Web portal za oglašavanje životopisa

Perak, Jure

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

Sveučilišni studij

WEB PORTAL ZA OGLAŠAVANJE ŽIVOTOPISA

Diplomski rad

Jure Perak

Osijek, 2018.

Sadržaj

1. UVOD

Web portal za oglašavanje životopisa služi za studente fakulteta FERITOs koji su u potrazi za praksom. Cilj ovog rada je olakšati put od traženja, do razgovora za praksu, te eventualnog zaposlenja. Student registriranjem na stranci (ili portalu) može napraviti profil, te ga urediti s podacima kao što su adresa, školovanje, radovi na kojima je radio samostalno ili u nekoj tvrtki, razinu svojih znanje iz različitih tehnologija, te je u mogućnosti postaviti kompletan životopis u .pdf dokumentu. Svi ovi podaci služe za tvrtke koje se također registriraju na portal, gdje mogu pretraživati studente u potrazi za praksom. Tvrtke mogu vidjeti samo one studente koji žele da se vidi njihov profil, odnosno student može odlučiti žele li da njihov profil bude vidljiv tvrtkama, i za koju granu posla su zainteresiranu (npr. web, mobilne aplikacije, system programeri, game programeri..). Tvrtke mogu pretraživati studente po različitim parametrima, tip tehnologije, ime, prezime, radovi napravljeni u određenim tehnologijama, adresa. Ako se tvrtka odluči za poziv, može poslati poruku studentu preko portala.

Prilikom izrade aplikacije naglasak će biti na korištenju slojevite arhitekture, te korištenju različitih obrazaca i praksi. Aplikacija će se razdvojiti u nekoliko slojeva od kojih će svaki imati svoju svrhu, dohvaćanje podatka iz baze, obrađivanje tih podataka, te prikaz na stranici. Svaki dio će biti povezan s ostalim preko interfejsa (engl. interface), odnosno kod će biti slabo povezan (engl. loosely coupled) te preko *Dependency Injection* principa prosljeđivati će se implementacija u klasu.

Ovaj problem može se rastaviti u nekoliko dijelova koji će se rješavati sustavno. U početku će naglasak biti na funkcionalnosti stranice (portala), tako da će izgled stranice u početku biti u drugom planu. Prvi dio je omogućiti korisniku registraciju i dodavanje profila, odnosno uređivanje osnovnih podataka. Drugi dio je napraviti mogućnost za registraciju tvrtki i pretraživanje svih korisnika. Zatim će se dodati mogućnost da studenti detaljnije opišu svoj profil, dodaju projekte na kojima su radili te tehnologije u kojima rade. Nakon toga omogućiti tvrtkama detaljnije pregledavanje tih studenata. Važan detalj je omogućiti tvrtkama poziv na razgovor, stoga će to biti sljedeća stvar za riješiti. Pred kraj će se napraviti da stranica poprimi ljepši i moderniji izgled.

1.1 Diplomski zadatak

Zadatak je napraviti web portal na kojemu će nezaposlene osobe moći objavljivati životopis i odabrati jednu ili više grana u kojoj traže posao, a potencijalni poslodavci će moći pozivati ljude na razgovor. Potrebno je, prije same izrade portala, napraviti model koji će biti okosnica gotovog proizvoda.

2. DEFINIRANJE PROBLEMA

Nakon što je u uvodu dan opis rješavanja problema web portala, u ovom poglavlju će se to provesti u djelo. Prije samog pisanja koda potrebno je dobiti određenu sliku kako cijeli proces od registracije do poziva na razgovor treba izgledati. U tu svrhu biti će prikazan dijagram korištenja (engl. use case diagram). Programski jezik korišten za izradu aplikacije je C# u razvojnoj platformi ASP.NET MVC 5. Korišteno je Microsoftovo integrirano razvojno okruženje Microsoft Visual Studio Enterprise 2017. Detaljniji opis korištenih alata biti će u drugom dijelu ovog poglavlja.

2.1 Podjela glavnih dijelova sustava

Glavni dijelovi rada se mogu podijeliti da dvije strane. Na jednoj je student, njegov profil i svi podaci vezani za praksu, a na drugoj je tvrtka koja ima mogućnost pretraživanja studenata i pozivanja na razgovor, stoga se sustav može podijeliti na dva sudionika:

- Student
- Tvrtka

Zbog toga što student i tvrtka imaju zajedničke elemente kao npr. email za prijavu, lozinku, potvrdu računa, možemo napraviti generalizaciju tako da student i tvrtka nasljeđuju svojstva korisnika koji će sadržavati ta zajednička svojstva. Studentova strana sadrži sljedeće slučajeve:

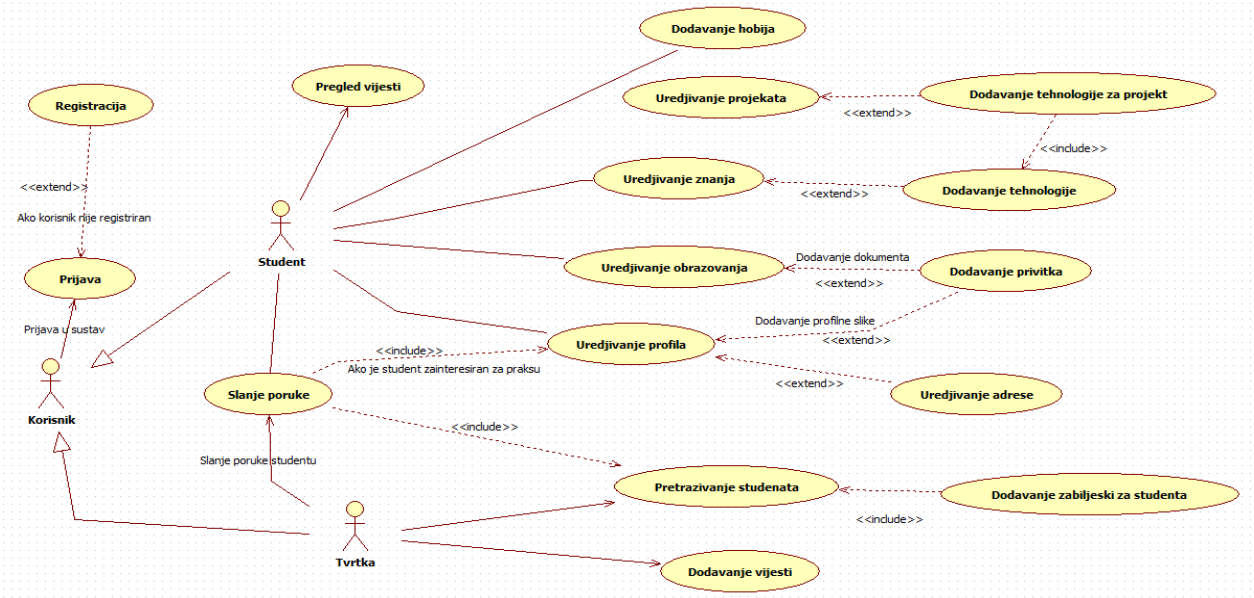
- Registracija/Prijava
- Uređivanje profila
    - Dodavanje profilne slike
- Uređivanje znanja
    - Dodavanje tehnologija
- Uređivanje obrazovanja
    - Dodavanje dokumenata
- Uređivanje projekata
    - Dodavanje tehnologija
- Uređivanje hobija
- Označavanje profila dostupnog za praksu
- Odgovor na poruke koje pošalje tvrtka
- Pregled vijesti

Na nabrojanim elementima temelji se studentov profil. Svaki element imati će mogućnost dodavanja dodatnih detalja koji omogućuju prednost nad profilima s manje detalja. Student dodaje profil i ima mogućnost dodavanja adrese. Zajednička stvar za uređivanje profila i obrazovanja biti će dodavanja privitka. Kod profila to predstavlja profilnu sliku, a kod obrazovanja diplome, svjedodžbe ili pohvalnice. Student kod uređivanja svojih projekata ima mogućnost odabira više znanja/tehnologija za jedan projekt. Tako će biti moguće detaljnije opisati projekte na kojima se radilo, a istovremeno ta znanja biti će korištena za opisivanje razine znanja iz određenih tehnologija.

Tvrtkina strana sadrži sljedeće elemente:

- Registracija/Prijava
- Pretraživanje studenata
- Dodavanje zabilješki za studenta
- Slanje poruke studentu koji je dostupan za praksu
- Dodavanje vijesti

Tvrtkina strana puno je jednostavnija zbog toga što je kompletan sustav zamišljen da se student stavlja u prvi plan, i da će se okrenuti procedura dobivanja prakse, odnosno tvrtke su te koje će tražiti studente jer se skraćuje vrijeme dobivanja prakse.Tvrtka prilikom registracije unosi svoj naziv. Imati će mogućnost dodavanja zabilješki za svakog studenta, te mogućnost slanja poruke onima koji su dostupni za praksu, dodavanje vijesti. U nastavku (sl. 2.1.) možemo vidjeti prikaz dijagrama korištenja (engl. use case diagram).



Sl.2.1. *Dijagram korištenja*

Jedan od zahtjeva pri izradi aplikacije je omogućiti da student ima mogućnost odabira je li zainteresiran za dobivanje prakse, odnosno može li ga tvrtka kontaktirati. Student neće imati mogućnost kontaktiranja tvrtki nego se cjelokupan proces prebacuje na tvrtku koja može izabrati studente na temelju njihovih znanja iz određenih tehnologija, obrazovanja te projekata. Student odabirom znanja određuje za koju granu tehnologija je zainteresiran što smanjuje izbor među tvrtkama. Samo tvrtka može pretraživati studente, odnosno studenti se ne mogu međusobno pretraživati.

2.2 Alati korišteni za rješavanje zadatka

Cjelokupni projekt je izveden u .Net okviru (engl. framework) koristeći razvojnu platformu je ASP.NET MVC 5. Relacijska baza podataka izrađena je u Microsoft SQL Server Management Studio 2017, a alat koji pretvara tablice iz baze u klase koje se kasnije koriste u aplikaciji je LLBLGen Pro 5.4.

2.2.1 **SQL Server Management Studio**

SQL Server Management Studio (SSMS) je integrirano okruženje za upravljanje SQL infrastrukturom. Osim upravljanja SQL servera te nadgledanja i administracije instance SQL servera, SSMS nam služi za prikaz podataka, kreiranje upita, te jednostavno građenje baze. Veze između tablica moguće dodati kroz grafičko sučelje bez poznavanja SQL (engl. Structured Query Language) jezika ili kroz upit (engl. query) na bazu.

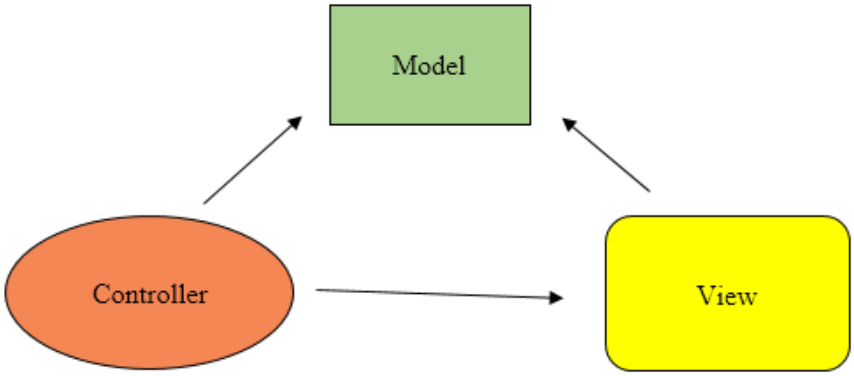
2.2.2 **LLBLGen**

LLBGen služi za objektno-relacijsko mapiranje (engl. object-relation mapping) podataka iz baze podataka u objektno orijentirani programski jezik. Tako sve tablice koje su kreirane u SSMS postoje u aplikaciji, što olakšava kreiranje upita na bazu. Pretvaranje tablica u klase se odvija u nekoliko koraka. Prvo je potrebno uspostaviti vezu između baze i LLBLGen desktop aplikacije. Nakon uspostave veze prikazati će se prozor gdje je potrebno označiti koje sve tablice se žele dodati u relacijski model. Prije konvertiranja u klase potrebno je kreirati *Entity Definitions* te zatim generirati izvorni kod (engl. source code). Prilikom generiranja, LLBLGen ponudi izbor u kojem programskom jeziku se žele generirati klase. Pošto je u radu korišten C#, klase su generirane u tom programskom jeziku. Generirane klase nalaze se u dva projekta, DatabaseGeneric i DatabaseSpecific. Projekte je potrebno dodati u glavni projekt kako bi se moglo s njima raditi.

Za sve operacije na bazi koristi se *DataAccessAdapter* klasa kojoj proslijedimo tekst koji sadrži informacije o bazi na koju se treba spojiti kako bi se dohvaćali podaci (engl. connection string).

2.2.3 **ASP.NET MVC**

ASP.NET MVC razvijen je od strane Microsofta 2009. godine. Prepoznatljiv je po MVC arhitekturnom obrascu (engl. pattern) koji dijeli aplikaciju na tri komponente, model, pogled (engl. view), kontroler (engl. controller)(sl. 2.2.). MVC obrazac razvio je Trygve Reenskaug 19790-ih za programski jezik Smalltalk.



Sl.2.2. *MVC arhitekturni obrazac*

- Model komponenta zadužena je za prijenos podataka od kontrolera do pogleda. On sadrži svojstva (engl. properties) koja se prikazuju na stranici. Praksa koja će se koristiti je da se za svaki pogled koristi jedan model. Tako se olakšava prikazivanje podataka, a taj način razdvajanja modela za prikaz se naziva Model-View-ViewModel princip (engl. MVVM pattern), odnosno model koji prenosi informacije do pogleda se naziva *ViewModel*, čime se postiže lakša održivost čitljivost koda. Tako kontroler daje pogledu samo ona svojstva koja su bitna za prikazivanje. (Freeman, 2018)

- Pogled služi za pretvaranje proslijeđenog modela u HTML kod te prikaz tog koda korisniku. Prilikom izrade pogleda korišten je Razor View Engine koji dolazi sa ASP.NET MVC razvojnim okruženjem. Razor omogućuje pisanje C# koda zajedno sa HTML kodom, te taj dokument ima ekstenziju cshtml. Osim pogleda koji predstavljaju cjelokupnu stranicu, kontroler može proslijediti i djelomični pogled, odnosno dio stranice, čime možemo mijenjati dijelove stranice bez potrebe za osvježavanjem cijele stranice.

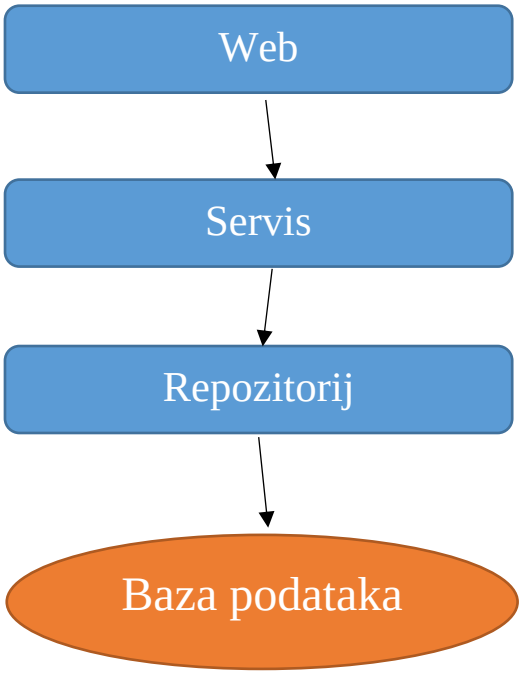
- Kontroler služi za primanje zahtjeva, izvršavanje operacija nad modelom te odabir odgovarajućeg pogleda za prikaz.(Freeman, 2018) Kontroler je poslužitelj između modela i pogleda. Dohvaća podatke iz baze, transformira ih u *ViewModel* te prosljeđuje u pogled. Kontroler je klasa izvedena is *System.Web.Mvc.Controller* klase koja sadrži akcijske metode (engl. action-method) zadužene za vraćanje pogleda. Kontroler i *action method* predstavljaju URL u sljedećem obliku: www.{domena}.com/{ime kontrolera}/{ime akcijske metode}/

3. RJEŠAVANJE PROJEKTA

U prethodnom poglavlju su spomenute tehnologije koje će biti korištene u projektu, a u ovom dijelu biti će opisan proces izrade te principi (engl. pattern) korišteni u aplikaciji. Aplikacija će biti podijeljena u više slojeva gdje je svaki sloj povezan s nadređenim. Ovakav pristup naziva se slojevita arhitektura MLA (engl. multilayered architecture) i glavni razlog odabira ove arhitektura je razdvajanje zadaća koju svaki sloj treba obaviti što predstavlja SoC obrazac (engl. separation of concerns).

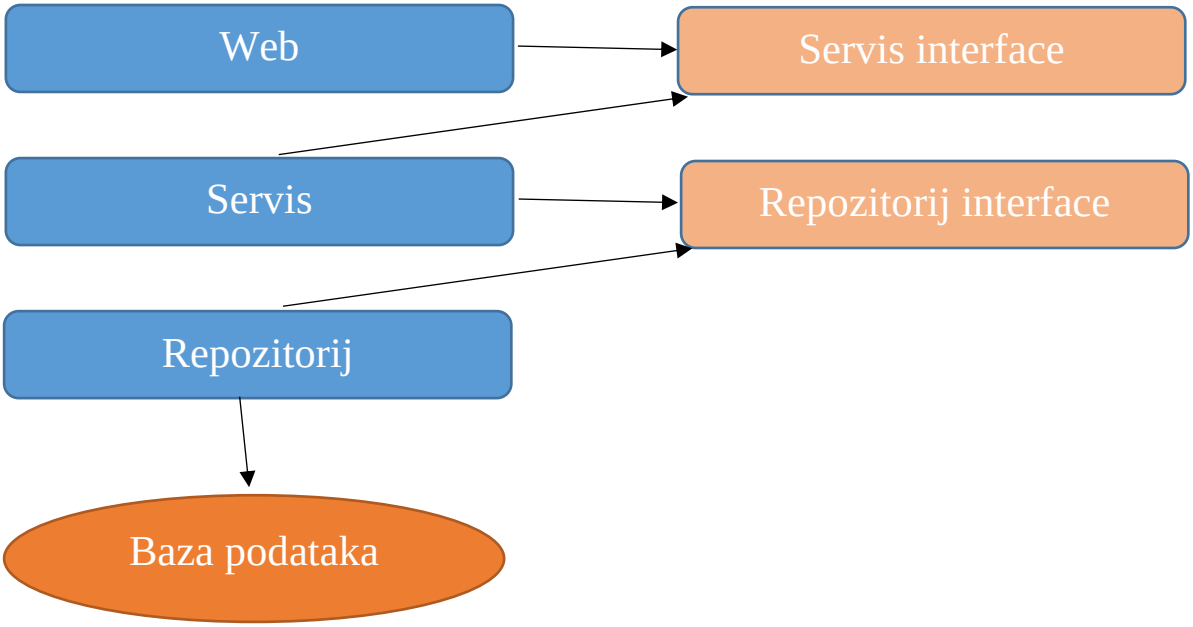
3.1 Arhitektura projekta

Slojevita arhitektura podrazumijeva razdvajanje aplikacije u više projekata. U web aplikaciji to najčešće predstavlja web projekt, poslovni projekt (servis), projekt za komunikaciju s bazom (repozitorij) i bazu podataka (sl. 3.1.)



Sl.3.1. *Slojevita arhitektura*

Tako se postiže razdvajane, međutim Web sloj ovisi o metodama koje su implementirane u Servisu, a Servis ovisi o metodama u Repozitoriju. To je značajka čvrsto povezanog koda (engl. tightly coupled). Umjesto da niži slojevi (engl. layers) definiraju više slojeve, uvesti će se interfejsi (engl. interface) koji će definirati niže slojeve (sl. 3.2.).



Sl.3.2. *Slojevita arhitektura sa interfejsima*

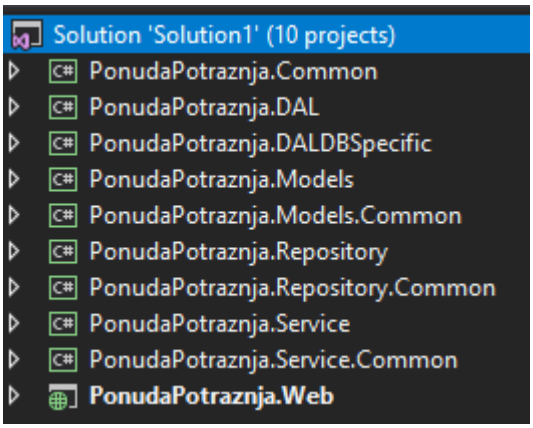
Web sloj definira Servis interface preko kojeg će djelovati sa Servis slojem. Web sloj ovisi o Servis interface-u, a Servis koji implementira taj interfejs također ovisi i njemu. Važno je naglasiti da je Web taj koja definira Servis interface. Na isti način funkcioniraju Servis i Repozitorij. Ako se nešto promijeni u Servis sloju, ništa se neće mijenjati u ostalim slojevima jer je Servis taj koji se mora prilagoditi Servis interface-u. Tako niži sloj ovisi o višem sloju. Niži sloj ovisi o apstrakciji tj. interfejsu koji je definiran od višeg sloja. Tako je postignut princip inverzije kontrole (engl. Inversion of Control pattern) što je jedan od preduvjeta za *Dependency Injection*.

3.1.1 **Dependecy Injection**

Glavna svrha *Dependecy Injection-a* (u nastavku DI) predstavlja održavanje (engl. maintainability). Jedan od najvažnijih načina za postizanje održivog koda je pisanje slabo povezanog koda (engl. loosely coupled). Glavna značajka slabo povezanog koda glasi: „*Program to an interface, not an implementation.*“ (Gamma, 1994). DI je način koji omogućuje slabo povezani kod. On prosljeđuje objekt drugim objektima kojima je potreban umjesto da ih oni sami kreiraju. To znači da će se klasi proslijediti objekti koji su joj potrebni i time ćemo smanjiti njena odgovornost da ih sama kreira. Prosljeđivanje objekta klasi se naziva *injection*. Načini prosljeđivanja objekta su *constructor injection*, *property injection* i *method injection*. Najrašireniji način je *constructor injection* te će on biti korišten u aplikaciji. Za implementaciju DI korišten je *Ninject*.

3.2 Postavljanje projekta

Projekt se sastoji od 10 potprojekata (sl. 3.3.) koji će biti opisani:



Sl.3.3. *Arhitektura PonudaPotraznja projekta*

PonudaPotraznja.Web je glavni projekt u aplikaciji od koji služi za prihvat zahtjeva od korisnika. Odabirom ASP.NET MVC 5 okvira riješen je problem prijave i registracije korisnika jer zadani predložak prilikom kreiranja Web projekta kreira i ASP.NET Identity sustav koji omogućava autorizaciju i autentifikaciju. Web projekt je zadužen za primanje HTTP upita, obrađivanje te odgovor. PonudaPotraznja.Web komunicira sa PonudaPotraznja.Service.Common projektom, te ima referencu na PonudaPotraznja.Models, PonudaPotraznja.Models.Common i PonudaPotraznja.Common projekte.

Projekti sa sufiksom Common označavaju projekte koji sadrže potpise (engl. interface) projekta bez tog sufiksa. Na taj način projekt koji želi implementirati metode projekta PonudaPotraznja.Service, referencirati će se na projekt PonudaPotraznja.Service.Common, i s time postižemo da kod bude slabo povezan (engl. loosely coupled). Slabo povezani kod puno je lakše izmjenjivati te nadograđivati.

PonudaPotraznja.Service je poslovna logika aplikacije u kojoj se nad modelima vrše operacije te ih se priprema za bazu podataka ili za prosljeđivanje prema Web projektu. Service projekt sadrži reference na PonudaPotraznja.Models, PonudaPotraznja.Models.Common, PonudaPotraznja.Repository.Common, PonudaPotraznja.Common projekte.

PonudaPotraznja.Repository služi za komunikaciju s bazom podataka. Sadrži metode za kreiranje, dohvaćanje, uređivanje, brisanje podataka u bazi. Kako su te stvari zajedničke za većinu modela, napravljen je jedan generički repozitorij koji će služiti za te operacije s čime se smanjilo

ponavljanje koda. Repozitorij projekt sadrži kod koji je dostupan preko alata LLBLGen. Pomoću tog alata znatno je pojednostavljen način kreiranja upita na bazu. Svaka operacija koja zahtjeva snimanje u bazu podataka izvedena je preko principa jedinice rada (engl. Unit of work pattern). *UnitOfWork* klasa osigurava da će sve operacije koji želimo snimiti biti sigurno snimljene, a ako jedna operacije ne prođe, cijela transakcija se obustavlja i ništa neće biti snimljeno. (Fowler, 2002)

PonudaPotraznja.DAL i PonudaPotraznja.DALDBSpecific su spomenuti ranije i oni su kreirani preko LLBLGen desktop aplikacije. PonudaPotraznja.DAL sadrži klase koje predstavljaju tablice u bazi. PonudaPotraznja.DALDBSpecific sadrži kod koji služi za ostvarivanje veze i komunikacije s bazom podataka, *DataAccessAdapter* klasa. Svaka operacija sa bazom podataka mora se izvršavati preko modela iz PonudaPotraznja.DAL projekta jer tako projekt razumije s kojom tablicom treba povezati upit.

PonudaPotraznja.Common sadrži kod relevantan za cijelu aplikaciju i zato svi projekti imaju reference na njega. Može sadržavati dodatne metode (engl. extension methods), podatke tipa enum, *lookup* podatke. *Lookup* podaci predstavljaju male nepromjenjive podatke koji definiraju puno veće tablice. Primjer jednog *lookup* podatka u aplikaciji je znanje/tehnologija.

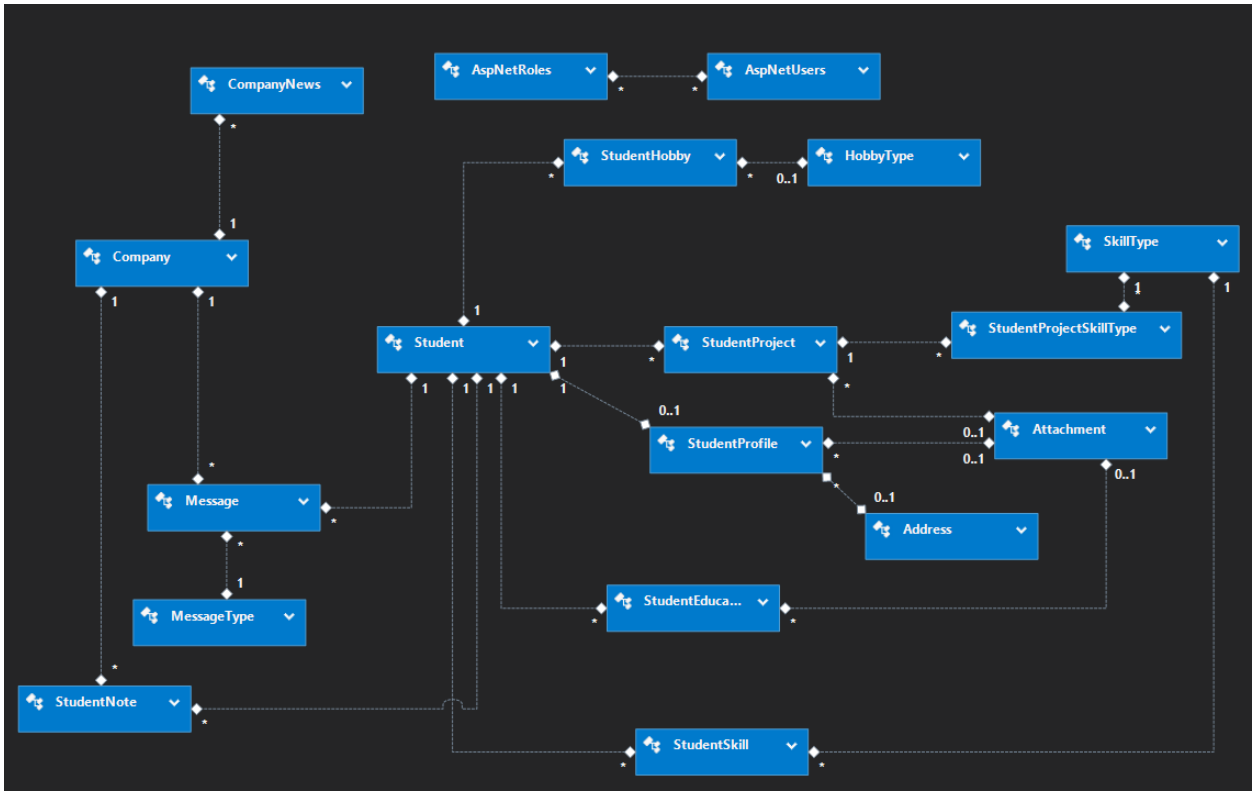
PonudaPotraznja.Models projekt sadrži modele koji će se koristiti u cijeloj aplikaciji, a nazivaju se POCO modeli, *PlainOldCLRObject*. Modeli služe za prijenos podataka od repozitorij projekta do web projekta. Ovime postižemo da service i web projekt ne znaju kako izgledaju pravi modeli koji se zapisuju u bazu. Ti modeli se nalaze u DAL projektu, a koristi ih jedino repozitorij projekt.

4. RAZVOJ APLIKACIJE

U ovom poglavlju biti će opisana programska izrada aplikacije. Na temelju zahtjeva aplikacije postavljena je baza podataka. Kreirane su tablice koje predstavljaju sudionike, Studenta i Tvrtku, te veze na ostale tablice. Nakon toga su kreirane klase u aplikaciji preko alata LLBLGen. U tom trenutku cijeli projekt je spreman za razradu zahtjeva i slučajeva zadanih na početku rada.

4.1 Modeliranje baze podataka

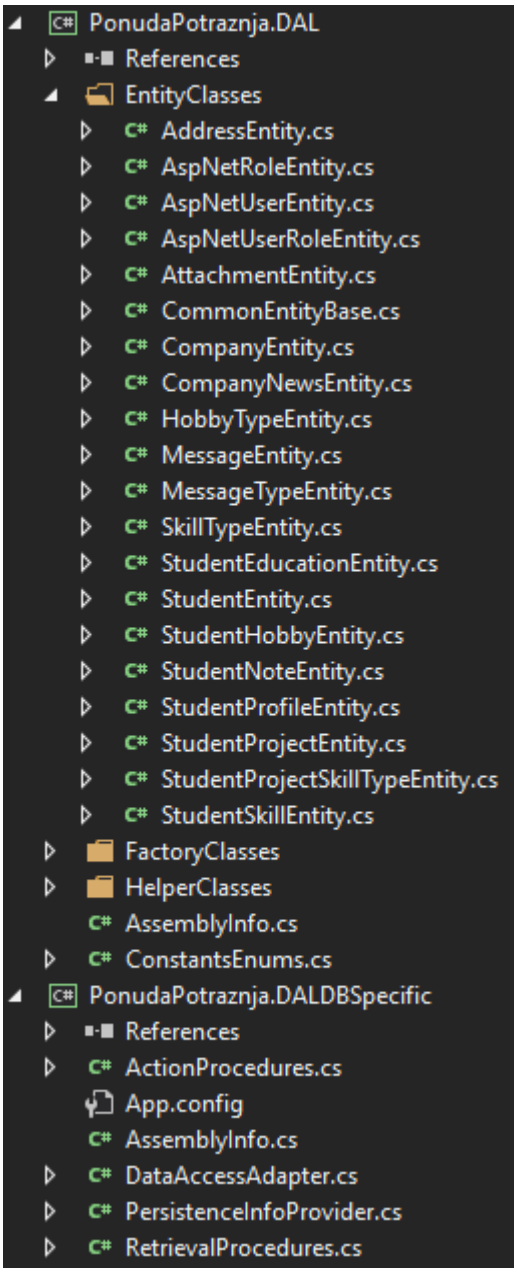
Baza podataka se može podijeliti na dvije strane, Student i Tvrtka. To su dva glavna sudionika koja određuju cijeli daljnji tok aplikacije. Student ima svoje akcije koje podrazumijevaju kreiranje profila, unos obrazovanja, tehnologija u kojima radi te projekte koje je do sada napravio. Na temelju tih zahtjeva izrađena je baza podataka (sl. 4.1.)



Sl. 4.1. *Dijagram baze podataka*

Prilikom modeliranja baze podataka posebno je razmatrana veza između studenta i profila. Kreirana je veza jedan Student na nijedan ili jedan profil (engl. one to zero, one relationship) iz razloga što će Student prilikom registriranja na PonudaPotraznja stranici imati mogućnost unosa Imena i Prezimena što su svojstva tablice Student. U tom trenutku registriranja student ima kreiran

zapis u tablici Student, ali nema u tablici StudentProfil. Tada će prilikom postavljanja profila biti kreiran zapis u tablici StudentProfil. Ostale studentove veze su kreirane na jedan Student, više obrazovanja, projekata, hobija, znanja (engl. one to many relationship). Nakon kreiranja baze podataka tablice su prebačene u C# klase preko alata LLBLGen. (sl. 4.2)



Sl.4.2. *Kreirani LLBLGen modeli*

Uz kreirane modele, LLBLGen kreira dodatne modele kojima olakšava kreiranje upita na bazu. U DALDBSpecific projektu se nalazi DataAccessAdapter preko kojega vršimo operacije na bazu podataka.

4.2 Kreiranje repozitorij sloja

Repozitorij sloj služi za kreiranje i izvršavanje upita na bazu da bi se dohvatili, kreirali, uredili ili obrisali određeni podaci. S obzirom na to da su upiti jednaki za većinu tablicu kreiran je jedan generički repozitorij koji služi za takve upite. U konstruktor generičkog repozitorija *inject-an* je *IDataAccessAdapterFactory* koji sadrži informacije o vezi prema bazi podataka, te pruža *IDataAccessAdapter* klasu preko koje izvršavamo upite na bazu (sl. 4.3.).

```
public GenericRepository(IDataAccessAdapterFactory dataAccessAdapterFactory)
{
    DataAccessAdapterFactory = dataAccessAdapterFactory;
}

protected IDataAccessAdapterFactory DataAccessAdapterFactory { get; private set; }
```

Sl. 4.3. *Konstruktor GenericRepository*

Na primjer metode za dohvaćanje podataka iz baze može se vidjeti način korištenja adapter klase (sl. 4.4).

```
public async Task<bool> FetchEntityAsync(IEntity2 entityToFetch, IPrefetchPath2 prefetchPath)
{
    using (var adapter = await GetAdapterAsync())
    {
        return adapter.FetchEntity(entityToFetch, prefetchPath);
    }
}
```

Sl. 4.4 *Generička metoda za dohvaćanje podataka FetchEntityAsync*

GetAdapterAsync metoda služi kreiranje instance *DataAccessAdapter* klase (sl. 4.5.).

```
public Task<IDataAccessAdapter> GetAdapterAsync()
{
    var adapter = DataAccessAdapterFactory.GetAdapter();
    return Task.FromResult(adapter);
}
```

Sl. 4.5. *Kreiranje DataAccessAdapter klase*

IEntity2 je interfejs koji predstavlja pretvorenu tablicu baze podataka. Preko tog interfejsa LLBLGen zna na koju tablicu treba vršiti upite. *IPrefetchPath2* interfejs predstavlja koje dodatne

tablice treba uključiti prilikom dohvata podataka iz baze. Ako želimo dohvatiti studenta, ali i njegov profil, taj profil bi naveli u *prefetchPath*. Svaka tablicu u bazi podataka ima kreiran svoj repozitorij. Svaki kreirani repozitorij implementira *GenericRepository* preko DI.

U repozitorij projektu treba naglasiti klasu DIModule koja predstavlja DI kontejner. DIModule derivira iz Ninject.Modules.NinjectModule klase i u njoj su definirana povezivanja (engl. bindings) između interfejsa i konkretnih implementacija (sl 4.6.). U repozitorij projektu to predstavlja interfejse repozitorija te konkretne repozitorij klase.

```
public class DIModule : Ninject.Modules.NinjectModule
{
    public override void Load()
    {
        Bind<IDataAccessAdapterFactory>().To<DataAccessAdapterFactory>();
        Bind<IGenericRepository>().To<GenericRepository>();

        Bind<IAddressRepository>().To<AddressRepository>();
        Bind<IAttachmentRepository>().To<AttachmentRepository>();
        Bind<ICompanyRepository>().To<CompanyRepository>();
        Bind<IEducationRepository>().To<EducationRepository>();
        Bind<IMessageRepository>().To<MessageRepository>();
        Bind<IStudentRepository>().To<StudentRepository>();
        Bind<IStudentHobbyRepository>().To<StudentHobbyRepository>();
        Bind<IStudentProfileRepository>().To<StudentProfileRepository>();
        Bind<IStudentSkillRepository>().To<StudentSkillRepository>();
        Bind<IStudentProjectRepository>().To<StudentProjectRepository>();
        Bind<IStudentNoteRepository>().To<StudentNoteRepository>();
        Bind<ICompanyNewsRepository>().To<CompanyNewsRepository>();
    }
}
```

Sl. 4.6. *DIModule klasa u repozitorij projektu*

Interfejsi za repozitorij project se nalaze u PonudaPotraznja.Repository.Common projektu. Za mapiranje LLBLGen modela I POCO modela korišten je alat AutoMapper. U klasi MappingProfile koja je derivirana od klase AutoMapper.Profile nalaze se mapiranja između modela (sl. 4.7.).

```
public class MappingProfile : Profile
{
    public MappingProfile()
    {
        CreateMap<AddressEntity, Address>();
        CreateMap<Address, AddressEntity>();
        CreateMap<AddressEntity, IAddress>();
        CreateMap<IAddress, AddressEntity>();
```

Sl. 4.7. *Primjer mapiranja između AddressEntity, Address i IAddress*

4.3 Kreiranje servis sloja

Servis sloj predstavlja poslovnu logiku aplikacije. Nalazi se između web i repozitorij projekta. Jednako kao za repozitorije, tako postoje i servisi za svaku tablicu u bazi. U svaki servis *inject-an* je odgovarajući repozitorij, a po potrebi mogu se *inject-ati* i dodatni. Tu dolazi do izražaja *UnitOfWork* princip. Uzmimo za primjer *StudentProfileService* klasu. U nju će se *inject-ati* *StudentRepository* i *StudentProfileRepository* (sl. 4.8.).

```
protected IStudentProfileRepository Repository { get; set; }
protected IStudentRepository StudentRepository { get; set; }
protected IAddressRepository AddressRepository { get; set; }
protected IAttachmentRepository AttachmentRepository { get; set; }

public StudentProfileService(IStudentProfileRepository repository, IStudentRepository studentRepository,
    IAddressRepository addressRepository, IAttachmentRepository attachmentRepository)
{
    Repository = repository;
    StudentRepository = studentRepository;
    AddressRepository = addressRepository;
    AttachmentRepository = attachmentRepository;
}
```

Sl. 4.8.. *StudentProfileService konstruktor*

Tada *StudentProfileRepository* kreira *UnitOfWork* klasu. Kad se žele spremiti promjene na tablici Student, pozove se *Update* metoda u *StudentRepository* gdje proslijedimo kreiranu *UnitOfWork* klasu. Repozitorije će dodati te promjene u *UnitOfWork* te neće još ništa spremiti u bazu. Tada se u servisu pozove *Update* metoda za StudentProfil gdje se isto proslijedi *UnitOfWork* klasa. Repozitorij doda i te promjene u *UnitOfWork*. Kada su dodane sve izmjene, u servisu se pozove metoda za spremanje svih izmjena, *Repository.CommitUnitOfWorkAsync(UnitOfWork klasa)*. *CommitUnitOfWorkAsync* će primiti instancu *UnitOfWork* klase te jednim pozivom snimiti sve promjene dodane u njoj. Ako jedna transakcija ne prođe, sve transakcije će biti otkazane. Tako možemo biti sigurni da će sve biti snimljeno ili ništa neće biti snimljeno (sl. 4.9.).

```
var uow = Repository.CreateUnitOfWork();
await OnBeforeUpdateStudentProfileAsync(studentProfile);
await Repository.UpdateAsync(uow, studentProfile);
await StudentRepository.UpdateAsync(uow, studentProfile.Student);

saved = await Repository.CommitUnitOfWorkAsync(uow);
```

Sl. 4.9. *Primjer korištenja UnitOfWork prinicipa*

Svaka akcija u servisu dodana je u *try-catch* blok koji osigurana da aplikacija nikad neće prekinuti zbog neke greške, nego će korisnik dobiti oblikovanu povratnu poruku da je nešto pošlo po krivu.

Kao i u repozitoriju, i servis ima DIModule klasu u kojoj su definirana povezivanja između interfejsa i konkretnih implementacija. Interfejsi za servise su definirani u PonudaPotraznja.Service.Common projektu.

4.4 Kreiranje web sloja

Web sloj napravljen je u ASP.NET MVC 5 okviru, te u zadanom predlošku se kreiran sustav namijenjen registraciji i prijavi korisnika. U tu svrhu ASP Identity sustav kreira određene tablice u bazi od kojih su za aplikaciju bitne sljedeće:

- AspNetUsers
    - Ovdje se sprema svaki registrirani korisnik, korisničko ime, lozinka
- AspNetUserRoles
    - Role koje će biti korištene u aplikaciji, Student rola za studenta i Company rola za tvrtke
- AspNetUserInRoles
    - Za svakog registriranog korisnika spremljeno u kojoj roli se on nalazi.

ASP.NET MVC dijelimo na tri glavna dijela, kontroler, model i pogled. Kontroler je zadužen za vraćanje modela i pogleda. On komunicira sa servis slojem na način da se u svaki kontroler, koji je zapravo klasa, preko DI *inject-a* servis koji se želi koristiti. Kontroler može biti označen s atributom *Authorize* koji omogućuje pristup metodama iz tog kontrolera samo ako je korisnik prijavljen. Dodatno se u *Authorize* atribut mogu dodati role pa je taj kontroler dostupan samo korisnicima u tim rolama.

Cjelokupna aplikacija je zamišljena na način da korisničko sučelje bude jednostavno, dinamično, brzo i bez puno učitavanja. To znači da pogledi (engl. views) trebaju sadržavati sve podatke, ali opet omogućiti korisniku da ih jednostavno izmjeni. Iz toga razloga, napravljen je pogled za studenta koji mu omogućuje gledanje svog profila onako kako bi ga tvrtka vidjela, te na istoj toj stranici uređivanje profila i svaka akcija uređivanja ne zahtjeva dodatno osvježavanje stranice. To se postiglo pomoću Javascript skriptnog jezika te dodatka Jquery. Akcije uređivanja i osvježavanja podataka izvedene su preko Ajax (engl. Asynchronous JavaScript And XML) zahtjeva. Tako se komunicira s kontrolerom bez potrebe da se cijela stranica osvježi (sl. 4.10.).

```
var pageIndex = 1;
$(document).on('click', '.btn-load-more', function () {
    $.ajax({
        url: '/Home/LoadCompanyNews/?pageIndex=' + (Number(pageIndex) + 1),
        type: 'GET',
        success: function (data) {
            $('.jq-company-news-list').append(data);
            pageIndex = pageIndex + 1;
        }
    });
});
```

Sl. 4.10. *Učitavanje dodatnih vijesti na naslovnoj stranici*

Odgovor kontrolera na taj zahtjev prikazan je na slici 4.11.

```
[Authorize(Roles = "Student")]
[AjaxOnly]
public async Task<ActionResult> LoadCompanyNews(int pageIndex)
{
    var filter = FiltersFactory.CreateCompanyNewsFilter();
    filter.Deleted = false;

    string[] includeItems = new string[]
    {
        PropertyName.GetPropertyName<ICompanyNews>(c => c.Company)
    };

    var paging = Factory.CreatePagingParameters(3, pageIndex);
    var companyNews = await CompanyNewsService.GetPagedCollectionAsync(filter, paging, includeItems);
    var homeCompanyNews = Mapper.Map<List<HomeCompanyNews>>(companyNews.ToList());

    return (ActionResult)PartialView("PartialViews/_ViewCompanyNews", homeCompanyNews);
}
```

Sl. 4.11. *Učitavanje dodatnih vijesti na kontroleru*

LoadCompanyNews metoda ograničena ja na zahtjeve koji dolaze od studenata te vrsta tog zahtjeva mora biti Ajax. Prima kao parametar pageIndex koji služi za dohvaćanje podataka iz baze s obzirom na to da će ovi podaci biti dohvaćani kao *Paged* podaci pa da se zna s koje stranice se trebaju uzeti podaci. Na slici se može vidjeti statička klasa *PropertyName* koja služi za dohvaćanje imena svojstava koje taj model sadrži, u ovom slučaju *ICompanyNews* sadrži *Company* model te će *GetPropertyName* metoda dohvatiti ime *Company*. To ime će se proslijediti u servis pa repozitorij i preko njega će se uz *CompanyNews* dohvatiti i ime tvrtke koja je objavila tu vijest. Nakon što se dobiju podaci od servisa, oni su u POCO obliku i potrebno ih je pretvoriti u ViewModele, a to postižemo pomoću AutoMappera i definiranih mapiranja. Možemo vidjeti da metoda vraća *PartialView* zajedno sa listom modela HomeCompanyNews što znači da vraćamo samo dio stranice. Taj dio stranice će služiti da se uz postojeće vijesti na stranici dodaju nove vijesti.

Spremanje i uređivanje podataka također koristi Ajax zahtjeve. U HTML kodu se definira forma s podacima koje se želi prenijeti do kontrolera (sl. 4.12.). Prilikom potvrde (engl. submit) te forme preko Javascript-a pronađemo formu na stranici te se ubacimo u event za potvrdu (sl.4.13.). Treba naglasiti da se ovaj proces događa na klijentovom računalu. Taj event za potvrdu će se prekinuti inače bi se cijela stranica ponovno učitala. Kad se event prekine, kreira se Ajax zahtjev kojem proslijedimo podatke iz forme. Tada će Ajax bez potrebe za učitavanjem cijele stranice prenijeti podatke do kontrolera. Kontroler će te podatke obraditi i vratiti jedinstveni model koji će sadržavati podatke o toj akciji, je li akcija izvršena uspješno i koja je povratna poruka. Treba naglasiti da svaki kontroler za ovakve akcije vraća isti model, tako da Ajax zahtjev zna koji povratni model očekuje.

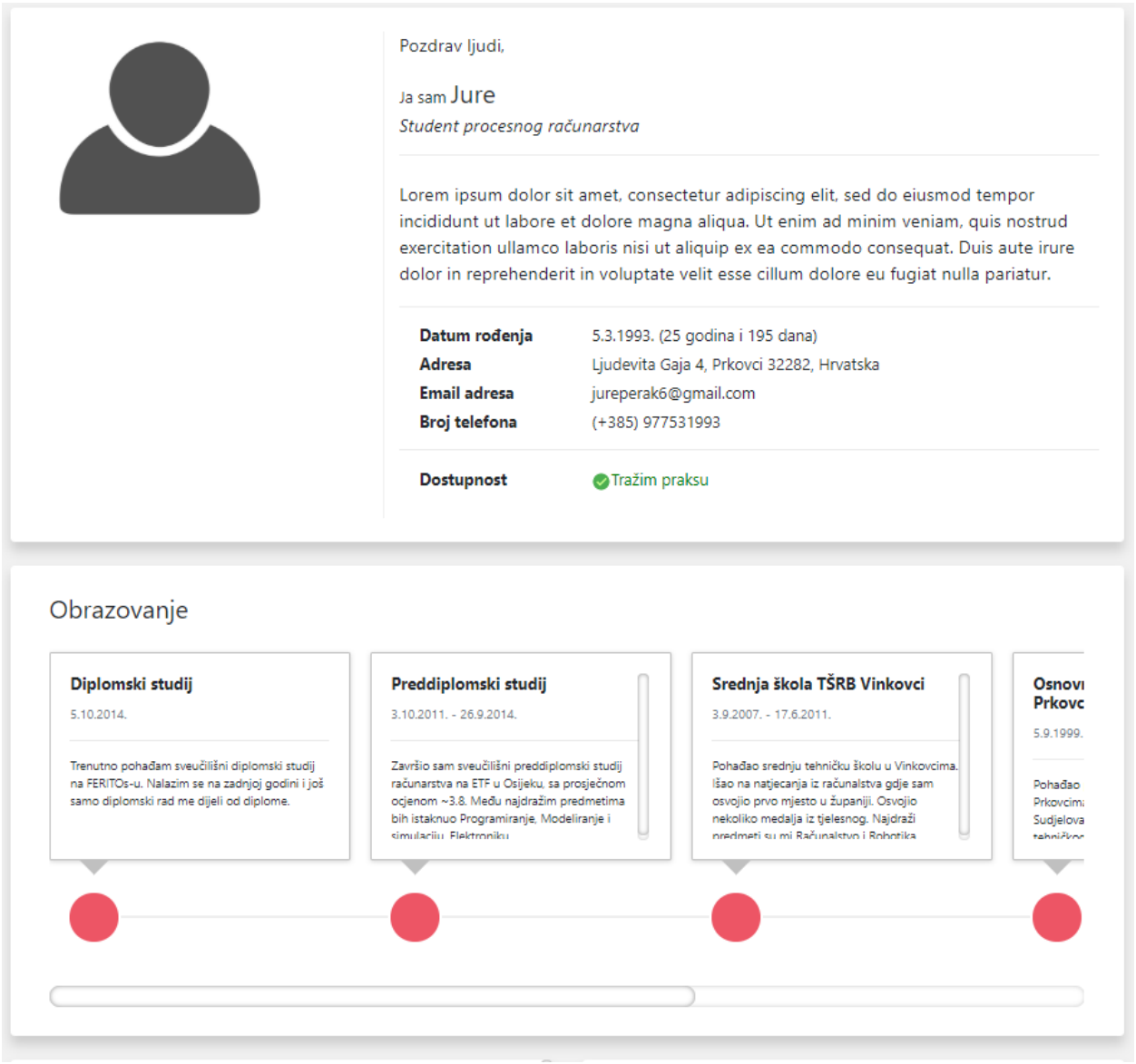
```
@model PonudaPotraznja.Web.ViewModels.CompanyNews.EditCompanyNews

@using (Html.BeginForm(Model != null && Model.Id.HasValue ? "Edit" : "Insert", "CompanyNews", FormMethod.Post, new { @class = "jqForm-company-news" }))
{
    @Html.AntiForgeryToken()
    @Html.HiddenFor(c => c.Id)

    <div class="form-row">
        @Html.LabelFor(m => m.Title, new { @class = "" })
        @Html.TextBoxFor(m => m.Title, new { @class = "form-control", autocomplete = "nope" })
        @Html.ValidationMessageFor(m => m.Title, null, new { @class = "text-danger" })
    </div>

    <div class="form-row">
        @Html.LabelFor(m => m.Body, new { @class = "" })
        @Html.TextAreaFor(m => m.Body, new { @class = "form-control", rows = "6", autocomplete = "nope" })
        @Html.ValidationMessageFor(m => m.Body, null, new { @class = "text-danger" })
    </div>

    <div class="form-group mt-3">
        <div class="col-md-offset-2 col-md-10">
            <input type="submit" class="btn btn-outline-primary" value="Spremi" />
        </div>
    </div>
}
```

Sl. 4.12. *Forma za uređivanje tvrtkinih vijesti*

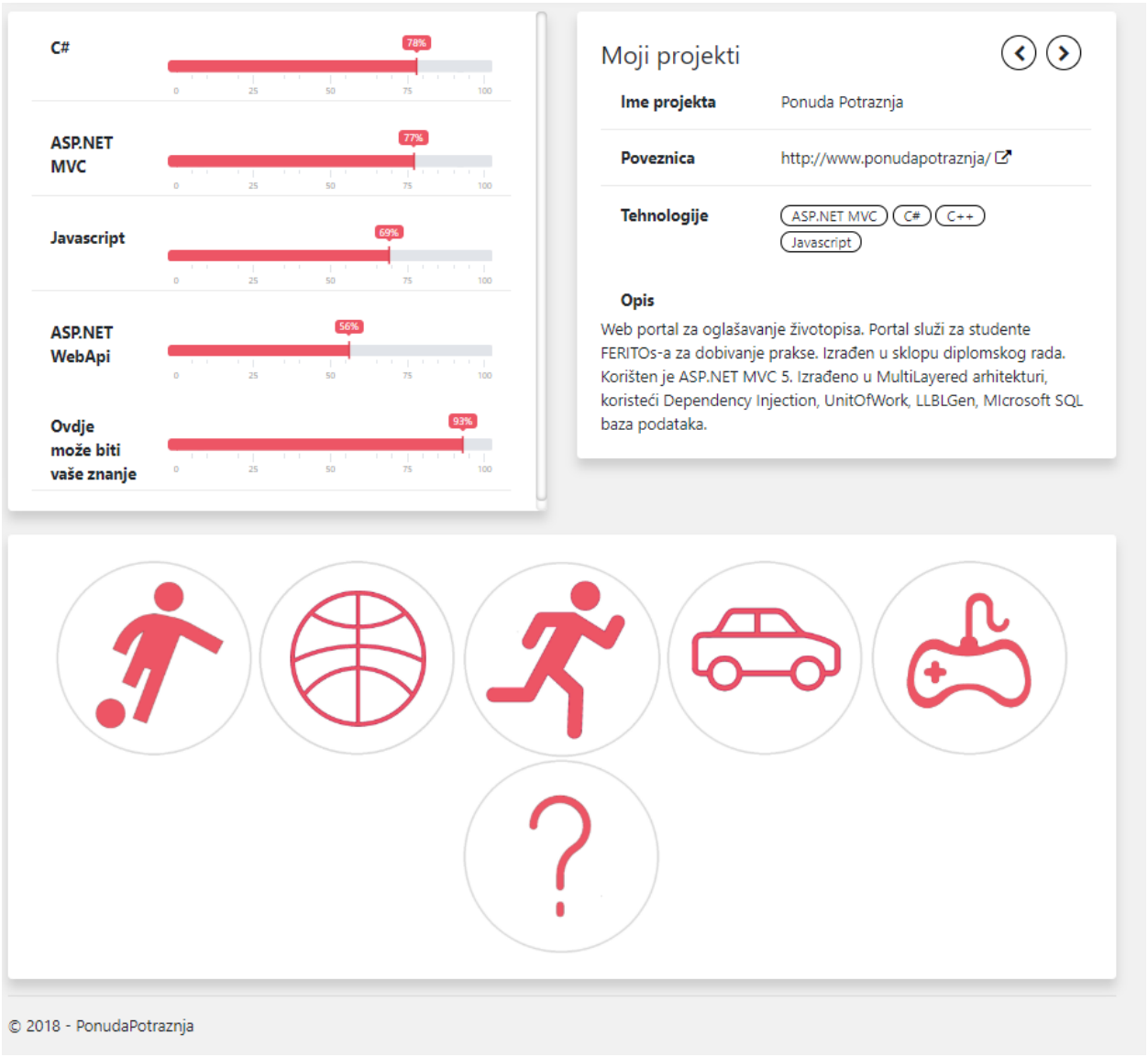
```
$(document).on('submit', '.jqForm-company-news', function (e) {
    e.preventDefault();
    var form = $(this);

    var formData = new FormData(form[0]);

    if (form.valid()) {
        $.ajax({
            url: form[0].action,
            type: form[0].method,
            data: formData,
            contentType: false,
            processData: false,
            success: function (data) {
                showMessage(data.Message, data.Type, false);
                if (data.Type === 1) {
                    $('.jqForm-search-company-news').submit();
                    $('#modal-company-news').modal('hide');
                }
            }
        })
    }
});
```

Sl. 4.13. *Ajax skripta za spremanje tvrtkinih vijesti*

Izgled studentovog profila za uređivanje prikazan je na slikama 4.14. i 4.15. gdje se može vidjeti kako su podijeljeni podaci profila, obrazovanja, znanja, projekata i hobija. Ono što olakšava uređivanje profila je izgled cijele stranice koja predstavlja stvarni izgled stranice kako će ju tvrtka vidjeti. Tako student može postići najviše iz svojih podataka bez potrebe za konstantnim učitavanjem cijele stranice.



Sl 4.14. *Izgled stranice za uređivanje osobnih podataka i obrazovanja*



Sl. 4.15. *Izgled stranice za uređivanje znanja/tehologija, projekata i hobija*

5. ZAKLJUČAK

PonudaPotraznja aplikacija izrađena je prvotno za studente FERITOs fakulteta. Cilj aplikacije je da se ubrza proces dobivanja prakse kod tvrtki te da tvrtke postanu te koje aktivno traže nove studente. Time se želi postići veća kompetitivnost među studentima jer oni koji imaju najbolji profil, odnosno koji imaju najviše za pokazati u svojem profilu automatski postaju bolji izbor za tvrtke. Aplikacija je izrađena na način da se korisniku pokaže profil, ali istovremeno omogući da ispravi željene podatke bez potrebe za odlaženjem na druge stranice. Jasan prikaz studentovih podataka koji su podijeljeni po karticama napravljen je koristeći Bootstrap okvir.

Aplikacija je izrađena koristeći prakse i obrasce koji se koriste u stvarnim sustavima. Takvim načinom postiglo se da aplikacija postane lako održiva i nadograđivana. Kod slojevite arhitekture, svaki sloj obavlja svoju zadaću i komunicira s drugim slojevima. Gledajući iz perspektive tog drugog sloja, njemu nije bitno kako je rezultat dobiven, sve dok je taj rezultat ispravan. Drugi princip koji se nadovezuje na slojevitu arhitekturu je Dependency Injection, koji omogućuje da projekti u slojevitoj arhitekturi surađuju s ostalim projektima koristeći apstrakcije. Tako će svaki sloj ovisiti o apstrakciji i time se dobiva mogućnost testiranja ili zamjene tog sloja. Recimo da se želi zamijeniti alat korišten za pretvaranje tablica u klase, LLBLGen, s nekim drugim alatom koji obavlja takvu zadaću, npr. Entity Framework. Bilo bi potrebno zamijeniti DAL projekte i prilagoditi repozitorij. Servis i web projekt nije potrebno mijenjati. Ili da se želi koristiti druga tehnologija umjesto ASP.NET MVC-a, tada bi bilo potrebno samo web projekt mijenjati.

Koristeći ovakvu arhitekturu sustava ne znači da je to jedino ispravno za izradu aplikacije. Potrebno je proučiti zahtjeve i slučajeve, te pretpostaviti mogući rast aplikacije. Tako se može predvidjeti kakva arhitektura je potrebna za izradu, jer složenija arhitektura predstavlja i duži period izrade, ali njene prednosti su daleko veće.

Neki od dodataka koji može predstavljati poboljšanje aplikacije je dodavanje treće role koja će biti zadužena za organiziranje podataka s kojima student raspolaže prilikom uređivanja svojeg profila, dodavanja novih vijesti vezanih za fakultet. Dodavanje zahtjeva koji će služiti kao ugovor za održavanje prakse, a na njemu mogu biti podaci kao vremenski period prakse, teme koje će biti obrađene na toj praksi.

LITERATURA

[1] Adam Freeman, Pro ASP.NET MVC 5, Apress, travnj 2014.

[2] Martin Fowler, Patterns of Enterprise Application Architecture, Pearson, travnj 2014.

[3] Erich Gamma, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, listopad 1994.

SAŽETAK

Izrađena je aplikacija za studente i tvrtke koja će služiti za oglašavanje profila studenata i njihovih projekata. Tvrtke imaju mogućnost pregleda tih podataka, zapisivanja zabilješki te slanja poruka tim studentima. Aplikacija je izrađena koristeći slojevitu arhitekturu, te razne principe među kojima treba istaknuti Dependency Injection koji se koristi za pružanje konkretne implementacije kod slabo povezanog koda. Za bazu je korišten Microsoft SQL server, a kao objektno relacijski alat LLBLGen. Web dio aplikacije je izrađen u ASP.NET MVC okviru, koristeći Bootstrap kao okvir za uređivanje i dizajniranje grafičkog sučelja.

Ključne riječi: principi, profil, slojevita arhitektura, student, tvrtka

ABSTRACT

TITLE: Web portal for advertising CV

An application for students and companies was created that will serve to advertise student profiles and their projects. Companies have the ability to view these data, adding notes and sending messages to students. Application was made by using multilayered architecture and various patterns, including Dependency Injection, which is used to provide concrete implementation for loosely coupled code. Database was created with Microsoft SQL server, and as object-relation mapper is used LLBLGen Pro. Web part of the application was made in the ASP.NET MVC framework, using Bootstrap framework for editing and designing graphical interface.

Key words: company, multilayered architecture, patterns, profile, student

ŽIVOTOPIS

Jure Perak rođen je 05. ožujka 1993. godine u Vinkovcima u Hrvatskoj. Prva četiri razreda osnovne škole pohađao je u Prkovcima, a ostatak u Retkovcima. Od malih nogu je pokazivao zanimanje za informatiku te je upisao Tehničku školu Ruđera Boškovića u Vinkovcima, smjer tehničar za mehatroniku. U srednjoj školi je išao na natjecanja iz predmeta Računalstvo i programiranje te je osvojio 1. mjesto na županijskom natjecanju u kategoriji strukovnih škola iz Osnova informatike. Srednja škola ga je potakla da se još više zainteresira za računarstvo i zbog toga upisuje 2011. godine sveučilišni preddiplomski studij računarstva na Elektrotehničkom fakultetu u Osijeku. Zvanje sveučilišni prvostupnik inženjer računarstva stječe 2014. godine, te iste te upisuje sveučilišni diplomski studij procesnog računarstva. U kolovozu 2016. godine kreće sa studentskim radom u tvrtki Mono kao software developer. Od studenog 2017. je zaposlen na neodređeno u istoj navedenoj tvrtki.

_____

Jure Perak

PRILOZI

Na optičkom disku u prilogu nalazi se ovaj rad u .docx i .pdf formatu kao i cjelokupni kod aplikacije opisane u radu.