SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 14

APLIKACIJA ZA UNOS PODATAKA O ALUMNIMA

Antonija Gužvinec

Zagreb, lipanj 2010.

Zahvaljujem svom mentoru Doc. dr. sc. Leonardu Jelenkoviću na strpljenju, pomoći i vodstvu pri izradi ovog diplomskog rada. Hvala svim kolegama i prijateljima što su mi studij učinili zabavnijim i manje stresnim.

Najveće hvala mojim roditeljima i obitelji na razumijevanju i podršci tokom studiranja.

Sadržaj

1. Uvod

Podaci o bivšim studentima Fakulteta elektrotehnike i računarstva zasad se još uvijek čuvaju u pisanom obliku, odnosno u knjigama diplomiranih. Pritom je riječ o starijim podacima - podacima o studentima koji su diplomirali prije 1992. godine, dok se informacije o studentima koji su diplomirali nakon 1992. godine već čuvaju u elektroničkom obliku (sustav ISVU). 19. siječnja 2000. godine osnovana je Hrvatska udruga diplomiranih inženjera Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu (AMAC-FER). Spomenuta je udruga nastavila rad Zajednice diplomiranih inženjera Elektrotehničkog fakulteta osnovane 4. lipnja 1992. godine. Budući da su ciljevi udruge povezivanje diplomiranih inženjera FER-a na unapređenju znanstvene i nastavne djelatnosti, kao i povezivanje članova zbog međusobne stručne i druge suradnje, unos prethodno spomenutih podataka i mogućnost njihova dohvata u elektroničkom obliku mogla bi znatno olakšati komunikaciju unutar udruge, odnosno pristup kontaktima i drugim informacijama o članovima udruge. Osim toga, spremanje podataka u elektroničkom obliku može olakšati i usklađivanje s podacima iz drugih izvora (kao što je ISVU), a i samo čuvanje i pristup podacima su time znatno olakšani.

Tema ovog rada je stoga bila proučiti strukturu podataka koji će se unositi te na temelju toga izraditi aplikaciju prikladnu za njihov unos.

U radu je opisana struktura podataka (tj. struktura same baze podataka) i korištene tehnologije (baza podataka *SQLite*, biblioteka *NHibernate* za povezivanje aplikacije s bazom podataka te grafički podsustav *Windows Presentation Foundation* za izradu grafičkog korisničkog sučelja). Budući da je rezultat ovog rada višeslojna aplikacija, ukratko je objašnjena i struktura samog programa, način podjele na slojeve i način na koji slojevi zajedno funkcioniraju.

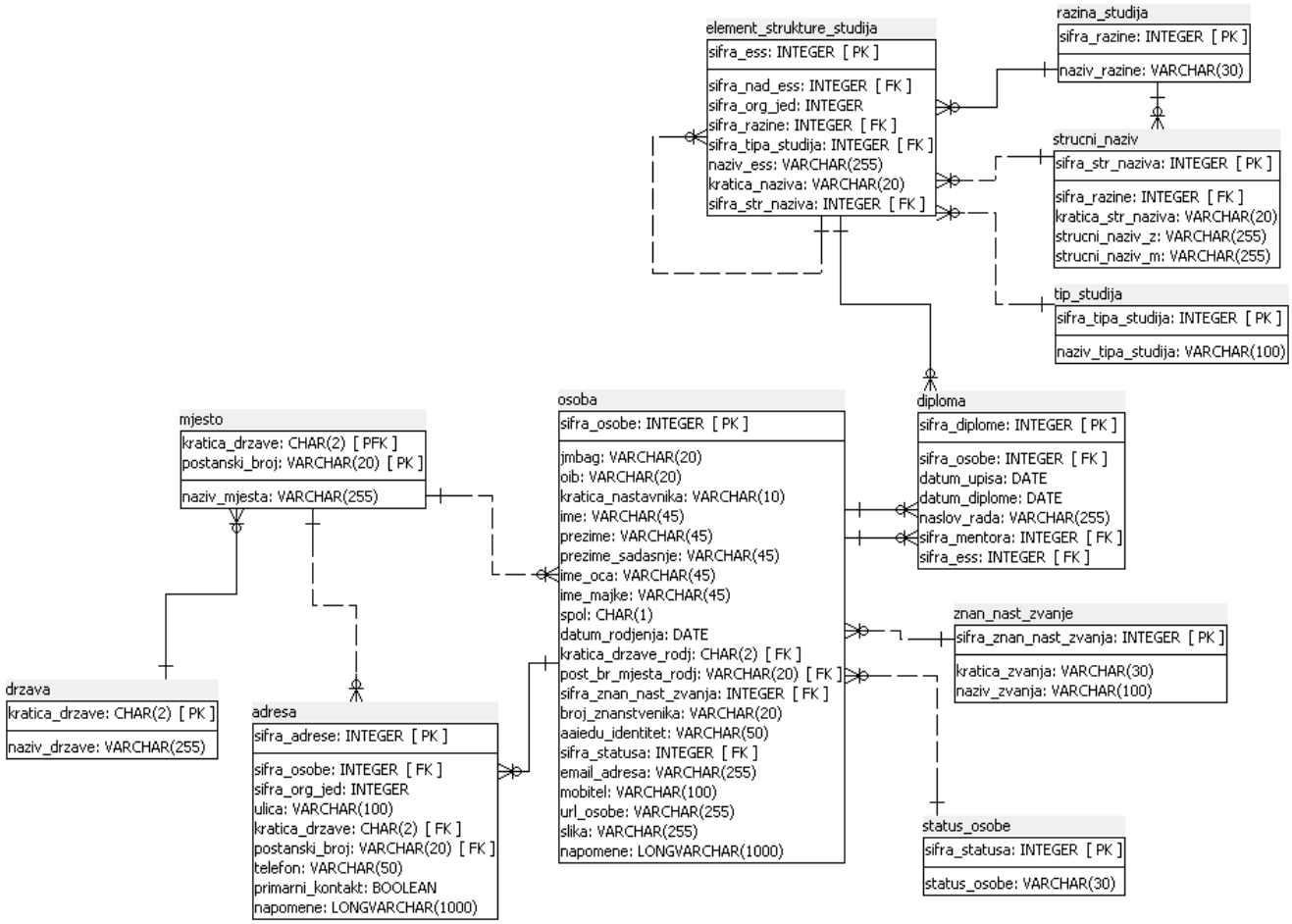
2. Organizacija podataka (struktura baze podataka)

Izrađena aplikacija bavi se upisom podataka o osobama (alumnima i njihovim mentorima) te njihovim diplomama. Za ispravan unos podataka o diplomama potrebno je znati i strukturu studija kojeg je pojedini alumni pohađao. Baza podataka koja je korištena za tu namjenu ima sljedeću strukturu (tablice):

- *osoba,*
- *mjesto,*
- *drzava,*
- *adresa,*
- *status osobe,*
- *znan_nast_zvanje,*
- *diploma,*
- *element_strukture_studija,*
- *razina_studija,*
- *tip_studija i*
- *strucni_naziv.*

Naknadno se planiraju dodati još podaci o AMAC udrugama, članstvu u pojedinoj udruzi i zaposlenju alumna pa će za to biti potrebno dodati još neke tablice. Dijagram baze, na kojem su vidljive veze između tablica, prikazan je u sljedećem odlomku.

2.1 Dijagram baze podataka



Slika 1: Dijagram baze podataka

2.2 Opis tablica Alumni baze

U nastavku će ukratko biti opisane tablice u bazi - koje atribute sadrže i koje značenje ima pojedini atribut.

Tablica drzava:

- *kratica_drzave* - kratica države, npr. "HR"; primarni ključ
- *naziv_drzave* - puni naziv države, npr. "Hrvatska"

Tablica mjesto:

- *postanski_broj* - poštanski broj mjesta, npr. "10 000"; dio primarnog ključa
- *kratica_drzave* - veza na tablicu "drzava"; dio primarnog ključa
- *naziv mjesta* - puni naziv mjesta, npr. "Zagreb"

Tablica adresa:

- *sifra_adrese* - primarni ključ
- *sifra_osobe* - veza na tablicu "osoba"
- *ulica* - naziv ulice i kućni broj
- *kratica_drzave* - veza na tablicu "mjesto"; dio stranog ključa
- *postanski_broj* - veza na tablicu "mjesto"; dio stranog ključa
- *telefon* - broj telefona na toj adresi
- *primarni_kontakt* - oznaka je li upisana adresa primarni kontakt preko kojeg se može kontaktirati osoba
- *napomene* - dodatne napomene (po potrebi)

Tablica osoba:

- *sifra_osobe* - primarni ključ
- *oib* - OIB osobe, ako je poznat; mora biti jedinstven
- *jmbag* - JMBAG osobe, nemaju ga sve osobe; mora biti jedinstven
- *ime* - ime osobe
- *prezime* - prezime osobe
- *prezime_sadasnje* - sadašnje prezime, ako je različito od prethodnog; može se upisati i naknadno
- *ime_oca* - ime oca, podatak koji postoji u starijim zapisnicima
- *ime_majke* - ime majke, podatak koji postoji u novijim zapisnicima
- *spol* - spol osobe

- *datum_rodjenja* - datum rođenja osobe
- *kratica_drzave_rodj* - veza na tablicu "mjesto"; dio stranog ključa
- *post_br_mjesta_rodj* - veza na tablicu "mjesto"; dio stranog ključa
- *sifra_znan_nast_zvanja* - znanstveno-nastavno zvanje osobe (npr. "dipl.ing."), veza na istoimenu tablicu
- *broj_znanstvenika* - matični broj u registru znanstvenika (npr. "223314"); može se koristiti za povezivanje s drugim bazama (npr. bib.irb.hr)
- *aaiedu_identitet* - AAI@EDU identitet; može se koristiti za povezivanje s drugim bazama
- *sifra_statusa* - status osobe (npr. "aktivan/aktivna"), veza na istoimenu tablicu
- *email_adresa* - adresa elektroničke pošte (za kontakt)
- *mobitel* - broj mobitela (za kontakt)
- *url_osobe* - adresa osobne web stranice
- *slika* - poveznica (eng. *link*) na sliku osobe
- *napomene* - dodatne napomene o osobi (po potrebi)

Tablica status_osobe:

- *sifra_statusa* - primarni ključ
- *status_osobe* - status, npr. "aktivan/aktivna", "preminuo/preminula"

Tablica znan_nast_zvanje:

- *sifra_znan_nast_zvanja* - primarni ključ
- *kratica_zvanja* - kratica znanstveno-nastavnog zvanja, npr. "prof.dr.sc"
- *naziv_zvanja* - puni naziv znanstveno-nastavnog zvanja, npr. "profesor", "docent"

Tablica element_strukture_studija:

- *sifra_ess* - šifra elementa strukture studija, primarni ključ
- *sifra_nadr_ess* - nadređen element strukture studija, veza na istu tablicu; služi za izgradnju hijerarhije elemenata (npr. Elektrotehnika → Elektroenergetika → Opća energetika)
- *sifra_org_jed* - šifra organizacijske jedinice (za FER iznosi 36)
- *sifra_razine_ess* - razina studija (npr. "preddiplomski"), veza na istoimenu tablicu
- *sifra_tipa_studija* - tip studija (npr. "studij", "smjer"), veza na istoimenu tablicu
- *naziv_ess* - puni naziv elementa strukture studija, npr. "Programsko inženjerstvo i informacijski sustavi"

- *kratica_naziva* - skraćeni naziv elementa strukture studija, npr. "PIIS"
- *sifra_str_naziva* - stručni naziv koji se dobije završetkom konkretnog studija, npr. "diplomirani inženjer elektrotehnike"; veza na istoimenu tablicu

Tablica razina_studija:

- *sifra_razine* - primarni ključ
- *naziv_razine* - puni naziv razine studija, npr. "dodiplomski"

Tablica tip_studija:

- *sifra_tipa_studija* - primarni ključ
- *naziv_tipa_studija* - puni naziv tipa studija, npr. "smjer" ili "usmjerenje"

Tablica strucni_naziv:

- *sifra_str_naziva* - primarni ključ
- *kratica_str_naziva* - kratica naziva, npr. "dipl.ing."
- *strucni_naziv_z* - puni stručni naziv za žene, npr. "magistra inženjerka računarstva"
- *strucni_naziv_m* - puni stručni naziv za muškarce, npr. "magistar inženjer računarstva"
- *sifra_razine_ess* - razina studija, npr. "diplomski"; veza na istoimenu tablicu

Tablica diploma:

- *sifra_diplome* - primarni ključ
- *sifra_osobe* - alumni (vlasnik diplome), veza na tablicu "osoba"
- *sifra_mentora* - mentor na konkretnoj diplomi, veza na tablicu "osoba"
- *datum_upisa* - datum upisa studija (ako je poznat)
- *datum_diplome* - datum diplomiranja
- *naslov_rada* - naslov rada
- *sifra_ess* - završeni studij/smjer/usmjerenje/modul/profil; veza na tablicu "element_strukture_studija"

3. Korištene tehnologije

Za spremanje podataka o alumnima korištena je baza podataka *SQLite*. Budući da je sama aplikacija zasnovana na objektnom modelu, bilo je potrebno nekako povezati objekte s tablicama u bazi - za to je korištena biblioteka *NHibernate*. Konačno, za prikaz podataka korisnicima izrađeno je grafičko sučelje, korištenjem grafičkog podsustava WPF (eng. *Windows Presentation Foundation*). Korištene tehnologije će ukratko biti opisane u nastavku dokumenta.

3.1 Baza podataka *SQLite*

SQLite je biblioteka koja implementira jednostavnu bazu podataka. Velikim je dijelom neovisna o drugim bibliotekama i operacijskom sustavu, što ju čini pogodnom za korištenje u aplikacijama koje se, bez dodatnih izmjena, planiraju koristiti na različitim računalima s različitim konfiguracijama. Pisana je u programskom jeziku C te ju je moguće prevesti s bilo kojim standardnim C prevoditeljem (eng. *compiler*).

Većina sustava za upravljanje bazama podataka implementirano je kao zasebni poslužiteljski proces. Zbog toga programi koji žele pristupati bazi podataka moraju komunicirati s poslužiteljem korištenjem neke vrste međuprocesne komunikacije (najčešće TCP/IP). Zahtjeve dakle šalju poslužitelju i od njega primaju rezultate. Baza podataka *SQLite* ne funkcionira na taj način, već programi koji žele pristupiti bazi čitaju i pišu direktno u datoteke baze na disku. Ovakav pristup ima i prednosti i nedostataka. Prednosti su sigurno što u slučaju baze podataka *SQLite* nema potrebe za instalacijom, konfiguracijom, inicijalizacijom i upravljanjem dodatnim poslužiteljskim procesom. Isto tako, za korištenje baze nisu potrebne administratorske ovlasti, već svaki program koji ima pristup disku može koristiti bazu. S druge strane, baze podataka koje koriste zasebni poslužiteljski proces nude bolju zaštitu od grešaka u klijentu te omogućavaju lakše zaključavanje baze pa time i bolju konkurentnost.

Baza podataka *SQLite* je transakcijska, što znači da se sve izmjene u bazi odvijaju putem transakcija. Baza osigurava ACID (eng. *Atomic, Consistent, Isolated, and Durable*) svojstva transakcija čak i kad se transakcija prekine uslijed rušenja programa, rušenja operacijskog sustava ili nestanka napajanja.

Nadalje, *SQLite* je vrlo kompaktna biblioteka pa čak i s uključenim svim mogućnostima ne zauzima puno prostora i radi dobro čak i u okruženjima s malo memorije.
Budući da sama aplikacija koja je nastala kao rezultat ovog diplomskog rada nije velika ni komplicirana, za bazu je odabrana upravo baza *SQLite* - zbog svih gore navedenih razloga, od kojih su dva bila ključna - jednostavnost korištenja i brzina.

3.2 O/R mapiranje podataka - biblioteka NHibernate

Iako u današnje vrijeme postoji niz različitih baza podataka koje se mogu koristiti u različite svrhe, pokazuje se kako relacijske baze u većini slučajeva i za većinu primjena i dalje pružaju najbolje mogućnosti. Budući da je Alumni aplikacija zasnovana na objektnom modelu podataka, logičan izbor za perzistenciju podataka bila bi neka od objektno orijentiranih baza, kojih danas ima već podosta. Međutim, problemi s objektno-orijentiranim bazama podataka su sljedeći:

- relacijske i objektne baze ne mogu međusobno razmjenjivati podatke,
- loša optimizacija deklarativnih upita i
- (možda najveći problem) nedostatak standardnog upitnog jezika (kao što je kod relacijskih baza upitni jezik SQL) i nekompatibilnost s upitnim jezikom SQL.

Upravo zbog spomenutih razloga, isključena je mogućnost korištenja objektno-orijentirane baze podataka i odabrana je relacijska. Problem koji se u tom slučaju javlja je tzv. *O/R Impedance Mismatch,* odnosno neslaganje između objektne i relacijske paradigme. Riječ je zapravo o nizu problema, od kojih su najvažniji:

- **neslaganje između objektne i relacijske tehnologije** - relacije predstavljaju činjenice o povezanosti podataka, dok su objekti modeli realnosti (sadrže i podatke i ponašanje)
- **neslaganje pravila o jedinstvenosti -** dva retka u tablici koji sadrže iste podatke smatraju se istima te u bazi nije ni dozvoljeno pohranjivanje dvaju istih redaka, dok objekti koji sadrže potpuno iste podatke i dalje mogu biti različiti (imaju različite memorijske adrese)
- **neslaganje topologije podataka -** objekti su najčešće nekako hijerarhijski ustrojeni, dok među tablicama u bazi ne postoji mogućnost izgradnje hijerarhije.

Kao rješenje prethodno opisanih problema pojavili su se tzv. "O/R maperi". Riječ je o alatima koji se koriste za automatsko i transparentno perzistiranje objekata u tablice relacijske baze podataka. Ta automatska transformacija iz jedne u drugu reprezentaciju podataka opisuje se najčešće preko određene vrste metapodataka. U nastavku će biti opisano jedno od postojećih rješenja za perzistenciju objekata u relacijske baze - biblioteka *NHibernate,* koja je ujedno korištena i u Alumni aplikaciji.

3.2.1 NHibernate

NHibernate je inačica popularne Java biblioteke *Hibernate* prilagođena razvojnom okruženju .NET. Cilj *NHibernate*-a je razvojnim programerima olakšati rad i umanjiti količinu kôda potrebnog za prevladavanje razlika između objektnog modela i relacijske baze. Pojednostavljeno opisano, *NHibernate* funkcionira na sljedeći način:

- na ulazu prima C# klasu sa svojstvima koje treba pohraniti u bazu i dohvatiti kasnije te XML datoteku s podacima za mapiranje,
- na izlazu daje automatski generirani SQL kod (kad se pojedini objekt sprema u bazu ili se dohvaća iz baze).

Ono što je najvažnije, *NHibernate* ne zahtjeva da se objekti na bilo koji način prilagođavaju načinu na koji će biti spremljeni u bazi. Tako je aplikaciju moguće graditi potpuno temelju na objektnom modelu, a tehniku spremanja podataka prepustiti alatu koji je za to zadužen. Kao primjer korištenja spomenute biblioteke, u nastavku je dan primjer razreda iz objektnog modela korištenog u izradi Alumni aplikacije te pripadna XML datoteka korištena za njegovu transformaciju u tablicu relacijske baze.

Razred Person:

```
public abstract class Person
{
    private int _id;
    private string _oib;
    private string _jmbag;
    private string _name;
    private string _surname;
    private string _actualSurname;
    private string _fatherName;
    private string _motherName;
    private char _sex;
```

```
        private DateTime? _dateOfBirth;
        private Locality _placeOfBirth;
        private ScienceTeachingProfession _scienceTeachingProfession;
        private string _scientistNum;
        private string _aaieduIdentity;
        private PersonStatus _status;
        private string _eMail;
        private string _cellPhoneNumber;
        private string _url;
        private string _picture;
        private string _annotation;
        private string _type;
    }
```

Pripadna XML datoteka:

```
<?xml version="1.0" encoding="utf-8" ?>

<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
assembly="Alumni_DomainModel" namespace="Alumni_DomainModel">

  <class name="Alumni_DomainModel.Person, Alumni_DomainModel"
      table="Osoba" discriminator-value="PersonType">

    <id name="Id" column="sifra_osobe" type="int">
      <generator class="identity" />
    </id>

    <discriminator column="Person_TYPE" type="string"/>

    <property name="Oib" type="string" column="oib" not-null="false"/>
    <property name="Jmbag" type="string" column="jmbag" not-
      null="false"/>
    <property name="Name" type="string" column="ime" not-null="true"/>
    <property name="Surname" type="string" column="prezime" not-
      null="true"/>
    <property name="ActualSurname" type="string"
      column="prezime_sadasnje"    not-null="false"/>
    <property name="FatherName" type="string" column="ime_oca" not-
      null="false"/>
    <property name="MotherName" type="string" column="ime_majke" not-
      null="false"/>
    <property name="Sex" type="char" column="spol" not-null="true"/>
    <property name="DateOfBirth" type="DateTime" column="datum_rodjenja"
      not-null="false"/>
    <many-to-one name="PlaceOfBirth" class="Locality" not-null="false"
      lazy="false" cascade="delete">
      <column name="post_br_mjesta_rodj"/>
      <column name="kratica_drzave_rodj"/>
    </many-to-one>
    <many-to-one name="ScienceTeachingProfession"
      column="id_znanstveno_nast_zvanja"
      class="ScienceTeachingProfession" not-null="false" lazy="false"/>
```

```
    <property name="ScientistNum" type="String" column="br_znanstvenika"
      not-null="false"/>
    <property name="AaieduIdentity" type="String"
      column="aaiedu_identitet" not-null="false"/>
    <many-to-one name="Status" column="id_statusa" class="PersonStatus"
      lazy="false"/>
    <property name="EMail" type="String" column="email_adresa" not-
      null="false"/>
    <property name="CellPhoneNumber" type="String" column="mobitel" not-
      null="false"/>
    <property name="Url" type="String" column="url_osobe" not-
      null="false"/>
    <property name="Picture" type="String" column="slike" not-
      null="false"/>
    <property name="Annotation" type="String" column="napomene" not-
      null="false"/>

    <subclass name="Alumni" discriminator-value="Alumni">
    </subclass>

    <subclass name="Mentor" discriminator-value="Mentor">
      <property name="TeacherKey" type="string"
      column="kratica_nastavnika"/>
    </subclass>

  </class>

</hibernate-mapping>
```

Na ovom je primjeru ujedno prikazano i kako je u *NHibernate*-u riješen problem nasljeđivanja među razredima (osoba može biti alumni ili mentor), ali budući da to nije tema ovog rada i ulaženje u detalje bi rezultiralo pretjeranom opširnošću, o tome ovdje neće biti riječi.

3.3 Grafičko korisničko sučelje - WPF

Budući da se radi o stolnoj aplikaciji, za izradu grafičkog korisničkog sučelja odabran je podsustav *Windows Presentation Foundation*, koji će detaljnije biti opisan u nastavku.

3.3.1 Windows Presentation Foundation

Windows Presentation Foundation (u nastavu kratko WPF) je grafički podsustav za iscrtavanje korisničkog sučelja u Windows aplikacijama. WPF se prvi put pojavljuje od inačice razvojnog okruženja .NET 3.0 pod nazivom "*Avalon*", s namjerom da ponudi moderne mogućnosti korisničkog sučelja, kao što su transparentnost, gradijenti i prilagodljivost samih kontrola. Neke od najznačajnijih prednosti WPF-a su sljedeće:

- iscrtavanje kontrola radi se neovisno o tzv. DPI (eng. *dots per inch*) postavkama operacijskog sustava,
- WPF sučelja iscrtavaju se unutar okruženja *DirectX* i nude podršku za razne grafičke elemente i animacije,
- sve su kontrole prilagodljive i moguće im je jednostavno promijeniti izgled korištenjem različitih stilova (definicijom stila za kontrolu određenog tipa moguće je odjednom promijeniti izgled svim kontrolama tog tipa),
- za specifikaciju dizajna, odnosno objekata i kontrola na sučelju koristi jezik za označavanje XAML.

Iako za izradu grafičkog korisničkog sučelja Windows aplikacija postoji i mogućnost korištenja sustava *Windows Forms*, upravo zbog većih mogućnosti prilagodbe grafičkih elemenata korišten je sustav WPF. U nastavku je prikazana specifikacija jednog (jednostavnog) prozora Alumni aplikacije u jeziku XAML.

```
<Window x:Class="Alumni_GUI.LoginScreen"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Prijava" WindowStartupLocation="CenterScreen"
    Background="#191970" WindowStyle="SingleBorderWindow" Height="172"
      Width="300">
    <Grid>

      <Grid.Resources>
        <ResourceDictionary>
           <ResourceDictionary.MergedDictionaries>
              <ResourceDictionary
      Source="/Alumni_GUI;component/skins/AppStyles.xaml" />
             </ResourceDictionary.MergedDictionaries>
          </ResourceDictionary>
       </Grid.Resources>

       <Grid.RowDefinitions>
         <RowDefinition Height="*" />
         <RowDefinition Height="*" />
         <RowDefinition Height="*" />
         <RowDefinition Height="*" />
       </Grid.RowDefinitions>
```

```
    <Grid.ColumnDefinitions>
       <ColumnDefinition Width="*" />
       <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>

       <Label Name="LabelUsername" Height="Auto"
     Style="{StaticResource GrayLabel}" Content="Korisničko ime: "
     Grid.Row="0" Grid.Column="0" Margin="0,0,8.336,0"
     VerticalAlignment="Center"/>

       <TextBox Name="textBoxUsername" Style="{StaticResource
     GrayTextBox}" Margin="3,3,3,3" Grid.Row="0"
     Grid.Column="1"></TextBox>

       <Label Name="LabelPassword" Height="Auto"
     Style="{StaticResource GrayLabel}" Content="Zaporka: "
     Grid.Row="1" Grid.Column="0" Margin="0,0,8.336,0"
     VerticalAlignment="Center"/>

       <PasswordBox Name="passwordBox" Grid.Row="1" Grid.Column="1"
     Background="#191970" Foreground="#E0E0E0" IsEnabled="true"
     Style="{StaticResource GrayPasswordBox}" Visibility="Visible"
     Margin="3,3,3,3"/>

       <Label Name="labelLoginFailed" Style="{StaticResource
     GrayLabel}" Content="Prijava nije uspjela! " Grid.Row="2"
     Grid.ColumnSpan="2" Visibility="Hidden"
     Margin="78,0,74,8.55"></Label>

       <Button Margin="30,0,18,8" Name="buttonLogin" Grid.Row="3"
     Click="buttonLogin_Click">Prijavi se</Button>

       <Button Margin="30,0,18,8" Name="buttonCancel" Grid.Row="3"
     Grid.Column="1" Click="buttonCancel_Click">Odustani</Button>

    </Grid>
</Window>
```

4. Opis izrađenog programa

Program je pisan u programskom jeziku C#, korištenjem razvojnog okruženja *Visual Studio 2008.*

Prilikom izrade aplikacije koja je rezultat ovog rada korištena je tzv. višeslojna arhitektura. Drugim riječima, aplikacija je podijeljena na način da je u slučaju potrebe u bilo kojem trenutku moguće promijeniti samo jedan "sloj" aplikacije, a aplikacija bi i dalje normalno radila. Sam objektni model domene, način na koji je obavljena podjela na slojeve, kao i opis pojedinih slojeva i njihove funkcionalnosti dan je u nastavku.

4.1 Višeslojna aplikacija - oblikovni obrazac MVC

Prilikom raslojavanja aplikacije korišten je oblikovani obrazac MVC (eng. *Model-View-Controller*). MVC je klasični oblikovni obrazac često korišten u aplikacijama u kojima postoji potreba za prikazom istih podataka na različite načine. Kao što sam naziv govori, sastoji se od tri dijela:

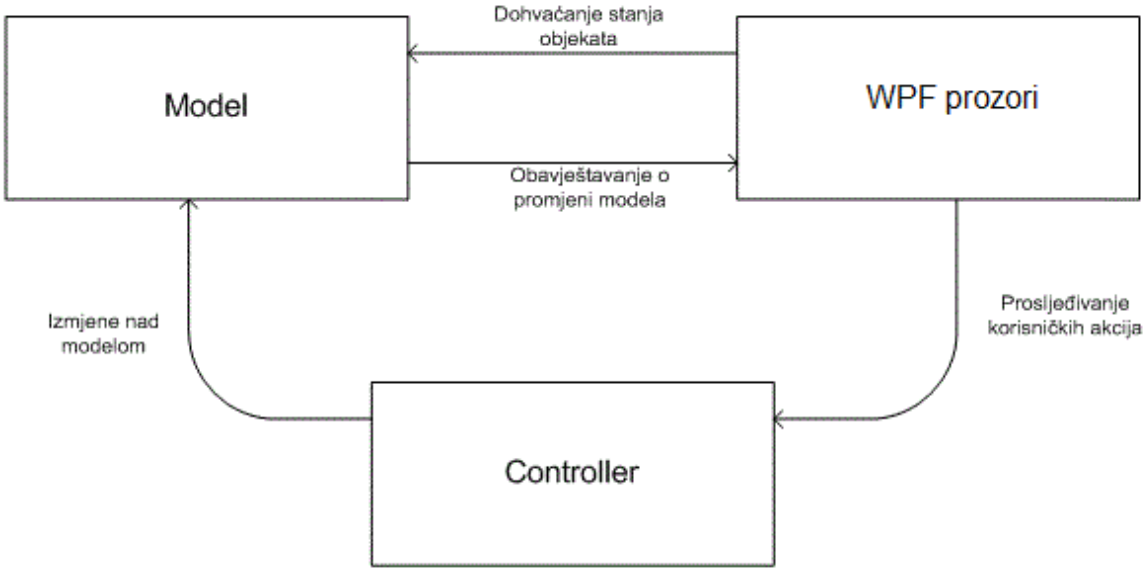
- **Model**
  - sadrži podatke koji predstavljaju problem (objektni model domene)
  - sadrži i pravila koja omogućavaju pristup do podataka i njihovu izmjenu

- **Controller**
  - odgovara na korisnikove akcije i obavlja tražene izmjene nad modelom
  - interakcije koje korisnik provodi s *view*-om provodi u akcije nad modelom
  - u ovisnosti o korisničkim akcijama odabire *view* koji će se sljedeći prikazati

- **View**
  - samo prikazuje trenutno stanje modela korisniku.

Dodatno, kako bi se osiguralo osvježavanje svakog *view*-a uslijed izmjena nad podacima koje prikazuju, korišten je oblikovni obrazac *Observer*. Spomenuti oblikovni obrazac definira 1 prema N ovisnost među objektima na način da kad jedan objekt promijeni stanje (u sklopu obrasca *Observer* objekti iz modela domene nazivaju se *Subject*- i), svi ovisni

objekti budu obaviješteni o načinjenoj izmjeni. Ti objekti koji "prate" stanje modela nazivaju se *Observer*-i. U MVC arhitekturi objekti koji se prate (*subject*) su objekti iz modela domene, a *observer*-i koji prate njihove izmjene su *view*-ovi, odnosno sami prozori u grafičkom sučelju. Pritom objekti iz modela domene nemaju informacije o detaljima implementacije grafičkog sučelja, već se sami *view*-ovi registriraju kod objekata kao njihovi *observer*-i te na taj način primaju obavijesti o izmjenama. Kako je to konkretno izvedeno u Alumni aplikaciji biti će opisano kasnije.
Prikaz opisane arhitekture dan je na sljedećoj slici.



Slika 2: Oblikovani obrazac MVC

4.1.1 Model domene

Model domene (koji predstavlja komponentu "Model" iz MVC obrasca) za Alumni aplikaciju nalazi se unutar projekta *Alumni_DomainModel,* a sastoji se od sljedećih klasa:

- *Address.cs,*
- *Alumni.cs,*
- *Country.cs,*
- *Diploma.cs,*
- *ElementOfStudyStructure.cs,*
- *Locality.cs,*
- *Mentor.cs,*
- *Person.cs,*

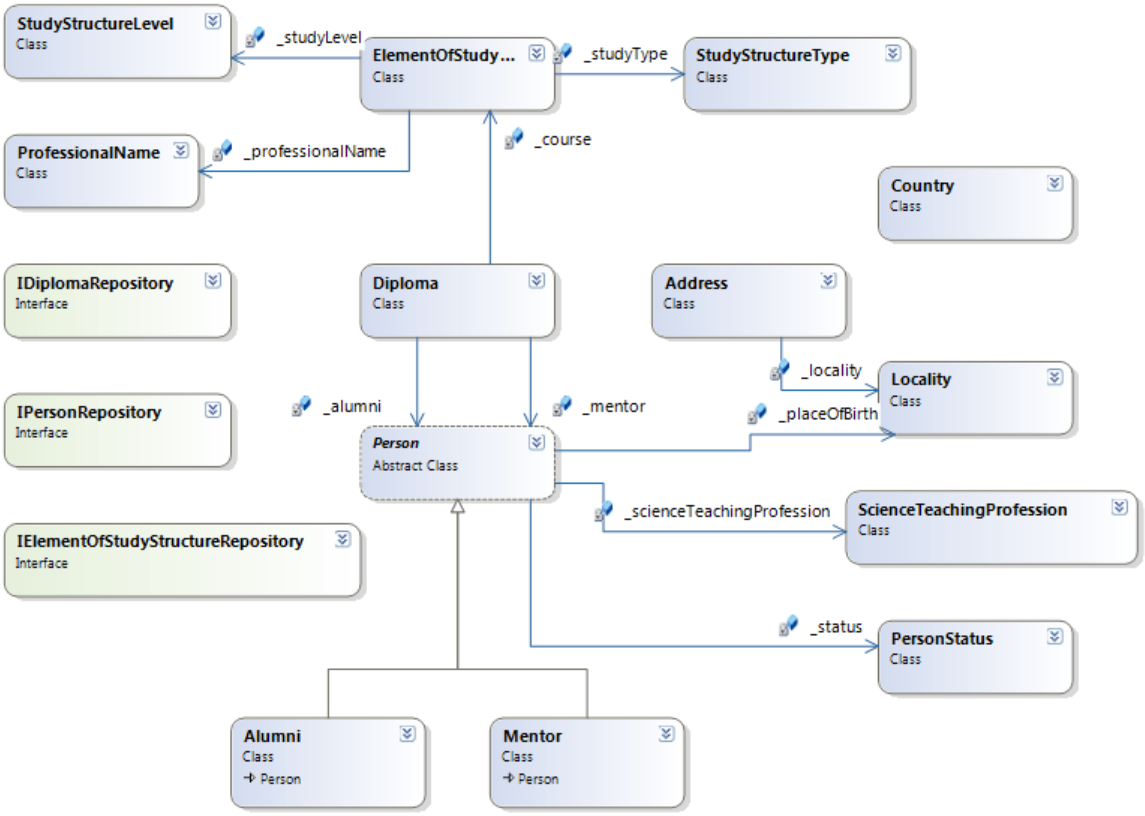
- *PersonStatus.cs,*
- *ProfessionalName.cs,*
- *ScienceTeachingProfession.cs,*
- *StudyStructureLevel.cs i*
- *StudyStructureType.cs.*

Dodatno, u modelu se nalaze i prethodno spomenuta "pravila" za rad s objektima iz domene. Riječ je o sljedećim sučeljima:

- *IDiplomaRepositoy.cs,*
- *IElementOfStudyStructureRepository.cs* i
- *IPersonRepository.*

Budući da je riječ o sučeljima, ona definiraju samo način korištenja objekata, a konkretne implementacije repozitorija koje se bave sa spremanjem, brisanjem, dohvaćanjem i izmjenom objekata nalaze se u podatkovnom sloju, odnosno sloju koji komunicira s bazom podataka pa će tamo naknadno biti i opisane.

U ovom se projektu nalazi još nekoliko razreda koji ne pripadaju u sam model domene, a to su razred *Exceptions.cs* koji sadrži popis svih korisnički definiranih iznimki do kojih može doći prilikom rada s aplikacijom te razred *User.cs* koji se koristi za prijavu u samu aplikaciju. Slijedi prikaz veza među objektima, odnosno dijagram razreda izgrađenog objektnog modela.



Slika 3: Korišteni objektni model

4.1.2 Komunikacija s bazom podataka - podatkovni sloj

Podatkovni se sloj nalazi u projektu *Alumni_DAL*. Sastoji se od nekoliko dijelova:

- konfiguracijska datoteka biblioteke *NHibernate,*
- XML datoteke za mapiranje objekata u tablice u bazi (direktorij *Mapping_Files*),
- Objekti za rad s modelom (repozitoriji i "tvornice", direktorij *Repositories_and_factories*) i
- Objekt za dohvaćanje *NHibernate* sjednice - datoteka *NHibernateHelper.cs*.

Konfiguracijska datoteka biblioteke *NHibernate*

Riječ je o XML datoteci *hibernate.cfg.xml* koja se koristi za općenitu konfiguraciju namijenjenu biblioteci *NHibernate.* U njoj je zapisan tip korištene baze podataka, staza do same baze na disku i još nekoliko detalja. U slučaju potrebe za korištenjem druge baze podataka (npr. *SQL Server*), potrebno je samo u ovoj konfiguracijskoj datoteci promijeniti nekoliko svojstava (eng. *property*) - *connection.driver_class, connection.connection_string i dialect.*

XML datoteke za mapiranje objekata u tablice u bazi (direktorij *Mapping_Files*)

Za svaki razred čiji se objekti perzistiraju u bazi postoji jedna XML datoteka naziva "ime_razreda.hbm.xml", primjerice za razred *Person* XML dokument ima naziv "Person.hmb.xml". Struktura jednog takvog XML dokumenta prikazana je ranije pa se ovdje neće ponavljati.

Objekti za rad s modelom (repozitoriji i "tvornice", direktorij *Repositories_and_factories*)

Razredi iz ovog direktorija vrlo su važni za ispravan rad aplikacije. Oni predstavljaju vezu aplikacije s bazom podataka - sve izmjene nad modelom spremaju se preko njih u bazu, a isto tako i svi postojeći podaci se preko njih dohvaćaju u aplikaciju. Riječ je o sljedećim razredima:

- *DiplomaRepository.cs,*
- *ElementOfStudyStructureRepository.cs,*
- *PersonFactory.cs,*
- *PersonRepository.cs i*
- *UsersRepository.cs.*

Razred ***DiplomaRepository.cs*** sadrži metode za spremanje podataka o diplomama u bazu, dohvaćanje diploma iz baze (na temelju različitih kriterija), izmjenu diploma i njihovo brisanje.

Razred ***ElementOfStudyStructureRepository.cs*** sadrži niz metoda za rukovanje elementima strukture studija, tipovima i razinama studija te stručnim nazivima.

Razred ***PersonFactory.cs*** sadrži metode za stvaranje instanci razreda Alumni i Mentor. Razlog zbog kojeg postoji poseban razred koji se bavi stvaranjem spomenutih objekata jest što je prije njihovog stvaranja potrebno stvoriti ili dohvatiti niz drugih objekata potrebnih za stvaranje samih osoba (npr. mjesto, država, adresa, status osobe, itd.). Da bi se osiguralo kako se sve potrebno za stvaranje tih složenijih objekata uvijek izvodi na jednom mjestu, stvoren je poseban razred koji se bavi upravo time.

Razred ***PersonRepository.cs*** sadrži metode za rukovanje podacima o osobama (alumnima i mentorima), njihovim adresama, zanimanjima, statusima i dr.

Konačno, razred ***UsersRepository.cs*** namijenjen je rukovanju informacijama o samim korisnicima aplikacije (njihovim dodavanjem, izmjenom podataka i brisanjem).

Objekt za dohvaćanje *NHibernate* sjednice - datoteka *NHibernateHelper.cs*

Za rad s bibliotekom *NHibernate* potrebna su dva objekta - *SessionFactory* i *Session*. Objekt *SessionFactory* preporuča se stvoriti samo jednom jer je njegovo stvaranje "skupo", a u daljnjem radu aplikacije taj se objekt koristi za stvaranje objekata *Session* po potrebi (svaka komunikacija s bazom mora se odvijati unutar *NHibernate* sjednice). U datoteci *NHibernateHelper.cs* nalaze se metode koje omogućavaju upravo tu funkcionalnost.

4.1.3 Upravljanje podacima - aplikacijski sloj

U aplikacijski sloj ubrajaju se dva projekta - *Alumni_Interfaces* i *Alumni_Controller*. Unutar projekta *Alumni_Interfaces* nalaze se sljedeći razredi i sučelja:

- sučelje *IController.cs,*
- sučelje *IMainController.cs,*
- sučelje *IObserver.cs* i
- apstraktni razred *Subject.cs.*

Sučelja *IController.cs* i *IMainController.cs* biti će detaljnije objašnjena u sklopu samih implementacija (razreda iz projekta *Alumni_Controller*). Sučelje *IObserver.cs* je sučelje koje implementiraju svi prozori u grafičkom sučelju. Na taj način svi prozori implementiraju metodu sučelja *Update()*, unutar koje se obavlja osvježavanje prozora u slučaju promjena modela podataka koje prikazuju. Kako bi se osvježavanje radilo samo u slučaju promjena podataka, svi objekti iz modela domene koji rade izmjene nad podacima (a to su razredi *DiplomaRepository.cs, ElementOfStudyStructureRepository.cs, PersonRepository.cs* i *UsersRepository.cs)* nasljeđuju apstraktni razred *Subject.cs* pa samim time nasljeđuju i metode:

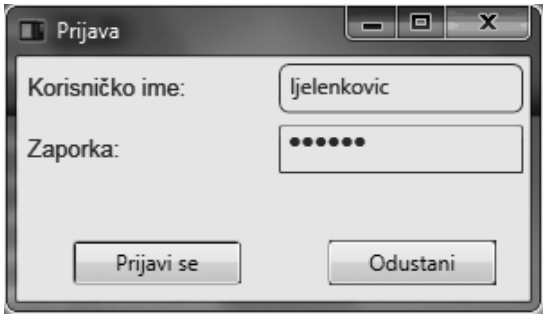
- `public void Attach(IObserver obs)` - za registraciju *obsever*- a nad određenim *subject*-om,
- `public void Delete(IObserver obs)` - za brisanje registracije i
- `public void NotifyObservers()` - za slanje obavijesti uslijed izmjena nad podacima.

U projektu *Alumni_Controller* nalaze se razredi *MainController.cs* i *Controller.cs*. Riječ je o implementacijama istoimenih sučelja iz projekta *Alumni_Interfaces*. Spomenuti razredi obavljaju ulogu *controller*-a iz MVC arhitekture, što znači da grafičko sučelje sve

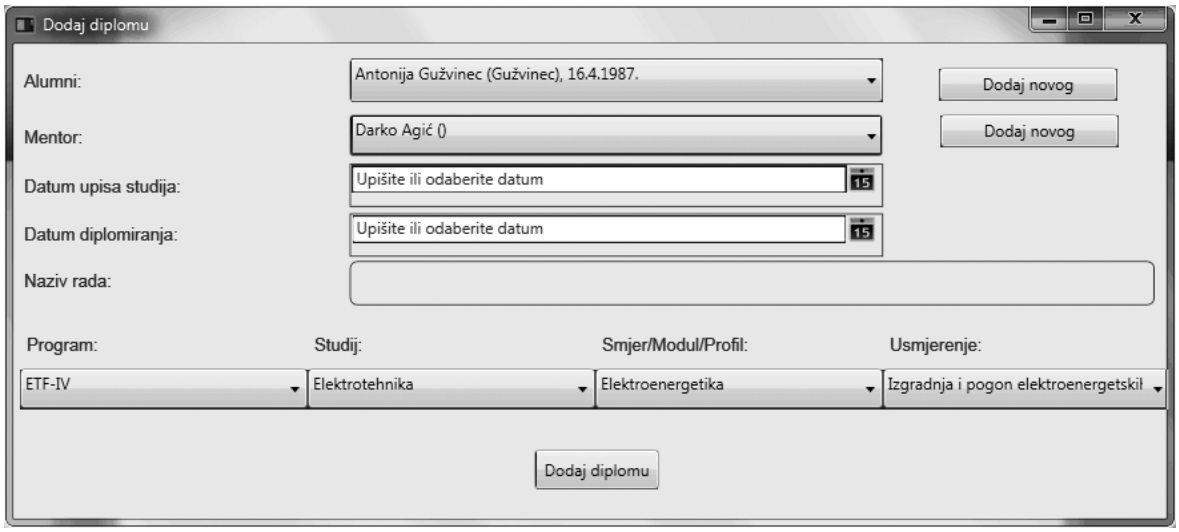
korisničke akcije prosljeđuje tim razredima, oni obavljaju potrebnu obradu podataka te izmjene prosljeđuju podatkovnom sloju, čime se spremaju u bazu podataka. Razlika između ta dva razreda je jedino u tome što *MainController* obrađuje korisničke akcije koje se dogode na "glavnom" prozoru aplikacije, dok *Controller* obrađuje sve akcije koje se dogode na ostalim prozorima.

4.1.4 Korisničko sučelje - prezentacijski sloj

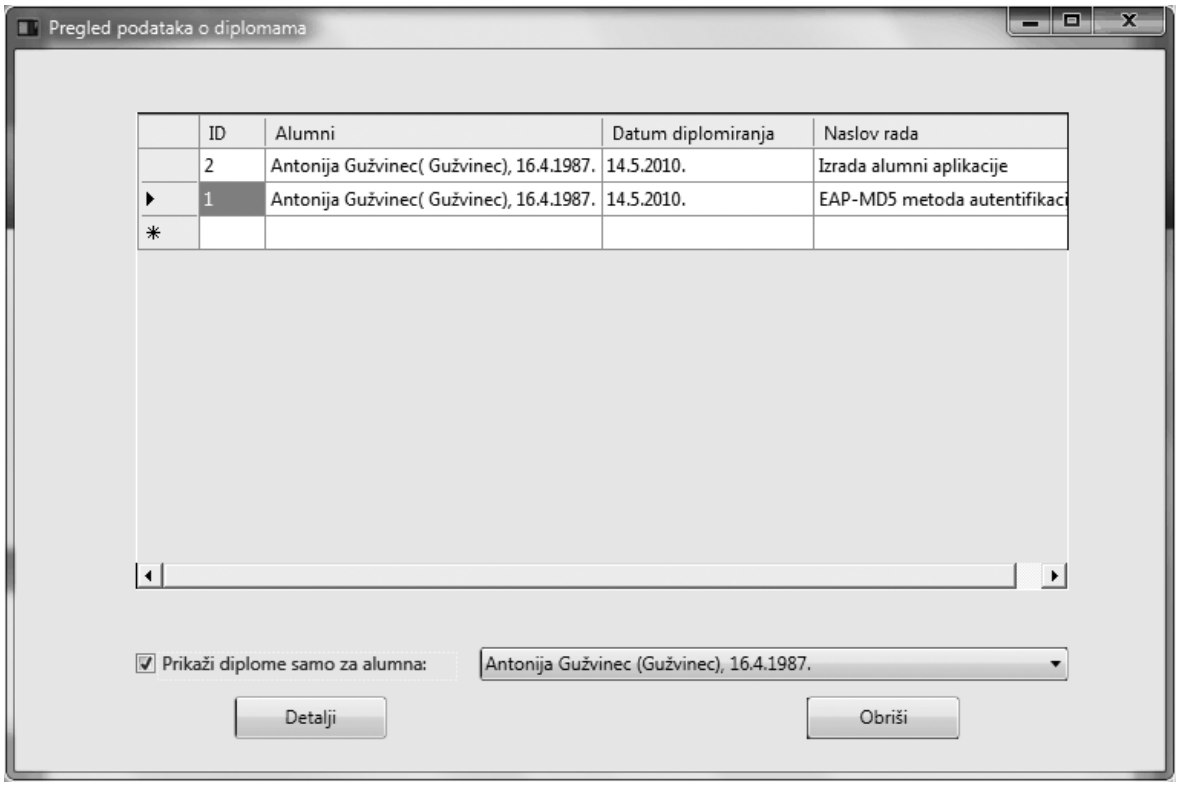
Korisničko je sučelje, kao što je prethodno već spomenuto, izgrađeno u grafičkom podsustavu WPF. Komponente grafičkog sučelja aplikacije nalaze se u projektu *Alumni_GUI*. U direktoriju *skins* nalazi se datoteka *AppStyles.xaml* koja definira korištene stilove različitih WPF kontrola koje se koriste u samom sučelju. Za svaki je prozor definirana jedna XAML datoteka koja definira dizajn (izgled i raspored kontrola) prozora te pripadna *.cs* datoteka koja definira "ponašanje" prozora, odnosno reakcije na korisničke akcije. U nastavku je prikazano nekoliko prozora izrađene aplikacije.



Slika 4: Prozor za prijavu u aplikaciju



Slika 5: Prozor za dodavanje nove diplome



Slika 6: Prozor za pregled podataka o diplomama

4.1.5 Stvaranje baze i sheme podataka

U projektu *Alumni_Tests*, koji je izvorno korišten za testiranje aplikacije, nalazi se datoteka *GenerateShema_Fixture.cs.* Spomenuta datoteka sadrži tzv. jedinični (eng. *unit*) test, odnosno testnu metodu *Can_generate_schema()* koja se koristi za generiranje sheme podataka. Drugim riječima, ova metoda služi za inicijalno stvaranje "veze" između objektnog modela i baze podataka. U slučaju korištenja baze *SQLite*, baza pri pokretanju spomenutog testa ne treba postojati, već ju sam test može stvoriti na temelju XML dokumenata i konfiguracijske datoteke biblioteke *NHibernate* iz projekta *Alumni_DAL*. U slučaju korištenja neke druge baze, npr. *SQL Server,* baza mora prethodno biti stvorena (i tablice moraju odgovarati podacima specificiranim u XML dokumentima za mapiranje objekata), a ova će testna metoda samo povezati tablice iz baze s objektima iz modela. Ako baza postoji, *Can_generate_schema()* će obrisati sve podatke u njoj. Osim toga, u testu se dodatno poziva i metoda za punjenje baze s potrebnim podacima - metoda *FillDatabase()*. Spomenuta metoda bazu puni s podacima o statusima osoba, strukturi studija, mjestima i državama, itd., odnosno s podacima koji su potrebni za ispravan rad aplikacije, a neće se naknadno mijenjati. Za punjenje baze s podacima o hrvatskim mjestima korištena je

datoteka "mjesta.txt". Budući da se prilikom svakog pokretanja testa kreira novi direktorij, samu datoteku s popisom mjesta nije bilo moguće staviti u izvršni direktorij pa je korištena apsolutna staza do datoteke. Stoga je u slučaju ponovnog pokretanja testa potrebno tu stazu promijeniti da odgovara stvarnoj stazi na računalu na kojem se test pokreće. Ovaj test potrebno je pokrenuti samo jednom, u slučaju da se baza želi ponovno stvoriti i konfigurirati. Ako je baza već prisutna, test nije potrebno (ni preporučljivo) pokretati kako se postojeći podaci iz baze ne bi obrisali.

4.1.6 Pokretanje aplikacije

Pokretanje aplikacije moguće je obaviti na dva načina:

- iz razvojnog okruženja *Visual Studio 2008* ili
- instaliranjem na korisničko računalo putem "*Setup*" programa pa pokretanjem izvršne datoteke *Start.exe*.

Ako se aplikacija pokreće iz razvojnog okruženja *Visual Studio 2008*, potrebno je najprije promijeniti "*Data Source*" svojstvo u datoteci *Alumni_DAL/hibernate.cfg.xml* tako da odgovara stazi do mjesta na disku na koje je prekopirana baza (datoteka *AlumniDatabase.db* iz direktorija "baza"). Nakon toga moguće je prevesti i pokrenuti aplikaciju.

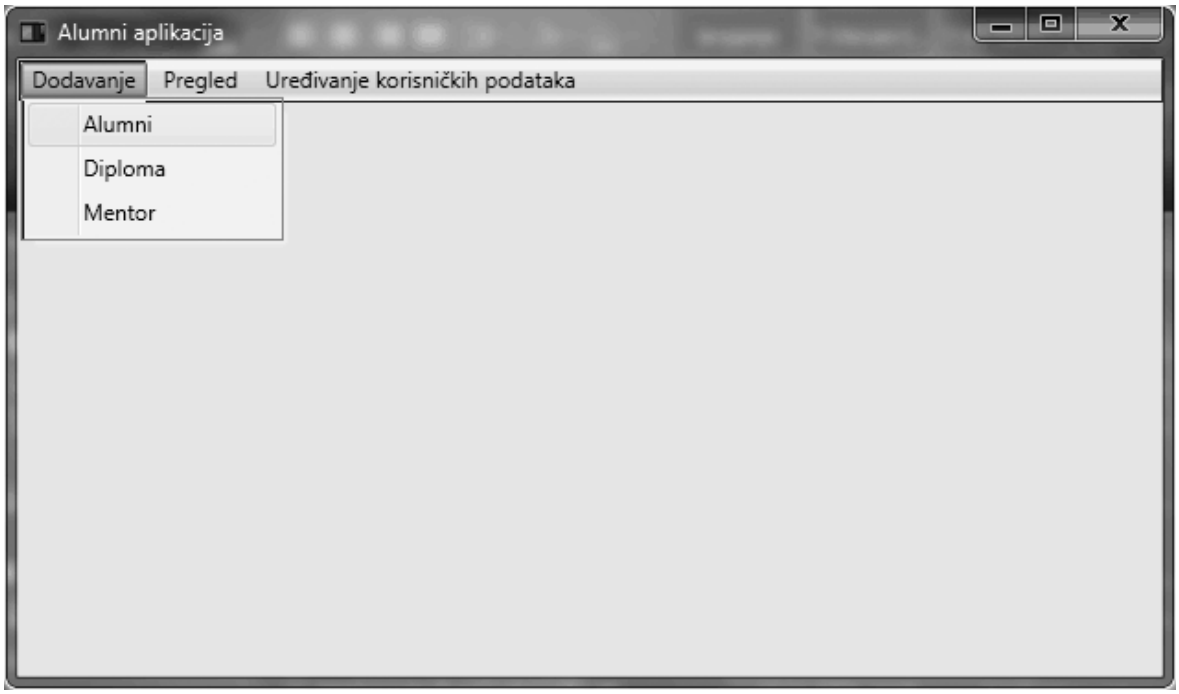
Ako se aplikacija instalira na korisničko računalo pomoću "*Setup*" programa (*Alumni_Application\Debug\setup.exe*), da bi nakon toga ispravno funkcionirala potrebno je također postaviti ispravnu stazu do baze u datoteci *hibernate.cfg.xml.*Ta će se datoteka, zajedno s ostalim potrebnim datotekama, bazom te izvršnom datotekom *Start.exe* nalaziti u direktoriju kojeg korisnik odabere za instalaciju. Ostali podaci za konfiguraciju i pokretanje aplikacije nalaze se u datoteci "*README.txt*" u instalacijskom direktoriju.

Korisnici se razlikuju po ulogama utoliko da korisnik s ulogom "user" može pregledavati i mijenjati samo svoje korisničke podatke (korisničko ime i zaporku), dok korisnik s ulogom "administrator" može, osim izmjene svojih podataka, mijenjati i podatke drugih korisnika, kao i dodavati i brisati korisnike.

4.1.7 Primjer rada aplikacije

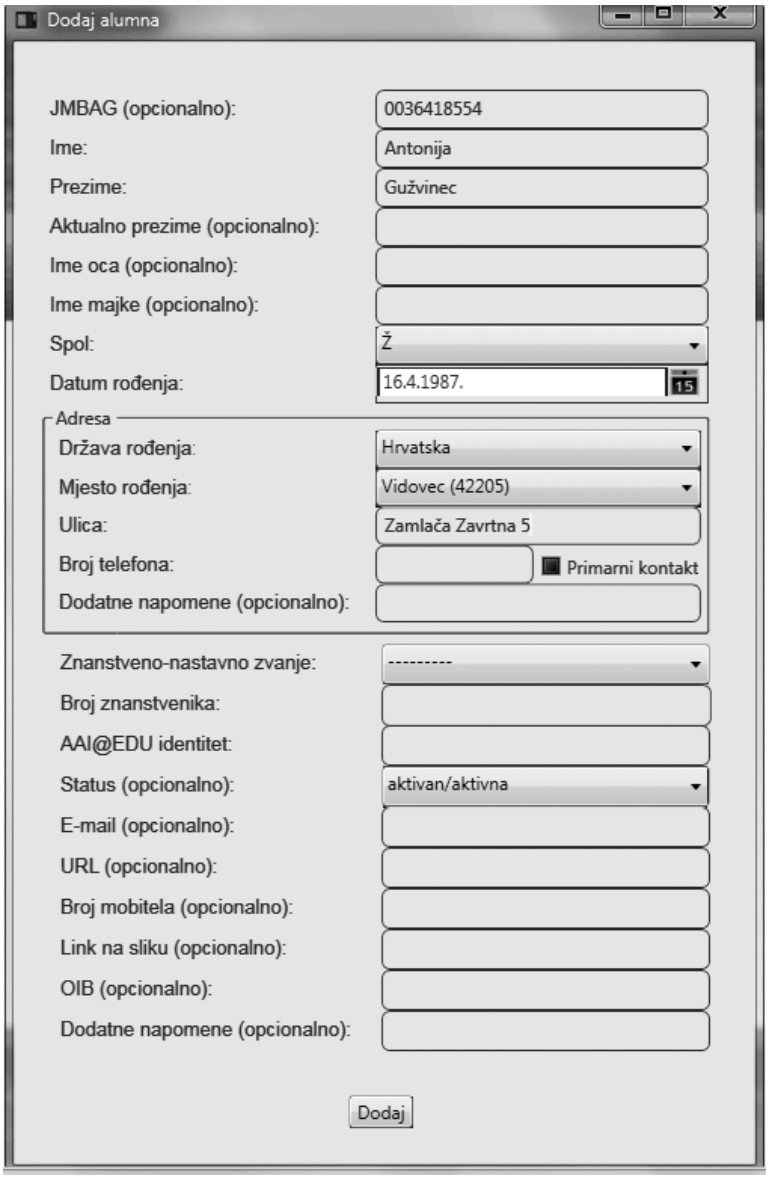
Za pojašnjenje rada aplikacije, u nastavku će biti opisan jedan scenarij korištenja - dodavanje novog alumna.

Prilikom pokretanja aplikacije, otvara se glavni prozor.



Slika 7: Početni prozor aplikacije

Nakon toga, korisnik može u padajućem izborniku "Dodavanje" odabrati opciju "Alumni ", nakon čega mu se otvara prozor za upis podataka o alumnu. "U pozadini" se dogodilo sljedeće: glavni prozor je proslijedio korisničku akciju glavnom kontroleru (*MainController.cs*), a on je reagirao otvaranjem prozora za dodavanje novog alumna. Prilikom otvaranja prozora za dodavanje alumna, glavni kontroler stvorenom prozoru ujedno prosljeđuje referencu na kontroler koji obrađuje ostale korisničke akcije (*Controller.cs*). Otvoreni prozor, koji implementira sučelje *IObserver*, "prijavljuje se" kao *Observer* nad određenim *Subjectom*, odnosno repozitorijom. Konkretno, u ovom slučaju, prozor za dodavanje alumna poziva nad repozitorijom *PersonRepository.cs* metodu *Attach()*.



Slika 8: Prozor za dodavanje novog alumna

Nakon što mu se otvori odgovarajući prozor, korisnik upisuje potrebne podatke o alumnu i odabire dugme "Dodaj". U tom se trenutku kontroleru prosljeđuje korisnikova akcija i on mijenja model - konkretno, pomoću razreda *PersonFactory.cs* stvara novi objekt *Alumni* i poziva u razredu *PersonRepository* metodu za dodavanje nove osobe u bazu. Budući da repozitoriji nasljeđuju klasu *Subject*, nakon svake izmjene podataka ili dodavanja novih (kao u ovom slučaju), repozitorij poziva funkciju *NotifyObservers()*, koja poziva metodu *Update()* za svakog prijavljenog *Observera* i na taj način dolazi do osvježavanja svih prozora kod kojih je to potrebno. Na isti način funkcionira i ostatak aplikacije.

5. Mogućnosti za proširenje

Ovako izgrađena aplikacija pogodna je za jednostavno provođenje izmjena, kao što je korištenje druge baze podataka, izmjena tablica u bazi i sl. U nastavku su opisane neke od mogućnosti.

5.1 Zamjena baze *SQLite* s nekom drugom SQL bazom

Kao što je prethodno napomenuto, zamjena baze *SQLite* s nekom drugom bazom podataka, koja se može nalaziti na udaljenom poslužitelju, ne predstavlja problem. Ako druga baza ima jednake tablice, jedino što je potrebno promijeniti jesu sljedeća svojstva u datoteci *hibernate.cfg.xml*:

- *connection.driver_class,*
- *connection.connection_string* i
- *dialect.*

Ako postoji potreba za istovremenim pristupom podacima, potrebno je obratiti pažnju na nekoliko stvari:

- nikad se ne smije stvoriti više od jedne instance objekta *ISession* ili *ITransaction* koje istovremeno pristupaju bazi. Objekt *ISession* prati izmjene načinjene samo u istoj sjednici, što znači da drugi *ISession* objekt može raditi sa zastarjelim podacima.
- objekt *ISession* ne podržava rad u višedretvenom okruženju pa se nikako ne preporuča istovremeno pristupati istom objektu *ISession* iz dviju različitih dretvi.

Objekt *ISession* se zbog prethodno navedenih ograničenja preporuča koristiti samo za male, atomske operacije, nakon čega je sjednicu potrebno zatvoriti. Na taj se način smanjuje mogućnost grešaka do kojih može doći u istovremenom pristupu podacima. Budući da je aplikacija za unos podataka o alumnima razvijena upravo na način da se bazi preko objekta *ISession* pristupa samo u kratkim intervalima za izvođenje atomskih operacija, istovremeni pristup podacima u kontekstu razvijene aplikacije ne bi trebao predstavljati problem.

5.2 Izmjena tablica u bazi

Ako se ukaže potreba za izmjenom tablica u postojećoj bazi podataka, takva će situacija tipično rezultirati sa nekoliko izmjena u samoj aplikaciji. Izmjena neke tablice najčešće će zahtijevati i izmjenu u objektnom modelu, odnosno razredu koji se perzistira u tu tablicu. Nakon toga, mijenja se i sama XML datoteka koja se koristi za mapiranje razreda u tablicu u bazi. Konačno, u ovisnosti o izmjenama načinjenim u tablici, potrebno je dodati ili promijeniti metode za pristup podacima iz te tablice. Ako se pritom mijenja i sučelje sloja za pristup podacima (mijenjanju se metode koje taj sloj nudi višim slojevima), može doći i do manjih izmjena u aplikacijskom, odnosno prezentacijskom sloju. Konkretno, na primjeru razvijene aplikacije, kad bi se mijenjala tablica u bazi, izmjene u aplikaciji bile bi:

- u projektu *Alumni_DomainModel* promijeniti odgovarajući razred,
- u projektu *Alumni_DAL* promijeniti XML datoteku za mapiranje razreda o kojem se radi,
- također u projektu *Alumni_DAL* u direktoriju *Repositories_and_factories* promijeniti (ako je potrebno) metode razreda koji se bavi dohvaćanjem/izmjenom/brisanjem podataka iz te tablice,
- ako se zbog izmjena u bazi promijenilo sučelje objekata za pristup podacima (npr. neka je metoda promijenjena na način da dohvaća podatke prema izmijenjenom kriteriju), potrebno je i u projektima *Alumni_Interfaces* i *Alumni_Controller* napraviti odgovarajuće izmjene kako bi se izmijenjene metode pozivale na ispravan način,
- konačno, i u prezentacijskom sloju (projekt *Alumni_GUI*) može doći do izmjena sučelja, ako je npr. u tablicu dodan novi podatak kojeg je potrebno na neki način prikazati na korisničkom sučelju.

5.3 Raspodijeljena baza

Moguć je i slučaj u kojem neće svi podaci biti u istoj bazi. Dio podataka, npr. podaci o osobama, mogu biti u jednoj bazi, dok npr. podaci o diplomama mogu biti spremljeni u neku drugu bazu. U tom je slučaju situacija nešto složenija, ali je pristup različitim bazama pomoću biblioteke *NHibernate* moguć.

Budući da je za komunikaciju s bazom potreban objekt *Session*, a njega je moguće stvoriti pomoću objekta *SessionFactory*, potrebno je konfigurirati onoliko *SessionFactory* objekata koliko ima baza podataka. Budući da se konfiguracija objekta *SessionFactory* specificira unutar konfiguracijske datoteke *hibernate.cfg.xml*, potrebno je dakle imati nekoliko konfiguracijskih datoteka s različitim podacima. Na konkretnom primjeru razvijene aplikacije, situacija bi bila sljedeća. U slučaju korištenja dvije baze, postojale bi dvije konfiguracijske datoteke, npr. *hibernate1.cfg.xml* i *hibernate2.cfg.xml.* U projektu *Alumni_DAL* nalazi se pomoćni razred za konfiguraciju - *NHibernateHelper.cs*. Umjesto stvaranja jednog objekta *SessionFactory*, ovaj bi razred sada bio zadužen za stvaranje dva objekta, npr. *sessionFactory1* i *SessionFactory2*. U nastavku je prikazano moguće rješenje.

```
private static SessionFactory sessionFactory1, sessionFactory2;


sessionFactory1 = new Configuration().configure("hibernate1.cfg.xml
").buildSessionFactory();

sessionFactory2= new Configuration().configure("hibernate2.cfg.xml
").buildSessionFactory();
```

SessionFactory objekti mogli bi se dohvaćati na sljedeći način:

```
public static SessionFactory getSessionFactory1()
{
         return sessionFactory1;
}

public static SessionFactory getSessionFactory2()
{
        return sessionFactory2;
}
```

U ovisnosti kojoj bazi se u kojem trenutku pristupa, pomoću odgovarajućeg objekta *SessionFactory* moguće je tada stvoriti objekt *Session*, odnosno otvoriti vezu prema odgovarajućoj bazi.

Ovakav je pristup (prema *NHibernate* dokumentaciji i izvorima s Interneta) moguć, ali u okviru ovog rada nije isproban pa je moguće da su za korištenje različitih baza podataka potrebne dodatne izmjene u aplikaciji.

6. Zaključak

Spremanje podataka o bivšim studentima u elektroničkom obliku ima nekoliko prednosti. Za početak, udruga AMAC-FER time dobiva sredstvo za olakšanje komunikacije među članovima, što može rezultirati samo dodatnim unapređenjem njihove znanstvene i nastavne djelatnosti te stručne suradnje. Same podatke o alumnima moguće je lakše povezati i uskladiti s drugim sustavima (npr. ISVU). Konačno, spremanje i održavanje podataka je znatno olakšano.

Za izradu aplikacije koja se koristi za spremanje podataka o alumnima korišteno je nekoliko tehnologija. Baza *SQLite* je odabrana prije svega radi jednostavnosti korištenja, brzine rada i pouzdanosti. Ipak, ona ne određuje sam rad aplikacije pa ju je, u slučaju da se za time ukaže potreba, vrlo lako moguće zamijeniti s nekom drugom bazom koja se u danom slučaju pokaže prikladnijom. Za testiranje aplikacije na nivou na kojem je to provedeno u sklopu ovog rada, baza *SQLite* pokazala se sasvim solidnim rješenjem.

Kao veza između aplikacije i baze podataka korištena je biblioteka *NHibernate* - inačica popularne Java biblioteke *Hibernate* prilagođena razvojnom okruženju .NET. Biblioteka *NHibernate* je odabrana jer na jednostavan način prevladava razlike između objektnog modela na kojem je aplikacija zasnovana i relacijske baze u koju se spremaju podaci.

Konačno, za izradu grafičkog sučelja korišten je grafički podsustav WPF (eng. *Windows Presentation Foundation*). Alternativno je za iscrtavanje korisničkog sučelja bilo moguće koristiti sustav *Windows Forms*, ali je radi nešto naprednijih mogućnosti i veće prilagodljivosti odabran upravo WPF.

7. Literatura

[1] *About SQLite*, http://www.sqlite.org/about.html, datum pristupa dokumentu 25.5.2010.

[2] The NHibernate FAQ, *Your firs NHibernate based application*, http://blogs.hibernatingrhinos.com/nhibernate/archive/2008/04/01/your-first-nhibernate-based-application.aspx, datum pristupa dokumentu 20.4.2010.

[3] Krzysztof Kozmic, *Testing with NHibernate and SQLite,* datum nastanka dokumenta 14.8.2009., http://devlicio.us/blogs/krzysztof_kozmic/archive/2009/08/14/testing-with-nhibernate-and-sqlite.aspx, datum pristupa dokumentu 20.4.2010.

[4] Pierre Henri Kuaté, Tobin Harris, Cristian Bauer, Gavin King; *NHibernate in Action*, 2008. godina

[5] AMAC-FER, Hrvatska udruga diplomiranih inženjera Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu, http://www.amac.fer.hr/, datum pristupa stranici 25.5.2010.

[6] *Windows Presentation Foundation,* http://en.wikipedia.org/wiki/Windows_Presentation_Foundation, datum pristupa dokumentu 20.4.2010.

[7] *WPF Tutorial - A Beggining,* datum nastanka 8.5.2010., http://www.dotnetfunda.com/articles/article882-wpf-tutorial--a-beginning--1-.aspx, datum pristupa dokumentu 25.5.2010.

[8] *Model-View-Controller Pattern, h*ttp://www.enode.com/x/markup/tutorial/mvc.html, datum pristupa dokumentu 25.5.2010.

8. Sažetak

U radu je prikazana problematika povezivanja bivših studenata s ciljem unapređenja njihove suradnje te znanstvene i nastavne djelatnosti. U tu je svrhu izrađena aplikacija za unos podataka o alumnima, njihovim diplomama i kontaktima. Pritom su korištene sljedeće tehnologije: baza *SQLite* za spremanje podataka o alumnima, biblioteka *NHibernate* za povezivanje objektnog modela podataka i načina njihove pohrane u bazu te grafički podsustav *Windows Presentation Foundation* za izradu grafičkog korisničkog sučelja. Rezultat rada je višeslojna aplikacija koja, u slučaju da se za time ukaže potreba, omogućava jednostavnu zamjenu prethodno spomenutih tehnologija nekim drugim rješenjima.

Ključne riječi: alumni, aplikacija, NHibernate, SQLite, WPF, AMAC-FER

9. Summary

Title: Application for updating Alumni database

The paper describes the problems of connecting alumni with the aim of enhancing their cooperation and scientific and educational activities. For this purpose, an application for saving data on alumni, their diplomas, and contacts was developed. The following technologies were used: SQLite database for storing data on alumni, NHibernate library for connecting the object model and database, and Windows Presentation Foundation graphics sub-system for development of graphical user interface. The result is a multi-layered application which, in case the need arises for a time, allows an easy replacement of the previously mentioned technologies with other solutions.

Keywords: alumni, application, NHibernate, SQLite, WPF, AMAC-FER