drugog tipa projekta.

ASP.NET *Identity*

ASP.NET *Identity* predstavlja sustav koji omogućava kontrolu nad podacima korisnika. Također je vrlo jednostavno dodati nove informacije o korisniku. Osim implementiranih klasa za prijavu i registraciju korisnika, ASP.NET *Identity* razumije da su korisnici često puta autentificirani putem društvenih pružatelja usluga (primjerice *Microsoft Account*, *Facebook*, *Twitter*).

Predlošci *Bootstrap*

Okvir *Bootstrap* kreiran je od strane programera i dizajnera u *Twitteru*, koji su kasnije tamo prestali raditi kako bi se u potpunosti mogli posvetiti *Bootstrapu*. Od MVC verzije 5, ASP.NET vizualno sučelje bazirano je upravo na *Bootstrapu*, koji omogućava jednostavnu i moćnu manipulaciju dizajnom aplikacije.

ASP.NET *Scaffolding*

Scaffolding je proces koji ubrzava generiranje koda, baziran prema postojećim klasama u modelu. Više detalja o tome biti će spomenuto u sljedećim poglavljima.

MVC unutar ASP.NET-a

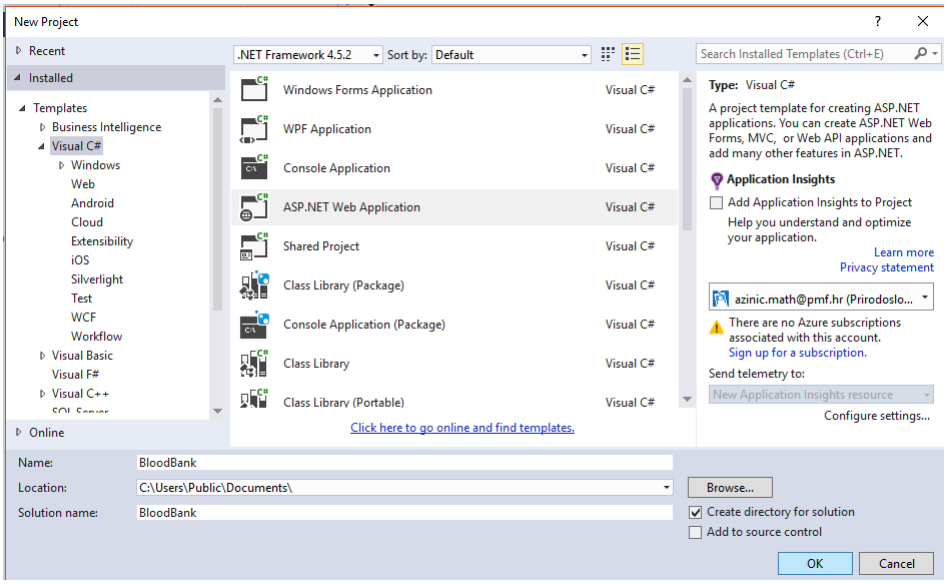
Uzorak MVC često je korišten u web-programiranju. Unutar ASP.NET MVC, **modele** čine klase koje predstavljaju domenu za koju smo zainteresirani. Ti objekti domene reprezentiraju podatke spremljene u bazu podataka, ali i kod koji manipulira podacima i nameće za domenu specifičnu poslovnu logiku. U okviru ASP.NET MVC, to će najčešće biti sloj za pristup podacima određenog tipa, uz pomoć alata poput *Entity Frameworka*. **Pogled** predstavlja predložak koji dinamički kreira *HTML*. **Upravitelje** čine specijalne klase koja upravljaju vezom između pogleda i modela. One odgovaraju na korisnički unos, komuniciraju s modelom, te odlučuju koji pogled prikazati.

1.3 Stvaranje ASP.NET MVC 5 aplikacije

MVC 5 aplikaciju možemo stvoriti koristeći Visual Studio 2013 ili noviju verziju. U daljnjim poglavljima, kroz razvijanje jednostavne web-aplikacije koja daje uvid u banku donacija krvi, bit će objašnjeni pojmovi i konvencije.

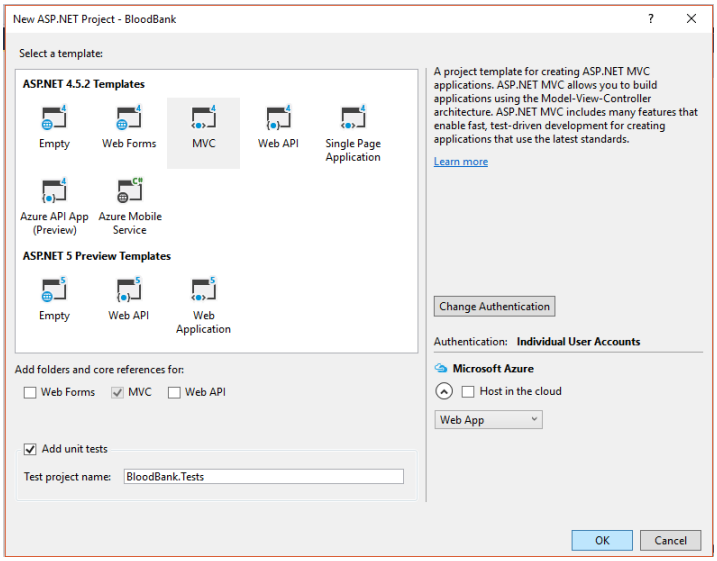
Kako bismo stvorili novu ASP.NET MVC 5 aplikaciju, unutar glavnog izbornika potrebno je izabrati (vidi Sliku 1.2):

File ⇒ *New* ⇒ *Project*

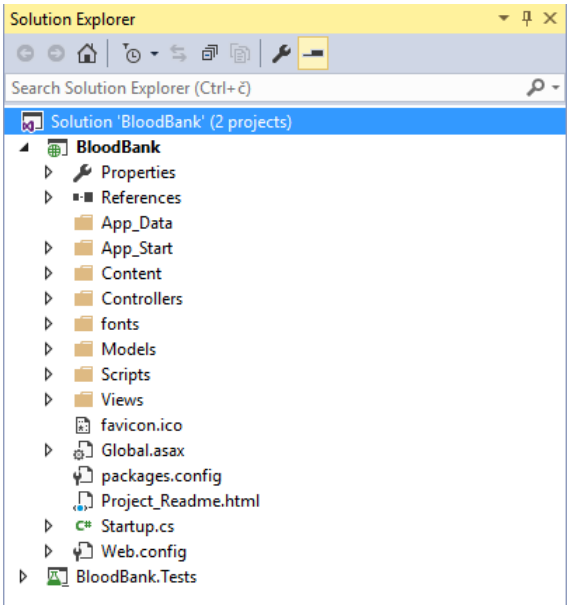


Slika 1.2: Stvaranje novog projekta

Zatim je potrebno izabrati *ASP.NET Web Application*. Pod *Name* napišemo željeno ime aplikacije, dok pod *Location* odaberemo željeno mjesto gdje će se naš projekt pohraniti. Odredišnu lokaciju mijenjamo klikom na *Browse*. Naposlijetku potvrdimo klikom na *OK*. Nakon toga nam se otvori novi prozor u kojemu pod *Select a template* odaberemo *MVC*. Kod opcije *Add unit tests* stavimo kvačicu. Potvrdimo klikom na *OK*. *Visual Studio* nakon toga stvara prazan projekt (vidi Sliku 1.3).



Slika 1.3: Stvaranje novog projekta (nastavak)



Slika 1.4: Solution Explorer

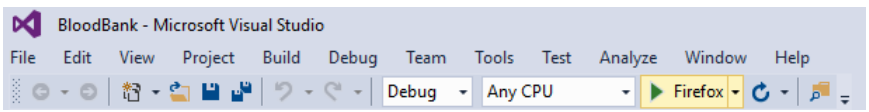
1.4 Struktura MVC aplikacije

Za početak će biti objašnjeno što se sve nalazi unutar novostvorenog ASP.NET MVC 5 projekta. Klikom u glavnom izborniku na *View* ⇒ *Solution Explorer*, Visual Studio će otvoriti cjelokupno rješenje projekta (engl. *project solution*).

Ako zavirimo unutar definiranih direktorija, lako je vidjeti da je u njima pohranjeno dosta već predefiniranih datoteka. Te dane datoteke pružaju osnovnu strukturu za razvijanje aplikacije, zajedno sa početnom stranicom (engl. *homepage*), "o nama" stranicom (engl. *about page*), stranicama za prijavu, odjavu i registraciju korisnika, te stranicom za nepredviđene pogreške (engl. *unhandled error page*) (za više detalja vidi Tablicu 1.1).

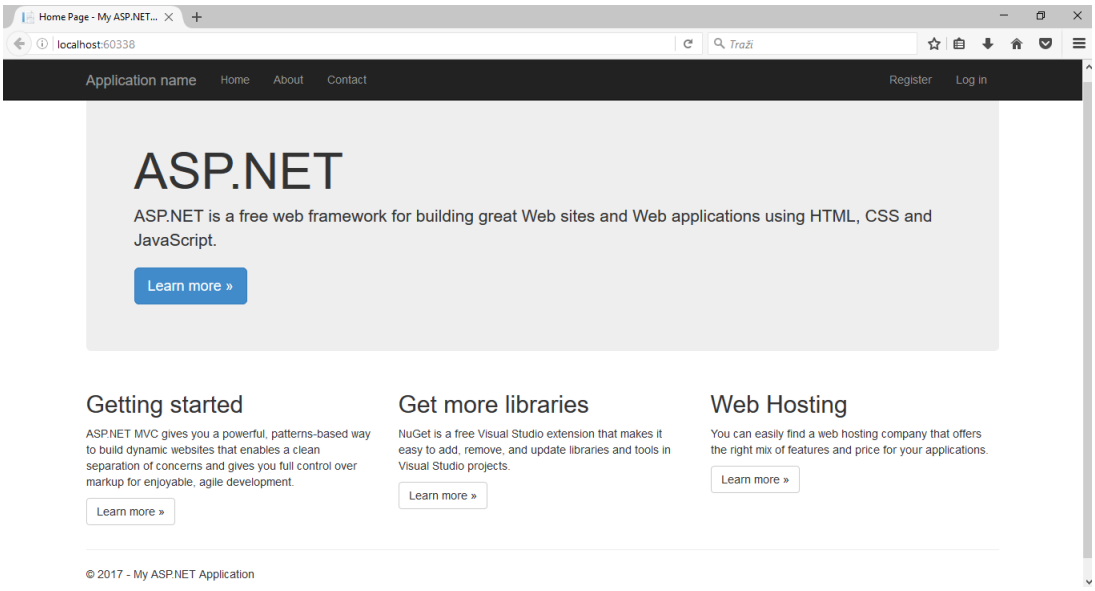
Tablica 1.1: Struktura novostvorenog projekta

Naziv direktorija	Opis direktorija
/Controllers	Direktorij u koji pohranjujemo klase upravitelja koji obrađuje URL zahtjeve.
/Models	Direktorij u kojem se nalaze klase koje predstavljaju modele i upravljaju podacima i poslovnim objektima.
/Views	Direktorij u koji stavljamo datoteke UI predložaka koje su odgovorne za prikazivanje izlaznih podataka (poput HTML koda).
/Scripts	Mjesto gdje pohranjujemo *JavaScript* biblioteke i skripte pisane u JavaScriptu.
/fonts	Predlošci *Bootstrap* uključuju određene prilagođene web fontove, koji su naposlijetku spremljeni u ovaj direktorij.
/Content	Mjesto za spremanje CSSa, slika i sličnog sadržaja, koji pritom isključuje skripte.
/App_Data	Ovdje se pohranjuju datoteke iz kojih čitamo i u koje zapisujemo (primjerice baza podataka, XML datoteka itd.).
/App_Start	Direktorij u kojemu se nalazi kod za konfiguraciju aplikacije. Primjerice za *Routing*, *Bundling* i *WebAPI*-je.



Slika 1.5: Pokretanje aplikacije

Pritiskom na ikonu izbornika na kojoj je napisano ime web-preglednika (*Firefox* na Slici 1.5) ili pritiskom na tipku *F5*, možemo pokrenuti aplikaciju. Na Slici 1.6 dan je prikaz njenog početnog izgleda.



Slika 1.6: Aplikacija pokrenuta u pregledniku

1.5 ASP.NET MVC i konvencije

ASP.NET MVC aplikacije izuzetno se oslanjaju na konvencije. To dozvoljava programerima da izbjegnu posao konfiguriranja i specificiranja stvari koje mogu biti standardizirane konvencijom.

Primjerice, u slučaju kada treba prikazati određeni pogled, MVC koristi strukturu nazivanja direktorija baziranu prema konvencijama, a to omogućuje da nije potrebno razmišljati o putu (engl. *route*) do lokacije pri upućivanju na pogled unutar klase upravitelja. Bez ikakve intervencije ili konfiguracije od strane programera, predložak *View* se automatski traži unutar direktorija ***/ Views / [ImeUpravitelja] / imePogleda.cshtml***. Bitno je naznačiti da je sve dane konvencije moguće promijeniti ukoliko za tim ima potrebe.

Neke od istaknutijih konvencija čine:

- Svaka klasa upravitelja završava sa ključnom riječi *Controller*: primjerice *HomeController*.
- Postoji jedinstven direktorij za poglede za cijelu aplikaciju, nazvan *Views*.

- Pogledi koje upravitelji koriste unutar poddirektorija od direktorija *Views* su nazvani prema imenu upravitelja (no bez sufiksa *Controller*). Primjerice, pogled od upravitelja *HomeController* bio bi spremljen unutar direktorija ***/ Views / Home***.

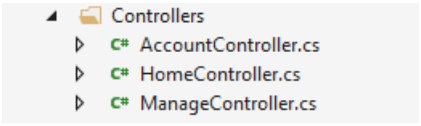
1.6 Upravitelji

Upravitelji unutar obrasca MVC zaduženi su za odgovaranje na unos korisnika, često radeći promjene u modelu kao odgovor na taj unos. U tom smislu, upravitelji se zaduženi za tijek aplikacije: oni obrađuju pristigle podatke i šalju izlazne podatke prema točno određenom pogledu. Umjesto toga da postoji direktna veza između URL-a i datoteke koja je pohranjena na tvrdom disku web servera, imamo vezu između URL-a i metode u klasi od upravitelja.

Pisanje vlastitog upravitelja

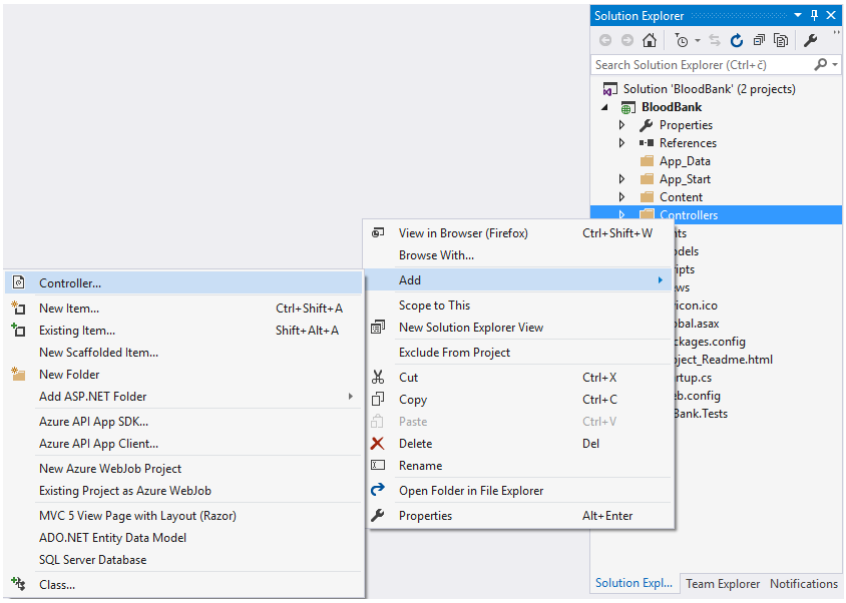
U ovom poglavlju bit će objašnjeno na koji način funkcioniraju upravitelji. Najčešće ćemo za svaki model konstruirati najviše jednog upravitelja, koji nakon toga svaku svoju akcijsku metodu proslijeđuje jednom pogledu. Primjerice, za prikazivanje i manipulaciju korisničkim podacima registriranih korisnika, stvorit ćemo upravitelja *UsersController*, dok ćemo za manipulaciju donacijama krvi stvoriti *BloodDonationsController* koji će sadržavati akcije za dodavanje novih donacija, te također uređivanje i pregled već postojećih donacija.

Kad tek stvorimo novi projekt, možemo uočiti da u direktoriju *Controllers* već postoje određeni upravitelji (vidi Sliku 1.7). *HomeController* prosljeđuje statičan sadržaj naslovne stranice, dok su preostala dva upravitelja, *AccountController* i *ManageController*, zadužena za stvaranje novih korisničkih računa i brigu o sigurnosti već postojećih korisničkih podataka.

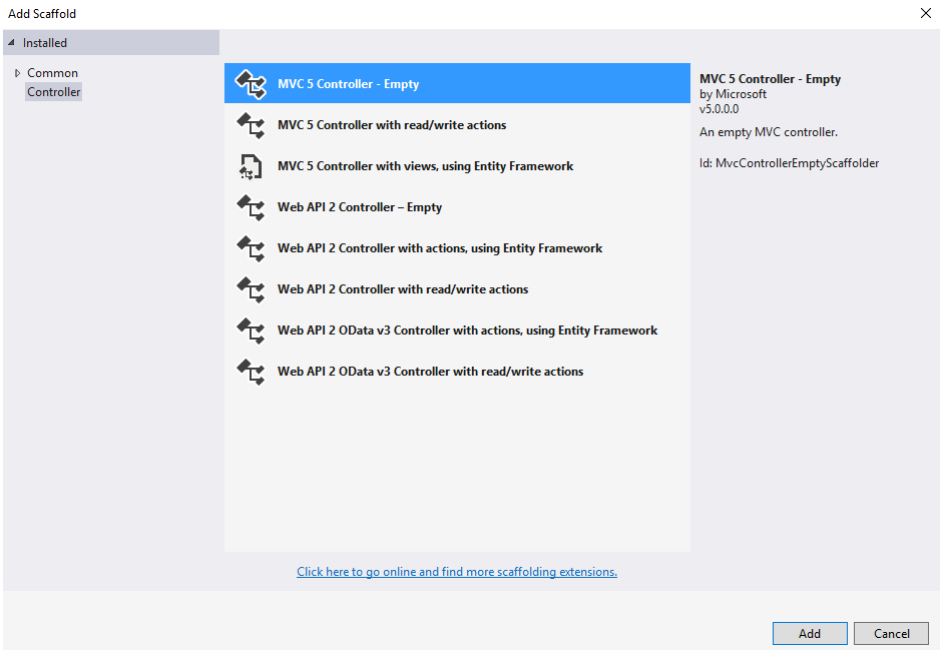


Slika 1.7: Direktorij *Controllers*

Da bismo dodali vlastitog upravitelja, potrebno je u *Solution Exploreru* pritisnuti desnom tipkom miša na direktorij *Controllers*, te nakon toga izabrati *Add* ⇒ *Controller...* (vidi Sliku 1.8). Zatim biramo *MVC 5 Controller - Empty* te potvrđujemo pritiskom na *Add* (vidi Sliku 1.9).

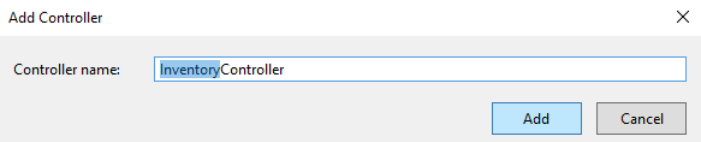


Slika 1.8: Dodavanje upravitelja



Slika 1.9: Dodavanje upravitelja (nastavak)

Naposlijetku upravitelja preimenujemo u *InventoryController* te potvrdimo pritiskom na gumb *Add* (vidi Sliku 1.10).



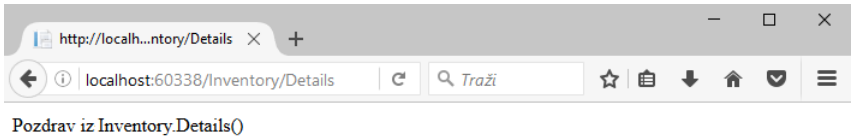
Slika 1.10: Dodavanje upravitelja (nastavak)

Kako bismo dobili bolji osjećaj za način funkcioniranja upravitelja, pokazat ćemo kako proslijeđivanjem određenih parametara, i pozivanjem različitih metoda upravitelja, dobivamo drugačiji odgovor unutar preglednika. U tu svrhu, unutar *InventoryController-a* metodu *Index* promijenimo na način da vraća *string* umjesto *ActionResult*. Zatim napišemo još jednu metodu *Details*, kao što je to prikazano na Slici 1.11.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace BloodBank.Controllers
{
    0 references
    public class InventoryController : Controller
    {
        // GET: Inventory
        0 references
        public string Index()
        {
            return "Pozdrav iz Inventory.Index()";
        }
        // GET: Inventory/Details
        0 references
        public string Details()
        {
            return "Pozdrav iz Inventory.Details()";
        }
    }
}
```

Slika 1.11: Prilagodba *InventoryControllera*

Projekt pokrenemo pritiskom na *F5* te možemo lako vidjeti da ukoliko otvorimo URL ***/Inventory***, u pregledniku će se ispisati string koji je vratila metoda *Index()*, dok će se pri otvaranju URL-a ***/Inventory/Details*** ispisati string koji je vratila metoda *Details()* (vidi Sliku 1.12).



Slika 1.12: Ispis u pregledniku

Iz ovog eksperimenta moguće je izvući nekoliko opažanja:

- Pregledavanje ***/Inventory/Details*** uzrokovalo je da se izvrši metoda *Details* koja pripada *InventoryControlleru*, bez ikakve dodatne konfiguracije.
- Upravitelji su stvoreni iz izuzetno jednostavnih klasa. Jedini način iz kojeg možemo vidjeti da su upravitelji jest to što su naslijeđeni iz *System.Web.Mvc.Controller*.
- Uspješno smo proslijedili tekst pregledniku samo uz pomoć upravitelja, dakle nije nam bio potreban niti model niti pogled. Premda su i modeli i pogledi izuzetno korisni unutar ASP.NET MVC-a, upravitelji su centralni dio svega. Svaki zahtjev prolazi kroz upravitelja, dok neki neće ni trebati koristiti modele ili poglede.

```
// GET: /Inventory/Details?bloodType=A
0 references
public string Details(string bloodType)
{
    string message = HttpUtility.HtmlEncode("Inventory.Details, bloodType= " + bloodType);
    return message;
}
// GET: /Inventory/Details2/5
0 references
public string Details2(int id)
{
    string message = HttpUtility.HtmlEncode("Inventory.Details 2, Id = " + id);
    return message;
}
```

Slika 1.13: Proslijeđivanje parametara metodama upravitelja

Korisno je naznačiti da i upraviteljima možemo proslijeđivati parametere. Ako izmijenimo *Details* kao što je to naznačeno na Slici 1.13, tako da sada prima parametar tipa *string*, te dodamo *Details2* koji prima *int*, vidjet ćemo da će se pri pristupanju URL-ovima ***/Inventory/Details?bloodType=A***, odnosno ***/Inventory/Details2/5***, ispisati unutar preglednika poruke *"Inventory.Details, bloodType = A"*, odnosno *"Inventory.Details 2, Id = 5"*.

Primijetimo da se parametar tipa *int* i naziva *id* ponaša drugačije. To je iznova *routing* konvencija unutar ASP.NET MVC prema kojoj će web-okvir drugačije tretirati parametre

naziva *id*. To je izuzetno pojednostavljenje koje omogućava lakše dohvaćanje određene vrijednosti. Upravo u komunikaciji s bazom podataka *id* igra bitnu ulogu, no više o tome bit će objašnjeno u Poglavlju 1.8.

1.7 Pogledi

Pogledi su ono što posjetitelj stranice vidi. Premda programeri troše puno vremena na razvijanje upravitelja i modela, taj dio posla korisniku nije vidljiv. Ipak se nameće pitanje - ako je moguće proslijediti sav sadržaj uz pomoć upravitelja, koja je svrha pogleda? Odgovor je vrlo jednostavan ako promatramo malo složeniju web aplikaciju. Većina upravitelja treba prikazati dinamičke informacije u HTML formatu. Ukoliko upravitelji vraćaju isključivo stringove, tada će biti potrebno raditi puno zamjena unutar stringa, a to se vrlo brzo može prilično zakomplicirati. Zato nam je potreban sustav koji generira predložak HTML, što je upravo uloga pogleda.

Pogled je zadužen za proslijeđivanje korisničkog sučelja korisniku. Nakon što je upravitelj izvršio prikladnu logiku vezanu uz URL kojem je pristupljeno, proslijeđuje prikaz pogledu. U jednostavnim slučajevima može se dogoditi da pogled treba malo ili uopće ne treba informacije od upravitelja. No ipak se češće događa da upravitelj mora proslijediti informacije pogledu, pa stoga proslijeđuje objekt za prijenos podataka, koji još zovemo model.

```
@{
    ViewBag.Title = "Home Page";
}

<div class="jumbotron">
    <h1>ASP.NET</h1>
    <p class="lead">ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.</p>
    <p><a href="http://asp.net" class="btn btn-primary btn-lg">Learn more &raquo;</a></p>
</div>

<div class="row">
    <div class="col-md-4">
        <h2>Getting started</h2>
        <p>
            ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that
            enables a clean separation of concerns and gives you full control over markup
            for enjoyable, agile development.
        </p>
        <p><a class="btn btn-default" href="http://go.microsoft.com/fwlink/?LinkId=301865">Learn more &raquo;</a></p>
    </div>
    <div class="col-md-4">
        <h2>Get more libraries</h2>
        <p>NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.</p>
        <p><a class="btn btn-default" href="http://go.microsoft.com/fwlink/?LinkId=301866">Learn more &raquo;</a></p>
    </div>
    <div class="col-md-4">
        <h2>Web Hosting</h2>
        <p>You can easily find a web hosting company that offers the right mix of features and price for your applications.</p>
        <p><a class="btn btn-default" href="http://go.microsoft.com/fwlink/?LinkId=301867">Learn more &raquo;</a></p>
    </div>
</div>
```

Slika 1.14: Pogled *Index* od *HomeControllera*

Osnove pogleda

Najbolji način za razumijevanje pogleda jest proučiti već predefinirano stvorene poglede unutar aplikacije. To je primjerice ***/Views/Home/Index.cshtml***. Ako zanemarimo mali dio koda na vrhu stranice, sve ostalo je samo standardni HTML (vidi Sliku 1.14). Promotrimo li metodu *Index()* unutar *HomeControllera* vidimo da je upravitelj izuzetno jednostavan i vraća samo pogled (vidi Sliku 1.15).

```
public ActionResult Index()
{
    return View();
}
```

Slika 1.15: Metoda *Index()* unutar *HomeControllera*

To je upravo i najjednostavniji slučaj u kojemu je, kada korisnik pošalje zahtjev upravitelju, vraćen pogled koji se oslanja samo na statičan HTML. Dobar početak proslijeđivanja podataka od upravitelja do pogleda jest tzv. *ViewBag*. Premda on ima dosta ograničenja, koristan je kada proslijeđujemo malu količinu podataka. Bitno je da je definiran prije nego upravitelj pozove naredbu *return View()*;

Unutar upravitelja definiramo ga na način:

```
ViewBag.Poruka = "Poruka koju proslijedujemo pogledu.";
```

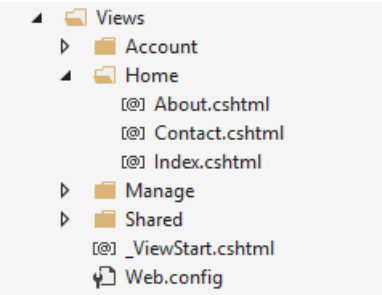
Dok ćemo tu istu vrijednost unutar pogleda dohvatiti pozivom:

```
@ViewBag.Poruka
```

Oznaka @ ispred varijable pripada tzv. *Razor* sintaksi i njezina ja najznačajnija karakteristika. Ona upućuje *Razor* mehanizmu pogleda (engl. *Razor view engine*) da su znakovi koji slijede specijalni kod, a ne HTML tekst. Više o *ViewBagu* biti će spomenuto u kasnijim poglavljima.

Razumijevanje konvencija pogleda

Do sada je promatrano kako koristiti pogled kako bismo prikazali HTML. U ovom dijelu će biti objašnjeno na koji način ASP.NET MVC pronalazi ispravan pogled koji će prikazati te kako zaobići to pravilo kako bismo specificirali proizvoljan željeni pogled kao akciju upravitelja. Nakon stvaranja novog predloška za projekt, možemo primijetiti da predložak također sadrži direktorij *Views* strukturiran na vrlo specifičan način (vidi Sliku 1.16).



Slika 1.16: Početni sadržaj direktorija Views

Unutar direktorija *Views* nalazi se niz direktorija koji nose ista imena kao njihovi pripadajući upravitelji. Svaki od tih direktorija sadrži datoteku pogleda za određenu akcijsku metodu, te je datoteka istog naziva kao i akcijska metoda. Logika odabira pogleda jest da će se tražiti pogled s istim imenom kao akcijska metoda unutar direktorija ***/Views/ControllerName*** (u ovom slučaju to je ime upravitelja bez sufiksa *Controller*). Pogled vraćen u slučaju poziva *Index()* metode unutar *HomeController-a* bio bi ***/Views/Home/Index.cshtml***.

Naravno, tu je konvenciju moguće promijeniti. Primjerice, u slučaju da želimo da metoda *Index* prikazuje drugi pogled, tada možemo proslijediti drugi pogled na način:

```
public ActionResult Index() {

   return View("DrugiPogled");

}
```

U toj će situaciji lokacija pogleda i dalje biti tražena unutar direktorija ***/Views/Home***, no bit će izabran pogled naziva *DrugiPogled.cshtml*. Postoje situacije kad želimo ipak prikazati pogled unutar potpuno drugog direktorija. Tada to radimo koristeći oznaku ~ unutar putanje, koja označava korijenski direktorij web-aplikacije:

```
public ActionResult Index() {

   return View("~/Views/DrugiDirektorij/Index.cshtml");

}
```

Strogo tipizirani pogledi

Postoje tri načina na koja možemo proslijediti informacije od upravitelja prema pogledu:

- strogo tipiziran objekt modela (engl. *Strongly typed model object*)
- dinamička vrsta (koristeći @*model dynamic*)
- koristeći *ViewBag*

ViewBag je koristan u jednostavnim slučajevima proslijeđivanja informacija iz upravitelja prema pogledu, no pri obradi velike količine podataka, on postaje izuzetno nepraktičan. Tu u igru dolaze strogo tipizirani pogledi. Strogo tipiziranje je obilježje koje definira izričito predefiniranje tipova svih varijabli. Dakle, to je metoda kojom možemo za svaki pogled postaviti proizvoljnu vrstu modela. To nam dozvoljava da objekt našeg modela proslijedimo iz upravitelja prema pogledu, te da taj objekt ostane strogo tipiziran u oba slučaja. Poglede kojima prosljeđujemo strogo tipizirane objekte zovemo strogo tipizirani pogledi. Time ostvarujemo prednosti korištenja *IntelliSense-a*, provjere kompajlera i slično. *IntelliSense* je naziv za skup svojstava unutar *Visual Studija*, koja omogućuju da naučimo više o kodu koji pišemo, pratimo parametre koje koristimo, dodamo pozive svojstava i metoda u samo nekoliko pritisaka po tipkovnici.

I strogo tipiziran objekt modela i *ViewBag* su informacije proslijeđene putem *ViewDataDictionary*. Generalno gledajući, svi podaci su uvijek proslijeđeni iz upravitelja prema pogledu putem specijalizirane klase *ViewDataDictionary* naziva *ViewData*. Moguće je postaviti i čitati vrijednosti rječnika *ViewData* na sljedeći način:

```
ViewData["TrenutnoVrijeme"] = DateTime.Now;
```

Važno je napomenuti da je, iako je ovakva sintaksa i dalje dostupna, nakon verzije MVC 3, primarni način za manipulaciju takvim varijablama upravo *ViewBag*. *ViewBag* predstavlja dinamički omotač (engl. *wrapper*) od *ViewData*. Omogućava nam da definiramo tu istu vrijednost "TrenutnoVrijeme" na sljedeći način:

```
ViewBag.TrenutnoVrijeme = DateTime.Now;
```

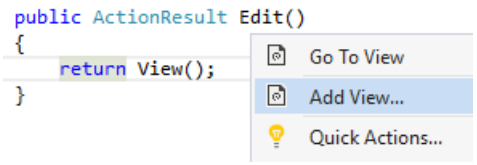
ViewBag.TrenutnoVrijeme jest ekvivalent varijabli *ViewData["TrenutnoVrijeme"]*.
O strogo tipiziranim objektima modela biti će više riječi u Poglavlju 1.8.

View Modeli

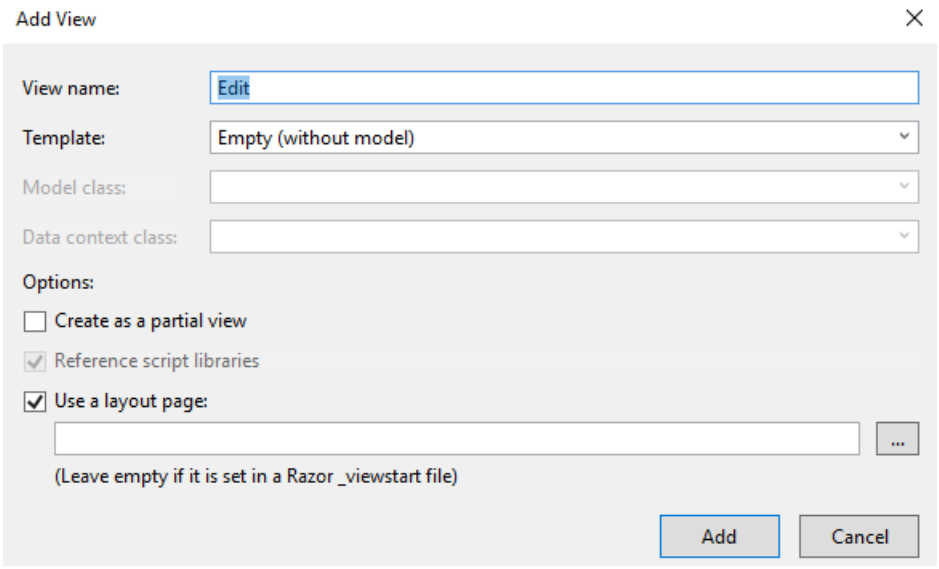
Često se može dogoditi da pogled treba prikazati podatke koji se ne preslikavaju svi u model domene. Primjerice, istovremeno se želi prikazati tablica svih donatora krvi, ali i omogućiti pritom pristup tim podacima isključivo ulogiranom korisniku koji ima administratorska prava. Alternativa ovom pristupu jest proslijeđivanje parametara unutar *ViewBaga*. To je česta praksa u slučajevima kod kojih želimo omogućiti padajući izbornik s fiksnim podacima. Primjerice, želimo omogućiti jednostavan odabir vrste krvi novog donatora, za što uvijek moramo izabrati jednu mogućnost od konačno mnogo ponuđenih.

Dodavanje pogleda

Kao što je opisano dodavanje upravitelja u Poglavlju 1.6, na isti način je moguće dodati i pogled, pritiskom desnom tipkom miša na direktorij *Views*. No ipak, najlakši način za dodavanje pogleda jest tako da otvorimo datoteku željenog upravitelja, te pritisnemo desnom tipkom miša na akcijsku metodu za koju želimo stvoriti pogled. Nakon toga u ponuđenom izborniku odaberemo *Add View*.



Slika 1.17: Dodavanje pogleda za postojeću akcijsku metodu



Slika 1.18: Dodavanje pogleda - nastavak

Obratimo sada pozornost na sadržaj dijaloškog prozora sa Slike 1.18:

View name Ako pokrenemo dijaloški prozor iz konteksta akcijske metode, tada će ime pogleda (engl. *View name*) već biti napisano i biti će jednako imenu akcijske metode iz koje smo otvorili prozor.

Template Nakon odabira vrste predloška (engl. *Template*), također će biti ponuđen odabir predloška modela, i oni će zajedno biti baza za kreiranje pogleda. Taj proces zajedno nazivamo *Scaffolding* (više detalja o svakom predlošku nalazi se u Tablici 1.2).

Reference script libraries Ova opcija daje mogućnost da pogled koji kreiramo uključuje reference na skup *JavaScript* datoteka (ukoliko to ima smisla za pogled). Ono što svakako znamo jest da *_Layout.cshtml* datoteka sadrži referencu na glavnu *jQuery* biblioteku, no ne primjerice i na *jQuery Validation* biblioteku ili *Unobtrusive jQuery Validation* biblioteku.

Create as a partial view Odabirom ove opcije naznačujemo da pogled koji stvaramo ne može biti stvoren u punom pogledu, te je pritom opcija *Layout* onemogućena. Stvoreni djelomični pogled (engl. *Partial View*) vrlo je sličan klasičnom pogledu, no u njemu nećemo pronaći *<html>* ili *<head>* tagove na vrhu pogleda.

Use a layout page Ova opcija odlučuje da li pogled koji će biti stvoren referencira na *_Layout* ili je potpuno samostalna neovisna stranica.

Tablica 1.2: Pojašnjenja *scaffold* tipova

Scaffold vrsta	Opis
Create	Stvara pogled sa obrascem (engl. *form*) za stvaranje novih instanci modela. Također, generira oznaku (engl. *label*) i polje za unos za svako svojstvo iz odabranog tipa modela.
Delete	Stvara pogled sa obrascem za brisanje postojećih instanci modela. Prikazuje oznaku te trenutnu vrijednost za svako svojstvo modela.
Details	Stvara pogled koji prikazuje oznaku te vrijednost svakog svojstva iz tipa modela.
Edit	Stvara pogled sa obrascem za uređivanje postojećih instanci modela. Generira oznaku te polje za unos za svako svojstvo tipa modela.
Empty	Stvara prazan pogled. Jedino što je specificirano jest tip modela, koristeći *@model* sintaksu.
Empty (without model)	Stvara prazan pogled. No kako nema modela, nije ni potrebno odabrati tip modela. To je jedini scaffold tip koji ne zahtijeva od nas da odaberemo tip modela.
List	Stvara pogled s tablicom instanci modela. Stvara tablicu za svako svojstvo modela. Treba obratiti pažnju da se iz akcijske metode prema pogledu proslijedi tip podataka *IEnumerable<TipModela>*. Pogled također sadrži poveznice na akcije za provođenje *create/edit/delete* operacija

Mehanizam pogleda *Razor*

Mehanizam pogleda *Razor* prvi puta je predstavljen sa ASP.NET MVC 3 te je glavni mehanizam pogleda, koji sve više napreduje (opcionalno, ASP.NET nudi i alternativni mehanizam pogleda *WebForms*). *Razor* omogućava pojednostavljenu sintaksu za prikaz pogleda, minimizirajući količinu sintakse i dodatnih znakova. Za razumijevanje je najbolje pogledati primjer na Slike 1.19. Primjer koda koristi C# sintaksu, što znači da datoteka ima ekstenziju *.cshtml*. Ekstenzija je izuzetno bitna jer upućuje kod pisan u C# sintaksi prema *Razor* parseru.

```
@{
    //Ovo je kod za demonstraciju upotrebe Razora
    var namirnice = new string[] { "mlijeko", "jaja", "sol" };
}

<html>
<head>
    <title>Pogled na namirnice</title>
</head>
<body>
    <h1>Lista @namirnice.Length namirnice </h1>
    <ul>
        @foreach(var namirnica in namirnice)
        {
            <li>Potrebno kupiti namirnicu @namirnica.</li>
        }
    </ul>
</body>
</html>
```

Slika 1.19: Primjer *Razor* sintakse

Ključan znak za prijelaz u *Razor* jest znak @. On je korišten za prijelaz iz HTMLa u kod, a nekad i obrnuto. Dva osnovna načina prijelaza koja imamo su prijelaz u *izraz* i u *blok koda*. Ono što *Razor* čini posebno kvalitetnim jest to da su svi izrazi unutar njega HTML enkodirani, te time sprječavaju mogućnost napada na stranicu umetanjem skripti, još poznato kao *XSS* ili *Cross Site Scripting*.

Za primjer promotrimo sljedeći kod:

```
@{
   string poruka = "<script>alert('Stranica je hakirana');</script>";
}
<span>@poruka</span>
```

On neće rezultirati sa alert prozorom s ispisanom porukom, već će generirani HTML kod izgledati ovako:

```
<span>&lt;script&gt;alert(&#39;Stranica je
    hakirana&#39;);&lt;/script&gt;</span>
```

No ipak, ako postoji slučaj u kojem želimo ispisati string točno onako kako je zapisan, tada to možemo napraviti pozivom *Razora* na način:

```
<span>@Html.Raw(poruka)</span>
```

To i dalje ostavlja unos putem *JavaScripta* ranjivim. Taj problem može se riješiti korištenjem

```
@Ajax.JavaScriptStringEncode
```

Primjer koda u kojem koristimo tu naredbu:

```
<script type="text/javascript">
$(function(){
   var poruka = "Pozdrav @Ajax.JavaScriptStringEncode(ViewBag.Username)";
});
</script>
```

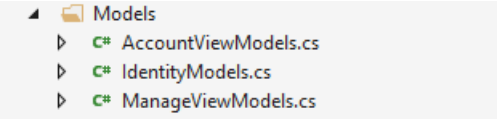
1.8 Modeli

U softverskom inženjerstvu riječ "model" koristi se kako bi definirala veliku količinu različitih koncepata. No u našem kontekstu, kada govorimo o modelu, mislimo na objekte koje koristimo kako bismo proslijedili informaciju bazi podataka, proveli poslovna izračunavanja, te ih prikazali unutar pogleda. Drugim riječima, objekti predstavljaju domenu na koju se se aplikacija fokusira, te modeli čine objekte koje želimo prikazati, spremiti, stvoriti, izmijeniti i izbrisati. Za ilustraciju postupka izrade modela unutar ASP.NET MVC, u narednim odjeljcima ćemo izgraditi model za banku donatora krvi.

Modeli za banku donatora krvi

Za početak stvorimo novi projekt ili otvorimo naš postojeći projekt koji je stvoren u Poglavlju 1.3. Biranjem MVC uzorka dobivamo sve što je potrebno da bi započeli s aplikacijom: osnovni raspored pogleda (engl. *layout view*), predefiniranu naslovnu stranicu (engl. *homepage*) sa linkom na prijavu korisnika, stranica je stilizirana. No ipak, direktorij *Models* je relativno prazan. U njemu se nalaze 3 datoteke: *AccountViewModels.cs*, *IdentityModels.cs* i *ManageViewModels.cs* (vidi Sliku 1.20). Sve te datoteke povezane su sa upravljanjem

korisničkim računima, te je za početak dobro znati da se sustav koji upravlja korisnicima u ASP.NET MVC pokreće na istom standardnom pogledu, modelima i upraviteljima koje ćemo koristiti za izgradnju ostatka aplikacije. Naravno, direktorij *Models* je s razlogom prazan, jer na nama je da izgradimo domenu na kojoj radimo te razjasnimo koji problem želimo rješavati.



Slika 1.20: Sadržaj direktorija *Models*

```
public class BloodDonation
{
    public virtual int BloodDonationId { get; set; }
    public virtual DateTime Date { get; set; }
    public virtual bloodType bloodType { get; set; }
    public virtual double bloodQuantity { get; set; }
    public virtual int InventoryId { get; set; }
    public virtual Inventory Inventory { get; set; }
}
```

Slika 1.21: *BloodDonation.cs*

Prvi korak neka bude implementiranje mogućnosti za pregled, dodavanje i uređivanje svih donacija krvi, kao i pregled zaliha pojedinih vrsta krvi. Stvaramo novu klasu pritiskom desnom tipkom miša na direktorij *Models*, te zatim odaberemo *Add* ⇒ *Class...* Klasu nazovemo *BloodDonation* (vidi Sliku 1.21). Ova klasa se neće kompajlirati zato jer smo referencirali svojstvo *Inventory* koje još uvijek nismo definirali. Zatim stvaramo novu klasu *Inventory*, kao što je prikazano na Slici 1.22.

```
public class Inventory
{
    public virtual int InventoryId { get; set; }
    public virtual bloodType bloodType { get; set; }
    public virtual double bloodQuantity { get; set; }
    public virtual List<BloodDonation> BloodDonations { get; set; }
}
```

Slika 1.22: *Inventory.cs*

Možemo primjetiti kako se u klasi *BloodDonation* pojavljuje referenca na svojstvo tipa *Inventory* u obliku:

```
public virtual int InventoryId { get; set; }

public virtual Inventory Inventory { get; set; }
```

dok se u klasi *Inventory* pojavljuje referenca na listu donacija krvi u obliku:

```
public virtual List<BloodDonation> bloodDonations { get; set; }
```

```
public enum bloodType
{
    O_plus = 1,
    O_minus = 2,
    A_plus = 3,
    A_minus = 4,
    B_plus = 5,
    B_minus = 6,
    AB_plus = 7,
    AB_minus = 8
}
```

Slika 1.23: *bloodType.cs*

To je način na koji *Visual Studiju* jasno dajemo direktivu da će pri kreiraju baze entiteti koji odgovaraju modelima *BloodDonation* i *Inventory* biti u vezi 1 naprama mnogo. To znači da jedna donacija krvi može biti pohranjena u jednom inventoriju, dok u jednom inventoriju može biti pohranjeno više donacija krvi. U slučaju da smo naš projekt započeli konstruiranjem baze, tada bi to značilo da nam je kod entiteta *BloodDonations* potreban strani ključ (engl. *foreign key*) koji upućuje na *InventoryId*. Također možemo uočiti da se u obje klase pojavljuje svojstvo *bloodType* za vrstu krvi. S obzirom da imamo točno predefinirane vrste krvi, možemo ih pohraniti u bazi kao tip *enum*. Prvo stvorimo novu

klasu u direktoriju *Models* naziva *bloodType*. Nakon toga promijenimo ključnu riječ *class* u riječ *enum*, kao što je to prikazano na Slici 1.23.

S obzirom da smo završili sa izgradnjom početnog koda modela, bitno je sve kompajlirati, a to radimo odlaskom u glavnom izborniku na *Build* ⇒ *Build Solution*. Druga alternativa je pritiskom na tipkovnici *Ctrl+Shift+B*. Ovo je izuzetno bitno napraviti, jer, prije svega omogućuje brzi uvid u to da li smo slučajno napraviti greške pri pisanju, a zatim i iz razloga što se prije toga klase neće pojaviti u dijaloškom prozoru *scaffold* za automatsko generiranje koda. Naravno, i dalje imamo opciju da samostalno kreiramo upravitelje i poglede, no nakon što to napravimo nekoliko puta, uočit ćemo da je to posao koji oduzima puno vremena, a izuzetno je ponavljajući.

Scaffold automatsko generiranje koda

Nakon generiranja naših klasa, možemo prijeći na generiranje upravitelja. Generiranje koda pomoću *scaffoldinga* može za nas ispitati način na koji smo definirali svaku od metoda unutar klasa, te na osnovu toga generirati funkcionalnost stvaranja, čitanja, izmjene i brisanja unutar aplikacije, također poznato kao CRUD (engl. *Create, Read, Update, Delete*). *Scaffolding* zna kako nazvati upravitelje, povezane poglede, a u nekim slučajevima i način pristupa podacima (primjerice ako nam je potreban padajući izbornik pri uređivanju i stvaranju). No ipak, ne smije se očekivati da će ovaj proces izgraditi cijelu aplikaciju. Zapravo, ovo automatsko generiranje koda oslobađa programera od stvaranja nezanimljivog dijela aplikacije. *Scaffolding* se pokreće isključivo kad mu mi kažemo da se pokrene, tako da nema brige da će doći do brisanja promjena koje sami napravimo u datotekama nakon inicijalnog generiranja koda.

Neke od ponuđenih opcija generiranja koda čine:

MVC 5 Controller - Empty Prazan predložak upravitelja dodaje klasu upravitelja s imenom koje specificiramo. Jedina akcija unutar upravitelja biti će akcija *Index* bez koda. Ovaj predložak neće stvoriti nikakve poglede.

MVC 5 Controller with read/write Actions Predložak s akcijama za čitanje/pisanje dodaje upravitelja u projekt s akcijama *Index, Details, Create, Edit i Delete*. Akcije nisu u potpunosti prazne, no također nisu sposobne izvoditi ikakav koristan rad sve dok ne dodamo vlastiti kod i stvorimo poglede za svaku akciju.

Web API 2 API Controller Scaffolders Nekoliko predložaka koji dodaju upravitelja naslijeđenog od bazne klase *ApiController*. Možemo koristiti ove predloške kako bismo izgradili Web API za svoju aplikaciju.

MVC 5 Controller with Views, Using Entity Framework Ovo je predložak koji ćemo najviše koristiti i on za nas ne generira samo upravitelja sa svim akcijama (*Index, Details, Create, Edit, Delete*), već također stvara potrebne poglede i kod, koji će se brinuti da baza bude sinkronizirana s promjenama.

Da bi predložak generirao ispravan kod, potrebno je označiti klasu modela (u našem slučaju, to je klasa *BloodDonation.cs*). Ono što je potrebno proslijediti jest i ime *data context* objekta. Možemo pri *scaffolding* procesu označiti već postojeću *data context* klasu, ili će *scaffolding* za nas kreirati novu *data context* klasu. Da bismo bolje razumjeli što ta klasa predstavlja, potrebno je prvo razumjeti osnove *Entity Frameworka*.

Entity Framework (skraćeno *EF*) je okvir za objektno-relacijsko preslikavanje (engl. *object-relational mapping (ORM)*) te razumije kako pohraniti .NET objekte u relacijsku bazu podataka, ali i kako dohvatiti te objekte uz pomoć *LINQ* upita (engl. *LINQ query*). *EF* podržava različite stilove razvijanja, primjerice možemo započeti razvoj od postojeće baze podataka (engl. *Database-First*), od modela (engl. *Model-First*) ili od kodiranja (engl. *Code-First*). *Code-First* pristup znači da možemo započeti pohranjivati i dohvaćati informacije iz *SQL Servera* bez stvaranja sheme baze podataka ili otvaranja dizajnera baze podataka unutar *Visual Studija*. Umjesto toga, pišemo jednostavne *C#* klase i *EF* će uspješno razumjeti kako i gdje pohraniti instance tih klasa.

Code-First konvencije

EF, baš kao i ASP.NET MVC, prati određene konvencije koje olakšavaju cijeli proces programiranja. Primjerice, ako želimo pohraniti objekt tipa *BloodDonation* u bazu podataka, *EF* pretpostavlja da želimo pohraniti podatke u tablicu naziva *BloodDonations*. On pretpostavlja da svojstvo sadrži vrijednost primarnog ključa te postavlja automatsko inkrementiranje na taj atribut unutar *SQL Servera*. Također ima konvencije za veze između stranih ključeva, imena baze i slično.

Code-First pristup funkcionira izuzetno dobro ako krenemo s izradom aplikacije od nule. Također, kao i sa svim konvencijama do sada, one mogu biti promijenjene. Primjerice, iako će prema konvenciji primarni ključ biti svojstvo tipa *int* i naziva *Id*, mi možemo to pisanjem određenih naredbi promijeniti, te ponovnim kreiranjem baze i ažuriranjem stare postaviti automatski željena svojstva. Konkretan alat za to će nam biti migracije podataka (engl. *Data Migrations*), no o tome će biti više riječi malo kasnije.

Klasa *DbContext*

Prilikom korištenja pristupa *code-first* unutar *Entity Framework*-a , poveznica s bazom podataka je klasa naslijeđena od *EF*-ove klase *DbContext*. Klasa sadrži jedno ili više svojstava tipa *DbSet<T>*, gdje *T* predstavlja vrstu objekta koju želimo koristiti. Primjerice, klasa na Slici 1.24 omogućava nam da upravljamo informacijama o inventaru i donacijama krvi. Na taj način možemo jednostavno dohvatiti sve donacije krvi prema datumu kada su provedene, od najnovijih prema najstarijima, slijedeći *LINQ* upit:

```
var db = new BloodBankDB();
var allBloodDonations = from donation in db.BloodDonations
                        orderby donation.Date descending
                        select donation;
```

```
public class BloodBankDB : DbContext
{

    public DbSet<BloodDonation> BloodDonations { get; set; }

    public DbSet<Inventory> Inventories { get; set; }
}
```

Slika 1.24: Klasa *BloodBankDB* naslijeđena od *DbContext*

Realiziranje *Scaffolding* predložaka

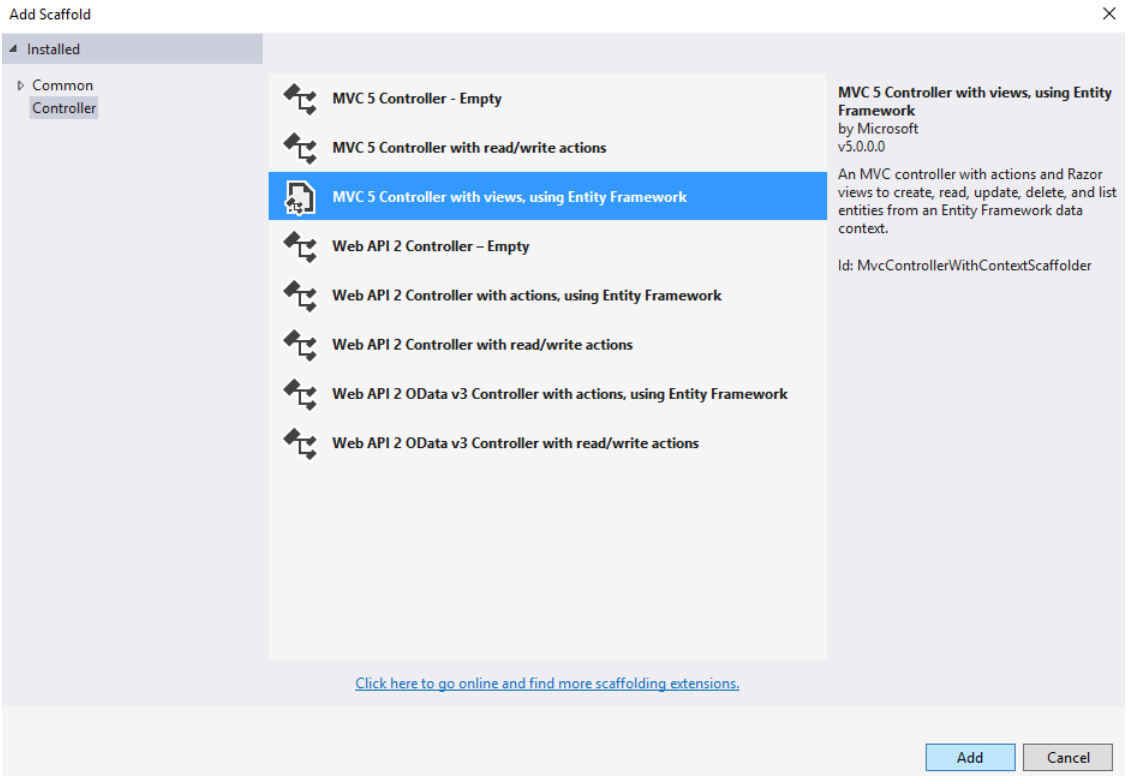
Korak 1 Desnom tipkom miša pritisnemo na direktorij *Controllers* te odaberemo *Add* ⇒ *Controller*

Korak 2 Od ponuđenih predložaka, odaberemo *MVC 5 Controller with Views, using the Entity Framework*, te potvrdimo pritiskom na *Add* (Slika 1.25)

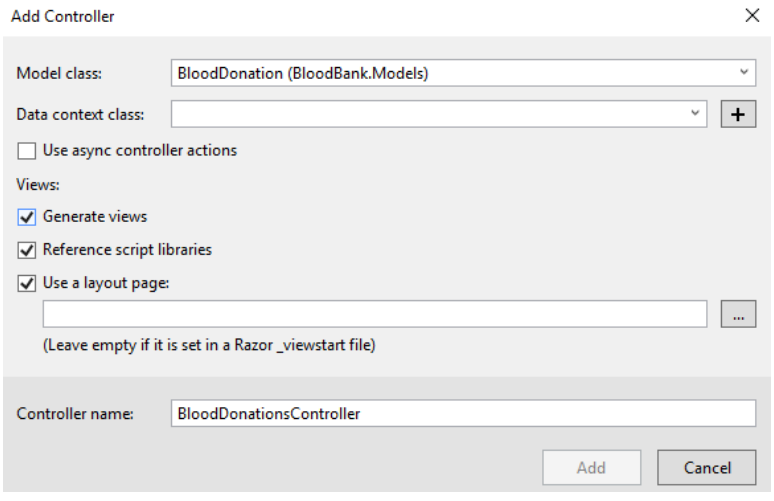
Korak 3 U novootvorenom *Add Controller* prozoru, postavimo ime upravitelja na *BloodDonationsController* te odaberemo *Blood Donation* kao klasu za model (engl. *Model class*). Možemo uočiti da gumb *Add* nije raspoloživ, jer još uvijek nismo označili *Data context* klasu (vidi Sliku 1.26)

Korak 4 Pritisnemo na znak +, te nam se otvori prozor kao na Slici 1.27. Preimenujemo klasu u *BloodBankDB*, te zatim potvrdimo pritiskom na *Add*.

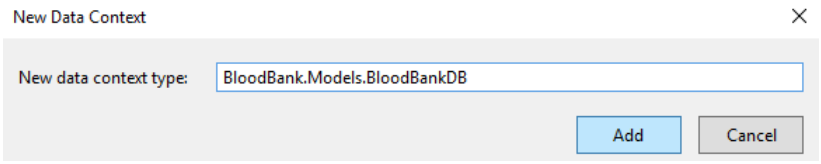
Korak 5 Nakon što još jednom provjerimo da li je sve ispravno definirano unutar prozora *Add Controller*, završimo pritiskom na (sada aktivan) gumb *Add* (Slika 1.28).



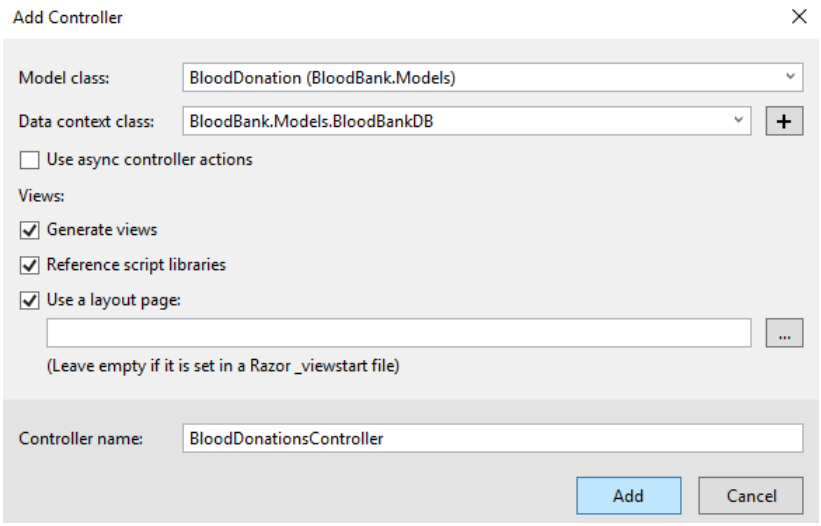
Slika 1.25: Stvaranje predloška *Scaffolding* (Korak 2)



Slika 1.26: Stvaranje predloška *Scaffolding* (Korak 3)



Slika 1.27: Stvaranje predloška *Scaffolding* (Korak 4)



Slika 1.28: Stvaranje predloška *Scaffolding* (Korak 5)

Promotrimo zatim našu novostvorenu klasu *BloodBankDB* unutar direktorija *Models* i naslijeđenu od klase *DbContext* (Slika 1.29).

```
public class BloodBankDB : DbContext
{
    // You can add custom code to this file. Changes will not be overwritten.
    //
    // If you want Entity Framework to drop and regenerate your database
    // automatically whenever you change your model schema, please use data migrations.
    // For more information refer to the documentation:
    // http://msdn.microsoft.com/en-us/data/jj591621.aspx

    public BloodBankDB() : base("name=BloodBankDB")
    {
    }

    public DbSet<BloodDonation> BloodDonations { get; set; }

    public DbSet<Inventory> Inventories { get; set; }
}
```

Slika 1.29: *BloodBankDB.cs*

Što se događa ako imamo potrebu izmijeniti naše klase unutar modela? Tada bitnu ulogu igraju migracije podataka (engl. *Data Migrations*). To su sistematične metode, bazirane na kodu, za provođenje promjena nad bazom podataka. Migracije dozvoljavaju da zadržimo postojeće podatke unutar postojeće baze tijekom izgrađivanja i promjena definicija modela. Pri promjenama, *EF* je sposoban uočiti što se točno promijenilo te stvoriti migracijske skripte koje možemo primijeniti na bazu podataka. Ovo je izuzetno praktično u počecima izrade aplikacije, kada još uvijek nismo sigurni u potpunosti kako želimo da naš model izgleda. Odgovor na pitanje koju će točno bazu podataka *data context* klasa koristiti bit će dan kasnije u poglavlju, kada dođemo do pokretanja aplikacije.

BloodDonationsController

Predložak *Scaffold* koji smo odabrali stvorio je i upravitelja *BloodDonations* u direktoriju *Controllers*. Upravitelj ima sav potreban kod da bi ispisao i uredio informacije o donacijama krvi. Zanimljivo je pogledati prve linije koda unutar upravitelja (vidi Sliku 1.30). Tu vidimo da *Scaffolding* dodaje privatnu varijablu tipa *BloodBankDB* upravitelju, te ga također i inicijalizira s novom instancom *data contexta*, s obzirom da svaki upravitelj zahtijeva pristup bazi.

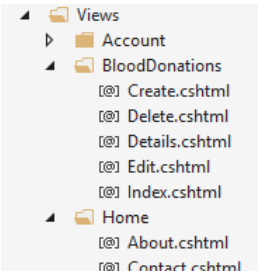
```
public class BloodDonationsController : Controller
{
    private BloodBankDB db = new BloodBankDB();

    // GET: BloodDonations
    0 references
    public ActionResult Index()
    {
        var bloodDonations = db.BloodDonations.Include(b => b.Inventory);
        return View(bloodDonations.ToList());
    }

    // GET: BloodDonations/Details/5
    0 references
    public ActionResult Details(int? id)
```

Slika 1.30: *BloodDonationsController.cs*

Pogledi

Također ćemo osim upravitelja, u direktoriju *Views* pronaći poddirektorij *BloodDonations*, kao što vidimo na Slici 1.31. Ti pogledi pružaju korisničko sučelje za ispis, uređivanje te brisanje donacija. Pogled *Index* ima sve što mu je potrebno da prikaže potpunu listu donacija. Model za pogled *Index* možemo pročitati u zaglavlju datoteke (vidi Sliku 1.32):

```
@model IEnumerable<BloodBank.Models.BloodDonation>
```



Slika 1.31: Direktorij pogleda *BloodDonations*

Vidimo da svaki red stvorene tablice uključuje poveznice za uređivanje, pregled i brisanje. Pogled uzima model te koristi *foreach* petlju kako bi stvorio HTML tablicu s informacijama vezanim uz svaku pojedinu donaciju. Index pogled ima ulogu prikaza liste svih trenutno postojećih zapisa u bazi podataka. To je napravljeno na način da se koristi kontekstna klasa *BloodDonationDb* kako bi se dohvatili podaci o donacijama krvi, te se proslijeđuju pogledu u obliku strogo tipiziranog objekta modela, kratko spomenutog u Poglavlju 1.7. Sukladno tome, tim podacima ćemo unutar pogleda pristupati uz pomoć

sintakse *Razor*. Primjerice, *Id* broju inventorija u koji će donacija krvi biti proslijeđena, unutar *foreach* petlje na Slici 1.32, pristupat ćemo pozivom

```
@item.Inventory.Id
```

U ovom će nam procesu od velike pomoći biti *Intellisense*, iz razloga što će nam automatski ponuditi sva dostupna imena varijabli koje smo dodijelili svakom modelu.

```
@model IEnumerable<BloodBank.Models.BloodDonation>

@{
    ViewBag.Title = "Index";
}
<h2>Index</h2>
<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table class="table">
    <tr>
        <th>@Html.DisplayNameFor(model => model.Inventory.Id)</th>
        <th>@Html.DisplayNameFor(model => model.Date)</th>
        <th>@Html.DisplayNameFor(model => model.bloodType)</th>
        <th>@Html.DisplayNameFor(model => model.bloodQuantity)</th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>@Html.DisplayFor(modelItem => item.Inventory.Id)</td>
        <td>@Html.DisplayFor(modelItem => item.Date)</td>
        <td>@Html.DisplayFor(modelItem => item.bloodType)</td>
        <td>@Html.DisplayFor(modelItem => item.bloodQuantity)</td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.Id }) |
            @Html.ActionLink("Details", "Details", new { id=item.Id }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.Id })
        </td>
    </tr>
}
</table>
```

Slika 1.32: Pogled *Index* od BloodDonations

Izvršavanje automatski generiranog koda

Prije nego krenemo s izvršavanjem koda, vratimo se izuzetno bitnom pitanju na koje nije dan odgovor ranije: Koju bazu podataka *BloodBankDB* koristi? Nismo do sad stvorili bazu koju bi aplikacija koristila ili specificirali vezu s bazom podataka.

```
<connectionStrings>
  <add name="BloodBankDB"
       connectionString="Data Source=(localdb)\MSSQLLocalDB;
                         Initial Catalog=BloodBankDB-20170725164238;
                         Integrated Security=True;
                         MultipleActiveResultSets=True;
                         AttachDbFilename=|DataDirectory|BloodBankDB-20170725164238.mdf"
        providerName="System.Data.SqlClient" />
</connectionStrings>
```

Slika 1.33: *connectionString* unutar datoteke *Web.config*

Code-First pristup *Entity Frameworka* koristi iznova prvenstveno konvenciju, a ne konfiguraciju, koliko god je moguće. Ako sami ne postavimo određena preslikavanja iz modela u bazu podataka, tada *EF* koristi konvencije kako bi stvorio shemu baze podataka. Bez posebno naznačenog načina povezivanja, *EF* će se probati povezati na *LocalDB* instancu *SQL Servera* te pronaći bazu istog imena kao klasa naslijeđena iz *DbContext* (u našem slučaju to je *BloodBankDB*). Postavke mijenjamo unutar datoteke *Web.config* našeg projekta, unutar *connection string-a*. Kako je to automatski *EF* postavio, možemo vidjeti na Slici 1.33.

```
public class BloodBankDbInitializer : System.Data.Entity.DropCreateDatabaseAlways<BloodBankDB>
{
    1 reference
    protected override void Seed(BloodBankDB context)
    {
        context.Inventories.Add(new Inventory {bloodType = bloodType.O_minus, bloodQuantity=20});
        context.Inventories.Add(new Inventory {bloodType = bloodType.B_plus, bloodQuantity=10});

        context.BloodDonations.Add(new BloodDonation {
            bloodType = bloodType.AB_plus,
            Date = DateTime.Now,
            bloodQuantity = 3,
            Inventory = new Inventory { bloodType = bloodType.AB_plus, bloodQuantity=3}
        });

        context.BloodDonations.Add(new BloodDonation {
            bloodType = bloodType.O_minus,
            Date = DateTime.Now,
            bloodQuantity = 2,
            Inventory = new Inventory { bloodType = bloodType.A_minus, bloodQuantity=2}
        });

        base.Seed(context);
    }
}
```

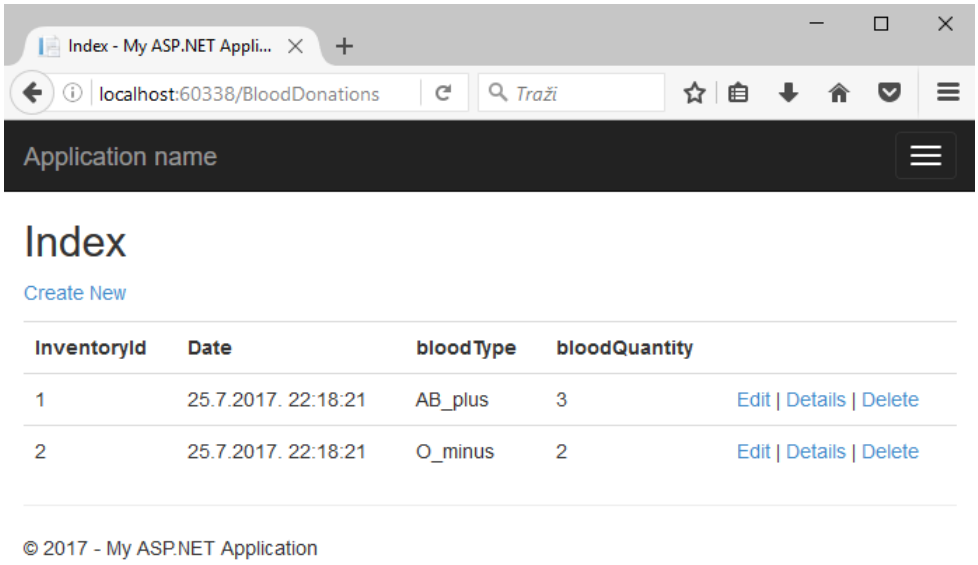
Slika 1.34: *BloodBankDbInitializer.cs*

U ovom trenutku baza nema nikakvih podataka, te ju želimo za početak napuniti određenim testnim podacima. Također želimo da se pri svakom pokretanju stvori potpuno nova baza podataka. Ta procedura se zove *seed* metoda (engl. *seed* = sjeme). Prvo stvorimo unutar direktorija *Models* novu klasu koju nazovemo *BloodBankDbInitializer*. Definiramo ju na način prikazan na Slici 1.34.

Preostalo je dodati u klasu *Global.asax.cs* kod kako bi se naša inicijalizacija iznova ponovila pri svakom pokretanju. To je napravljeno na način prikazan na Slici 1.35. Sada našu aplikaciju napokon možemo pokrenuti i vidjeti kako automatski generiran kod izgleda.

```
protected void Application_Start()
{
    Database.SetInitializer(new BloodBankDbInitializer());

    AreaRegistration.RegisterAllAreas();
    FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
    RouteConfig.RegisterRoutes(RouteTable.Routes);
    BundleConfig.RegisterBundles(BundleTable.Bundles);
}
```

Slika 1.35: Global.asax.cs



Slika 1.36: Pokrenuta aplikacija u pregledniku

U ovom poglavlju smo vidjeli kako unutar ASP.NET napraviti elemente MVC web-aplikacije: upravitelje, poglede i modele. Iako je sve ove elemente moguće napraviti ”od nule”, *Visual Studio*, slijedeći logiku “konvencija prije konfiguracije” omogućava da tehnikom *Scaffoldinga* brzo i automatski generiramo značajan dio koda. Ova tehnika pruža i elegantan način automatske izgradnje baze podataka na temelju modela, kao i generiranje koda za poglede i upravitelje koji realiziraju sve CRUD operacije.

Poglavlje 2

Web-aplikacija za donacije izrađena pomoću ASP.NET MVC 5

Aplikacija za donacije razvijena je pomoću ASP.NET MVC 5 okvira, koristeći *Visual Studio 2015*. Također je za izradu aplikacije korišten okvir naziva *Identity Framework*, za implementaciju korisnika aplikacije, i *Entity Framework*, za prevođenje napisanog modela u bazu podataka. U nastavku će aplikacija biti detaljnije opisana te će biti dan pregled modela, upravitelja i pogleda korištenih za njenu realizaciju. Naposlijetku će biti dan i uvid u izgled aplikacije unutar web-preglednika.

2.1 Opis aplikacije

Aplikacija za donacije jest web-platforma namijenjena onima kojima je potrebna donacija te onima koji bi željeli donirati ili podržati postojeće akcije. Pokretač kampanje prikupljanja novaca mora biti korisnik registriran unutar aplikacije. Pri stvaranju kampanje on samostalno određuje naziv kampanje, koliko novaca je potrebno da bi akcija bila uspješna, opis akcije, fotografije, te u koje kategorije akcija ulazi. Jedna kampanja može biti istovremeno u više kategorija.

Svaki neregistrirani korisnik ima mogućnost pregledavanja svih kampanja, njenih komentara te donacija, kao i pretraživanja svih kampanja, dok registrirani korisnik, pored već navedenog, može dodavati kampanje u listu omiljenih, može komentirati kampanju, označiti da mu se određeni komentar sviđa, donirati novce kampanji, te upravljati svojim korisničkim postavkama. Ima mogućnosti i stvarati, uređivati i brisati vlastite kampanje.

Da bi se novi korisnik registrirao, potrebno je da upiše svoju *e-mail* adresu, lozinku i ponovljenu lozinku. Kako bi registracija bila uspješna, *e-mail* adresa ne smije već postojati u sustavu aplikacije, te upisana lozinka mora imati najmanje 6 znakova, od kojih je barem jedan broj, barem jedan veliko slovo, barem jedan malo slovo i barem jedan specijalni

znak. Svaki novi korisnik ima dodijeljenu ulogu običnog korisnika, dok određeni korisnici mogu imati ulogu administratora aplikacije.

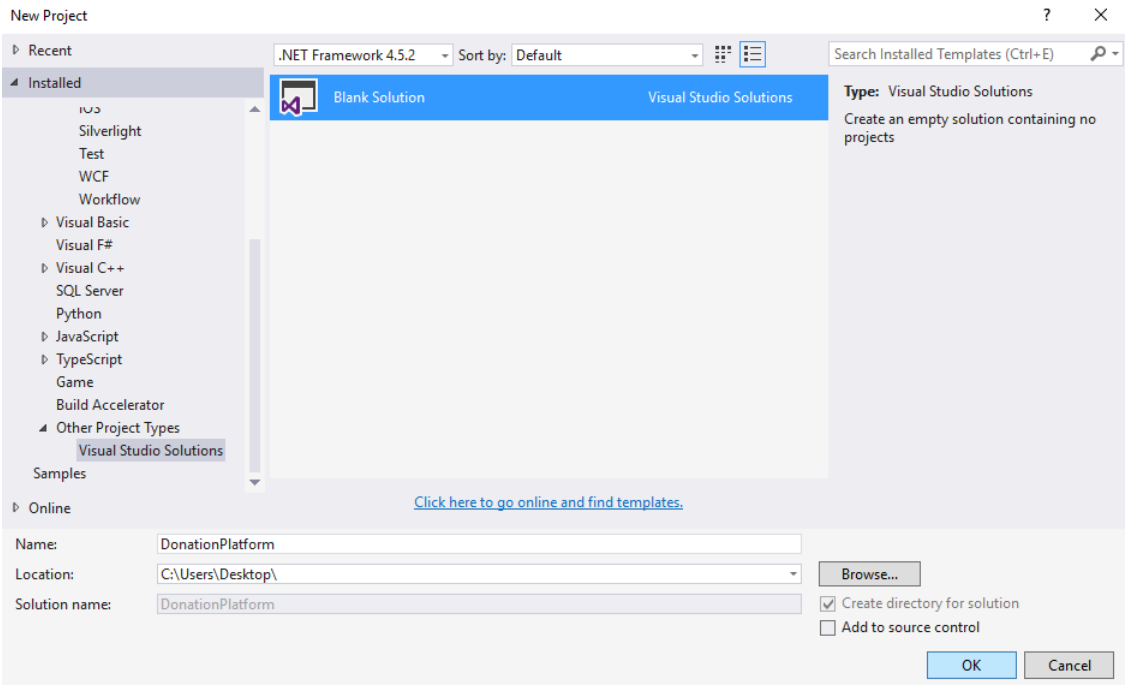
U početnom stadiju aplikacije, postoji jedan administrator, koji ima prava izmijeniti prava pristupa drugih korisnika te dodijeliti i drugima administratorska prava. Za razliku od običnih korisnika, administrator ima mogućnost stvaranja novih kategorija, te uređivanja i brisanja istih. On ima uvid u sve korisnike, te također može uređivati njihove podatke (izuzev lozinke) ili obrisati korisničke račune. Također upravlja sadržajem naslovne stranice, tako da odredi u kojem će se vremenskom razdoblju na naslovnoj stranici prikazivati željene kampanje.

Nakon što je jedna kampanja ostvarila svoj novčani cilj kroz donacije, i dalje je moguće donirati novce, no kampanja ima oznaku koja pokazuje da daljnje donacije nisu potrebne.

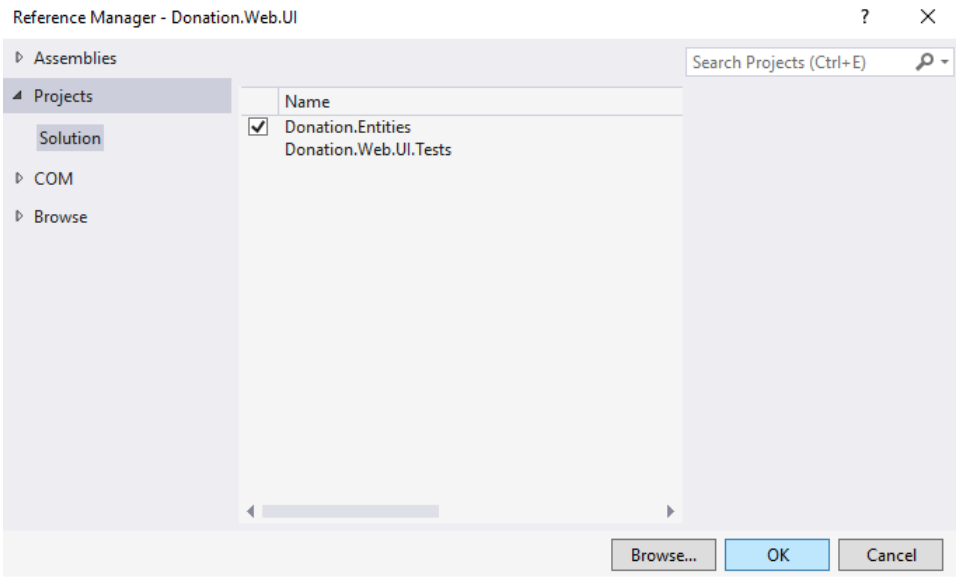
2.2 Struktura aplikacije

Aplikacija se sastoji od 3 projekta, naziva *Donation.Entities*, *Donation.Web.UI.* i *Donation.Web.UI.Tests*. U prvom projektu se nalazi model aplikacije, dok se unutar *Donation.Web.UI* nalaze upravitelji i pogledi. Treći projekt ne obuhvaća temu diplomskog rada te o njemu neće biti više riječi, no s obzirom da ga je korisno imati u kasnijem stadiju aplikacije, zbog lakšeg testiranja funkcionalnosti aplikacije, on postoji i ovdje. Kako bismo stvorili projekt u koji je moguće dodati više zasebnih projekata, potrebno je u glavnom izborniku *Visual Studija* otići na *File* ⇒ *New* ⇒ *Project* te unutar dijaloškog okvira izabrati *Other project types* ⇒ *Visual Studio Solutions* ⇒ *Blank Solution* (vidi Sliku 2.1). Nakon toga se pritiskom desne tipke miša na ime projekta unutar Solution Explorera te odabirom *Add* ⇒ *New Project...* zasebno doda svaki projekt. Odvojiti projekt u kojem se nalazi model od projekta u kojem se nalaze pogledi i upravitelji je odlična praksa iz razloga što je tada isti model vrlo jednostavno upotrijebiti za više projekata. Primjerice, u slučaju kada želimo izgraditi aplikaciju u kojoj je sučelje administratora potpuno odvojeno od sučelja koje koristi obični korisnik.

U ovoj cjelini bit će dan detaljniji uvid u korištene modele, upravitelje i poglede unutar aplikacije.



Slika 2.1: Dijaloški okvir *New Project*

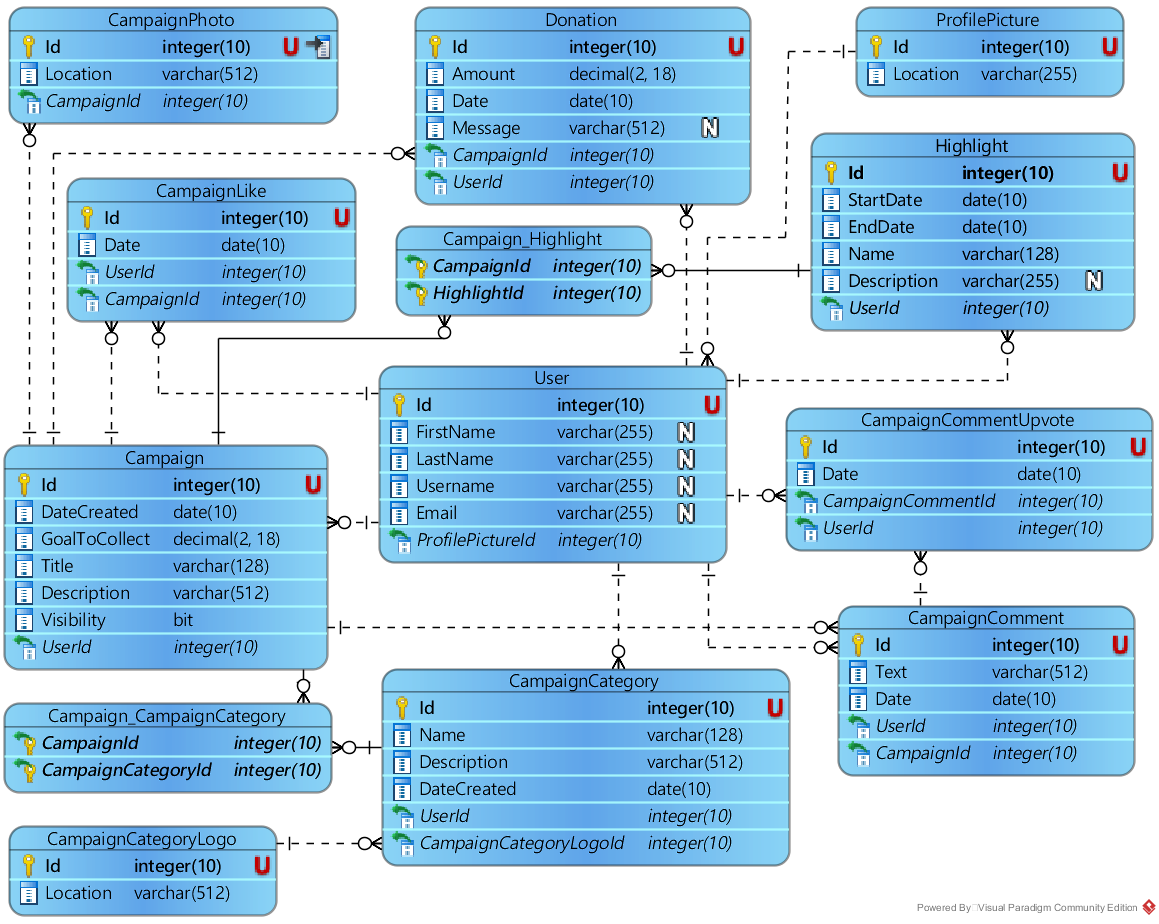


Slika 2.2: *Reference Manager*

Modeli

Za početak, kako bi bilo moguće pristupiti modelu u projektu *Donation.Entities*, bilo je potrebno unutar projekta *Donation.Web.UI* dodati referencu na prvi projekt. To se postiže pritiskom desne tipke miša na naziv drugog projekta unutar *Solution Explorera*, te odabirom *Add* ⇒ *Reference*. Nakon toga se unutar *Projects* označi *Donation.Entities* te potvrdi pritiskom na *OK* (vidi Sliku 2.2)

Prvi korak u razvijanju aplikacije bio je stvaranje ERD dijagrama (engl. *Entity Relationship Diagram*) unutar kojega će se jasno vidjeti kakve veze između različitih entiteta unutar baze želimo postići. Na temelju tog dijagrama napisani su modeli aplikacije (vidi Sliku 2.3). Pojašnjenje svih modela moguće je pronaći u Tablici 2.1.

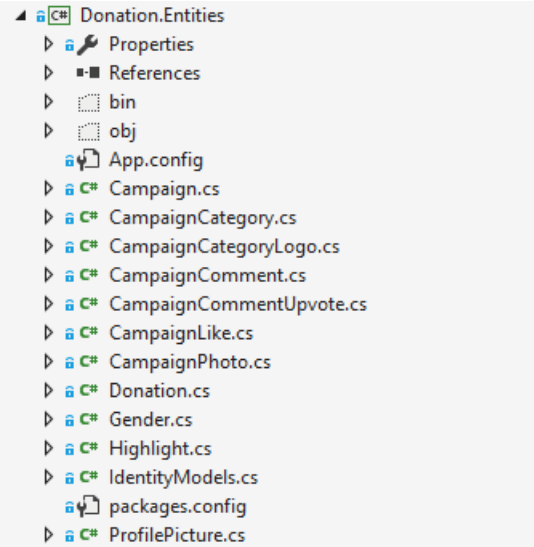


Slika 2.3: ERD dijagram

Tablica 2.1: Modeli i njihove uloge

Model	Opis modela
User	Svaki registrirani korisnik aplikacije, koji može imati ili obična korisnička prava ili administratorska prava, te ovisno o tome dopuštenja za pristup određenim elementima aplikacije.
Campaign	Predstavlja jednu aktivnu pokrenutu kampanju, koju isključivo administrator i korisnik koji ju je stvorio mogu uređivati i brisati.
CampaignCategory	Kategorija u koju svrstavamo kampanje. Primjer naziva kategorija bio bi 'Životinje', 'Zdravlje', 'Obrazovanje' i slično.
CampaignCategoryLogo	Slika koja je korištena za opis jedne kategorije kampanje. Svaka instanca *CampaignCategory* može imati najviše jednu sliku.
ProfilePicture	Fotografija vezana uz registriranog korisnika aplikacije. Svaki korisnik tipa *User* može imati najviše jednu profilnu sliku tipa *ProfilePicture*.
CampaignComment	Komentar koji je usko vezan uz pojedinu kampanju. Svaki komentar može pripadati jednoj kampanji tipa *Campaign*, dok svaka kampanja može imati više komentara tipa *CampaignComment*.
CampaignLike	Svaka kampanja tipa Campaign može imati više vezanih instanci *CampaignLike*, koje označavaju da se određenom korisniku kampanja sviđa, dok jedna instanca *CampaignLike* pripada jednoj kampanji. Svaki registrirani korisnik je pritom u mogućnosti označiti da mu se svaka dostupna kampanja sviđa.
CampaignCommentUpvote	Model je usko vezan uz model *CampaignComment*. Svaki registrirani korisnik je u mogućnosti označiti komentar tipa *CampaignComment* da mu se sviđa, stvarajući time novu instancu modela *CampaignCommentUpvote*.
Highlight	Predstavlja svojstvo dostupno za izmjenu isključivo administratoru. To je sadržaj koji će se pojavljivati na naslovnoj stranici aplikacije.
CampaignPhoto	Usko je vezano uz instancu tipa *Campaign*. Predstavlja fotografije koje pripadaju jednoj kampanji. Jedna kampanja može imati više fotografija, dok jedna fotografija može pripadati isključivo jednoj kampanji.
Donation	Model koji predstavlja jednu novčanu donaciju i usko je vezan uz instancu tipa *Campaign*. Jedna kampanja može imati više donacija tipa *Donation*.

Sukladno tome, unutar *Donation.Entities* stvorene su klase koje predstavljaju model. Klasa naziva *IdentityModels* jest klasa koja implementira sučelje korisnika, o kojoj se u pozadini brine *Identity Framework*. Klase koje projekt sadrži moguće je vidjeti na Slici 2.4.



Slika 2.4: Modeli aplikacije

```
public class CampaignCategory
{

    public int Id { get; set; }

    public string Name { get; set; }

    public string Description { get; set; }

    public DateTimeOffset? DateCreated { get; set; }

    public virtual ApplicationUser User { get; set; }

    public virtual CampaignCategoryLogo CampaignCategoryLogo { get; set; }

    public virtual ICollection<Campaign> Campaigns { get; set; }

}
```

Slika 2.5: CampaignCategory.cs

Za primjer implementacije, promotrimo klase *Campaign* i *CampaignCategory*. Prva klasa sadrži podatke vezane uz jednu postojeću kampanju, dok druga klasa sadrži podatke

vezane uz kategorije u koje svrstavamo kampanje. Također uočimo da će ta dva modela unutar baze podataka biti u odnosu mnogo naprama mnogo, što znači da jedna kategorija može sadržavati više kampanja, dok jedna kampanja može biti svrstana u više kategorija (vidi Slike 2.5 i 2.6).

```
public class Campaign
{

    public Campaign()
    {
        CampaignCategories = new List<CampaignCategory>();
    }

    public int Id { get; set; }
    [Required]

    public string Title { get; set; }

    public DateTimeOffset? DateCreated { get; set; }

    public decimal GoalToCollect { get; set; }

    public string Description { get; set; }

    public virtual ApplicationUser User { get; set; }

    public virtual ICollection<CampaignPhoto> CampaignPhotos { get; set; }

    public virtual ICollection<CampaignComment> CampaignComments { get; set; }

    public virtual ICollection<CampaignLike> CampaignLikes { get; set; }

    public virtual ICollection<Donation> Donations { get; set; }

    public virtual ICollection<CampaignCategory> CampaignCategories { get; set; }

    public virtual ICollection<Highlight> Highlights { get; set; }
}
```

Slika 2.6: Campaign.cs

Iz navedenih primjera možemo uočiti kako prevodimo programskom jeziku *C#* tipove podataka kako bi ih on razumio. Cijele brojeve smo proglasili tipom *int*, tekst ili nazivi su tipa *string*, dok primjerice broj izražen kao novac jest tipa *decimal* (on se smatra pogodnijim za financijske proračune od standardnih tipova poput *double*). Datum stvaranja (*DateCreated*) je tipa *DateTimeOffset?*. Stavljajući upitnik na kraj, ovom atributu unutar baze će biti dozvoljeno da bude tipa *NULL* (nedefiniran), dok nam tip *DateTimeOffset* omogućuje da spremimo datum prema standardnom vremenu, te time znamo postaviti vrijeme uvijek ovisno o vremenskoj zoni prijavljenog korisnika, a ne vrijeme jedinstveno samo za određenu regiju svijeta. Korisnik (*User*) je tipa *ApplicationUser*, što je klasa implementirana unutar *Identity Frameworka*. Uočimo i da unutar klase *Campaign* vršimo referencu na više elemenata klase *CampaignCategories*, te također unutar klase *CampaignCategories* referencu na više elemenata klase *Campaign*. To je upravo način na koji će

Entity Framework jasno razumijeti pri prevođenju modela u bazu da su ta dva elementa u vezi mnogo naprama mnogo. Analogno je napravljeno i sa svim drugim klasama, ovisno da li su u vezi mnogo naprama mnogo, mnogo naprama jedan ili, upravo spomenutoj, mnogo naprama mnogo vezi.

Upravitelji

Dodavanje migracije podataka

Pri dodavanju prvog upravitelja, unutar datoteke *DataContexts* stvorene unutar Donation.Web.UI projekta, stvorena je kontekstna klasa za komunikaciju upravitelja i modela, naziva *DonationDb*. Na Slici 2.7 moguće je vidjeti kako kontekstna klasa izgleda.

```
public class DonationDb : ApplicationDbContext
{
    9 references
    public DonationDb()
        : base ()
    {
    }
    31 references
    public DbSet<Campaign> Campaigns { get; set; }
    18 references
    public DbSet<CampaignCategory> CampaignCategories { get; set; }
    3 references
    public DbSet<CampaignCategoryLogo> CampaignCategoryLogos { get; set; }
    9 references
    public DbSet<CampaignComment> CampaignComments { get; set; }
    3 references
    public DbSet<CampaignCommentUpvote> CampaignCommentUpvotes { get; set; }
    3 references
    public DbSet<CampaignLike> CampaignLikes { get; set; }
    7 references
    public DbSet<CampaignPhoto> CampaignPhotos { get; set; }
    5 references
    public DbSet<Entities.Donation> Donations { get; set; }
    9 references
    public DbSet<Highlight> Highlights { get; set; }
    2 references
    public DbSet<ProfilePicture> ProfilePictures { get; set; }
}
```

Slika 2.7: *DonationDb.cs*

U toj fazi bilo je za očekivati da će se model zasigurno mijenjati, te je iz tog razloga omogućena opcija migriranja baze podataka, spomenuta u Poglavlju 1.8. To je napravljeno na način da otvorimo *Package Manager Console* (odabirom u glavnom izborniku *Tools* ⇒ *Nuget Package Manager*), te upišemo u konzolu

```
PM> enable-migrations -ContextTypeName
    Donation.Web.UI.DataContexts.DonationDb -MigrationDirectory
    DataContexts\DonationMigrations
```

```
internal sealed class Configuration : DbMigrationsConfiguration<DonationDb>
{
    0 references
    public Configuration()
    {
        AutomaticMigrationsEnabled = false;
        MigrationsDirectory = @"DataContexts\DonationMigrations";
    }

    0 references
    protected override void Seed(DonationDb context)
    {


    }
}
```

Slika 2.8: *Configuration.cs*

Ukoliko je sve prošlo u redu, vidjet ćemo da se unutar direktorija ***DataContexts/DonationMigrations*** pojavila klasa naziva *Configuration* (vidi Sliku 2.8), koja nam je potrebna za daljnje stvaranje skripti za ažuriranje baze podataka. Sljedeći korak, dodavanje migracije, radimo na način

```
PM> add-migration -ConfigurationTypeName
    Donation.Web.UI.DataContexts.DonationMigrations.Configuration
    "InitialCreate"
```

Nakon uspješnog izvođenja će unutar direktorija ***DataContexts/DonationMigrations*** biti generirane skripte za provođenje izmjena na bazi podataka. U ovom trenutku još uvijek baza nije generirana, što znači da će prve skripte biti generiranje baze podataka. Posljednji korak je ažuriranje baze podataka, tako da se u *Package Manager Console* upiše

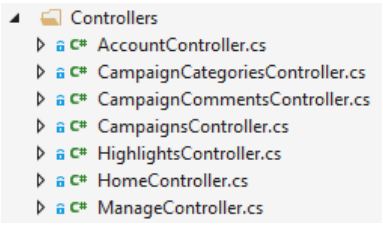
```
PM> update-database -ConfigurationTypeName
    Donation.Web.UI.DataContexts.DonationMigrations.Configuration -verbose
```

Ako je sve prošlo u redu, sada bismo trebali imati postojeću bazu podataka koju ćemo kasnije, sa svakom izmjenom modela, nadograđivati ponovnim upisivanjem *add-migration* i *update-database* unutar *Package Manager Console*.

Upravitelji unutar aplikacije

Na Slici 2.9 moguće je vidjeti koji su sve upravitelji dodani, pored već postojećih upravitelja *AccountController* i *ManageController*, koji služe za upravljanje korisničkim podacima. Upravitelj *CampaignCategories* se brine o upravljanju kategorijama, *CampaignComments* o komentarima unutar pojedinih kampanja, *Campaigns* oko stvaranja, uređivanja i brisa-

nja kampanji, te njenih donacija, *Highlights* oko stvaranja, uređivanja i brisanja istaknutih kampanja na naslovnoj stranici, dok *Home* sadrži metode koje implementiraju prikaz određenih dijelova naslovne stranice.

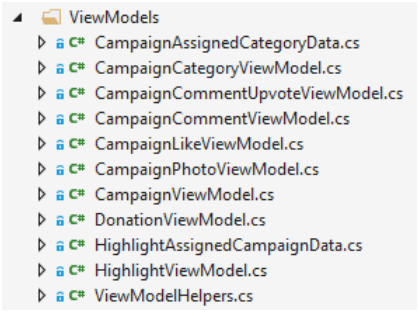


Slika 2.9: *Direktorij Controllers*

View Modeli

Istaknimo kako upravitelji ne prosljeđuju pogledima modele iz *Donation.Entities* projekta. Upravitelji dohvaćaju podatke iz baze uz pomoć modela i *Entity Frameworka*, te ih potom prevode u tip *ViewModel* (odnosno primaju *ViewModel* od pogleda, te ga prevode u tip koji je prepoznatljiv *Entity Frameworku*). Primjerice, kampanja s modelom *Campaign* je prevedena u *ViewModel* naziva *CampaignViewModel*. Prema konvenciji, spremamo ih unutar ručno generiranog direktorija naziva *ViewModels* unutar projekta *Donation.Web.UI*.

ViewModeli nam omogućuju da definiramo određene elemente koji nam unutar baze i modela domene nisu potrebni. Primjerice, unutar *ViewModela* možemo definirati varijablu koja vraća ukupan broj komentara unutar jedne kampanje. To nam nije bilo potrebno u modelu, no u našem pogledu i upravitelju želimo na jednostavan način moći pristupiti ukupnom broju komentara jedne kampanje. Pregled svih generiranih *ViewModela* moguće je vidjeti na Slici 2.10.



Slika 2.10: Direktorij *ViewModels*

Klasa *ViewModelHelpers* služi kao pomoćna klasa koja sadrži metode za prevođenje *ViewModela* u model domene, ali i obrnuto. Također, specifične klase su i *CampaignAssignedCategoryData* i *HighlightAssignedCampaignData*. Prije svega, uočimo da su to dva slučaja kada su kampanje i kategorije (odnosno highlight i kampanja) u odnosu mnogo naprama mnogo. *Scaffold* automatsko generiranje koda neće samo razumjeti da pri stvaranju nove kampanje ili uređivanju iste želimo da nam se učitaju sve postojeće kategorije, te da je moguće odabrati samo određene kategorije od ponuđenih. Iz tog razloga stvaramo specijalizirane poglede za određene situacije koji se razlikuju isključivo po jednoj varijabli naziva *Assigned*. Ukoliko jednu kampanju želimo svrstati u određenu kategoriju, tada će vrijednost *Assigned* biti postavljena na *true*, dok će u suprotnom biti postavljena na *false*. Bolji uvid u *CampaignViewModel* te *CampaignAssignedCategoryData* može se vidjeti na Slikama 2.11 i 2.12.

```
public class CampaignViewModel
{
    10 references
    public int Id { get; set; }
    7 references
    public string Title { get; set; }
    7 references
    public DateTimeOffset? DateCreated { get; set; }
    10 references
    public decimal GoalToCollect { get; set; }
    8 references
    public string Description { get; set; }
    6 references
    public Boolean Visibility { get; set; }
    7 references
    public decimal AmountCollected { get; set; } //read this from DonationViewModel
    5 references
    public int LikesCollected { get; set; } //read this from CampaignLikeViewModel
    4 references
    public int CommentsNumber { get; set; } //read this from CampaignCommentViewModel
    4 references
    public int DonationsNumber { get; set; } //read this from DonationViewModel
    12 references
    public virtual ApplicationUser User { get; set; }
    6 references
    public virtual ICollection<CampaignAssignedCategoryData> CampaignCategories { get; set; }
    7 references
    public virtual ICollection<CampaignPhotoViewModel> CampaignPhotos { get; set; }
    4 references
    public virtual ICollection<DonationViewModel> Donations { get; set; }
    4 references
    public virtual ICollection<CampaignCommentViewModel> CampaignComments { get; set; }
    2 references
    public virtual ICollection<CampaignLikeViewModel> CampaignLikes { get; set; }
}
```

Slika 2.11: *CampaignViewModel.cs*

```
public class CampaignAssignedCategoryData
{
    //Category data that we send to our Campaign View
    6 references
    public int Id { get; set; }
    4 references
    public string Name { get; set; }
    3 references
    public string Description { get; set; }
    6 references
    public bool Assigned { get; set; }
    2 references
    public virtual CampaignCategoryLogo CampaignCategoryLogo { get; set; }


}
```

Slika 2.12: *CampaignAssignedCategoryData.cs*

Prava pristupa korisnika

Kako bismo postavili restrikcije na pristup određenim stranicama, unutar upravitelja iznad njegove metode potrebno je staviti određeno pravo pristupa. Upravo ono najpotrebnije jest *[Authorize]* koje omogućava samo registriranim i prijavljenim korisnicima da upravljaju tom stranicom. Također, ono što nam je svakako u određenim trenucima bilo potrebno je pravo *[Authorize("Admin")]* prema kojemu isključivo admin ima pravo pristupa određenoj metodi upravitelja. Primjer za korištenje moguće je vidjeti na Slici 2.13, na kojoj vidimo metodu *Donate*, koja pripada *CampaignsController*-u.

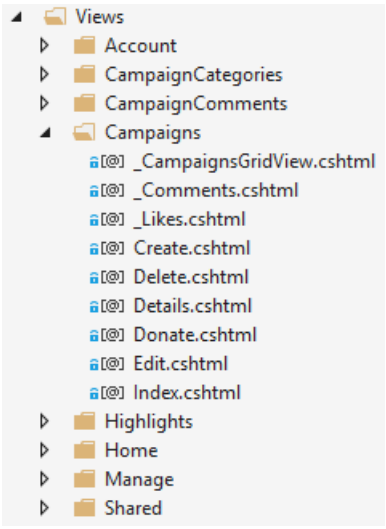
```
[Authorize]
0 references
public ActionResult Donate(int? id = 0)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    var campaign = db.Campaigns.Find(id);
    var donationViewModel = new DonationViewModel { Campaign = campaign };
    return View(donationViewModel);
}
```

Slika 2.13: Metoda *Donate* unutar *CampaignsController*

Nakon što smo se upoznali ukratko s upraviteljima aplikacije za donacije, slijedi opis pripadajućih pogleda aplikacije, kao i kratki uvid u izgled aplikacije u web-pregledniku.

Pogledi

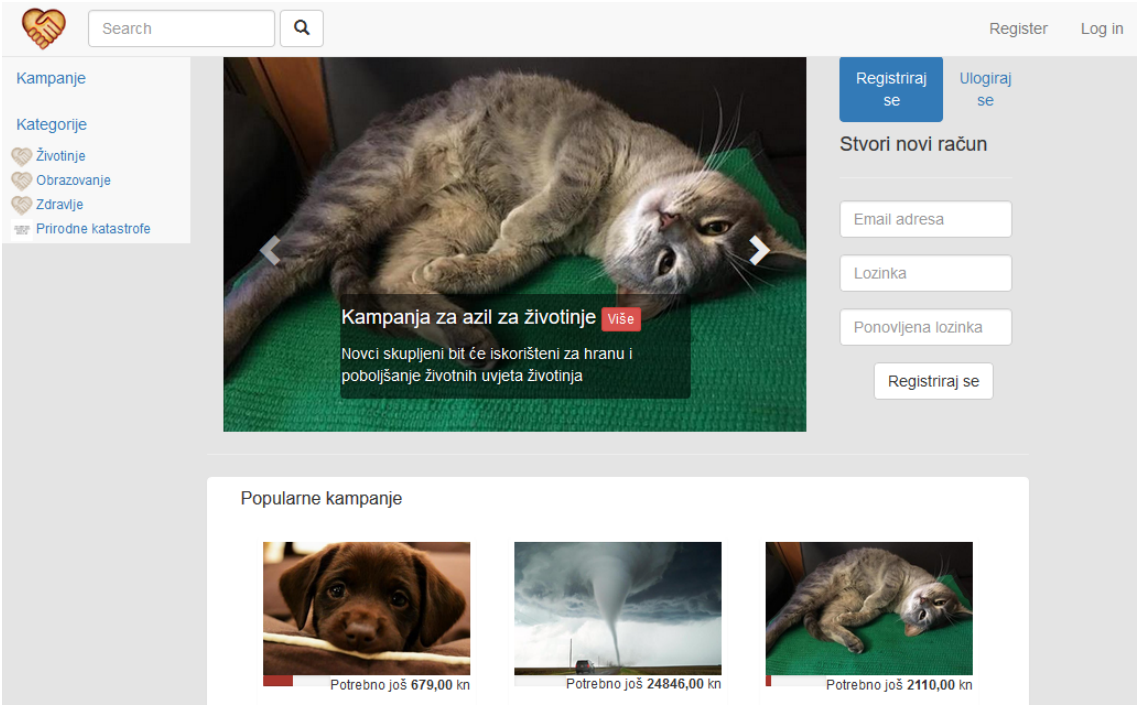
Pregled svih postojećih pogleda moguće je vidjeti na Slici 2.14. Svaki poddirektorij direktorija *Views* sadrži poglede koji prikazuju liste svih postojećih elemenata (primjerice, listu postojećih kampanja), opcije za detaljniji pregled određenog elementa te također opcije za uređivanje i brisanje istog. Generirani su iz upravitelja, no u većini slučajeva imaju izmijenjeni HTML kod, za ugodnije iskustvo korisnika aplikacije. Za prilagodljivi prikaz aplikacije na svim veličinama ekrana, korišteni su predlošci *Bootstrap* (za više informacija pogledati izvor [3]), a za ikone aplikacije *Font Awesome* (više informacija na [4]).



Slika 2.14: Direktorij *Views*

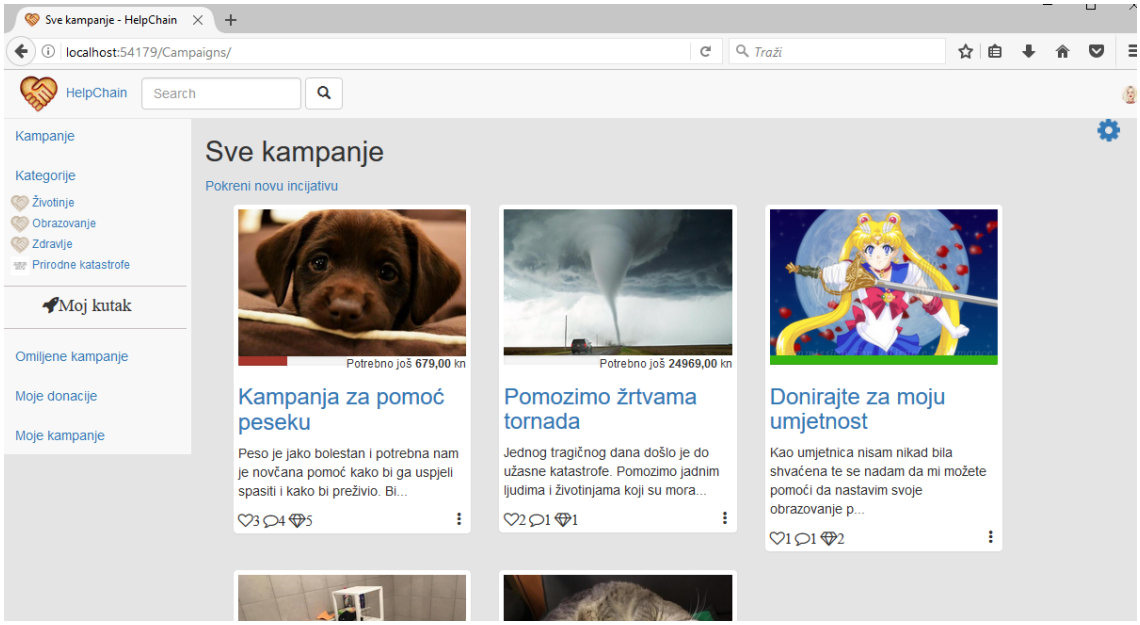
Na Slici 2.15 možemo vidjeti kako izgleda naslovna stranica aplikacije unutar web-preglednika. Administrator ima dodijeljena prava da upravlja centralnim dijelom naslovne stranice, koji prikazuje *Bootstrap Carousel* sa istaknutim kampanjama, a učitava se kao djelomični pogled (*partial view*) naziva *_Carousel*, koji se nalazi unutar direktorija ***/Views/Home***. Na desnoj strani od Carousela, nalaze se forme za registraciju novih i prijavu već postojećih korisnika. Za taj dio se u pozadini brinu *Identity Framework* te upravitelji *Manage* i *Account*. U gornjem lijevom uglu možemo vidjeti formu za pretraživanje kampanja. Ona vraća pogled *Search* unutar ***/Views/Home***, a rezultati su obrađeni i vraćeni uz pomoć istoimene metode *Search* koja pripada upravitelju *Home*. Ispod *Carousela* možemo uočiti odsječak naziva "Popularne kampanje". On se učitava kao djelomični pogled *_TopCampaigns*, a metoda unutar upravitelja *Home* je naziva *TopCampaigns*. Popularne kampanje se generiraju na način da se iz baze dohvate tri kampanje, za koje nije

skupljen dovoljan novčani iznos, a ističu se po broju komentara i broju stavljanja kampanje u omiljene.

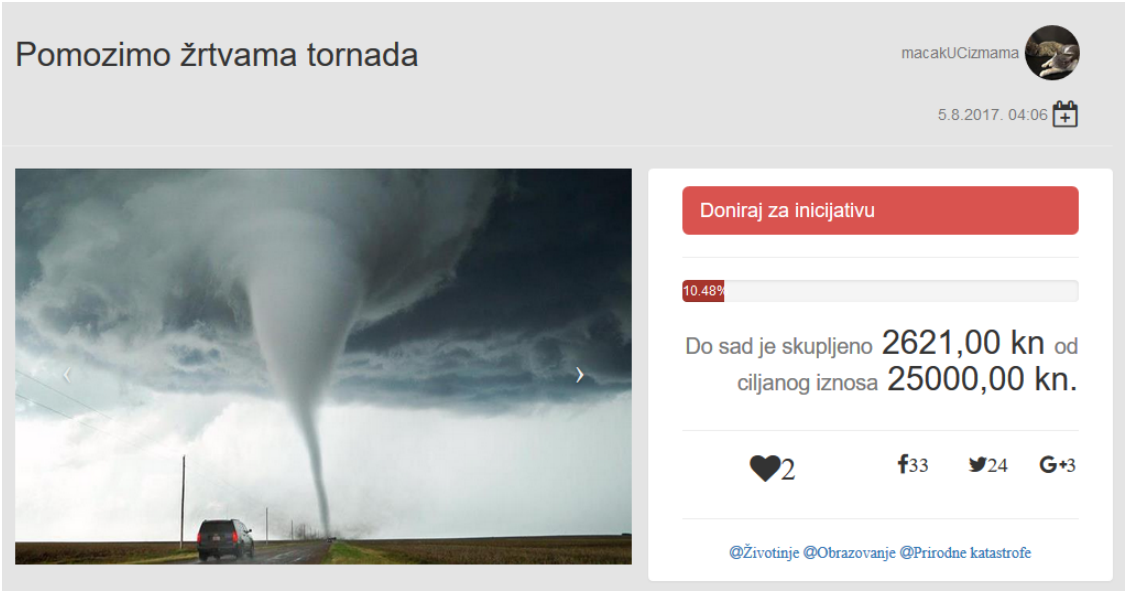


Slika 2.15: Naslovna stranica web-aplikacije

Na Slici 2.16 vidimo pogled ***Campaigns/Index***. Tu nam se učitavaju sve postojeće kampanje te njihovi pripadajući podaci. Direktno iz tog prikaza moguće je stvoriti novu kampanju, koja otvara pripadajući *Create* pogled, te pritiskom na vertikalne tri točkice u donjem desnom kutu pojedine kampanje, urediti istu ili ju obrisati. Ta prava imaju isključivo korisnici koji su kampanju stvorili te administratori aplikacije. S obzirom da je u primjeru sa Slike 2.16 korisnik ulogiran, možemo vidjeti i prošireni lijevi izbornik. Dodatno, jer je korisnik istovremeno i administrator, zdesna se nalazi plavi gumb koji omogućava pristup administratorskom dijelu stranice. U lijevom izborniku je dio naziva "Moj kutak" rezerviran za određenog prijavljenog korisnika, koji pritom dobiva uvid u kampanje koje je stavio u omiljene (pritiskom na ikonu srca na Slici 2.17), njegove dosadašnje donacije i njegove vlastite stvorene kampanje. Upravitelj u kojemu su implementirane pripadajuće metode jest upravitelj *Manage*, iz razloga jer je usko vezan uz pojedinog korisnika.

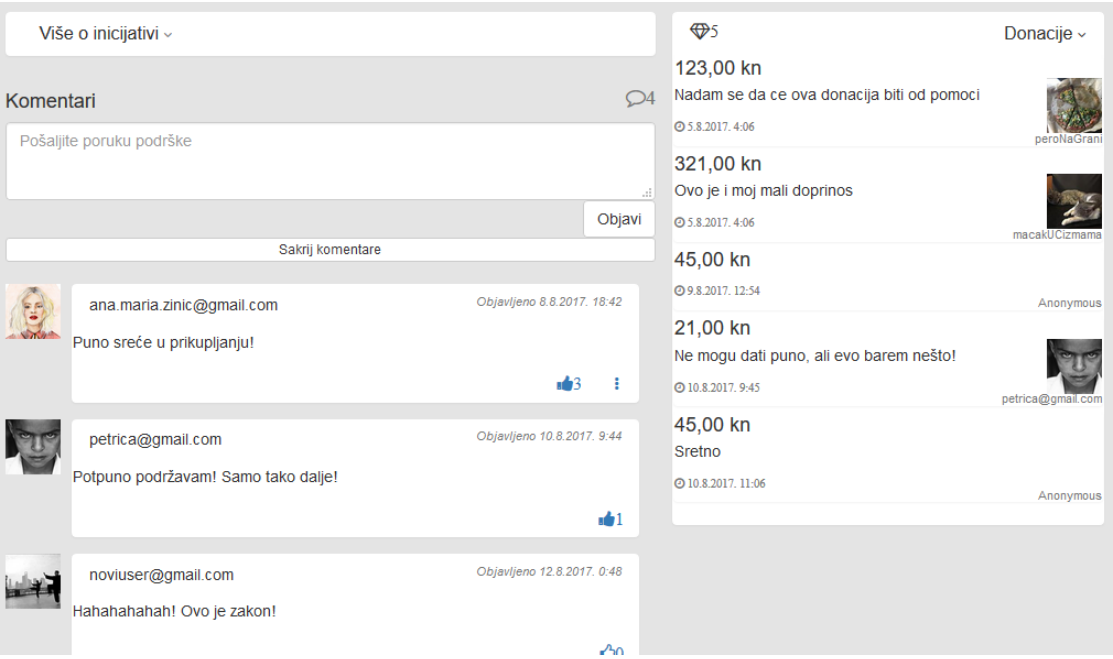


Slika 2.16: Pregled svih kampanja unutar web-preglednika



Slika 2.17: Prikaz detalja jedne kampanje (Prvi dio)

Naposlijetku, na Slikama 2.17 i 2.18 možemo vidjeti prikaz u web-pregledniku pojedine kampanje. Prva slika prikazuje gornji, dok druga slika donji dio pogleda. Pritiskom na "Doniraj za inicijativu", svaki ulogirani korisnik je u mogućnosti donirati određeni novčani iznos kampanji. Za upravljanje doniranjem je zadužena metoda *Donate* unutar upravitelja *Campaigns*. Donji dio pogleda daje uvid u komentare i dotadašnje donacije za kampanju. Ako je korisnik ulogiran, tada ima mogućnost i ostaviti komentar.



Slika 2.18: Prikaz detalja jedne kampanje (Drugi dio)

Dodatno

Premda je *Scaffold* automatsko generiranje koda u većini slučajeva bilo dovoljno, u određenim situacijama, osvježavanje cijele web-stranice za svaku akciju nije bilo poželjno. Iz tog razloga je za dodavanje, uređivanje, brisanje te "sviđanje" komentara, kao i za stavljanje kampanja u favorite, korišten *Ajax* za komunikaciju s upraviteljima. Na taj način smo mogli ažurirati podatke u bazi, a da smo pritom osvježili samo željeni dio otvorene stranice.

Javascript Jquery i *Ajax* funkcije nalaze se unutar projekta *Donation.Web.UI* unutar direktorija ***Scripts/App*** (prema konvenciji) te je naziv datoteke koja sprema skripte *DonationScripts.js*. Da bi sve te funkcije radile, prvo je bilo potrebno otvoriti *Package Manager Console* unutar glavnom izbornika pritiskom na *Tools* ⇒ *Nuget Package Manager*, te instalirati *Micosoft.JQuery.Unobtrusive.Ajax*. Nakon toga je unutar ***App_Start/BundleConfig.cs***

potrebno dodati reference na biblioteke *Ajax* i *jQuery* (vidi Sliku 2.19), kao i referencu na vlastite skripte. Naposlijetku treba dodati referencu i u *_Layout.cshtml*, tako da neposredno prije kraja <*body*> taga ubacimo

```
@Scripts.Render("~/bundles/myscripts")
```

```
public class BundleConfig
{
    public static void RegisterBundles(BundleCollection bundles)
    {
        bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
                    "~/Scripts/jquery-{version}.js",
                    "~/Scripts/jquery.unobtrusive-ajax.min.js"));

        bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
                    "~/Scripts/jquery.validate*"));

        bundles.Add(new ScriptBundle("~/bundles/myscripts").Include(
                    "~/Scripts/App/DonationScripts.js"));

        bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(
                    "~/Scripts/modernizr-*"));

        bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(
                  "~/Scripts/bootstrap.js",
                  "~/Scripts/respond.js"));

        bundles.Add(new StyleBundle("~/Content/css").Include(
                  "~/Content/bootstrap.css",
                  "~/Content/font-awesome.css",
                  "~/Content/site.css"));

    }
}
```

Slika 2.19: *BundleConfig.cs*

Zaključak poglavlja

U ovom poglavlju smo vidjeli kako je u *Visual Studiju* izgrađena složenija aplikacija za donacije, koristeći ASP.NET MVC 5 okvir i procedure pojašnjene u prvom dijelu rada. Pored toga, ukratko su objašnjene neke složenije procedure kojima smo došli do željenih karakteristika aplikacije, poput primjerice samo djelomičnog osvježavanja podataka na trenutno aktivnoj HTML stranici. Napravljen je pregled samo nekih elemenata web-aplikacije i objašnjeni neki detalji njezine implementacije. Cjeloviti izvorni kod je dostupan na CD-u priloženom uz ovaj rad.

Bibliografija

[1] K. S. Allen D. Matson J. Galloway, B.Wilson, *Professional ASP.NET MVC 5*, Wrox, 2014.

[2] *Web stranice platforme .NET MVC 5*, `https://www.asp.net/mvc/mvc5`, pristupljeno u kolovozu 2017.

[3] *Web stranice predložaka Bootstrap*, `http://getbootstrap.com/`, pristupljeno u kolovozu 2017.

[4] *Web stranice predložaka Font Awesome*, `http://fontawesome.io/`, pristupljeno u kolovozu 2017.

[5] *Izrada ViewModela za modele u odnosu mnogo naprama mnogo*, `https://goo.gl/rosCEh`, pristupljeno u kolovozu 2017.

[6] *Manipulacija slikama u MVC 5 i C#*, `goo.gl/3UgYN7`, pristupljeno u kolovozu 2017.

[7] *Uvod u Code-First pristup Entity Frameworka*, `http://www.entityframeworktutorial.net`, pristupljeno u kolovozu 2017.

Sažetak

U ovom diplomskom radu obrađena je problematika razvijanja web-aplikacija uz pomoć *Visual Studija*. Korišteni okvir jest okvir ASP.NET MVC 5, koji olakšava razvoj dinamičkih web-stranica, pritom primjenjujući uzorak Model-Pogled-Upravitelj (MVC).
MVC obuhvaća tri različita dijela aplikacije: Model predstavlja aplikacijsku logiku, Pogled definira korisničko sučelje aplikacije, dok Upravitelj brine o komunikaciji između Pogleda i Modela.

U prvom dijelu rada, razvijanjem jednostavne aplikacije banke krvi prikazan je proces kojim je moguće ostvariti funkcionalnu aplikaciju te su objašnjene konvencije, koje igraju veću ulogu unutar okvira, nego sama konfiguracija. U drugom dijelu diplomskog rada opisana je procedura razvoja složenije aplikacije za donacije, koja je izgrađena koristeći procedure uvedene u prvom dijelu rada. Svim registriranim korisnicima aplikacija omogućuje pokretanje kampanje u kojoj prikupljaju novce za određenu svrhu, ali i donaciju novca kampanjama.

Summary

In this master thesis, the problem of developing web-applications with the help of Visual Studio has been covered. The framework that has been used is ASP.NET MVC 5, which makes the development of dynamic web-pages easier, by implementing the Model-View-Controller (MVC) pattern. MVC is consists of three parts: the Model represents the application logic, the View defines the user interface of the application and the Controller takes care of the communication in between Views and Models.

The first part of the thesis explains the process of creating a functional application. This has been demonstrated through the development of the simple application for a blood bank. Besides that, the most important conventions have been explained, considering that conventions play a bigger role within this framework than the configuration. In the second part of the thesis, the process of developing a more complex web-application has been shown. The developed application is an application for donations. This application provides a possibility to start a fundraiser for a certain cause to all registered users, and also to support other campaigns by making their own donations.

Životopis

Ana Maria Žinić rođena je 9.11.1992. u Sisku. Ondje pohađa Osnovnu školu Braća Ribar te sudjeluje na natjecanjima iz matematike, fizike, te programiranja u Logo-u. U Sisku završava Osnovnu glazbenu školu Fran Lhotka te po završetku osnovne škole upisuje Prirodoslovno-matematičku gimnaziju u Sisku. Tijekom srednje škole sudjeluje na natjecanjima iz matematike te pohađa satove crtanja i dramske grupe. S Amaterskim kazalištem mladih sudjeluje na internacionalnim festivalima u Grenoble-u, u Francuskoj, i Oxfordu, u Velikoj Britaniji. Maturirala je 2011. godine, te obrazovanje nastavila na Prirodoslovno-matematičkom fakultetu u Zagrebu, upisujući Preddiplomski sveučilišni studij matematike. Studij završava 2014. godine, nakon čega upisuje Diplomski studij Računarstvo i matematika. Akademske godine 2016./2017. provodi dva semestra na Erasmus+ razmjeni na Sveučilištu u Beču, u Austriji, na Fakultetu za matematiku. Tijekom studija bila je dvije godine aktivna članica studentske udruge eSTUDENT, te članica Rotaract kluba Zagreb. Posljednju godinu studija u Zagrebu radila je nekoliko mjeseci kao frontend programerka u tvrtki Combis.