Web aplikacija za objavu i prijavu na natječaj za stipendije

Zorić, Dušan

Master's thesis / Diplomski rad

2018

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:200:170078

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

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 diplomski studij

WEB APLIKACIJA ZA OBJAVU I PRIJAVU NA NATJEČAJ ZA STIPENDIJE

Diplomski rad

Dušan Zorić

Osijek, 2018.

SADRŽAJ

1. UVOD 1
1.1 Zadatak diplomkog rada 2
2. VRSTE STIPENDIJA I PREDNOSTI RADA PUTEM APLIKACIJE 3
3. KORIŠTENE TEHNOLOGIJE 5
3.1. Opisni jezik HTML – HyperText Markup Language 5
3.1.1 Inačice HTML-a 5
3.1.2. Struktura HTML dokumenata 6
3.2. Stilski jezik CSS – Cascading Style Sheets 7
3.2.1. Sintaksa CSS-a 7
3.3. Objektno orijentirani programski jezik C# 8
3.3.1. Sintaksa C#-a 9
3.4. Entity Framework 10
3.5. ASP.NET MVC 11
3.5.1. Modeli 11
3.5.2. Pregledi 12
3.5.3. Kontroleri 12
4. PROGRAMSKO RIJEŠENJE WEB APLIKACIJE 13
4.1. Programske komponente 13
4.2. Baza podataka 16
4.3. Registracija i prijava korisnika 19
4.4. Popunjavanje prijavnice, potvrda i rezultati 23
5. TESTIRANJE I UPOTREBA APLIKACIJE 31
5.1. Upotreba prije prijave i registracijska forma 31
5.2. Prijavnica, potvrda i rezultati 34
5.3. Administratorsko sučelje 38
6. ZAKLJUČAK 41
LITERATURA 42
SAŽETAK 43
ŽIVOTOPIS 45

1. UVOD

Tema ovog diplomskog rada je izrada web aplikacije koja omogućuje objavu natječaja za stipendije, prijavu studenata za istu, te rangiranje studenata prilikom prijave. Rad se sastoji od praktičnog dijela izrade web aplikacije i pripadajuće dokumentacije. Web aplikaciji se pristupa putem jednog od mnogobrojnih internet preglednika. Jednostavnost pristupa ovakvoj vrsti aplikacija čini ih neizostavnima u svakodnevnom korištenju. Korištenje web aplikacija dostupno je svima, putem svih vrsta računala ili mobilnih telefona, stoga ne čudi značajan napredak u razvoju istih.

Objava natječaja započinje u listopadu. Ukoliko žele sudjelovati u natječaju za dodjelu stipendije, svi studenti trebaju osobno ispuniti molbu za dodjelu stipendije te priložiti tražene dokumente na za to predviđenom mjestu. Uz molbu za dodjelu stipendije, student je dužan priložiti presliku rodnog lista, kao i preslike potvrde statusa redovitog studenta, potvrde nadležnog tijela o prebivalištu te pisanu izjavu kandidata da nema zaključen ugovor o stipendiranju s drugim stipenditorom. Kriteriji na temelju kojih se odabire predloženik za stipendiju su: utvrđena deficitarnost zanimanja, opći uspjeh iz svih predmeta za prethodnu godinu obrazovanja, socijalni status, pohađanje druge škole odnosno paralelno studiranje, završavanje srednjoškolskog obrazovanja u vremenu kraćem od propisanog, objavljeni radovi, izlaganje studenata na znanstvenim skupovima, sudjelovanje i uspjeh studenata na natjecanjima i izložbama inovacija, osvojene nagrade, posebni kriteriji te prethodan status korisnika stipendije za vrijeme trajanja preddiplomskog ili specijalističkog stručnog studija.

U nastavku je obrađena analiza i usporedba dosadašnje metode objave i rangiranja stipendista s metodom objave i rangiranja istih putem web aplikacije. Nakon toga opisane su tehnologije koje su potrebne za provođenje istih. U pretposljednjem dijelu rada objašnjeni su pojedini dijelovi koda, stvaranje aplikacije i način primjene. Na kraju je prikazan i konačan rezultat stvaranja aplikacije te obavljena testiranja.

Analizom u radu ukazano je na dobrobiti korištenja ovakve aplikacije koja se može pokazati kao značajna potpora IT-ja (Information Technology) u sustavu rangiranja i objave stipendija. Pristupačnija, jeftinija i prijava u kraćem vremenskom roku prednost je korištenja u odnosu na prijašnje metode. Korisnicima je dovoljno imati pristup internetu kako bi kroz nekoliko minuta postupak bio gotov.

1.1 Zadatak diplomkog rada

Zadatak ovog rada je izraditi web aplikaciju koja objašnjava način objave uvjeta i prijavu na natječaj za dodjelu stipendija. Izraditi model baze podataka koja će moći podržati rad web aplikacije za objavu i prijave na takve natječaje. Dizajnirati korisničko sučelje kao i funkcionalnosti aplikacije. Opisati postupak izrade aplikacije i obaviti potrebna testiranja rada aplikacije.

2. VRSTE STIPENDIJA I PREDNOSTI RADA PUTEM APLIKACIJE

Stipendije su oblik financijske pomoći za studente tijekom njihova obrazovanja, profesionalnog usavršavanja ili istraživanja. Dodjeljuju se na temelju akademskog uspjeha, posebnih talenata, socijalnog statusa ili pripadnosti određenoj društvenoj ili etničkoj skupini.[1]

Programi stipendiranja odnose se na studijske programe koji uključuju postizanje određenog akademskog stupnja. Stipendije mogu biti potpune ili djelomične. Potpune stipendije pokrivaju troškove školarine, smještaja, hrane, osobnih troškova te dodatnih studijskih materijala, dok djelomične stipendije pokrivaju jedan ili više navedenih troškova, ali ne sve.

Stipendije dodjeljuje čitav niz institucija i organizacija iz svih sektora društva – vladinog, nevladinog i privatnog. Među njima su najčešće: visoka učilišta, tijela državne ili lokalne vlasti (vlade, ministarstva, županije, općine, gradovi), međunarodne organizacije, zaklade, nevladine organizacije, privatne tvrtke.[2]

Stipendije otvaraju pristup obrazovanju pojedincima koji to zaslužuju i onima kojima bez dodatnih financijskih sredstava obrazovanje ne bi bilo moguće. Stipendije ne samo da povećavaju pristup obrazovanju, nego također doprinose demokratizaciji visokog obrazovanja i povećanju broja visokoobrazovanih građana. Program stipendiranja direktno potiče suradnju između visokih učilišta i vladinih, nevladinih i privatnih sektora. Stipendije tako povezuju širu društvenu zajednicu s visokim učilištima omogućujući samim time naobrazbu kadra potrebnog u svim sektorima društva. Jedan od osnovnih ciljeva Bolonjske deklaracije je stvaranja visokoobrazovanog kadra, kako studenata, tako i istraživača i nastavnika, što je dodatno potaknuto stipendiranjem istih.

Prijava na natječaj zahtjeva prikupljanje određene dokumentacije. Ponekad se radi samo o dokumentima koje je potrebno ispuniti (prijavni obrasci), a ponekad dokumenti koje treba prikupiti (obrazovni dokumenti). Nerijetko prijava na natječaj zahtjeva i prilaganje osobno sastavljenih dokumenata (životopis, pismo motivacije, akademski esej, itd.).

Nakon pronalaska željenog natječaja, potrebno je proučiti njegove uvjete. Ukoliko su uvjeti zadovoljeni i korisnikove kvalifikacije odgovaraju uvjetima natječaja, dokumentacija treba biti prikupljena i osobno proslijeđena na za to predviđeno mjesto. Upravo se tu vidi razlika u efikasnosti i ekonomičnosti između takove vrste objave i prijave na natječaj i vrste predložene ovim radom. Posao koji rade zaposlenici uvelike bi bio olakšan činjenicom kako ne bi morali

rukovati s velikom količinom papira, već bi samo morali putem aplikacije provjeriti valjanost dokumenata priloženih od strane korisnika prijavljenog na natječaj. Korisnik bi tako uz prijavu priložio tražene dokumente, zaposlenici bi provjerili njihovu valjanost, a aplikacija bi nakon verifikacije dokumenata odradila posao rangiranja prijavljenih. Samim time rezultati natječaja bi bili gotovi ranije i prijavljeni korisnici bi ranije saznali jesu li u krugu izabranih za stipendiranje.

3. KORIŠTENE TEHNOLOGIJE

Tehnologije korištene za prikaz izgleda web aplikacije su Cascading Style Sheets (CSS) i Hypertext Markup Language (HTML). Algoritmi za spremanje podataka i računanje bodova izrađeni su pomoću objektno-orijentiranog programskog jezika C#. Za manipulaciju na strani korisnika (izvođenje operacije na strani korisnika/klijenta) korišten je ASP.NET MVC arhitekturalni dizajn, a okvir za upravljanje bazom podataka (MSSql) je Entity Framework.

3.1. Opisni jezik HTML – HyperText Markup Language

Skraćenica HTML ima korijen u engleskom jeziku, od riječi HyperText Markup Language. HyperText označava dio teksta koji ima ulogu poveznice. Markup Language predstavlja način unosa izgleda informacija unutar dokumenta. Razlog popularizacije i općeprihvaćenosti HTML-a je jednostavno rukovanje i lako učenje. Besplatan je i dostupan svima. Prikaz ovih dokumenata omogućen je internet pretraživačima.[3] Osnovna zadaća HTML jezika je uputiti internet pretraživač kako prikazati hypertext dokument bez obzira na platformu na kojoj se koristi. HTML nije programski jezik, on služi za opis hipertekstualnih dokumenata.

3.1.1 Inačice HTML-a

Prva inačica pojavila se 1990. Godine. Potpuno odvajanje sadržaja od dizajna omogućeno je pojavom CSS-a (Cascadin Style Sheets-a). HTML-u je vraćena namjena označavanja i određivanja sadržaja, a ne izgleda istog. [4]

Novije inačice HTML-a su:

HTML 4.01	1999.
XHTML 1.0	2000.
HTML5	2012.
XHTML5	2013.

3.1.2. Struktura HTML dokumenata

Ekstenzija HTML datoteka je *.html* ili *.htm*. HTML datoteke su obične tekstualne datoteke, svaki HTML dokument sastoji se od HTML elemenata – građevnih blokova. HTML element sastoji se od para HTML oznaka . Svaki element može imati atribute kojim se definiraju svojstva tog elementa. Na početku HTML dokumenta postavlja se <!DOCTYPE> element, kojim se označava DTD (engl. *Document Type Definition*), čime se definira točna inačica standarda koja se koristi za izradu HTML dokumenta. [4]

Elementom <html> označava se početak HTML dokumenta. Unutar <html> elementa nalaze se elementi <head> i <body>. Element <head> je zaglavlje HTML dokumenta u kojemu se specificiraju jezične značajke HTML dokumenta kao i naslov stranice. Pomoću određenih HTML elemenata unutar zaglavlja mogu se dodati i stilska obilježja stranice i to na dva načina, direktno ili kao referenca na vanjski CSS dokument. Unutar zaglavlja često se još definiraju i skripte kreirane u JavaScript jeziku. U elementu <body> nalazi se sadržaj HTML dokumenta, odnosno, stranice koju on reprezentira. HTML oznake počinju znakom < (manje od), a završavaju znakom > (više od). Zatvarajuća HTML oznaka kreira se na isti način kao i otvarajuća, ali se prije završnog znaka < dodaje kosa crta /.

```
<!DOCTYPE html>  deklaracija html dokumenta
<html>
    <head>
        <title>Naziv</title>    zaglavlje
    </head>
    <body>
        <p>Sadržaj</p>          sadržaj
    </body>
</html>
```

Sl.3.1. *Struktura jednostavnog HTML dokumenta.*

3.2. Stilski jezik CSS – Cascading Style Sheets

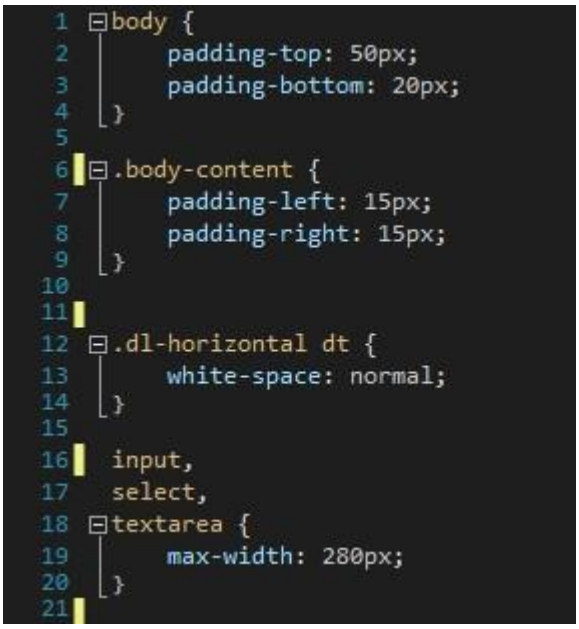
CSS kratica dolazi iz engleskog naziva Cascading Style Sheets. CSS je stilski jezik, koji se koristi za opis prezentacije dokumenta napisanih pomoću *markup* (HTML) jezika. [5]
Razvojem web-a, u HTML su ubacivani elementi za definiciju prezentacije, brzo je uočena potreba za stilskim jezikom koji će HTML rasteretiti potrebe prikazivanja sadržaja i njegovog oblikovanja (što je uloga CSS-a danas). Izgled i raspored stranice uređuju se CSS-om. Pomoću CSS-a je definirano kako se HTML elementi prikazuju (slike, slova, linkovi itd.). Stilovi se spremaju u odvojene datoteke izvan HTML dokumenta. Odvajanje CSS-a od HTML-a poboljšalo je dostupnost sadržaja, pružilo veću fleksibilnost i kontrolu kod prikazivanja obilježja. Omogućeno je dijeljenje istog stila oblikovanja između više html dokumenata kreiranjem .css datoteke koja se pozove u svakom html dokumentu.

3.2.1. Sintaksa CSS-a

Sintaksa CSS-a se razlikuje od sintakse HTML-a. Sastoji se samo od tri dijela:

```
1  selektor {
2      svojstvo : vrijednost;
3  }
```

Selektor je element u html-u kojem se želi dodijeliti vrijednost svojstva. Svojstvo je oznaka koja definira koje svojstvo se mijenja, a vrijednost je stil koji se dodjeljuje svojstvu. Selektori mogu imati više svojstava. Svojstva i vrijednosti odvajaju se dvotočkom i nalaze se unutar vitičastih zagrada. Iza vrijednosti svojstva nalazi se graničnik točka-zarez koji odvaja svojstva. Višestruke vrijednosti jednog svojstva odvojene su zarezom, a ako vrijednost svojstva ima više od jedne riječi stavlja se unutar dvostrukih navodnika. [6]



Sl. 3.2. *Primjer CSS datoteke.*

3.3. Objektno orijentirani programski jezik C#

C# je Microsoftov programski jezik, razvijen kao konkurencija Sun-ovom Java jeziku. C# je objektno orijentirani programski jezik baziran na XML web servisima na .NET platformi i dizajniran za poboljšanje produktivnosti u izradi web aplikacija. Početni naziv za C# bio je „COOL“, značenja C-like Object Oriented Language, a taj naziv promijenjen je nakon samo godinu dana. C# programski je jezik koji je izradio Andres Hejlsberg i njegov tim 1999. godine. Sintaksa samog C# jezika dosta je slična sintaksi programskog jezika C++, a važno je i napomenuti da je C# modernizirana inačica istog. Prednosti korištenja C# jezika prilikom izrade web aplikacija su u tome što koristi .NET okvir, što znači da pristupa brojnim aplikacijskim okvirima. U C# dodane su i neke nove vrste podataka, primjer je decimalna vrsta podataka koja služi obavljanju financijskih kalkulacija što značajno olakšava dio posla. [7]

Sigurnost C# jezika leži u tome što nije dozvoljeno upotrebaljavanje neinicijaliziranih varijabli, dok je, na primjer, u C++ jeziku lako provjeriti vrijednost varijable koja je prethodno proglašena. Na taj način, prikazalo bi se ono što se nalazi u memorijskoj adresi koja je dana tim varijablama, što onemogućuje ispravno korištenje aplikacije. Prilikom kompiliranja C# obavještava pokušaj upotrebe varijable prije njene inicijalizacije na neku ispravnu vrijednost.

3.3.1. Sintaksa C#-a

U C#-u postoji čitav niz različitih tipova podataka. Podaci se dijele na dvije skupine, value tipove i reference tipove. Value tipovi podataka imaju nekakav oblik nule za nultu vrijednost, dok reference tipovi mogu biti *null*, odnosno prazni (ništa). Kada se govori o value tipovima podataka valja ih podijeliti na nekoliko skupina, to su cijelobrojni tipovi podataka, na primjer, *byte, short, int, long* itd., realni tipovi podataka, odnosno *float, double* i *decimal*, te logički tip podataka, odnosno *bool* koji prima samo vrijednosti *true* ili *false*. Kada se govori o reference tipovima podataka, oni uključuju podatke tipa *string, array, class* i *delegates*. [8]

```
static void Main()
        {
            string dan = "ponedjeljak";
            int a = 7;
            Console.WriteLine(dan);
            Console.WriteLine(a);
            Console.Read();
        }
```

Sl. 3.3. *Primjer korištenja string i int tipa podataka te ispis istih unutar funkcije Main u konzolnoj aplikaciji C# programa.*

Prilikom pisanja koda u C#-u neizbježno je korištenje operatora. Operatori se dijele na operator pridruživanja, aritmetičke operatore, relacijske operatore, logičke operatore, operatore inkrement i dekrement te operatore zadatka (Sl. 3.4.).

```
//operatori
        //Vrste operatora:
        //-operator pridruživanja =
        int a = 7;
        a = 8;
        //-aritmetički operatori + - * / % (modulo)
        int b = 4;
        int c = 2;
        int d = b + c;
        //-relacijski operatori < > <= >= == !=
        if(d==c)
        {
            Console.WriteLine("ovo je super");
        }
        //-logicki operatori && || !
        //logičko &&
        if(a==5 && b==20)
        {

            Console.WriteLine("jeeea");
        }
```

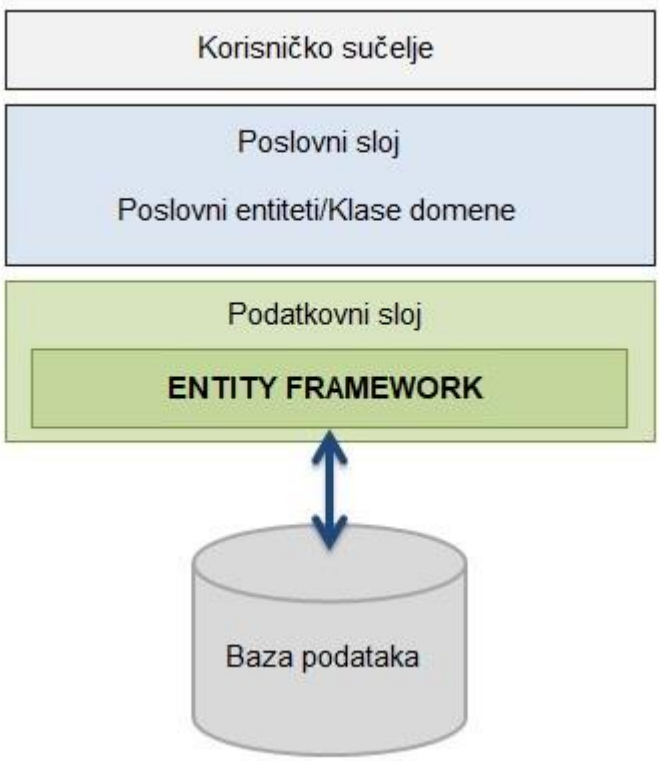
```
// || logičko ili
if (a < 1 || a > 5)
{
    Console.WriteLine("ne postoji takva ocjena");
}
//! logičko ne
bool ukljuceno = false;
if (!ukljuceno)
{
    Console.WriteLine("iskljuceno");
}
//-operator inkrement ++
int godine = 15;
godine++;
Console.WriteLine(godine);
//operator dekrement --
godine--;
Console.WriteLine(godine);
//operatori zadataka += -= *= /=
int tecaj = 5;
tecaj = tecaj + 5;
```

Slika 3.4. Vrste operatora u C#-u

3.4. Entity Framework

Entity Framework je O/RM (Object/Relational Mapper) okvir otvorenog tipa koji se koristi u .NET aplikacijama podržanim u Microsoftu i omogućuje developerima rad s bazama podataka koristeći .NET objekte. Eliminira potrebu pisanja koda za pristup podacima koji je obično potreban programeru. [9]

Svojevrsna je nadogradnja za ADO.NET. Microsoft je omogućio Entity Framework kako bi automatizirao sve procese kreiranja baza podataka i upravljanja istima. S Entity Frameworkom arhitekti i programeri mogu raditi na višoj razini apstrakcije kada rade s podacima, također mogu kreirati i održavati podatkovno-orijentirane aplikacije s manje koda u usporedbi s aplikacijama prije razvoja Entity Frameworka.



Sl. 3.5. *Princip rada Entity Frameworka.*

EF Verzija	Godina nastanka	.NET Framework
EF 6	2013	.NET 4.0 & .NET 4.5, VS 2012
EF 5	2012	.NET 4.0, VS 2012
EF 4.3	2011	.NET 4.0, VS 2012
EF 4.0	2010	.NET 4.0, VS 2010
EF 1.0 (or 3.5)	2008	.NET 3.5 SP1, VS 2008

Sl. 3.6. *Inačice i godine nastanka Entity Frameworka*

3.5. ASP.NET MVC

Model-View-Controller (MVC) arhitekturalni je dizajn, odnosno radni okvir Microsoft Web Framework-a, korišten prilikom izrade web stranica, usluga i aplikacija.

MVC odvaja aplikaciju na tri glavne komponente: model, pregled i kontroler (Sl. 3.7.). Okvir ASP.NET MVC pruža alternativu u odnosu na rad ASP.NET Web Formi prilikom kreiranja web aplikacija. ASP.NET MVC okvir visoko je testirani prezentacijski okvir integriran u već postojeće ASP.NET značajke kao što su „*master*“ stranice i ovjere bazirane na članovima. MVC okvir definiran je unutar System.Web.Mvc. Nekim aplikacijam ovaj okvir bit će izuzetno koristan, međutim određene vrste Web aplikacija nastavit će s korištenjem ASP.NET tradicionalnih dizajnova baziranih na Web Form-ama, kao što će neke vrste aplikacija kombinirano koristiti oba pristupa.[10]

MVC okvir sastoji se od sljedećih dijelova:

3.5.1. Modeli

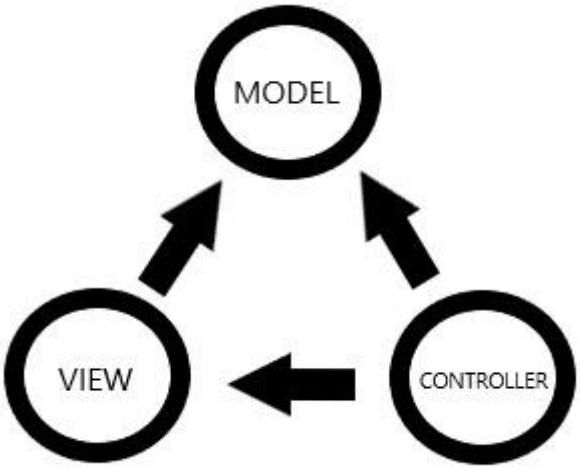
Objekti unutar modela dijelovi su aplikacije koji sadrže logiku domene podataka aplikacije. Ne rijetko, objekti modela spremaju i vraćaju stanje modela iz baze podataka. Na primjer, objekt može izvući informaciju iz baze podataka, raditi operacije s prikupljenim informacijama te zatim zapisati novu ažuriranu informaciju u bazu podataka.

3.5.2. Pregledi

Komponenta pregledi služi za prikaz aplikacije u korisničkom sučelju. Obično se korisničko sučelje kreira od podataka unutar modela. Primjer toga bio bi prikaz uređivanja tablice putem tekstualnih okvira, padajućih izbornika ili potvrtnih okvira što bi se temeljilo na trenutno stanju objekta proizvoda.

3.5.3. Kontroleri

Kontroleri su dio koji se bavi interakcijom s korisnikom, rade s modelima i odabiru pregled koji će se iscrtati u korisničkom sučelju. U MVC aplikaciji pregled se koristi samo za prikaz informacija korisniku, kontroler je zadužen i reagira na korisnikove unose i interakcije s aplikacijom. U praksi, kontroler upravlja nizom vrijednosti, prenosi te vrijednosti modelu, koji bi te vrijednosti mogao, na primjer, spremiti u bazu podataka.



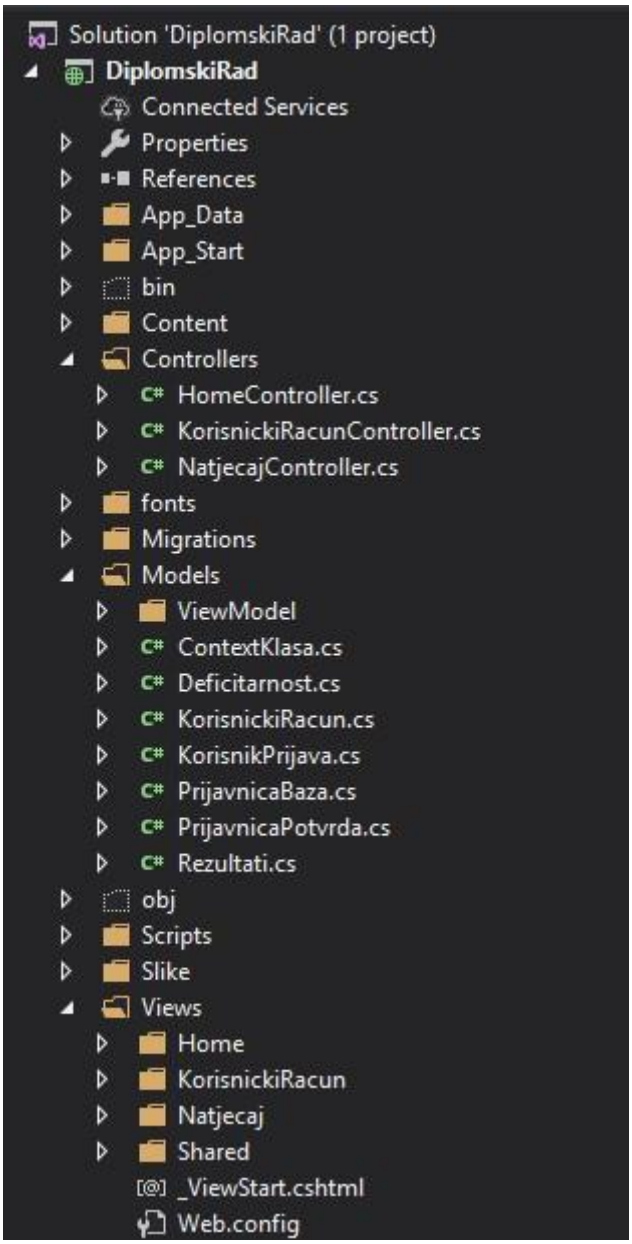
Sl. 3.7. *Komponente i princip rada ASP.NET MVC aplikacije*

4. PROGRAMSKO RIJEŠENJE WEB APLIKACIJE

Kako su u prethodnom poglavlju objašnjene tehnologije korištene prilikom izrade ovoga rada (poglavlje 3.), u nastavku će biti objašnjena programska riješenja i komponente web aplikacije. Prilikom izrade težilo se jednostavnosti i funkcionalnosti aplikacije te prilici za upoznavanje s osnovnim mogućnostima ASP.NET MVC okvira, njegovog načina funkcioniranja i principa rada.

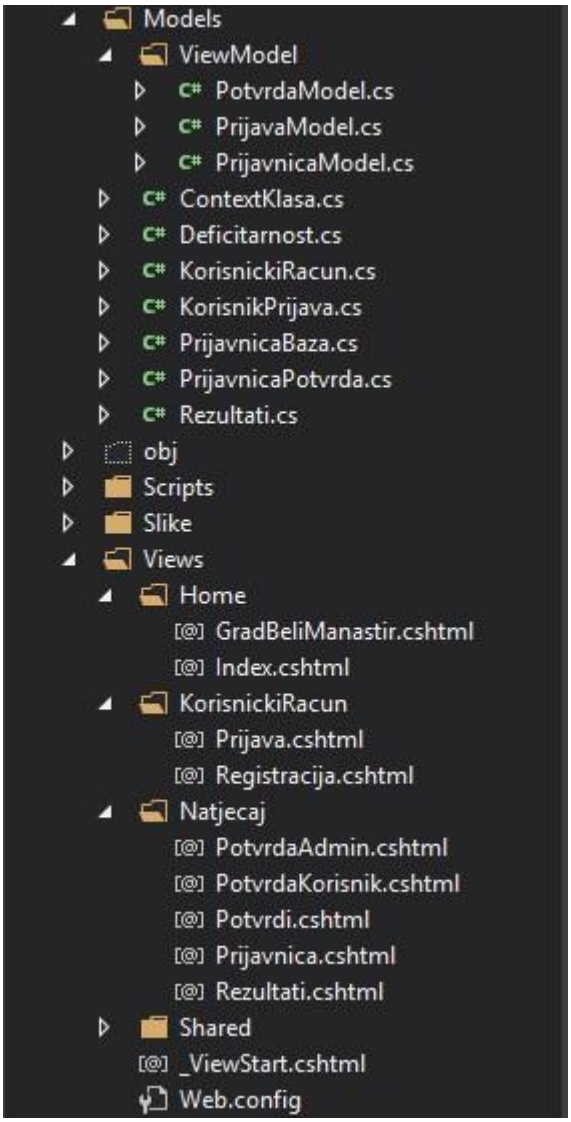
4.1. Programske komponente

Projekt (aplikacija) sastoji se od nekoliko datoteka (Sl. 4.1.). Naglasit će se samo najbitnije datoteke. Mapa *App_Data* sadrži bazu podataka koja se koristi (registriranog korisnika, korisnikovu prijavnicu, potvrdu prijavnice i rezultate nakon bodovanja), u mapi *Controllers* nalaze se korišteni kontroleri (*HomeController, KorisnickiRacunController te NatjecajController*), u mapi *Models* nalaze se korištene klase odnosno modeli podataka (*ContextKlasa* kao konekcija sa Entity Framework-om, klasa *Deficitarnost* čija funkcija je tipa enum, *KorisnickiRacun, KorisnikPrijava, PrijavnicaBaza, PrijavnicaPotvrda, Rezultati*), unutar mape *Models* također se nalazi mapa *ViewModels* koja sadrži klase (*PotvrdaModel, PrijavaModel i PrijavnicaModel*) (Sl. 4.2.), korištene za pretvorbu tipa podataka koje unosi korisnik. Mapa *Views* sadrži preglede organizirane u nekoliko podmapa (Sl. 4.2.). Mapa *Home*, generirana automatski razvojnim okruženjem, mapa *KorisnickiRacun* (s pregledima vezanim za sustav prijave i registracije korisnika), *Natjecaj* (s pregledima vezanim za popunjavanje prijavnice na natječaj, potvrdu iste te rezultate), također postoji i *Shared* mapa s dijeljenim pregledima koja sadrži *Layout* zajednički dio pregleda svih ostalih datoteka i pregleda.



Sl. 4.1. *Komponente riješenja web aplikacije*

Izgled odnosno ponašanje određenih elemenata aplikacije sadržano je u mapama *Content* i *Scripts*. Biblioteke za *CSS* dizajn, među kojima i *bootstrap* biblioteka, nalaze se u mapi *Content*, a gotove *JavaScript* funkcije u mapi *Scripts.*



Sl. 4.2. *ViewModel, Home, KorisnickiRacun, Natjecaj i Shared datoteke*

Prema metodama unutar *HomeControllera* stvoreni pregledi u mapi ***Home***. *Index* pregled za prikaz početne stranice aplikacije, odnosno *GradBeliManastir* za prikaz informacija i kontakta u gradskoj upravi. Sukladno tome unutar mape ***KorisnickiRacun*** stvoreni su pregledi prema metodama korištenim u kontroleru *KorisnickiRacunController*, a njihovi nazivi su *Prijava*¸ odnosno *Registracija*, a prikazuju korisničko sučelje registracije odnosno prijave novog, odnosno postojećeg korisnika. U mapi *Natjecaj* prema metodama pisanim u ***NatjecajController*** nalaze se prikazi: *PotvrdaAdmin* koji će administratorskom računu prikazati korisnikove informacije i omogućiti potvrđivanje prijavnice, *PotvrdaKorisnik* za prikaz informacije korisniku o statusu predane potvrde, *Potvrdi* za prikaz dokumenata korisnika, *Prijavnica*, za prikaz prijavnice i *Rezultati* za listu rezultata.

4.2. Baza podataka

Kako bi se pohranjivali i dohvaćali podaci potrebni za rad aplikacije, stvorena je baza podataka. U realizaciji programskog rješenja pisani su modeli podataka, što znači da je korišten *Code-first* pristup. Kreirane su klase *KorisnickiRacun*, *PrijavnicaBaza*, *PrijavnicaPotvrda* i *Rezultati*. Modeli predstavljaju podatke za registraciju korisnika, podatke upisane u prijavnicu, je li prijavnica potvrđena i rezultate, odnosno broj bodova koje je kandidat prikupio. Sadržaj klasa prikazan je slikama 4.3., 4.4., 4.5. i 4.6., klase su pisane u programskom jeziku C#.

```
public class KorisnickiRacun
{
    [Key]
    7 references
    public int KorisnikID { get; set; }
    [Display(Name = "Ime")]
    [Required(AllowEmptyStrings = false, ErrorMessage = "Ime je obavezno")]
    2 references
    public string Ime { get; set; }
    [Display(Name = "Prezime")]
    [Required(AllowEmptyStrings = false, ErrorMessage = "Prezime je obavezno")]
    2 references
    public string Prezime { get; set; }
    [Display(Name = "Korisničko ime")]
    [Required(AllowEmptyStrings = false, ErrorMessage = "Korisničko ime je obavezno")]
    3 references
    public string KorisnickoIme { get; set; }
    [Display(Name = "E-mail")]
    [Required(AllowEmptyStrings = false, ErrorMessage = "E-mail je obavezan")]
    [DataType(DataType.EmailAddress)]
    7 references
    public string Email { get; set; }
    [Display(Name = "Datum rođenja")]
    [DataType(DataType.Date)]
    [DisplayFormat(ApplyFormatInEditMode = true, DataFormatString = "{0:dd/MM/yyyy}")]
    1 reference
    public DateTime DatumRodjenja { get; set; }
    [Required(AllowEmptyStrings = false, ErrorMessage = "Lozinka je obavezna")]
    [DataType(DataType.Password)]
    [MinLength(6, ErrorMessage = "Potrebno minimalno 6 znakova")]
    4 references
    public string Lozinka { get; set; }
    [Display(Name = "Potvrdi lozinku")]
    [DataType(DataType.Password)]
    [Compare("Lozinka", ErrorMessage = "Lozinke se ne poklapaju")]
    3 references
    public string PotvrdiLozinku { get; set; }

    3 references
    public string Role { get; set; } = "Korisnik";

    10 references
    public virtual PrijavnicaBaza Prijavnica { get; set; }

    11 references
    public virtual PrijavnicaPotvrda PrijavnicaPotvrda { get; set; }
}
```

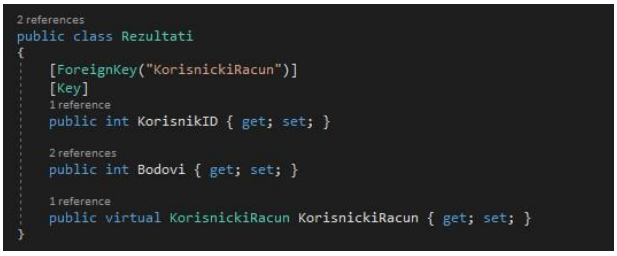
Sl. 4.3. *Klasa KorisnickiRacun*



Sl. 4.4. *Klasa PrijavnicaBaza*



Sl. 4.5. *Klasa PrijavnicaPotvrda*



Sl. 4.6. *Klasa Rezultati*

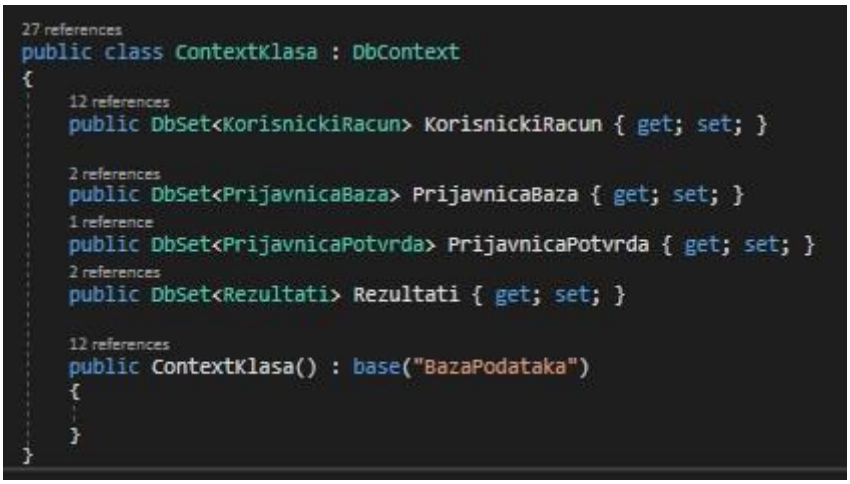
U klasi *KorisnikRegistracija* (Sl. 4.3.) nalaze se podaci koji predstavljaju stupce u tablici baze podataka. Imena stupaca jednaka su imenima svojstava unutar klase pa se tako od korisnika traži unos imena, prezimena, korisničkog imena, adrese e-pošte, datuma rođenja, zaporke, odnosno potvrde zaporke, a automatski mu se dodaje uloga korisnika. U klasi se također nalaze svojstva *PrijavnicaBaza* i *PrijavnicaPotvrda* koji su strani ključevi koji upućuju na istoimene klase, odnosno povezuju se s klasom *KorisnikRegistracija* u odnosu jedan-na-jedan (*eng. one-to-one*). Osim navedenih svojstava, nužno je napomenuti da prvi stupac u tablici pokazuje na svojstvo *KorisnikID* što je primarni ključ ove tablice. Primarni ključ je jedinstven i povećava se automatski sa svakim novim dodanim korisnikom, što znači da osigurava integritet tablice. Primarni ključ ne smije se obrisati jer pokazuje na redak u drugoj tablici.

Klasa *PrijavnicaBaza* (Sl. 4.4.) sadrži svojstva koja će korisniku biti prikazana na način da će trebati unijeti podatke o prosjeku ocjena, deficitarnosti, broju članova kućanstva, visini godišnjeg dohotka svih članova kućanstva zajedno, živi li s oba roditelja, studira li paralelno na drugom fakultetu i je li primao stipendiju grada u prošlosti. Ove podatke student će potkrijepiti dokazima u vidu dokumenata koji će se spremiti u svojstva koja su za to zadužena (državljanstvo, status redovitog studenta, potvrda o prebivalištu, prijepis ocjena). Kao što je već navedeno u prethodnom paragrafu, posljednje svojstvo strani je ključ koji pokazuje na ID korisnika u tablici *KorisnickiRacun*.

Klasa *PrijavnicaPotvrda* (Sl. 4.5.) sadrži podatke koji se prikazuju korisnicima s ulogom administratora. Klasa sadrži strani ključ na ID registriranog korisnika, te dohvaća njegove dokumente i omogućuje administratoru potvrdu istih.

Klasa *Rezultati* (Sl. 4.6.) također sadrži strani ključ na ID registriranog korisnika i stvara stupac u tablici u koji će se zapisati broj bodova korisnika nakon što ih algoritam zbroji te će se ispisati u vidu tablice rezultata u korisničkom sučelju.

Kako bi relacije između navedenih modela bile valjane, potrebno je kreirati kontekstnu klasu *ContextKlasa.cs* koja će naslijediti svojstva klase *DbContext* i u kojoj će se objediniti navedeni modeli (Sl. 4.7.). To će omogućiti *ORM-u* (Entity Framework) rad u *Code-First* pristupu.



Sl. 4.7. *Kontekstna klasa*

4.3. Registracija i prijava korisnika

Za pristup sadržaju aplikacije i prijavu na natječaj korisnik mora obaviti registraciju. Nakon što pravilno ispuni podatke u registracijskoj formi, automatski biva registriran i daje mu se mogućnost prijave. Svakom korisniku dodjeljuje se jedinstveni ID koji se automatski povećava prilikom svake sljedeće registracije. Dio unutar ASP.NET MVC okvira koji obavlja ove funkcije naziva se kontroler (potpoglavnje 3.5.). U kontroleru *KorisnickiRacun* kreirane su dvije akcijske metode s jednakim nazivom *Registracija()*, ali s drukčijim atributima. Atribut *[HttpGet]* označava akciju koja će se izvršiti prilikom poziva adrese (*URL*) s istoimenim kontrolerom i akcijskom metodom. Metoda s ovim atributom kao rezultat vraća istoimeni pregled (Sl. 4.8.). Druga akcijska metoda s istim imenom sadrži *[HttpPost]* (Sl. 4.9.) atribut ili oznaku, a njena funkcija je dohvatiti podatke koje korisnik unese u svom sučelju, te ih poslati odgovarajućem modelu koji je spojen s tablicom u bazi podataka i sprema unesene podatke. Navedena akcijska metoda prvo će izvršiti provjeru postoji li korisnikova adresa e-pošte u bazi podataka na način da će pozvati metodu *EmailPostoji* (Sl. 4.10.). U slučaju postojanja, korisniku će se pokazati odgovarajuća poruka. Korisnikova zaporka bit će kriptirana pozivom klase *RasprsivanjeLozinke.cs* unutar koje se nalazi metoda *Rasprsivanje* (Sl. 4.11), nakon čega će se podaci spremiti u bazu podataka, a korisniku prikazati odgovarajuća poruka.

```
[HttpGet]
0 references
public ActionResult Registracija()
{
    return View();
}
```

Sl.4.8. *Akcijska metoda Registracija s HttpGet atributom*

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public ActionResult Registracija(KorisnickiRacun racun)
{
    bool status = false;
    string poruka = "";

    if(ModelState.IsValid)
    {
        var mailPostoji = EmailPostoji(racun.Email);

        if (mailPostoji)
        {
            ModelState.AddModelError("EmailPostoji", "Email već postoji");
            return View(racun);
        }

        racun.Lozinka = RasprsivanjeLozinke.Rasprsivanje(racun.Lozinka);
        racun.PotvrdiLozinku = RasprsivanjeLozinke.Rasprsivanje(racun.PotvrdiLozinku);

        using (ContextKlasa db = new ContextKlasa())
        {
            db.KorisnickiRacun.Add(racun);
            db.SaveChanges();
        }

        poruka = racun.Ime + " " + racun.Prezime + ", uspješno ste se registrirali.";
        status = true;
    }
    else
    {
        poruka = "Zahtjev nije valjan";
    }

    ViewBag.Message = poruka;
    ViewBag.Status = status;
    return View(racun);
}
```

Sl. 4.9. *Akcijska metoda Registracija s HttpPost atributom*

Metoda *EmailPostoji* koristi atribut *NonAction* kako bi joj se onemogućio pristup putem web adrese. Ova metoda tipa je *boolean* što znači da vraća vrijednost istinito ili neistinito. Metoda će provjeriti unesenu adresu e-pošte korisnika prilikom registracije, te vidjeti postoji li

takava adresa, na način da će dohvatiti adrese svih registriranih korisnika i usporediti ih s unesenom vrijednošću (Sl. 4.10.).

```
[NonAction]
1 reference
public bool EmailPostoji (string email)
{
    using (ContextKlasa db = new ContextKlasa())
    {
        var v = db.KorisnickiRacun.Where(a => a.Email == email).FirstOrDefault();
        return v != null;
    }
}
```

Sl. 4.10. *EmailPostoji metoda*

Klasa *RasprsivanjeLozinke.cs* sadrži metodu *Rasprsivanje* koja kao ulazni parametar prima vrijednost koju je korisnik unio kao zaporku prilikom registracije. Metoda pretvara vrijednost u podatak tipa *byte*, a zatim raspršuje zaporku koristeći *SHA-256* (Secure Hash Algorithm 256) nakon čega tu kriptiranu vrijednost vraća kao izlazni parametar (Sl. 4.11.).

```
5 references
public static class RasprsivanjeLozinke
{
    5 references
    public static string Rasprsivanje(string vrijednost)
    {
        return Convert.ToBase64String(
            System.Security.Cryptography.SHA256.Create()
            .ComputeHash(Encoding.UTF8.GetBytes(vrijednost))
            );
    }
}
```

Sl. 4.11. *Klasa RasprsivanjeLozinke*

Prilikom prijave od korisnika se zahtjeva unos svoje adrese e-pošte i zaporke. Akcijska metoda *Prijava* potom će izvršiti provjeru korisnikovih podataka na način da će dohvatiti podatke registriranih korisnika iz baze podataka te ih usporediti s unesenim podacima. Ukoliko korisnik unese ispravne podatke i odabere opciju da aplikacija zapamti njegovu prijavu, korisniku će se spremiti kolačić koji će mu omogućiti pregled stranice bez ponovne prijave u

rasponu od minimalno 20 minuta do najviše godinu dana. U slučaju neispravnih podataka korisniku će se prikazati poruka o neispravnosti unesenih podataka (Sl. 4.12.).

```
[HttpGet]
0 references
public ActionResult Prijava()
{
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
0 references
public ActionResult Prijava(KorisnikPrijava prijava, string PovratniUrl)
{
    string poruka = "";
    using (ContextKlasa db = new ContextKlasa())
    {
        var v = db.KorisnickiRacun.Where(a => a.Email == prijava.Email).FirstOrDefault();
        if (v != null)
        {
            if (string.Compare(RasprsivanjeLozinke.Rasprsivanje(prijava.Lozinka),v.Lozinka) == 0)
            {
                int timeout = prijava.ZapamtiPodatke ? 525600 : 20;
                var karta = new FormsAuthenticationTicket(prijava.Email, prijava.ZapamtiPodatke, timeout);
                string enkriptirano = FormsAuthentication.Encrypt(karta);
                var kolacic = new HttpCookie(FormsAuthentication.FormsCookieName, enkriptirano);
                kolacic.Expires = DateTime.Now.AddMinutes(timeout);
                kolacic.HttpOnly = true;
                Response.Cookies.Add(kolacic);

                if (Url.IsLocalUrl(PovratniUrl))
                {
                    return Redirect(PovratniUrl);
                }
                else
                {
                    return RedirectToAction("Index", "Home");
                }
            }
            else
            {
                poruka = "Podaci nisu ispravni";
            }
        }
        else
        {
            poruka = "Podaci nisu ispravni";
        }
    }
    ViewBag.Message = poruka;
    return View();
}
```

Sl. 4.12. *Akcijska metoda Prijava*

Ukoliko se korisnik želi odjaviti iz svog računa, pozvat će se akcijska metoda *Odjava*. Ovu metodu neće biti moguće pozvati, nit će biti prikazana korisniku ukoliko korisnik nije

prijavljen, zbog atributa *[Authorize]*, a po pozivu, korisnika će preusmjeriti na pregled za prijavu (Sl. 4.13.).

```
[Authorize]
[HttpPost]
0 references
public ActionResult Odjava()
{
    FormsAuthentication.SignOut();
    return RedirectToAction("Prijava", "KorisnickiRacun");
}
```

Sl. 4.13. *Akcijska metoda Odjava*

4.4. Popunjavanje prijavnice, potvrda i rezultati

Po uspješno obavljenoj registraciji i prijavi, korisniku se nudi opcija popunjavanja prijavnice za natječaj. Prilikom otvaranja prikaza za prijavnicu, aplikacija će provjeriti u bazi podataka postoji li već popunjena prijavnica za trenutno prijavljenog korisnika. Ukoliko je korisnik već popunio svoju prijavnicu, bit će preusmjeren na prikaz potvrde, gdje će mu se prikazati odgovarajuća poruka. Ukoliko korisnik nije ispunio prijavnicu, akcijska metoda *Prijavnica* (Sl. 4.14.) iscrtat će formu prijavnice koju korisnik mora ispuniti ranije navedenim podacima (pododlomak 4.2.).

```
public ActionResult Prijavnica()
{
    using (ContextKlasa db = new ContextKlasa())
    {
        try
        {
            var korisnickiRacun = db.KorisnickiRacun.Include(pb => pb.Prijavnica).First(kr => kr.Email == User.Identity.Name);
            if (korisnickiRacun.Prijavnica != null)
            {
                return RedirectToAction("PotvrdaKorisnik", "Natjecaj");
            }
            PrijavnicaModel model = new PrijavnicaModel();
            model.KorisnikID = korisnickiRacun.KorisnikID;
            return View(model);
        }
        catch (Exception ex)
        {
            return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
        }
    }
}
```

Sl. 4.14. *Akcijska metoda Prijavnica (GET)*

Kada korisnik unese odgovarajuće podatke, provjeri ih i spremi svoju prijavnicu, pozvat će se akcijska metoda *Prijavnica* s atributom *[HttpPost]* koja će prikupiti unesene podatke, spremiti ih u bazu podataka nakon čega će preusmjeriti korisnika na prikaz o statusu njegove prijavnice (Sl. 4.15.). Prilikom spremanja datoteka s *.pdf* ekstenzijom u bazu podataka iste će biti pretvorene u podatak tipa *byte* pomoću klase *PDFConverter* (Sl. 4.16.).

```
[HttpPost]
0 references
public ActionResult Prijavnica(PrijavnicaModel model)
{
    if (ModelState.IsValid)
    {
        try
        {
            if (model.KorisnikID != 0)
            {
                PrijavnicaBaza prijava = new PrijavnicaBaza
                {
                    Deficitarnost = model.Deficitarnost,
                    BrojClanovaKucanstva = model.BrojClanovaKucanstva,
                    Drzavljanstvo = PDFConverter.FromHttpPostFileToByteArray(model.Drzavljanstvo.InputStream),
                    ParalelnoStudiranje = model.ParalelnoStudiranje,
                    ZiviSObaRoditelja = model.ZiviSObaRoditelja,
                    Prebivaliste = PDFConverter.FromHttpPostFileToByteArray(model.Prebivaliste.InputStream),
                    ProsjekOcjena = model.ProsjekOcjena,
                    PrimaoStipendiju = model.PrimaoStipendiju,
                    ProsjekOcjenaPDF = PDFConverter.FromHttpPostFileToByteArray(model.ProsjekOcjenaPDF.InputStream),
                    StatusRedovitogStudenta = PDFConverter.FromHttpPostFileToByteArray(model.StatusRedovitogStudenta.InputStream),
                    VisinaDohotka = model.VisinaDohotka,
                    KorisnikID = model.KorisnikID
                };
                using (ContextKlasa db = new ContextKlasa())
                {
                    db.PrijavnicaBaza.Add(prijava);
                    db.SaveChanges();

                }
            }
        }
        catch (Exception ex)
        {
            return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
        }

    }
    return RedirectToAction("PotvrdaKorisnik", "Natjecaj");
}
```

Sl. 4.15. *Akcijska metoda Prijavnica (POST)*

```
4 references
public static class PDFConverter
{
    4 references
    public static byte[] FromHttpPostFileToByteArray(Stream stream)
    {
        MemoryStream ms = stream as MemoryStream;
        if (ms == null)
        {
            ms = new MemoryStream();
            stream.CopyTo(ms);
        }
        return ms.ToArray();
    }
}
```

Sl. 4.16. *Klasa PDFConverter*

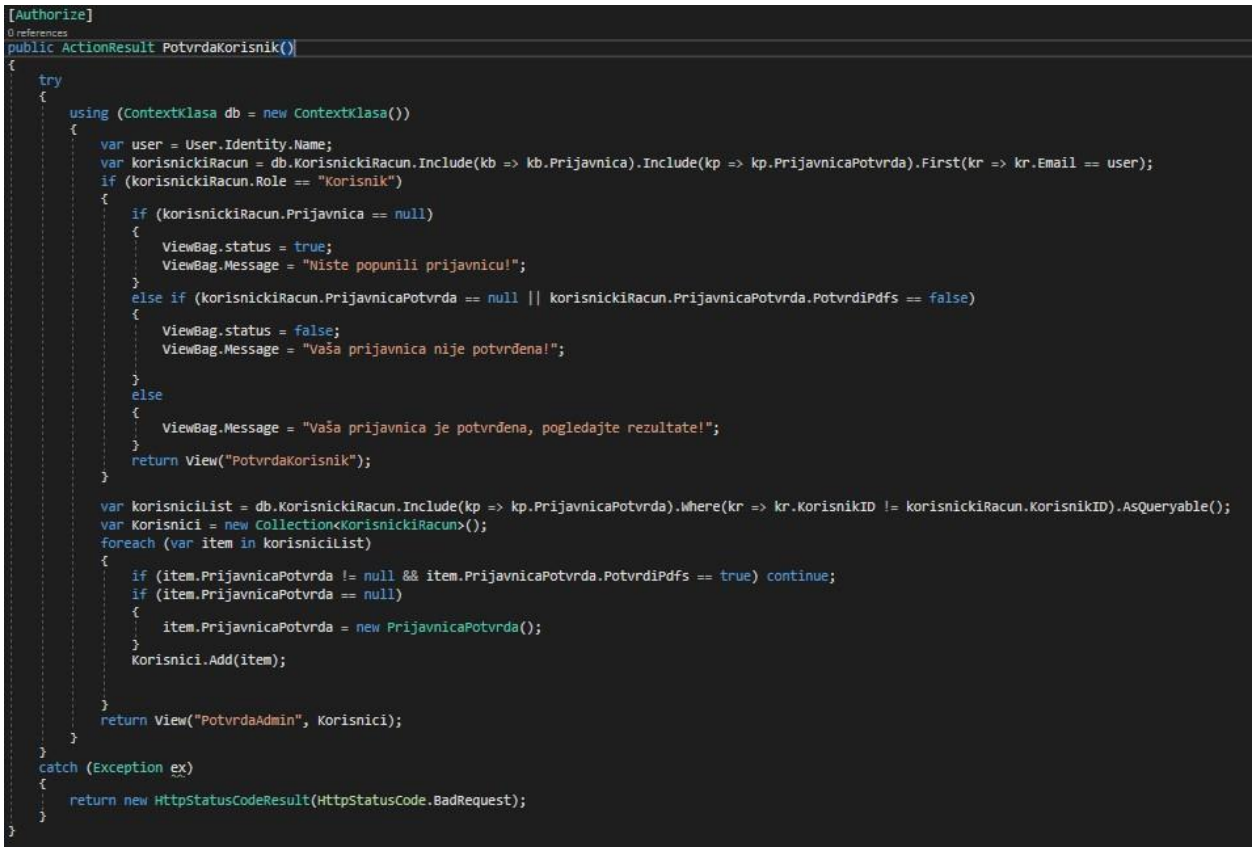
Nakon spremanja podataka korisniku se onemogućuje ponovni prikaz prijavnice i umjesto toga preusmjeren je na prikaz statusa prijavnice. Akcijska metoda koja se poziva tom prilikom naziva se *Potvrda*. Ova metoda dohvaća korisnikovu ulogu u bazi podataka, ukoliko se radi o korisniku, metoda će istog preusmjeriti na pregled naziva *PotvrdaKorisnik*. Ukoliko je trenutno prijavljeni korisnik s ulogom administratora, ista metoda će spremiti sve korisnike čije prijavnice nisu potvrđene u listu i izlistati ih administratoru kako bi ih mogao provjeriti i potvrditi (Sl. 4.17.).

```
[Authorize]
0 references
public ActionResult Potvrda()
{
    using (ContextKlasa db = new ContextKlasa())
    {
        var user = User.Identity.Name;
        var korisnickiRacun = db.KorisnickiRacun.First(kr => kr.Email == user);
        if (korisnickiRacun.Role == "Korisnik")
        {
            return View("PotvrdaKorisnik");
        }

        var korisniciList = db.KorisnickiRacun.Include(kp => kp.PrijavnicaPotvrda).Where(kr => kr.KorisnikID != korisnickiRacun.KorisnikID).AsQueryable();
        var Korisnici = new Collection<KorisnickiRacun>();
        foreach (var item in korisniciList)
        {
            if (item.PrijavnicaPotvrda == null) item.PrijavnicaPotvrda = new PrijavnicaPotvrda();
        }
        return View("PotvrdaAdmin", Korisnici);
    }
}
```

Sl. 4.17. *Akcijska metoda Potvrda*

Po otvaranju prikaza statusa prijavnice, pozvat će se akcijska metoda *PotvrdaKorisnik*. Ova metoda iz baze podataka dohvaća korisnikove do sad unesene podatke nakon čega provjerava ulogu trenutno ulogiranog korisnika. Ukoliko se radi o korisniku koji se prijavljuje na natječaj, petlja će provjeriti status njegove prijavnice i prikazati mu odgovarajuću poruku. Ukoliko je korisnik sa svojstvima administratora, u kolekciju će se spremiti podaci o statusu korisnikove prijavnice. Postoji li prijavnica i ukoliko nije potvrđena, administratoru će se prikazati popis prijavnica koje nije potvrdio (Sl. 4.18.).



```
[Authorize]
0 references
public ActionResult PotvrdaKorisnik()
{
    try
    {
        using (ContextKlasa db = new ContextKlasa())
        {
            var user = User.Identity.Name;
            var korisnickiRacun = db.KorisnickiRacun.Include(kb => kb.Prijavnica).Include(kp => kp.PrijavnicaPotvrda).First(kr => kr.Email == user);
            if (korisnickiRacun.Role == "Korisnik")
            {
                if (korisnickiRacun.Prijavnica == null)
                {
                    ViewBag.status = true;
                    ViewBag.Message = "Niste popunili prijavnicu!";
                }
                else if (korisnickiRacun.PrijavnicaPotvrda == null || korisnickiRacun.PrijavnicaPotvrda.PotvrdiPdfs == false)
                {
                    ViewBag.status = false;
                    ViewBag.Message = "Vaša prijavnica nije potvrđena!";

                }
                else
                {
                    ViewBag.Message = "Vaša prijavnica je potvrđena, pogledajte rezultate!";
                }
                return View("PotvrdaKorisnik");
            }

            var korisniciList = db.KorisnickiRacun.Include(kp => kp.PrijavnicaPotvrda).Where(kr => kr.KorisnikID != korisnickiRacun.KorisnikID).AsQueryable();
            var Korisnici = new Collection<KorisnickiRacun>();
            foreach (var item in korisniciList)
            {
                if (item.PrijavnicaPotvrda != null && item.PrijavnicaPotvrda.PotvrdiPdfs == true) continue;
                if (item.PrijavnicaPotvrda == null)
                {
                    item.PrijavnicaPotvrda = new PrijavnicaPotvrda();
                }
                Korisnici.Add(item);


            }
            return View("PotvrdaAdmin", Korisnici);
        }
    }
    catch (Exception ex)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
}
```

Sl. 4.18. *Akcijska metoda PotvrdaKorisnik*

Nakon izlistanog popisa korisnika kojima prijavnice još nisu potvđene, administrator, ukoliko odluči pregledati nečije dokumente može to učiniti na način da pozove akcijsku metodu *Potvrdi* klikom na za to odgovarajući gumb u prikazu. Ova metoda prikazat će administratoru dokumente korisnika čiju prijavnicu želi pregledati na način da iz baze podataka dohvati podatke korisnika čiju prijavnicu pregledava. Potom će se instancirati novi objekt u modelu *PrijavaModel* (Sl. 4.20.) gdje će se podaci koji su u bazi podataka spremljeni pod tipom varijable *byte* dohvatiti i pretvoriti u tip podatka *string* što će administratoru omogućiti pregled dokumenata (Sl. 4.19.).

```
0 references
public ActionResult Potvrdi(int id)
{
    try
    {
        using (ContextKlasa db = new ContextKlasa())
        {
            var korisnik = db.KorisnickiRacun.Include(kb => kb.Prijavnica).First(kr => kr.KorisnikID == id);
            if (korisnik.Prijavnica == null)
            {
                ViewBag.status = false;
                return View("Potvrdi");
            }
            PrijavaModel model = new PrijavaModel
            {
                ProsjekOcjenaPDF = string.Format("data:application/pdf; base64,{0}", Convert.ToBase64String(korisnik.Prijavnica.ProsjekOcjenaPDF)),
                Prebivaliste = string.Format("data:application/pdf; base64,{0}", Convert.ToBase64String(korisnik.Prijavnica.Prebivaliste)),
                Drzavljanstvo = string.Format("data:application/pdf; base64,{0}", Convert.ToBase64String(korisnik.Prijavnica.Drzavljanstvo)),
                StatusRedovitogStudenta = string.Format("data:application/pdf; base64,{0}", Convert.ToBase64String(korisnik.Prijavnica.ProsjekOcjenaPDF)),
                potvrda = new PotvrdaModel { KorisnikID = korisnik.KorisnikID }
            };

            return View(model);
        }
    }
    catch (Exception ex)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
}
```

Sl. 4.19. *Akcijska metoda Potvrdi (GET)*

```
2 references
public class PrijavaModel
{
    1 reference
    public string ProsjekOcjenaPDF { get; set; }

    1 reference
    public string Drzavljanstvo { get; set; }

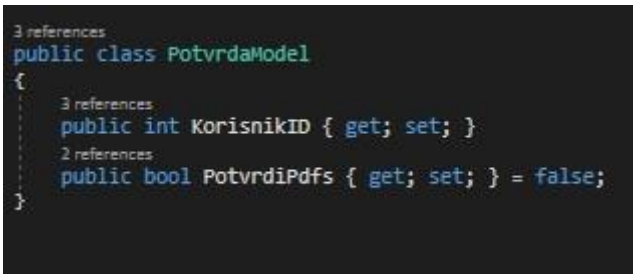
    1 reference
    public string StatusRedovitogStudenta { get; set; }

    1 reference
    public string Prebivaliste { get; set; }

    1 reference
    public PotvrdaModel potvrda { get; set; }
}
```

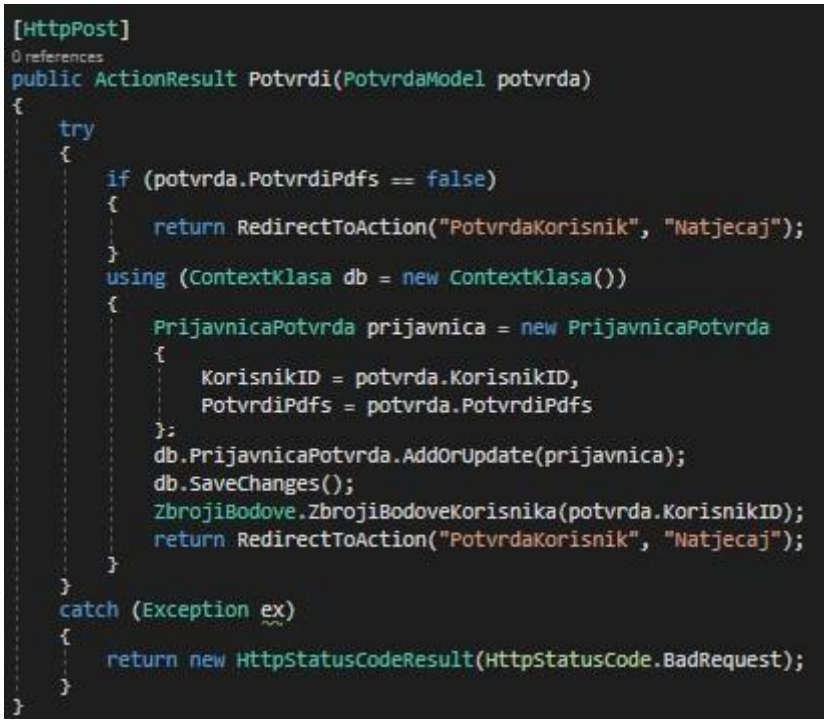
Sl. 4.20. *Klasa PrijavaModel*

Ukoliko administrator, nakon pregledavanja korisnikovih dokumenata, ustanovi da su svi podaci koje je korisnik unio ispravni, označi ih ispravnima i potvrdi, podaci će biti spremljeni u klasu *PotvrdaModel* (Sl. 4.21.). Akcijska metoda koja će se izvršiti potvrđivanjem dokumenata naziva se *Potvrdi* (Sl. 4.23.) i sadrži *HttpPost* atribut. Iz baze podataka instancirat će se novi objekt iz klase *PrijavnicaPotvrda* te će se administratorov unos spremiti u taj objekt. Objekt će se zatim dodati bazi podataka, što pokreće algoritam za zbrajanje bodova u klasi *ZbrojiBodove.cs* (Sl. 4.22.), upisuje bodove korisnika te vraća administratora na prikaz još uvijek ne potvrđenih prijavnica.



```
3 references
public class PotvrdaModel
{
    3 references
    public int KorisnikID { get; set; }
    2 references
    public bool PotvrdiPdfs { get; set; } = false;
}
```

Sl. 4.21. *Klasa PotvrdaModel*



```
[HttpPost]
0 references
public ActionResult Potvrdi(PotvrdaModel potvrda)
{
    try
    {
        if (potvrda.PotvrdiPdfs == false)
        {
            return RedirectToAction("PotvrdaKorisnik", "Natjecaj");
        }
        using (ContextKlasa db = new ContextKlasa())
        {
            PrijavnicaPotvrda prijavnica = new PrijavnicaPotvrda
            {
                KorisnikID = potvrda.KorisnikID,
                PotvrdiPdfs = potvrda.PotvrdiPdfs
            };
            db.PrijavnicaPotvrda.AddOrUpdate(prijavnica);
            db.SaveChanges();
            ZbrojiBodove.ZbrojiBodoveKorisnika(potvrda.KorisnikID);
            return RedirectToAction("PotvrdaKorisnik", "Natjecaj");
        }
    }
    catch (Exception ex)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
}
```

Sl.4.22. *Akcijska metoda Potvrd (POST)*

Algoritam zbrajanja bodova radi na jednostavan način (Sl. 4.23.). Prosjek ocjena koji je korisnik unio množi se sa sto, dobije se okrugli broj na ukupan broj bodova, tome se pridodaju još različite vrijednosti u ovisnosti o prihodu i broju članova kućanstva korisnika, deficitarnosti korisnikove struke, paralelnom studiranju na drugom fakultetu, živi li s oba roditelja i je li primao stipendiju prethodnih godina. Ukupna suma svih bodova spremit će se u bazu podataka kako bi se mogla prikazati prilikom objave rezultata.

```
public static class ZbrojiBodove
{
    1 reference
    public static void ZbrojiBodoveKorisnika(int id)
    {
        using (ContextKlasa db = new ContextKlasa())
        {
            int bodovi = 0;

            var prijavnica = db.PrijavnicaBaza.First(kr => kr.KorisnikID == id);

            float prihodPoGlavi = prijavnica.VisinaDohotka / prijavnica.BrojClanovaKucanstva;
            float prosjekSto = prijavnica.ProsjekOcjena * 100;

            if (prijavnica.ParalelnoStudiranje == true)
                bodovi += 100;
            if (prijavnica.Deficitarnost != Deficitarnost.NištaOdNavedenog)
                bodovi += 50;
            if (prihodPoGlavi <= 10000)
            {
                bodovi += 200;
            }
            else if (prihodPoGlavi > 10000 && prihodPoGlavi <= 20000)
            {
                bodovi += 150;
            }
            else if (prihodPoGlavi > 20000 && prihodPoGlavi <= 30000)
            {
                bodovi += 100;
            }
            else if (prihodPoGlavi > 30000 && prihodPoGlavi <= 40000)
            {
                bodovi += 50;
            }
            else if (prihodPoGlavi > 40000)
            {
                bodovi += 10;
            }

            if (prijavnica.PrimaoStipendiju == false)
                bodovi += 50;

            if (prijavnica.ZiviSObaRoditelja == false)
            {
                bodovi += 100;
            }
            else
            {
                bodovi += 20;
            }
            int x = Convert.ToInt32(prosjekSto);
            bodovi += x;
            db.Rezultati.AddOrUpdate(new Rezultati { Bodovi = bodovi, KorisnikID = id });
            db.SaveChanges();
        }
    }
}
```

Sl. 4.23. *Algoritam za zbrajanje bodova prijavljenih korisnika*

Na poslijetku, po potvrdi svih valjanih dokumenata prijavljenih korisnika od strane administratora, korisnicima se omogućuje pregled rezultata. Pregled rezultata obavit će se pozivom akcijske metode *Rezultati*. Ova metoda jednostavno će spremiti broj bodova svih

korisnika, poredati ih padajućim redom po broju bodova i kao izlaznu varijablu ju poslati na pregled što korisniku omogućuje uvid u stanje na tablici rezultata (Sl. 4.24.).

```
[Authorize]
0 references
public ActionResult Rezultati()
{
    try
    {
        using (ContextKlasa db = new ContextKlasa())
        {
            var rezultatiList = db.Rezultati.Include(kr => kr.KorisnickiRacun).OrderByDescending(bod => bod.Bodovi).ToList();
            return View(rezultatiList);
        }
    }
    catch (Exception ex)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
}
```

Sl. 4.24. *Akcijska metoda Rezultati*

Uloga korisnika i administratora riješena je vrlo jednostavnom metodom. Unutar mape *Migrations* nalazi se klasa *Configuration.cs*. Ova klasa pokreće se prilikom prvog pokretanja aplikacije i služi za kreiranje baze podataka. Unutar klase nalazi se *Seed* metoda, u su zapisani podaci o administratorskom računu. To znači da će se prilikom pokretanja programa u bazi podataka, u tablici *KorisnickiRacun*, na prvo mjesto upisati upravo podaci administratorkog računa (Sl. 4.25.). Sve što je potrebno nakon toga je korisnicima s administratorskim ovlastima, dati te podatke.

```
0 references
protected override void Seed(DiplomskiRad.Models.ContextKlasa context)
{
    if(!context.KorisnickiRacun.Any())
    {
        KorisnickiRacun korisnicki = new KorisnickiRacun()
        {
            DatumRodjenja = DateTime.Today,
            Email = "admin@gmail.com",
            Ime = "Admin",
            KorisnickoIme = "Admin",
            Lozinka = RasprsivanjeLozinke.Rasprsivanje("password"),
            PotvrdiLozinku = RasprsivanjeLozinke.Rasprsivanje("password"),
            Prezime = "Admin",
            Role = "Admin",
        };
        context.KorisnickiRacun.Add(korisnicki);
        context.SaveChanges();
    }
}
```

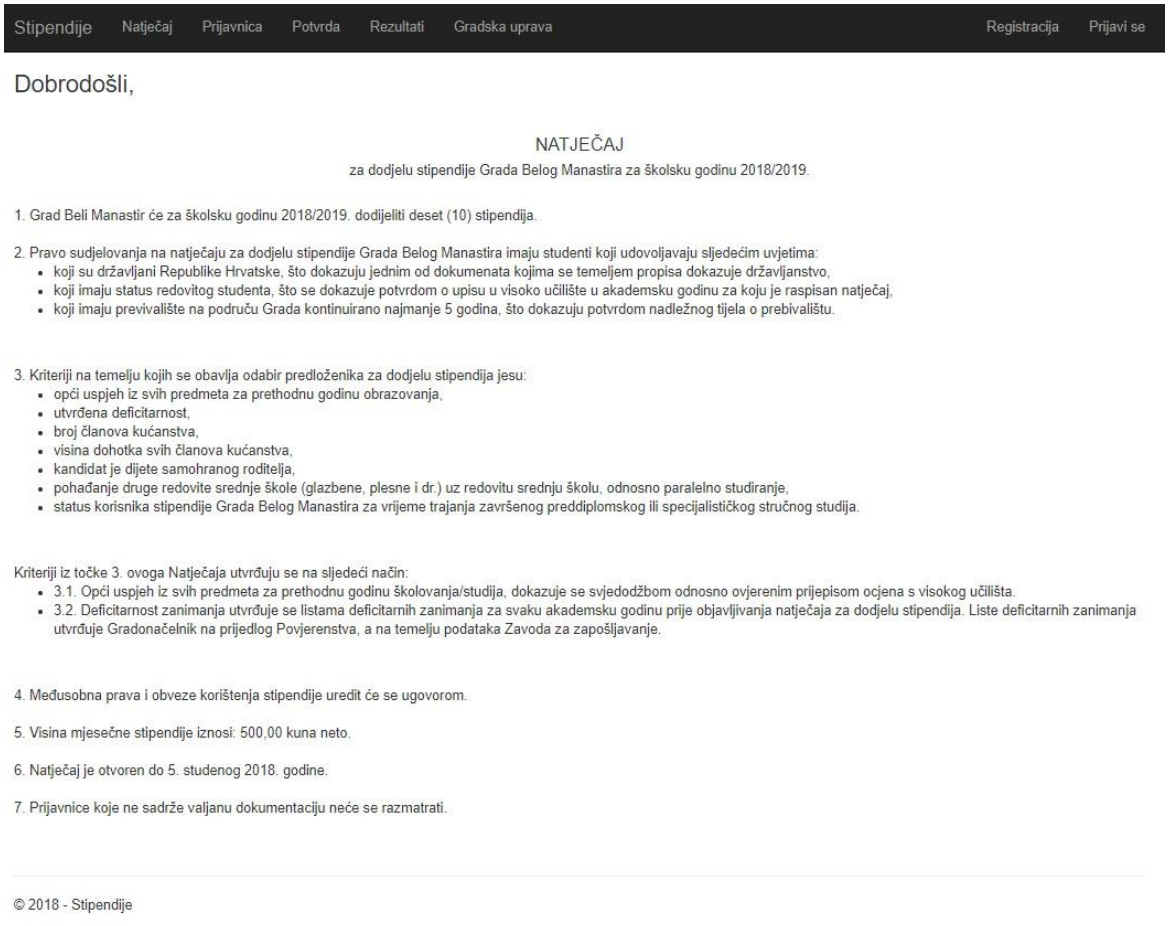
Sl. 4.25. *Seed metoda unutar Configuration klase*

5. TESTIRANJE I UPOTREBA APLIKACIJE

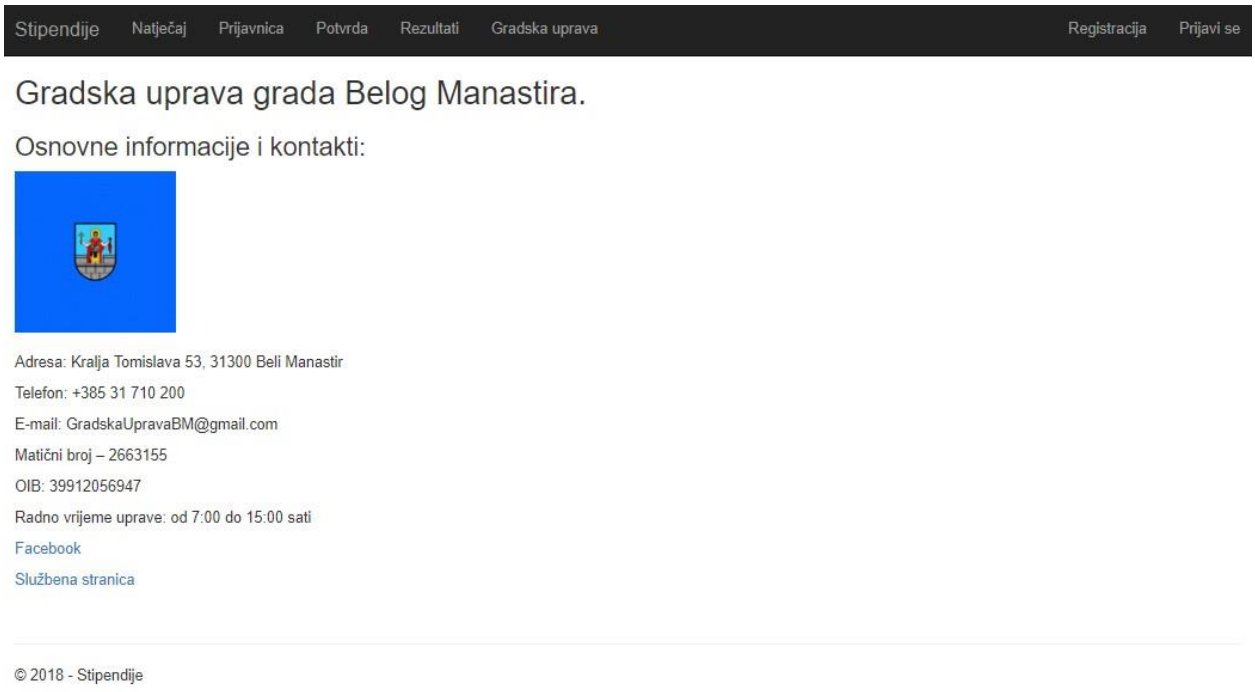
U ovom poglavlju objasnit će se način upotrebe aplikacije. U nastavku će biti prikazani pogledi korisničkog sučelja prije same prijave, nakon prijave i administratorsko sučelje.

5.1. Upotreba prije prijave i registracijska forma

Prilikom pokretanja aplikacije natječaja za stipendije korisniku se automatski prikazuje početna stranica (pregled *Index*) (Sl. 5.1.). Korisniku je na raspolaganju izbornička traka, ukoliko korisnik nije registriran i prijavljen, bit će u mogućnosti pregledati samo početnu stranicu i stranicu *Gradska uprava* (pregled *GradBeliManastir*) (Sl. 5.2.).

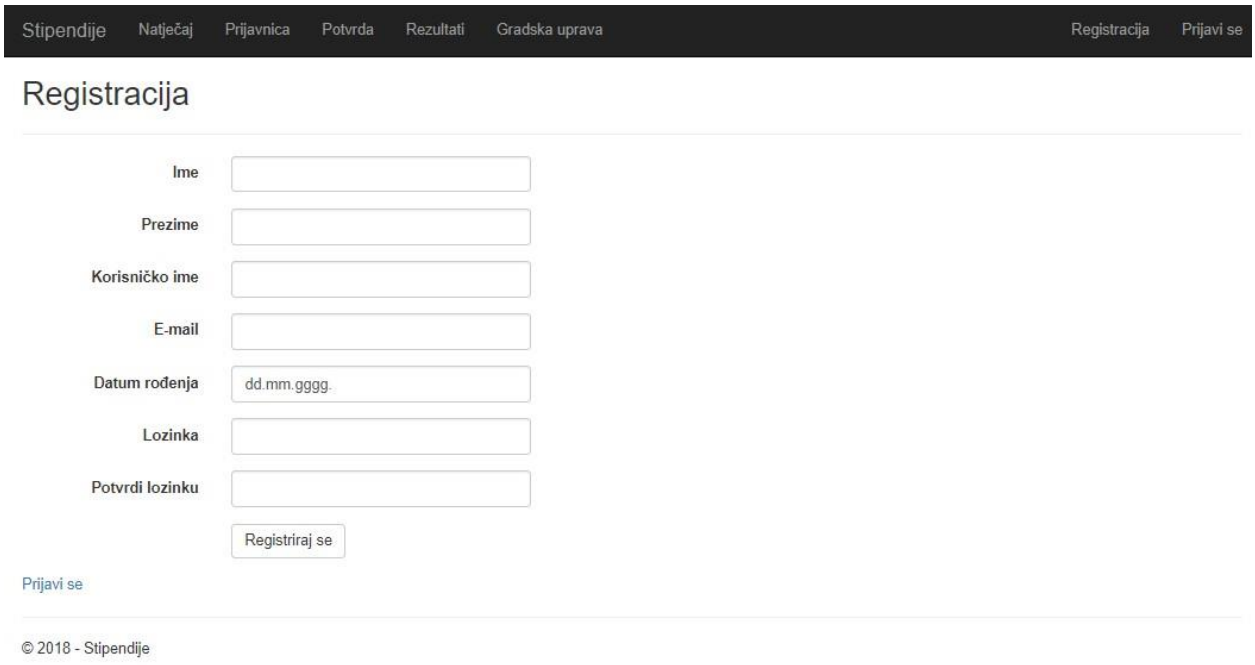


Sl. 5.1. *Početna stranica aplikacije*

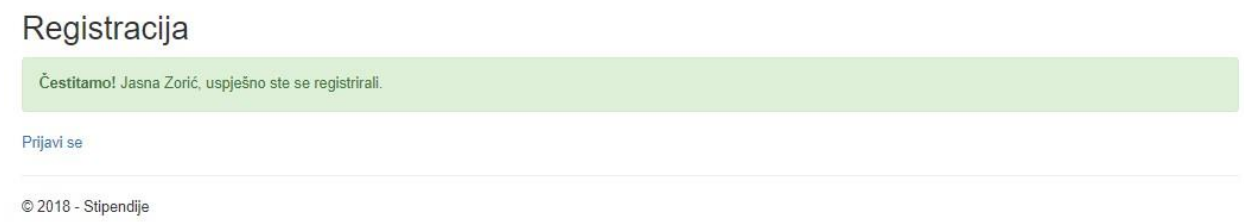


Sl. 5.2. *Stranica s informacijama o gradskoj upravi*

Klikom na gumb *Registracija*, korisniku se prikazuje stranica za registraciju (pregled *Registracija*). Prikazana mu je forma s potrebnim poljima koja mora ispuniti (Sl. 5.3.). Korisnik zatim unosi ime, prezime, korisničko ime, adresu e-pošte, datum rođenja, zaporku i potvrdu zaporke. Ukoliko korisnik upotrijebi već postojeću adresu e-pošte, bit će o tome obaviješten. Korisniku neće biti dozvoljen nastavak ukoliko svi podaci ne budu uneseni ispravno. Kada korisnik ispravno ispuni sve tražene podatke o uspješnosti registracije bit će obaviješten odgovarajućom porukom (Sl. 5.4.).

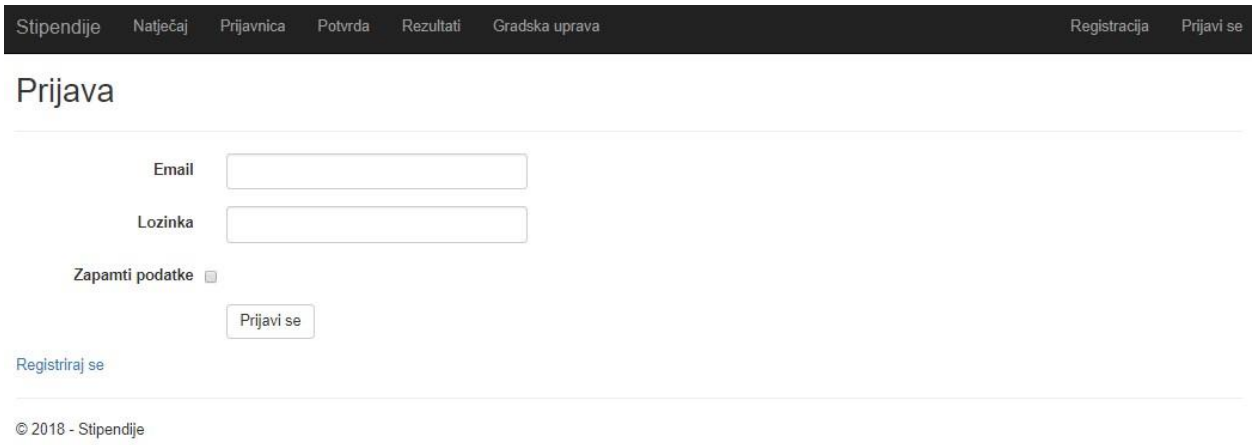


Sl. 5.3. *Prikaz stranice registracije*



Sl. 5.4. *Prikaz uspješne registracije*

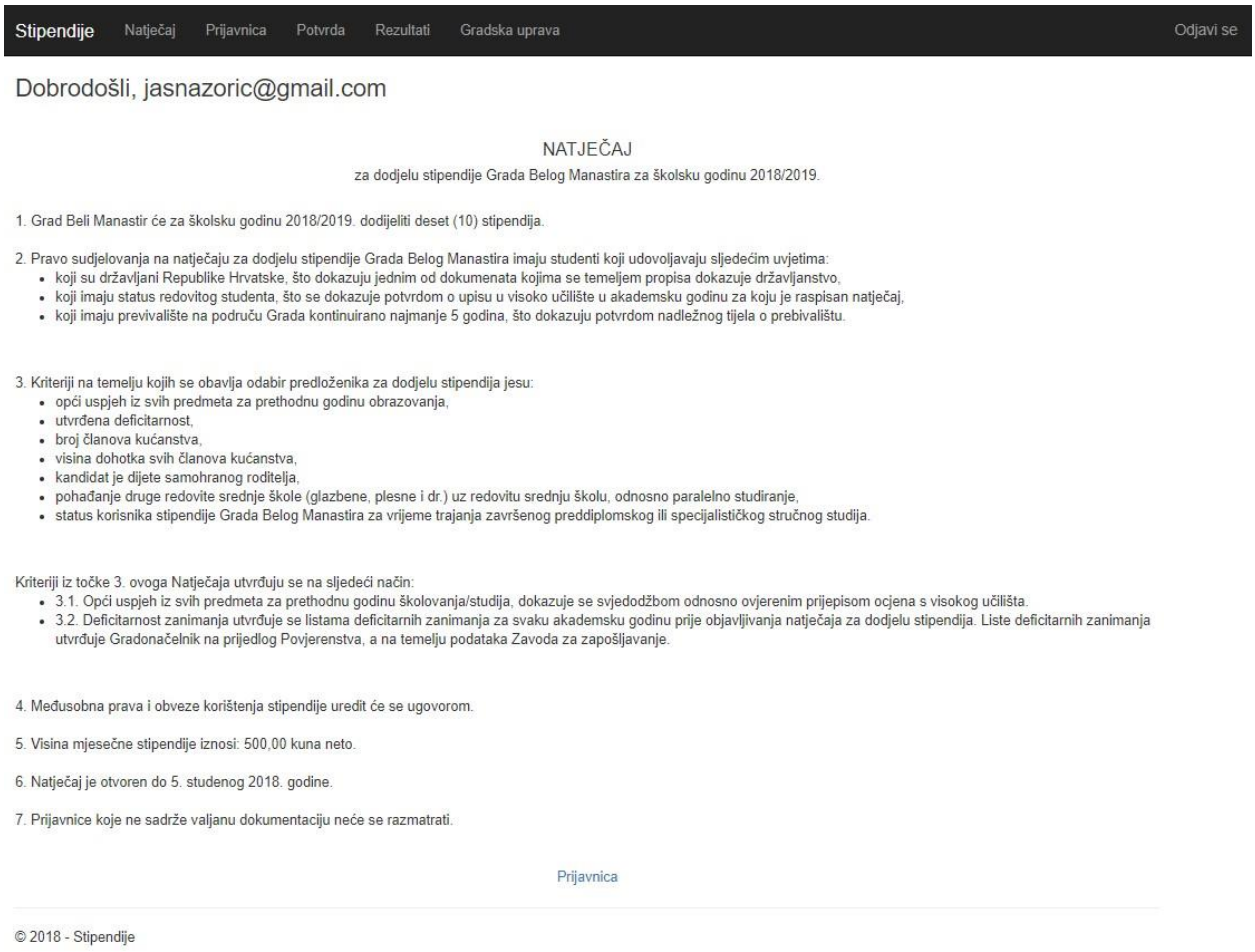
Uspješnom registracijom i stvorenim korisničkim računom, korisniku je omogućena prijava na stranicu. Stranici prijave korisnik može pristupiti, klikom na link koji se prikaže ispod poruke o uspješnoj registraciji ili klikom na gumb *Prijavi se* u izborničkoj traci. Kada korisnik pristupi stranici prijave (pregled *Prijava*), od njega će se zahtijevati unos korisničkog imena i zaporke (Sl. 5.5.).



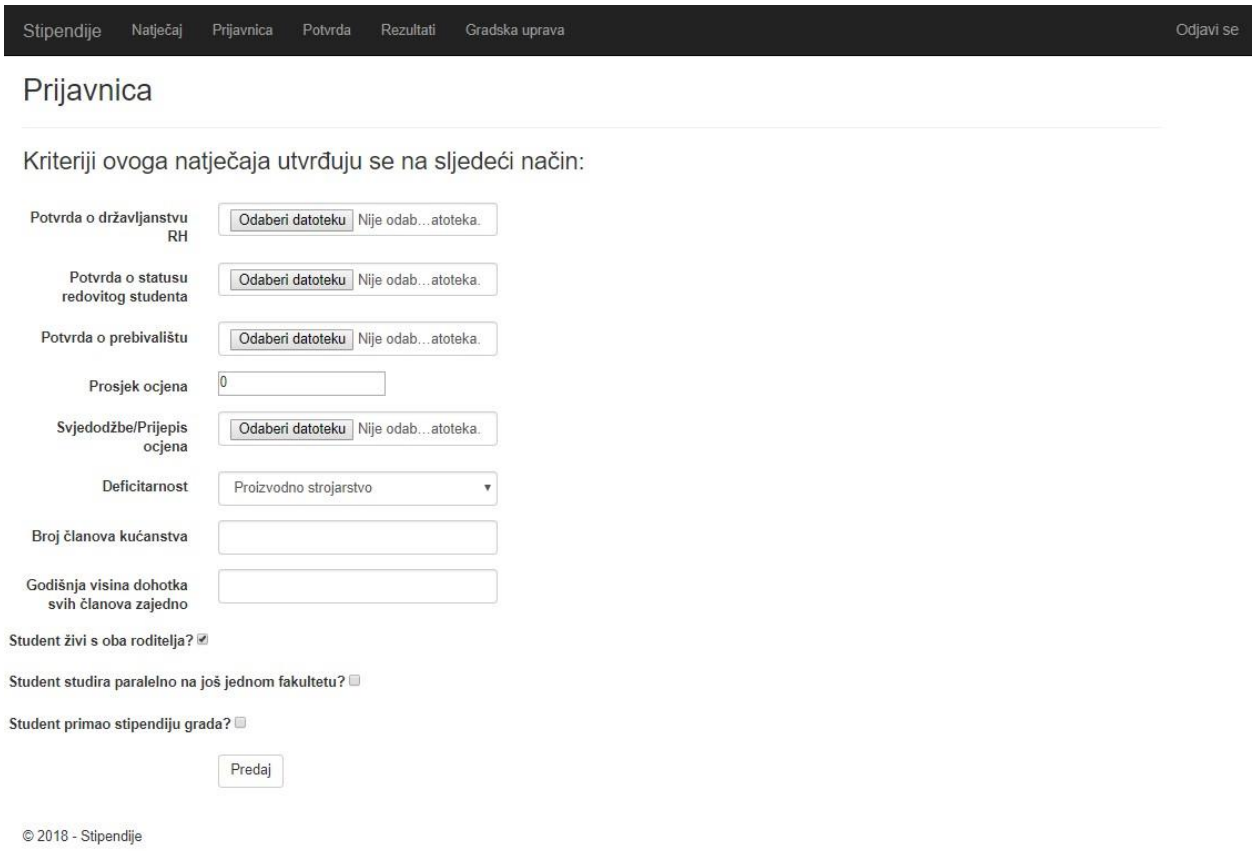
Sl. 5.5. *Izgled stranice za prijavu*

5.2. Prijavnica, potvrda i rezultati

Prijavljenom korisniku neznatno se mijenja izgled početne stranice (Sl. 5.6.). Kod poruke „Dobrodošli“ dodaje se korisnikova adresa e-pošte te mu se u dnu početne stranice dodaje veza kojom, osim gumbom u izborničkoj traci, može pristupiti prijavnici, a natpisi *Registracija* i *Prijavi se* u izborničkoj traci, mijenjaju se u *Odjavi se*. Klikom na jednu od te dvije opcije korisniku se prikazuje forma prijave (pregled *Prijavnica*) (Sl. 5.7.).



Sl. 5.6. *Izgled početne stranice nakon prijave korisnika*



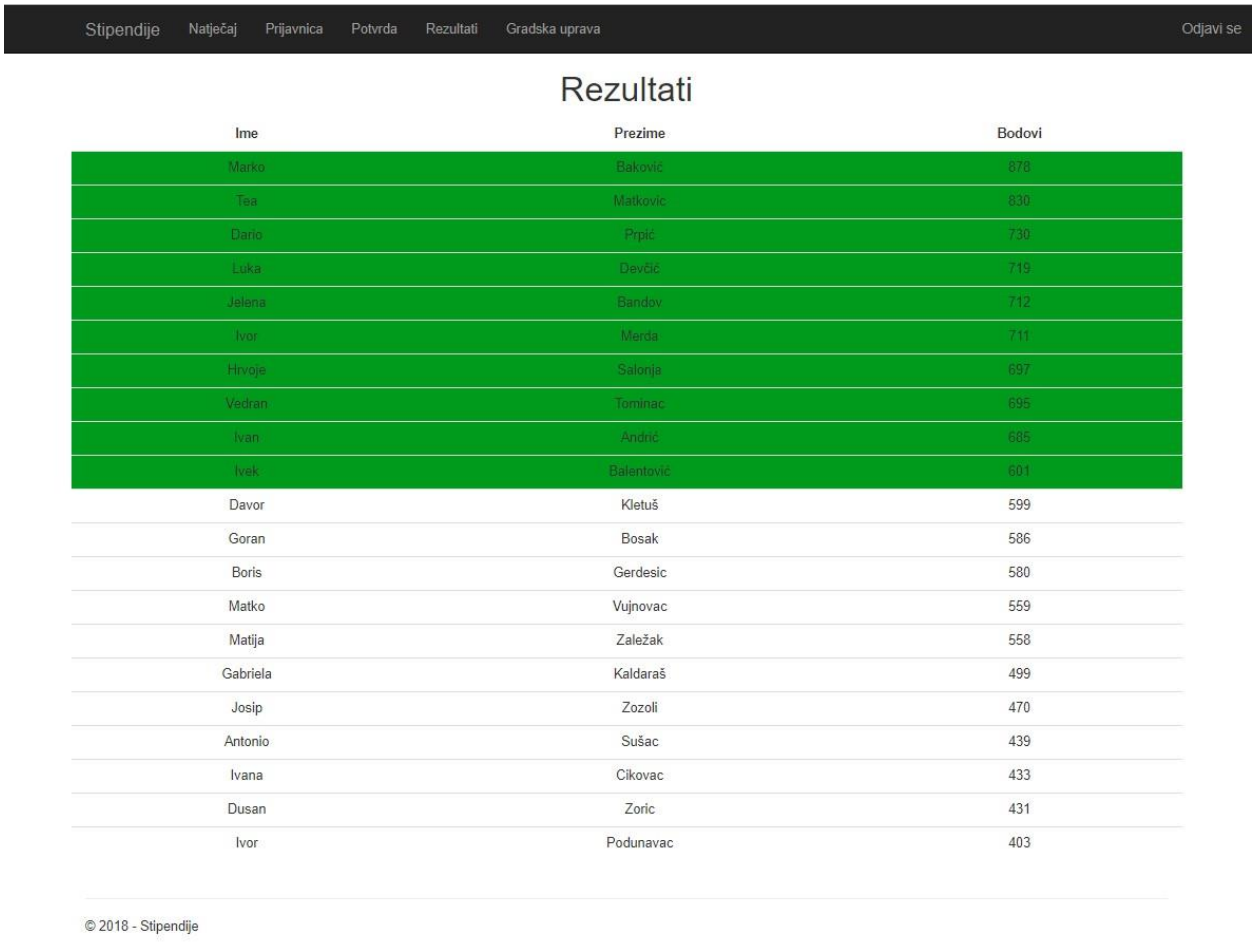
Sl. 5.7. *Izgled stranice Prijavnica*

Uspješnim ispunjavanjem prijavnice i klikom na gumb *Predaj*, korisniku se sprema prijavnica i prikazuje mu se stranica (pregled *Potvrdi*) s odgovarajućom porukom u ovisnosti o tome je li njegova prijavnica pregledana ili ne. Ukoliko bi korisnik pokušao pristupiti stranici potvrda (pregled *PotvrdaKorisnik*) prije nego li je prijavnica ispunjena, korisniku će se prikazati stranica s obavijesti o neispunjavanju prijavnice (Sl. 5.8.).



Sl. 5.8. *Poruka koja se ispisuje korisniku ovisno o statusu prijavnice*

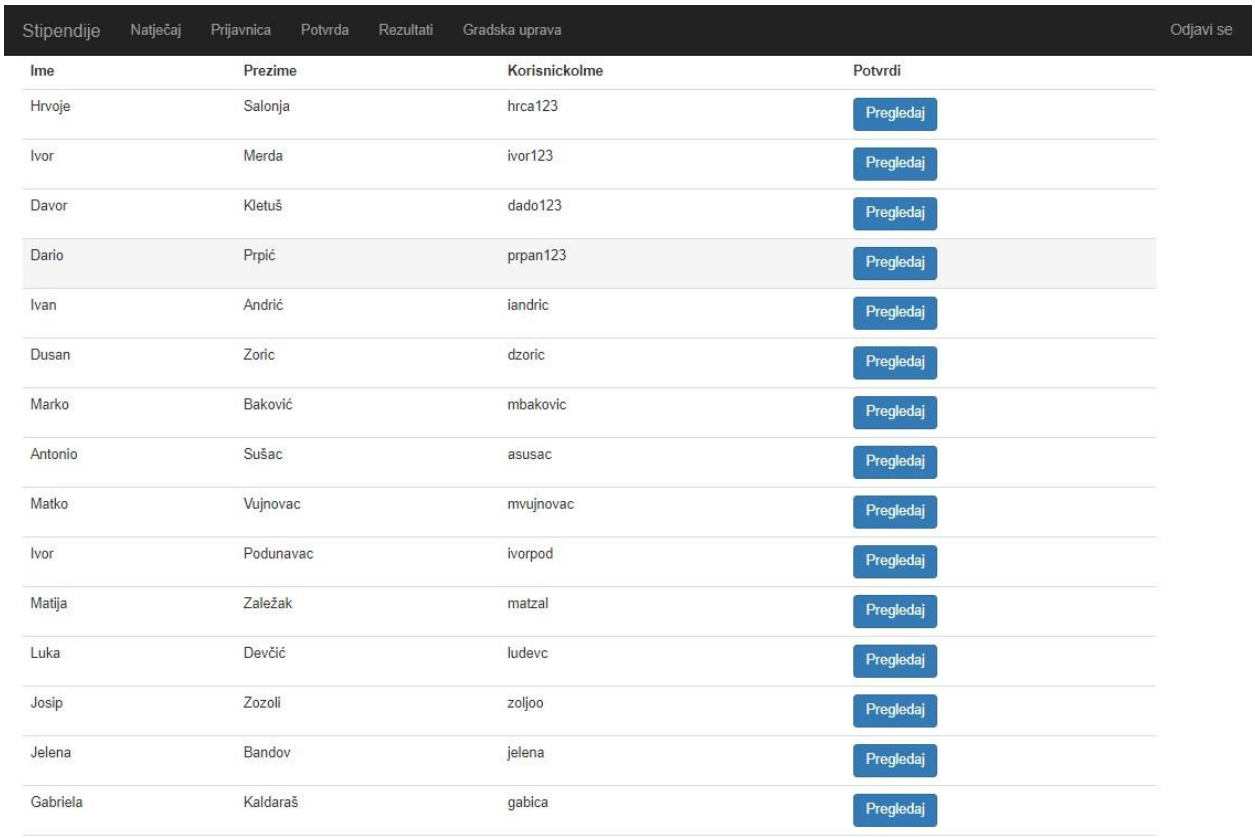
Po završetku procesa prijave i nakon administratorske potvrde korisničkih podataka, korisnici mogu vidjeti rezultate spremljene u tablici klikom na gumb *Rezultati* u izborničkoj traci (pregled *Rezultati*). Korisnicima će biti prikazana stranica na kojoj će pisati imena, prezimena i korisničko ime svih kandidata s valjano ispunjenim podacima (Sl. 5.9.). Prvih deset korisnika u tablici bit će osjenčani zelenom bojom, što označava da su ti korisnici ostvarili pravo primanja stipendije za akademsku godinu u kojoj je natječaj raspisan.



Sl. 5.9. *Prikaz stranice s rezultatima*

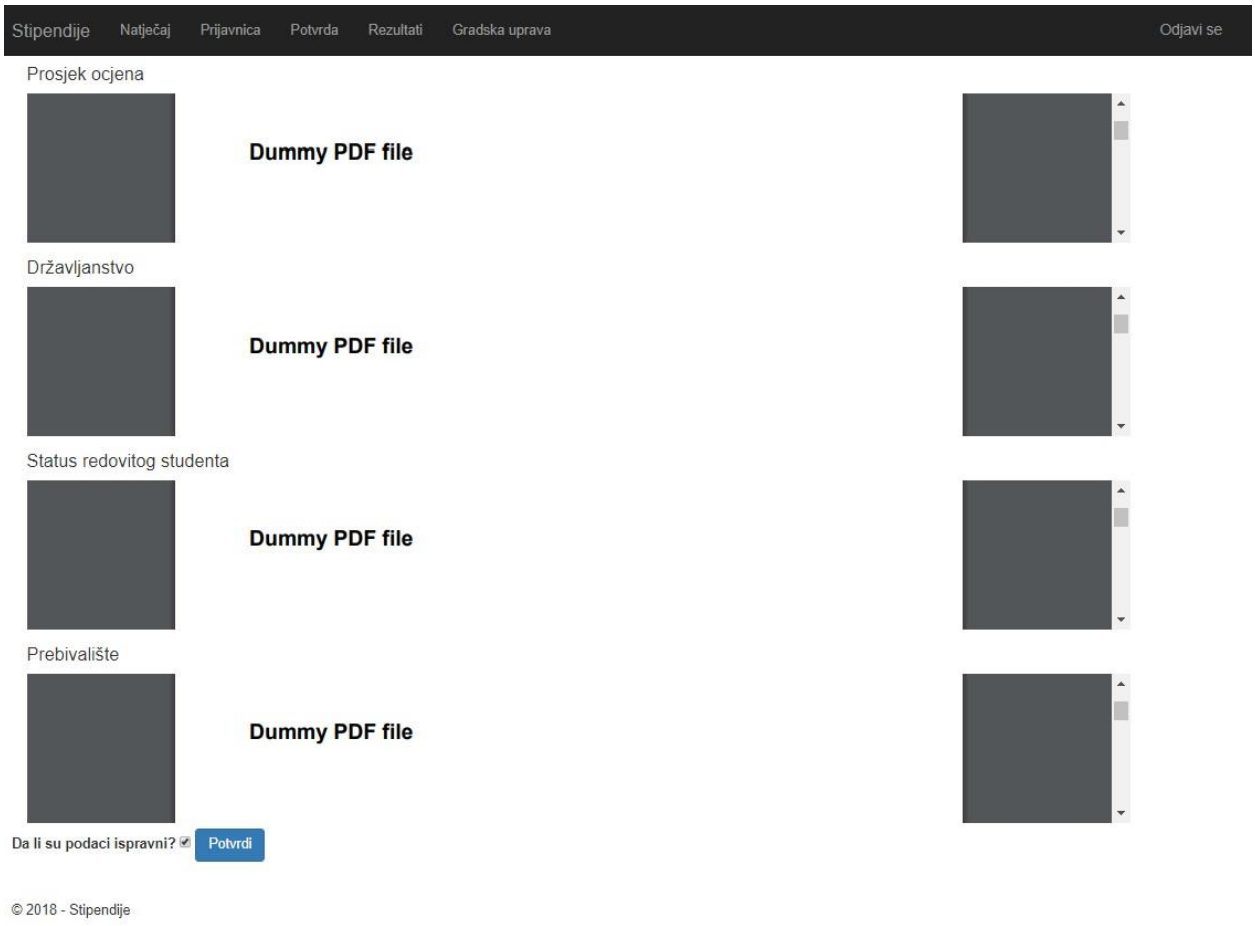
5.3. Administratorsko sučelje

Kao što je već objašnjeno (potpoglavlje 4.4.) administrator aplikaciji pristupa jedinstvenim korisničkim imenom i zaporkom. Uloga administratora automatski se dodjeljuje prvom računu u tablici *KorisnickiRacun* u bazi podataka. Osobe koje budu zadužene za pregledavanje korisničkih dokumenata moći će putem tog računa pristupiti aplikciji i pregledavati dokumente korisnika koji se prijavljuju na natječaj. Nakon prijave s administratorskim računom, upravitelj računa može otići na stranicu za prijave (pregled *Potvrda*) i vidjeti ima li nepotvrđenih prijavnica (Sl. 5.10.).



Sl. 5.10. *Stranica prikaza nepotvrđenih prijavnica u administratorskom sučelju*

Klikom na gumb *Pregledaj*, administratoru se otvara stranica s pregledom tekstualnih dokumenata korisnika (Sl. 5.11.). Po završetku pregleda i ukoliko je ustanovljeno da je prijavnica određenog korisnika ispravna može se kliknuti kvačica i zatim gumb potvrdi. To će korisnika upisati u tablicu na stranici *Rezultati* na mjesto koje mu pripada imajući u vidu korisnikov ukupan broj bodova. Ukoliko korisnik nije ispunio prijavnicu, administrator će za to dobiti obavijest (Sl. 5.12.). Korisnike čiji dokumenti nisu valjani ili su nepotpuni, administrator neće potvrditi te im tako neće dati pristup u tablicu na stranici *Rezultati*.



Sl. 5.11. *Stranica koja prikazuje administratoru korisnikove dokumente i omogućuje mu potvrdu istih*



Sl. 5.12. *Prikaz informacije administratoru o korisniku koji nije unio podatke*

6. ZAKLJUČAK

Web aplikacija objašnjena ovim radom nudi jednostavnost korištenja, korisnicima bi bio dovoljan bilo kakav web preglednik i pristup Internetu. Korisnika je bilo potrebno informirati o uvjetima natječaja, dati mu mogućnost prijave putem web preglednika i omogućiti mu brzu provjeru rezultata natječaja. Aplikacija je stvarana u programskom okruženju Microsoft Visual Studio. Pomoću znanja i potrebnih tehnologija koje zahtjevaju okviri ASP.NET MVC i Entity Framework kao Object Relational Mapper, HTML i slične potrebne tehnologije te C# programski jezik, ostvarena je web aplikacija i baza podataka. Prednost korišenja ovakve aplikacije je višestruka, prijava na daljinu značila bi značajnu uštedu vremena i novca s obzirom da nije potrebna fizički prikupljati i donositi dokumente prilikom prijave. Većina traženih dokumenata može se prikupiti na stranicama kao što su „SRCE – Sveučilišni računski centar“ gdje korisnik može skinuti ovjereni potvrdu o redovitom studiranju, „NCVVO“ gdje učenici koji se prijavljuju za stipendije mogu skinuti prijepis ocjena ili na portalu „e-Građani“ gdje su dostupni ovjereni dokumenti potrebni za prijavu.

LITERATURA

[1] http://www.beli-manastir.hr/natjecaja-za-dodjelu-stipendija-grada-belog-manastira-za-sk-g-20172018 [04.06.2018.]

[2] http://www.stipendije.info/hr [14.06.2018.]

[3] https://www.w3schools.com/html/ [15.09.2018.]

[4] http://www.mojwebdizajn.net/skriptni-jezici/vodic/html/uvod-u-html.aspx [11.09.2018.]

[5] https://hr.wikipedia.org/wiki/CSS [11.09.2018]

[6] http://www.mojwebdizajn.net/skriptni-jezici/vodic/css/css-syntax.aspx [11.09.2018.]

[7] Ian Griffiths, Programming C# 5.0: Building Windows 8, Web, and Desktop Applications for the .NET 4.5 Framework, O'Reilly Media, Inc., Sebastopol, CA, 2007. [14.09.2017]

[8] https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/ [31.05.2018.]

[9] https://en.wikipedia.org/wiki/Entity_Framework [13.09.2018.]

[10] https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx [02.06.2018.]

[11] http://www.beli-manastir.hr/wp-content/uploads/2017/10/Natjecaj-2017-deficitarna-zanimanja.pdf [04.06.2018.]

[12] Jon Galloway, Brad Wilson, K. Scott Allen, David Matson, Professional ASP.NET MVC 5, John Wiley & Sons, Inc., Indianapolis, IN, 2014. [22.12.2017.]

[13] https://www.c-sharpcorner.com/UploadFile/g_arora/Asp-Net-mvc-series-for-beginners-part-1/ [08.05.2018.]

SAŽETAK

Zadatak rada bio je izrada web aplikacije za objavu natječaja i rangiranje stipendista. Aplikacija podržava dva načina rada, korisnički i administratorski. Student, prilikom prijave na natječaj, prvo mora obaviti valjanu registraciju. Nakon prijave u sustav, student u prijavnicu unosi podatke o prosjeku ocjena, broju članova kućanstva, prihode kućanstva, živi li s oba roditelja, studira li paralelno i je li primao stipendiju prethodnih godina. Student također mora priložiti dokumente koji potvrđuju njegov prosjek ocjena, status redovitog studenta, državljanstvo te prebivalište. Nakon što student podnese prijavnicu, administrator prijavnicu može prihvatiti. Ukoliko su dokumenti zadovoljavajući, korisnik se, po odobrenju administratora, dodaje u tablicu. Dosadašnja prijava na natječaj za stipendije nije bila pristupačna kao ideja web aplikacije, što zbog svoje učinkovitosti, što zbog uštede. Dovoljno je pristupiti Internetu i završiti prijavu u par minuta.

Ključne riječi: student, stipendija, web aplikacija, ASP.NET, C#

ABSTRACT

Web application for scholarship announcement and registration

The task for this work was developing web application for scholarship announcement and scholars rankings. Application supports functionality for two user profiles, user and administrator. Student, trying to apply for scholarship, first of all has to do be registrated. After logging in, student has to fill the data on the application registration form with GPA (academic achievement at school), number of household members, household income, student also has to provide information whether he is living with one or both parents, if he is enroled in parallel studies and if he recieved scholarship in the past. Student also has to provide valid documents which confirm his GPA, his student status, citizenship, and residence. After registrating for scholarship, administrator can accept application form. If administrator finds all required data valid, user, with administrator permission, is added in results table. Up to this date, scholarship application forms were not so accessible like web application, because of it's effectiveness and savings. The only requirement for the user is available Internet connection and device to submit the form.

Key words: student, scholarship, web application, ASP.NET, C#

ŽIVOTOPIS

Dušan Zorić rođen je 12.04.1991. u Osijeku. Odrasta i živi u Belom Manastiru gdje završava osnovnu školu s odličnim uspjehom. Poslije osnovne škole upisuje srednju školu, Gimnazija Beli Manastir, opći smjer u Belom Manastiru. Srednju školu završava vrlo dobrim uspjehom.

Godine 2010. upisuje preddiplomski studij računarstva na Fakultetu elektotehnike, računarstva i informacijskih tehnologija u Osijeku. Godine 2014. Završava preddiplomski studij, a potom na istom fakultetu upisuje diplomski studij Procesnog računarstva.

Svoje slobodno vrijeme upotpunjava baveći se sportom, igranjem košarke u klubu „KK Beli Manastir“, treniranjem mlađih kategorija u istoimenom klubu i biciklizmom.