UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Kaja Krnec

Primerjava dveh ogrodij za izdelavo grafičnih uporabniških vmesnikov: Windows Presentation Foundation in Windows Forms

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2015

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Kaja Krnec

Primerjava dveh ogrodij za izdelavo grafičnih uporabniških vmesnikov: Windows Presentation Foundation in Windows Forms

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: viš. pred. dr. Igor Rožanc

Ljubljana, 2015

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Naslov: Primerjava dveh ogrodij za izdelavo grafičnih uporabniških vmesnikov: Windows Presentation Foundation in Windows Forms

Tematika naloge:

V diplomski nalogi primerjajte ogrodji za izdelavo grafičnih uporabniških vmesnikov Windows Presentation Foundation in Windows Forms, ki omogočata dva različna načina oblikovanja enakih vizuelno in vsebinsko bogatih uporabniških vmesnikov v tehnologiji .NET. Za obe ogrodji predstavite osnovne značilnosti, njuno uporabo pa prikažite na praktičnem primeru izdelave enakega vmesnika za manjšo aplikacijo. Nalogo zaključite z oceno obeh ogrodij po več izbranih kriterijih.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisana **Kaja Krnec**, z vpisno številko **63100004**, sem avtor diplomskega dela z naslovom:

Primerjava dveh ogrodij za izdelavo grafičnih uporabniških vmesnikov: Windows Presentation Foundation in Windows Forms

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelala samostojno pod mentorstvom **viš. pred. dr. Igorja Rožanca**,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne _____ Podpis avtorja: _____

Kazalo

Kazalo slik

Slovar pojmov

Ogrodje (angl. framework): ogrodje programske opreme je univerzalno okolje, ki je namenjeno večkratni uporabi in razvijalcem nudi določeno funkcionalnost za razvoj programskih aplikacij.

Aplikacija (angl. application): računalniški program, ki izvede zaporedje ukazov za reševanje konkretnega problema.

Uporabniški vmesnik (angl. user interface): grafično okolje, v katerem uporabnik komunicira z napravo.

Zaslonska maska (angl. form): posamezni del uporabniškega vmesnika, vsak uporabniški vmesnik je lahko sestavljen iz več zaslonskih mask.

Komponenta (angl. control): posamezni element na zaslonski maski kot gumb ali drsnik.

Lastnost (angl. property): lastnost neke komponente, ki je odvisna od zunanjih vplivov, recimo animacij, slogov ali povezav z določenimi podatki.

Povezava (angl. binding): način, kako komponente v uporabniškem vmesniku komunicirajo s podatki v podatkovnem objektu.

Seznam uporabljenih kratic

kratica	angleško	slovensko
MVVM	Model-View-View Model	Model-pogled-model pogleda
XAML	Extensible Application Markup Language	Razširljiv aplikacijski označevalni jezik
XML	Extensible Markup Language	Razširljiv označevalni jezik
SQL	Structured Query Language	Strukturirani povpraševalni jezik

Povzetek

Namen diplomske naloge je primerjava ogrodij Windows Presentation Foundation in Windows Forms za izdelavo uporabniških vmesnikov. Kljub temu, da obe ogrodji služita istemu namenu, je postopek izdelave vmesnikov z ogrodjema konceptualno drugačen. Ogrodje Windows Presentation Foundation je tesno povezano z arhitekturnim vzorcem načrtovanja model-pogled-model pogleda, ki loči grafično plast uporabniškega vmesnika od podatkovne plasti aplikacije, poleg tega pa za grafično podobe aplikacije uporablja označevalni jezik XAML. Ogrodje Windows Forms nudi razširljiv nabor objektno usmerjenih razredov, ki so namenjeni razvoju vizualno in vsebinsko bogatih aplikacij.

Posamezen pristop razvoja je v diplomski nalogi prikazan na praktičnem primeru aplikacije. Aplikacija je zasnovana kot domača knjižnica, ki uporabniku omogoča pregled in urejanje podatkov, shranjenih v lokalni podatkovni bazi. Na podlagi empirične in teoretične primerjave so na koncu izpeljane prednost in slabosti posameznega ogrodja. Kljub marsikaterim prednostim ogrodja Windows Presentation Foundation so rezultati primerjave pokazali, da je ogrodje Windows Forms bolj primerno za tip aplikacije kot je domača knjižnica, saj ponuja bolj ustrezne rešitve.

Ključne besede: .NET, WPF, Windows Forms, MVVM, XAML, uporabniški vmesnik.

Abstract

The purpose of this thesis is to compare Windows Presentation Foundation and Windows Forms as the framework of graphical user interfaces. Although both tools serve the same purpose, they tackle it with a different approach. Windows Presentation Foundation is tightly bound to the architectural pattern model-view-view model that separates the visual aspect of a graphical user interface from the business and back end logic as well as using the markup language XAML for the design of the graphical interface. Windows Forms offers an extensible set of object oriented classes that enable the development of content and visually rich applications.

The individual approach to the development in both frameworks is shown on a practical example in the form of an application. The application is designed as a home library that allows the user to view and edit the data stored in a local database. The pros and cons of each framework are presented on the basis of empirical and theoretical comparison. The results of the comparison have shown that Windows Forms is more applicable for the type of applications such as the home library as it provides more conducive solutions despite many advantages and benefits of Windows Presentation Foundation.

Keywords: .NET, WPF, Windows Forms, MVVM, XAML, user interface.

Poglavje 1 Uvod

Za razvoj in oblikovanje namiznih aplikacij in uporabniških vmesnikov je na voljo veliko ogrodij, ki razvijalcem ponujajo raznovrstne rešitve. Dve izmed takih tehnologij sta ogrodji Windows Presentation Foundation (WPF) in Windows Forms. Obe izhajata iz Microsoftovega ogrodja .NET [7], podpirata programski jezik C# [18] in sta namenjeni razvoju vizualno in vsebinsko bogatih uporabniških vmesnikov [14,15]. Kljub navideznim podobnostim ogrodij je postopek izdelave vmesnikov v obeh ogrodjih drugačen. Kakšne so te razlike smo odkrili tako, da smo se postavili v vlogo razvijalca in si zamislili primer uporabniškega vmesnika. Zamišljeno aplikacijo smo razvili tako v ogrodju Windows Presentation Foundation kot tudi v ogrodju Windows Forms in nato primerjali postopek izdelave ter končni rezultat.

Zanimalo nas je, kakšen je pristop posameznega ogrodja pri izdelavi in oblikovanju zaslonske maske, kako poteka povezovanje vmesnika s podatkovno bazo, ali je ogrodje zapleteno za uporabo in kakšno dokumentacijo nudi. Predvsem so nas zanimale tudi prednosti in slabosti enega ali drugega ogrodja ter katero izmed njih je bolj primerna izbira glede na zahteve in namembnost posamezne aplikacije.

V diplomski nalogi sta v prvem delu opisani ogrodji, v drugem delu pa so njune razlike in podobnosti raziskane na praktičnem primeru uporabniškega vmesnika. Enak uporabniški vmesnik smo razvili najprej v eni tehnologiji in nato v drugi ter beležili postopek oblikovanja zaslonske maske, izvedbo programske logike in dostopanja do podatkov v podatkovni bazi. V tretjem poglavju smo se osredotočili na primerjavo ogrodij ter pojasnili, kako so se med seboj razlikovali postopki iz drugega poglavja.

V zadnjem poglavju sledi povzetek ugotovitev, do katerih smo prišli s teoretično in empirično primerjavo tehnologij.

Poglavje 2 Uporabljena orodja in tehnologija

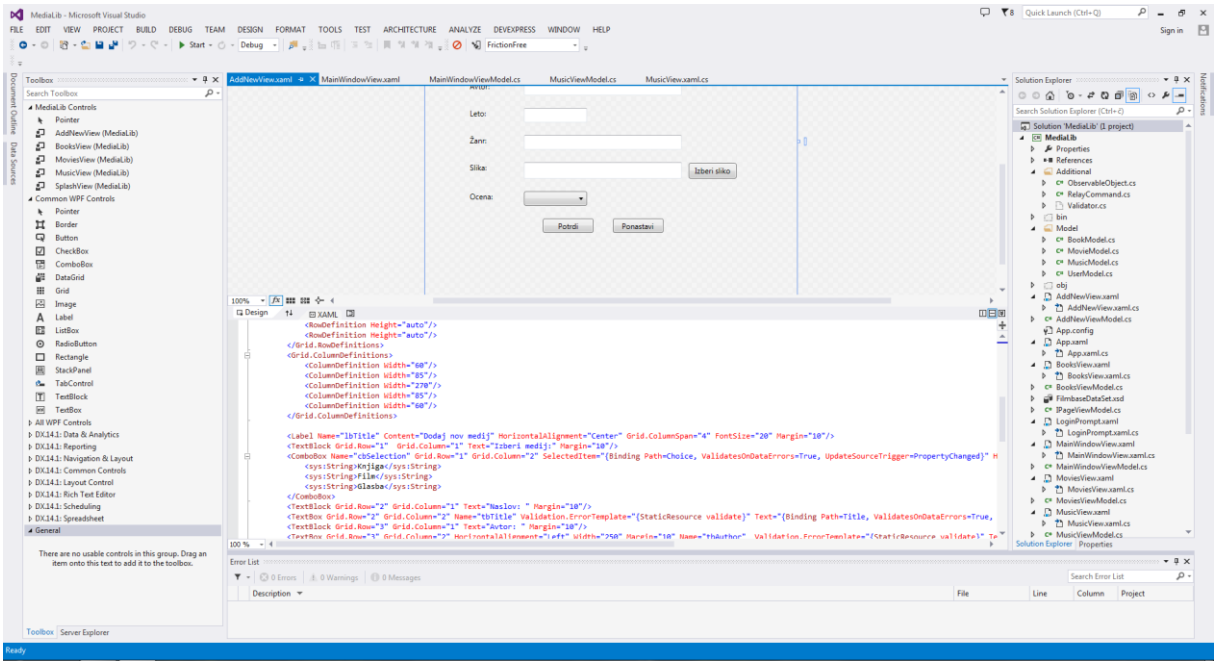
Poglavje je namenjeno predstavitvi razvojnega okolja Visual Studio kot tudi poglobljeni predstavitvi uporabljenih ogrodij Windows Presentation Foundation in Windows Forms.

2.1 Razvojno okolje Visual Studio

Visual Studio je razvojno okolje (IDE) podjetja Microsoft. Uporablja se za razvoj aplikacij in grafičnih vmesnikov za operacijski sistem Windows pa tudi za razvoj spletnih strani, spletnih aplikacij, medmrežnih storitev in mobilnih aplikacij [13].

Urejevalnik besedila v Visual Studiu podpira širok nabor jezikov, kot so C#, VB.NET, C++, HTML, JavaScript, XAML, SQL in še mnogo drugih [12].

Za razvoj tako razgibane palete aplikacij Visual Studio ponuja različna ogrodja, med njimi tudi Windows Forms in WPF [8]. Na sliki 2.1 je prikazano razvijanje aplikacije z ogrodjem WPF.



Slika 2.1: Razvojno okolje Visual Studio 2013.

2.2 Ogrodje Windows Presentation Foundation

Windows Presentation Foundation (v nadaljevanju WPF) je Microsoftovo najnovejše ogrodje za izdelavo razkošnih uporabniških vmesnikov [15]. Z ogrodjem je možno sestaviti tako minimalistične uporabniške vmesnike, namenjene poslovnim uporabnikom, kot tudi vmesnike bogate z animacijami in grafičnimi učinki [1]. Jedro ogrodja WPF temelji na vektorskem upodabljanju komponent in je neodvisno od ločljivosti. Ponuja širok spekter elementov za razvoj aplikacij, kot so opisni jezik XAML [19], povezovanje podatkov, podporo za elemente v 2D in 3D grafiki, sloge in podobno. WPF je vključen v ogrodje .NET [7].

2.2.1 Označevalni jezik XAML

XAML je označevalni jezik, ki temelji na razširljivem označevalnem jeziku XML [19]. Uporablja se za deklarativno izvedbo grafične podobe aplikacije kot so okna, pogovorna okna in strani, na katerih so postavljene različne komponente. Temeljni razlog za uporabo jezika XAML je ločitev predstavitvene logike od kode grafičnega vmesnika, kar pomeni, da programer lahko ločeno razvija kodo od grafičnega oblikovalca, ki skrbi za vizualno podobo aplikacije [1].

Uporaba označevalnega jezika XAML v ogrodju WPF ni obvezna, je pa priporočljiva, saj le tako razvijalec izkoristi funkcionalnosti, ki jih ogrodje ponuja. Brez jezika XAML postane aplikacija bolj zaprt sistem, ki je namenjen le programerjem in izgubi vso prednost ločenega razvijanja kode [6].

Vsak element v datoteki XAML je instanca .NET razreda, kar pomeni, da se mora ime elementa na dokumentu natanko ujemati z imenom razreda [6]. Na sliki 2.2 je prikazana koda zaslonske maske. Element definiran z oznako "<Button .../>" ustvari objekt razreda `Gumb` (angl. button).

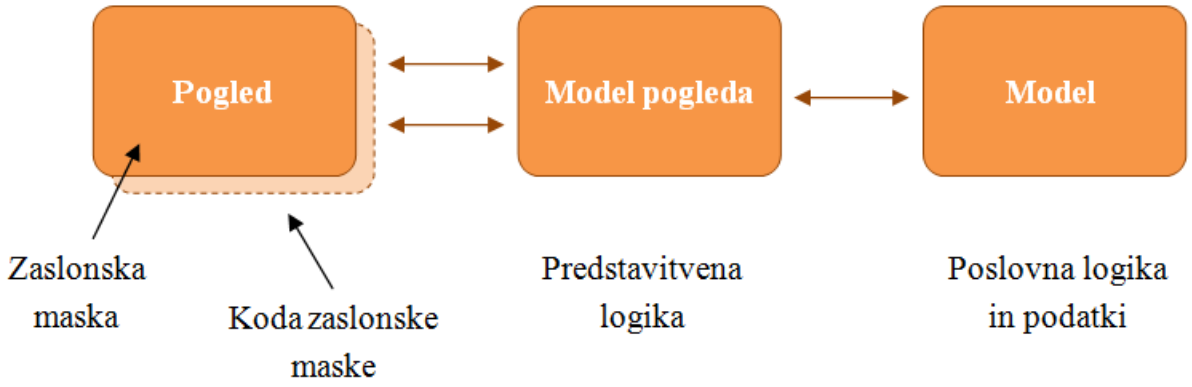
```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="MediaLib.LoginPrompt"
  Title="Prijava" Height="200" Width="280" ResizeMode="NoResize"
  WindowStartupLocation="CenterOwner">
    <Grid>
      <Grid.ColumnDefinitions>
         <ColumnDefiniton Width="auto"/>
         <ColumnDefinition/>
         <ColumnDefinition/>
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
         <RowDefinition Height="auto"/>
         <RowDefinition/>
         <RowDefinition/>
         <RowDefinition/>
      </Grid.RowDefinitions>

      <Label x:Name="lbUsername" Content="Uporabniško ime"/>
      <TextBox x:Name="tbUsername" Grid.Column="1" Grid.Row="0" Grid.ColumnSpan="2"
             HorizontalAlignment="Center"/>
      <Label x:Name="lbPwd" Grid.Column="0" Grid.Row="1" Content="Geslo:"/>
      <PasswordBox x:Name="pbPwd" Grid.Column="1" Grid.Row="1" Grid.ColumnSpan="2"
                   HorizontalAlignment="Center"/>
      <Label x:Name="lbStatus" Grid.Column="0" Grid.Row="2" Grid.ColumnSpan="3"
             Foreground="Red" Content=""/>
      <Button Click="btOk_Click" Grid.Row="3" Grid.Column="1" Content="Potrdi"
              HorizontalAlignment="Right"/>
      <Button Click="btCancel_Click" Grid.Row="3" Grid.Column="2"
              Content="Prekliči" HorizontalAlignment="Left"/>
   </Grid>
</Window>
```

Slika 2.2: Primer kode v označevalnem jeziku XAML.

2.2.2 Vzorec Model-Pogled-Model Pogleda

Razvoj aplikacij ni preprosta naloga, saj zahteva kombiniranje podatkov, dostopanje do podatkov, izvedbo zaslonskih mask, izvedbo validacije in testiranja in predvsem sodelovanje vseh naštetih elementov [20]. Za lažji razvoj se poleg razvoja ogrodij za uporabniške vmesnike hkrati razvijajo tudi vzorci, ki so namenjeni lažjemu načrtovanju in izdelavi le-teh. Leta 2005 je John Gossman, razvijalec programske opreme WPF v podjetju Microsoft, predstavil nov vzorec za načrtovanje imenovan Model-Pogled-Model Pogleda (ang. Model-View-View Model ali krajše MVVM) [1, 2].



Slika 2.3: Diagram vzorca MVVM [11].

Vzorec MVVM sestoji iz treh glavnih komponent, ki so predstavljene na sliki 2.3:

- **Pogled** predstavlja zaslonsko masko oziroma uporabniški vmesnik in je edina komponenta, do katere ima uporabnik dostop. V najboljšem primeru koda v ozadju zaslonske maske vsebuje metodo, ki izriše komponente. Koda v ozadju pa lahko vseeno vsebuje tudi programsko logiko, vendar le tako, ki se nanaša na neposredno spremembo komponent na zaslonski maski in bi jo bilo nesmiselno vpeljati v predstavitveno logiko [1,11].

- **Model pogleda** hrani predstavitveno logiko in podatke, ki so predstavljeni v pogledu. Ne vsebuje nobenega neposrednega sklica na pogled ali komponente znotraj pogleda. Model pogleda ima definirane lastnosti in ukaze, preko katerih so povezani podatki na pogled, poleg tega pa model pogleda skrbi za poročanje o stanju lastnosti in ukazov s pomočjo vmesnikov za proženje sprememb (vmesnik `INotifyPropertyChanged`) [1,11].

- **Model** vsebuje poslovno logiko in podatke. Poslovna logika je definirana kot način pridobivanja in upravljanja podatkov ter zagotavljanje skladnosti in veljavnosti le-teh. Model nima zaslonske maske in je namenjen le hranjenju podatkov. Nima neposrednega sklica na model pogleda ali model in ne vpliva na njihovo izvedbo. Ponavadi model vsebuje vmesnik za proženje sprememb, s katerim lahko posreduje podatke modelu pogleda in posledično tudi pogledu [1,11].

Pogled torej vsebuje dogodke in komponente, na katere se lahko sklicuje s pomočjo lastnosti `DataContext`, ki poveže komponente na lastnosti in ukaze v modelu pogleda [1,11]. Lastnost `DataContext` predstavlja povezavo med podatkovno plastjo aplikacije in

zaslonsko masko. S pomočjo povezave lastnosti `DataContext` komponente na zaslonski maski lahko prikažejo vrednosti in podatke iz podatkovne plasti (preko modela pogleda).

```
class Knjiga
{
    private string _naslov="Knjiga";

    public string Naslov
    {
       get
       {
           return _naslov;
       }
    }
}

public partial class Okno: Window
{
    public Okno()
    {
       InitializeComponent();
       this.DataContext = new Knjiga();
    }
}
```

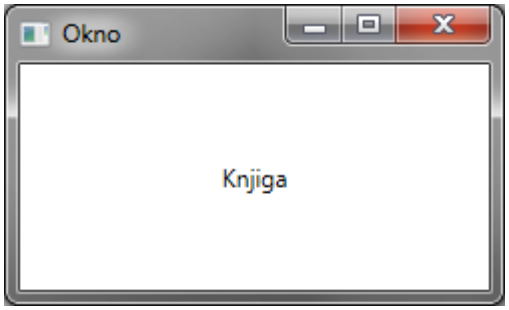
Slika 2.4: Povezava z lastnostjo `DataContext`.

```
<Window Title="Okno" Height="150" Width="150">
    <Grid>
        <Label>
            Content="{Binding Naslov}"
            HorizontalAlignment="Center"
            VerticalAlignment="Center"
        />
    <Grid>
</Window>
```

Slika 2.5: Povezava z lastnostjo `DataContext` v označevalnem jeziku XAML.

Na slikah 2.4 in 2.5 je predstavljen preprost praktičen primer uporabe lastnosti `DataContext`. V razredu `Knjiga` je definirana lastnost `Naslov`, ki vrne niz "Knjiga", ko je poklicana. Definirano imamo tudi okno, ki ga sestavljata koda v ozadju (slika 2.4, spodnji del) in koda XAML v pogledu (slika 2.5). V kodi v ozadju se nastavijo komponente, poleg tega pa povežemo lastnost `DataContext` z razredom `Knjiga`, ki predstavlja naš podatkovni model.

V pogledu nastavimo še zaslonsko masko, ki bo prikazovala podatkovni model. Okno je majhnih dimenzij in ima le eno komponento v sredini. Ta komponenta je oznaka, ki ima vsebino povezano na lastnost `Naslov`.



Slika 2.6: Prikaz komponente na zaslonski maski.

Na sliki 2.6 je prikazana zaslonska maska, ko program poženemo. Ker imamo v kodi v ozadju pogleda nastavljeno lastnost `DataContext` na razred `Knjiga`, se izpiše niz, ki ga hrani lastnost `Naslov`.

2.3 Ogrodje Windows Forms

Windows Forms je računalniško okolje za razvoj grafičnih vmesnikov, ki temeljijo na ogrodju .NET. To ogrodje nudi razširljiv nabor objektno usmerjenih razredov, ki so namenjeni razvoju vizualno in vsebinsko bogatih aplikacij [14].

2.3.1 Obrazec

Obrazec (angl. form) je osnovna enota aplikacije. To je neke vrste virtualna nepremičnina pravokotne oblike, na kateri so uporabniku predstavljene informacije, ki mu omogočajo vnos. Obrazci so lahko navadna okna, pogovorna okna ali površine za grafično predstavitev. Uporabniški vmesnik je le obrazec, ki na svoji površini vsebuje komponente. Obrazci so pravzaprav objekti, ki z lastnostmi definirajo svoj videz, z metodami definirajo svoje vedenje in z dogodki definirajo svoje sodelovanje z uporabnikom.

Obrazci v projektu z ogrodjem Windows Forms predstavljajo osrednji vmesnik za medsebojno sodelovanje z uporabnikom. S kombiniranjem različnih komponent in lastno kodo lahko aplikacija od uporabnika zahteva informacije ali mu jih podaja. Aplikacija lahko deluje in uporablja obstoječe hranilnike podatkov in izvaja poizvedbe in zapisuje v datotečni sistem [14].

Najbolj razširjen način ustvarjanja obrazcev je z orodjem Windows Forms Designer. V njem na obrazec vlečemo komponente, čeprav je možno le-te oblikovati tudi zgolj z uporabo urejevalnika besedila [14].

Poglavje 3 Izdelava aplikacije v ogrodjih WPF in Windows Forms

Ogrodji WPF in Windows Forms služita istemu namenu izdelave grafičnih vmesnikov, vendar se podobnost tu tudi konča. Windows Forms je zgolj sloj nad standardnimi gradniki kot so vnosna polja ali oznake, WPF pa od takih gradnikov ni odvisen, kar pomeni, da je bolj prilagodljiv, saj lahko katerikoli gradnik služi katerikoli funkciji. Za snovanje in urejanje grafičnega vmesnika uporablja jezik XAML, kar pomeni, da je grafični del aplikacije ločen od kode v ozadju.

Zanimalo nas je, kaj razlike med ogrodjema pomenijo v praksi, kako izgleda razvoj aplikacije z uporabo Windows Forms in kako z uporabo WPF. Treba je bilo torej razviti preprost grafični vmesnik, ki uporablja najbolj pogoste gradnike in prikazuje splošno rabo poljubne aplikacije. Vmesnik naj vsebuje dostop do podatkov v podatkovni bazi s katerimi lahko uporabnik upravlja.

Za primer aplikacije smo si zamislili navidezno domačo knjižnico, ki simulira uporabnikovo resnično zbirko medijev - knjig, filmov in glasbe. Aplikacija ima sledeče specifikacije:

- podatki so shranjeni v podatkovni bazi na lokalnem strežniku,
- glavno okno prikazuje vsebino, poleg tega pa vsebuje tudi orodno in statusno vrstico,
- možno je pregledovanje po seznamu poljubnega medija (po knjigah, filmih ali po glasbi) s klikom na specifično delo pa se uporabniku prikažejo še ostale bistvene informacije,
- uporabnik se lahko v aplikacijo prijavi, s tem pa pridobi možnost dodajanja novih del, popravljanja že obstoječih vnosov ali brisanje le-teh.

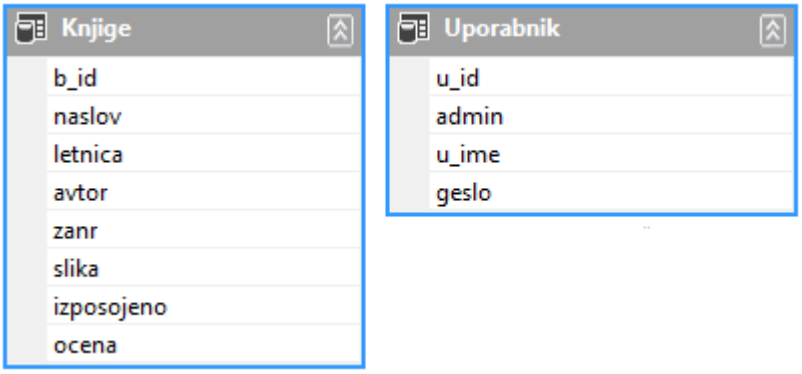
Za potrebe tako grafičnega vmesnika kot podatkovne baze smo potrebovali naslednji orodji:

- Visual Studio Ultimate 2013 [17],
- Microsoft SQL Server Management Studio 2014 [9].

3.1 Postavitev okolja in podatkovne baze

Postavitev samega okolja je nezahtevna, saj je potrebna zgolj namestitev programa za izdelavo grafičnih vmesnikov in programa za upravljanje s podatkovno bazo. Visual Studio Ultimate 2013 podpira tako izdelavo vmesnika z ogrodjem WPF kot tudi z ogrodjem Windows Forms. V primeru obeh je uporabljen programski jezik C#.

Podatkovno bazo smo za namene aplikacije postavili na lokalni strežnik. Tabele v bazi podatkov smo ohranili preproste, saj podatki v njih služijo zgolj kot vzorec v aplikaciji. V primeru, da bi aplikacijo želeli razširiti in dodati nove podatke v tabele ali celo nove tabele, bi to storili brez težav.



Slika 3.1: Tabeli `Knjige` in `Uporabnik`.

Podatkovna baza je tako sestavljena iz štirih tabel: `Filmi`, `Glasba` `Knjige` in `Uporabnik`. Na sliki 3.1 sta prikazani tabeli Knjige in Uporabnik. Tabele `Filmi`, `Glasba` `Knjige` imajo identične stolpce. Nismo jih želeli postaviti v eno samo tabelo, čeprav so trenutno med seboj enake. Kasneje bi namreč tabele lahko razširili in vanje dodali še specifične stolpce, kot na primer stolpec producent in režiser v tabelo `Filmi`. Poleg tega je smiselno, če so različni tipi objektov shranjeni v različnih tabelah. Vrstice, ki predstavljajo knjige v tabeli `Knjige`, se ne bodo v trenutni aplikaciji nikjer pojavile na istem seznamu kot vrstice, ki predstavljajo filme v tabeli `Filmi`. Stolpci, ki definirajo tabele `Knjige`, `Filmi` in `Glasba` so:

- Identifikacijska številka, ki služi kot primarni ključ in je v posamezni tabeli definirana kot `x_id`, kjer x pomeni prvo črko tipa medija (na primer: `k_id` za tabelo `Knjige`). Identifikacijsko številko vsak vnos avtomatično določi, ko je dodan v tabelo, saj ta enolično določa vsako vrstico. Uporabnik do te številke nima neposrednega dostopa.

- `Naslov`, kjer so shranjeni naslovi posameznih del. Naslov je obvezno polje in ne sme biti prazno (ne sme biti `null`).
- `Letnica`, kjer je shranjeno leto izdaje dela. `Letnica` je obvezno polje.
- `Avtor`, kjer je shranjen avtor ali večje število le-teh. Polje je obvezno.
- `Zanr`, kjer lahko v eni besedi ali več definiramo tip vsebine. Polje ni obvezno in je lahko tudi prazno.
- `Slika`, kjer je z nizom shranjeno ime slikovne datoteke. Polje lahko ostane prazno.
- `Izposojeno` je polje tipa `boolean`, kjer lahko uporabnik označi, če je delo trenutno v njegovi domači knjižnici ali ne. Polje je obvezno, privzeta vrednost ob dodajanju v bazo pa je `0` (neizposojeno).
- `Ocena` je številka od 0 do 5, s katero uporabnik dodeli lastno oceno poljubnega dela. Polje ni obvezno.

Tabela uporabnik ima štiri stolpce, od katerih nas v aplikaciji zanimata zgolj uporabniško ime in geslo. Slednje je v bazi shranjeno s kodiranjem SHA1 [3,4].

Podatkovno bazo smo pred začetkom razvoja grafičnih vmesnikov napolnili s podatki za lažje sprotno testiranje.

3.2 Razvoj v ogrodju WPF

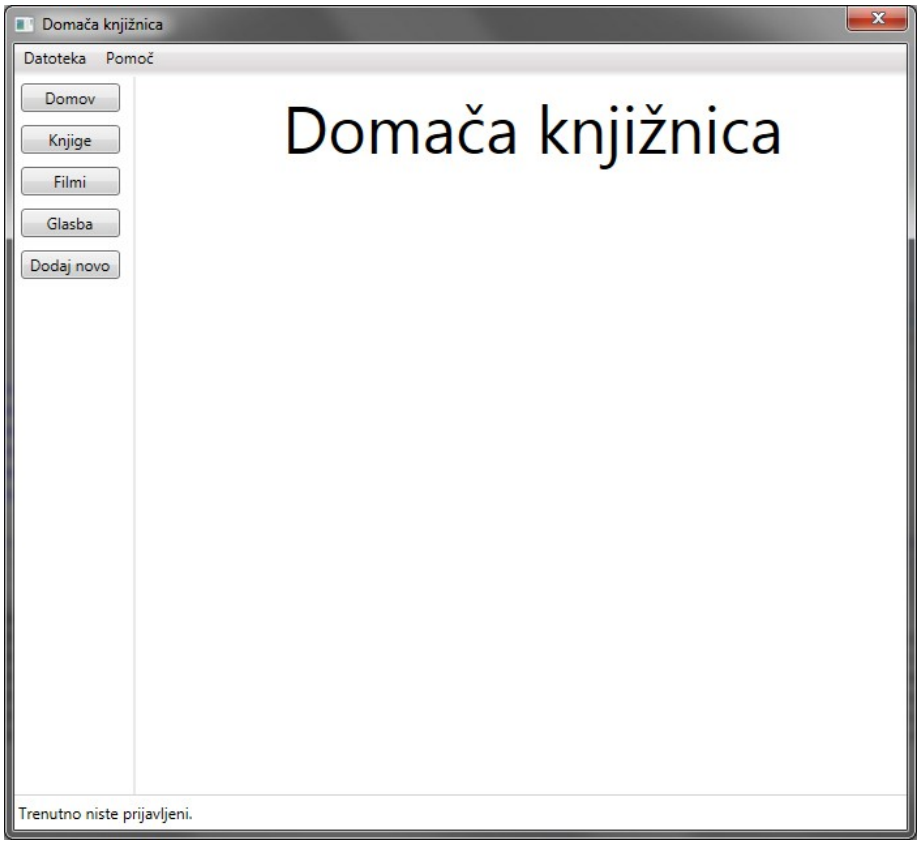
Sledil je razvoj grafičnega vmesnika v ogrodju WPF. V orodju Visual Studio je bilo za začetek potrebno izbrati izdelavo novega projekta in izbrati ogrodje, s katerim ga bomo izdelali.

V ogrodju WPF oblikovanje grafičnega vmesnika poteka po vzorcu MVVM, torej je grafični del neodvisen od logičnega. Namesto interaktivnega izbiranja gradnikov in vlečenja le-teh na delovno površino, grafični vmesnik v WPF pišemo v označevalnem jeziku XAML. Vsaka maska, ki jo uporabnik vidi, ima v ozadju vsaj dve datoteki. Ena je pogled, kjer se prikazujejo podatki. Pogledu lahko dodamo kodo v ozadje, vendar zgolj v primeru, če se ta nanaša na pogled. Druga datoteka je pogled modela, kjer pa se dogaja dostop do podatkovne baze, validacija in vsa logika, ki je vezana na delovanje vmesnika. Po vzorcu MVVM ima vsaka

maska tudi svoj model, ki hrani podatke, za delovanje pogleda modela. V vmesniku glavna maska nima modela, imajo pa ga preostale maske, kjer dostopamo do podatkov iz baze.

3.2.1 Oblikovanje osrednjega okna vmesnika in prijavnega okna

Ko uporabnik aplikacijo odpre, se mu prikaže glavno okno, ki vsebuje orodno vrstico, statusno vrstico, navigacijo med seznami medijev in glavno polje, kjer se prikazuje vsebina, kot je prikazano na sliki 3.2.

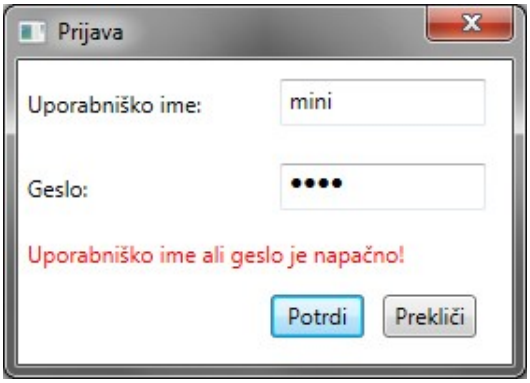


Slika 3.2: Glavno okno aplikacije.

Glavna maska nima modela, saj služi zgolj ogrodju vmesnika. V pogledu je definirana orodna vrstica, ki vsebuje gumbe za vpis uporabnika, izhod iz programa in gumb za dostop do navodil in podatkov o programu in avtorju. Koda v ozadju pogleda skrbi za akcije, ki jih sprožijo pritiski na gumbe v orodni vrstici.

Ob prijavi se odpre prijavno okno v katero uporabnik vpiše svoje uporabniško ime in geslo. Če poizvedba SQL v tabeli `Uporabniki` vrne zadetek, se prijavno okno zapre, na statusni vrstici pa je sedaj zavedeno, da je uporabnik prijavljen. Na statusni vrstici je zavedeno tudi, s katerim uporabniškim imenom je vpisan kot je to prikazano na sliki 3.4. Če se uporabnik

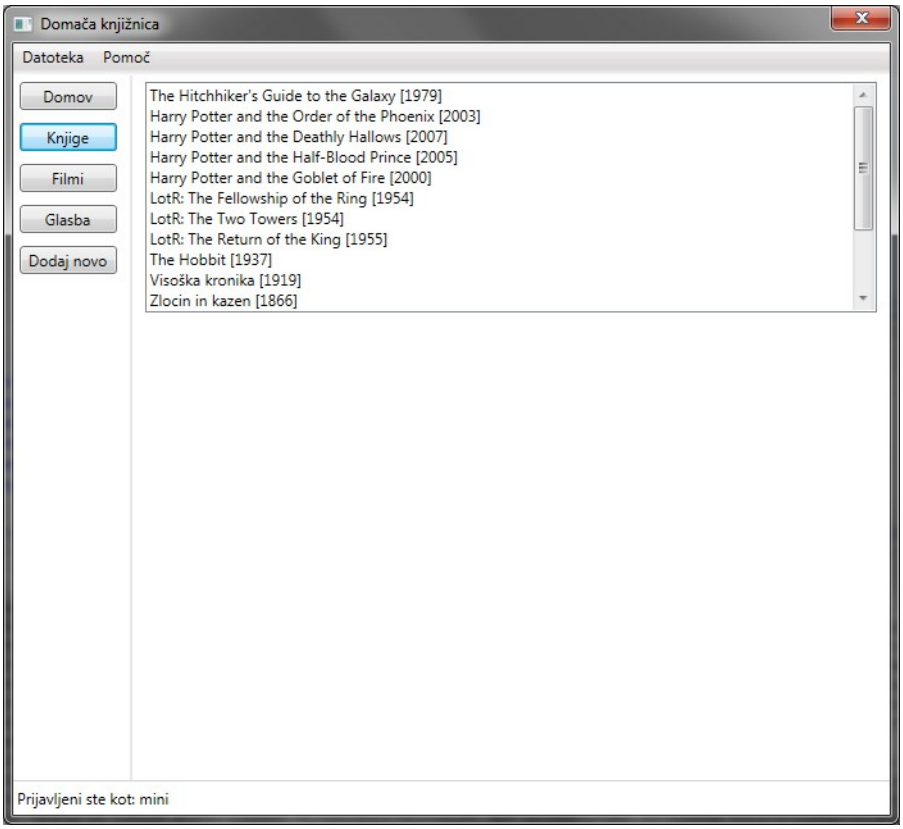
zmoti pri pisanju gesla ali uporabniškega imena ali če pusti polji prazni, poizvedba SQL ne vrne zadetka, v prijavnem oknu pa se izpiše opozorilo, kot je razvidno iz slike 3.3.



Slika 3.3: Prijavno okno z opozorilom.

3.2.2 Navigacija med maskami

Osrednji del glavne maske je navigacija med pogledi, kjer se izrisuje vsebina aplikacije. Navigacija v obliki gumbov se nahaja na levi strani vmesnika in je z delilnikom ločena od desne plošče, kjer se prikazuje in spreminja vsebina glede na uporabnikove ukaze.



Slika 3.4: Zaslonska maska s seznamom knjig.

Vsebina, ki se na plošči spreminja, je pravzaprav sestavljena iz več različnih pogledov oziroma mask, ki izpisujejo podatke iz baze v različnih gradnikih, ki so opisani v naslednjem podpoglavju. Da lahko dosežemo samodejno posodabljanje plošče glede na ukaze uporabnika, moramo v model pogleda glavne maske vgraditi vmesnik `INotifyPropertyChanged`. Slednjega smo implementirali v naš razred v skladu z navodili na spletnem naslovu [5]. Gumbov na vmesnik nismo izpisali ročno, temveč smo definirali kontrolo za predstavitev postavk (ang. ItemsControl) kot je prikazano na sliki 3.5. V kontroli si lahko poljubno izberemo na kakšen način so postavke predstavljene (na primer z gumbi). Gumbov je torej toliko, kolikor je v modelu pogleda definiranih postavk. V našem primeru je postavka pogled, ki se izrisuje na plošči. Kontrola za predstavitev postavk je povezana na seznam pogledov, ki ga imamo definiranega na modelu pogleda imenovanim `PageViewModels`. Ob pritisku na gumb se tako sproži metoda (na sliki 3.5 je to metoda `ChangePageCommand`), ki preveri, ali je vrednost plošče enaka vrednosti pritisnjenega gumba in ploščo ustrezno posodobi. Vrednost je definirana kot pogled, ki ga mora izrisati (na sliki 3.4 je to povezava na lastnost `CurrentPageViewModel`).

```
<ItemsControl ItemsSource="{Binding PageViewModels}" Grid.Row="1" ...
  <ItemsControl.ItemTemplate>
    <DataTemplate>
      <Button Content="{Binding Name}"
          Command="{Binding DataContext.ChangePageCommand, Rel...
          CommandParameter="{Binding}"
          Margin="2,5"/>
    </DataTemplate>
  </ItemsControl.ItemTemplate>
</ItemsControl>
<Border Brush="LightGray" BorderThickness="0,0,1,0", Grid.Row="..
<ContentControl Content="{Binding CurrentPageViewModel}" Grid.Row="1..
```

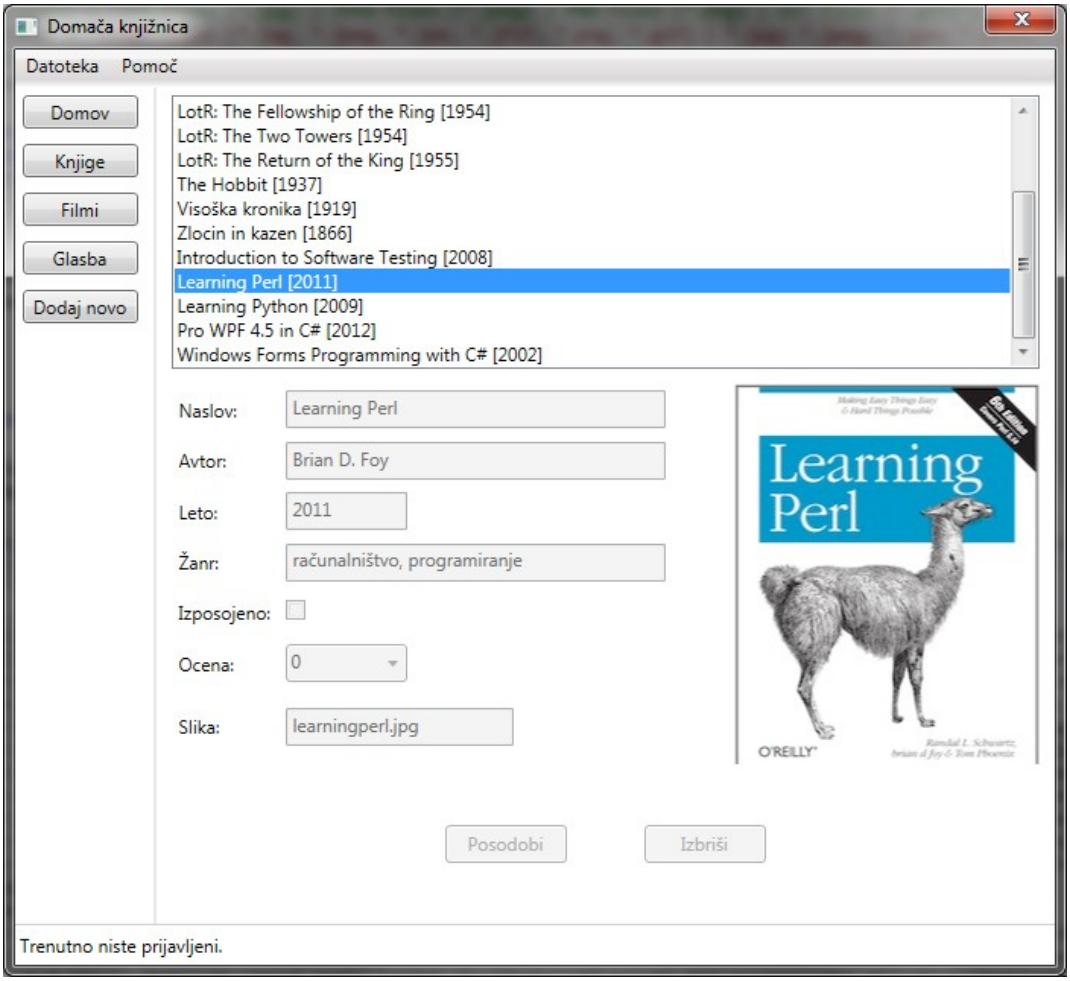
Slika 3.5: Dinamična povezava zaslonskih mask.

3.2.3 Maska za izpis podatkov

Osrednji del uporabniškega vmesnika je brskanje po vsebini knjižnice. S klikom na tip medija se na glavni plošči odpre gradnik seznam (angl. ListView) in izpiše zadetke iz poizvedbe SQL. Koda dostopanja do podatkovne baze se nahaja v modelu pogleda. Poizvedba SQL je statični niz v obliki `SELECT` stavka. Ob inicializaciji maske je poklicana metoda `Get()`, ki vzpostavi povezavo z bazo in ob uspešni izvršitvi poizvedbe v obliki seznama vrne podatke.

Podatke o posameznem delu hranimo v objektu. Objekt hrani vse lastnosti, ki so shranjene v tabelah `Knjige`, `Filmi` ali `Glasba`. Ko je povezava z bazo odprta, metoda sproži bralca (angl. `SqlDataReader`). Če bralec prebere vsaj eno vrstico, v metodi ustvarimo nov objekt z zahtevanimi parametri in ga dodamo v seznam objektov.

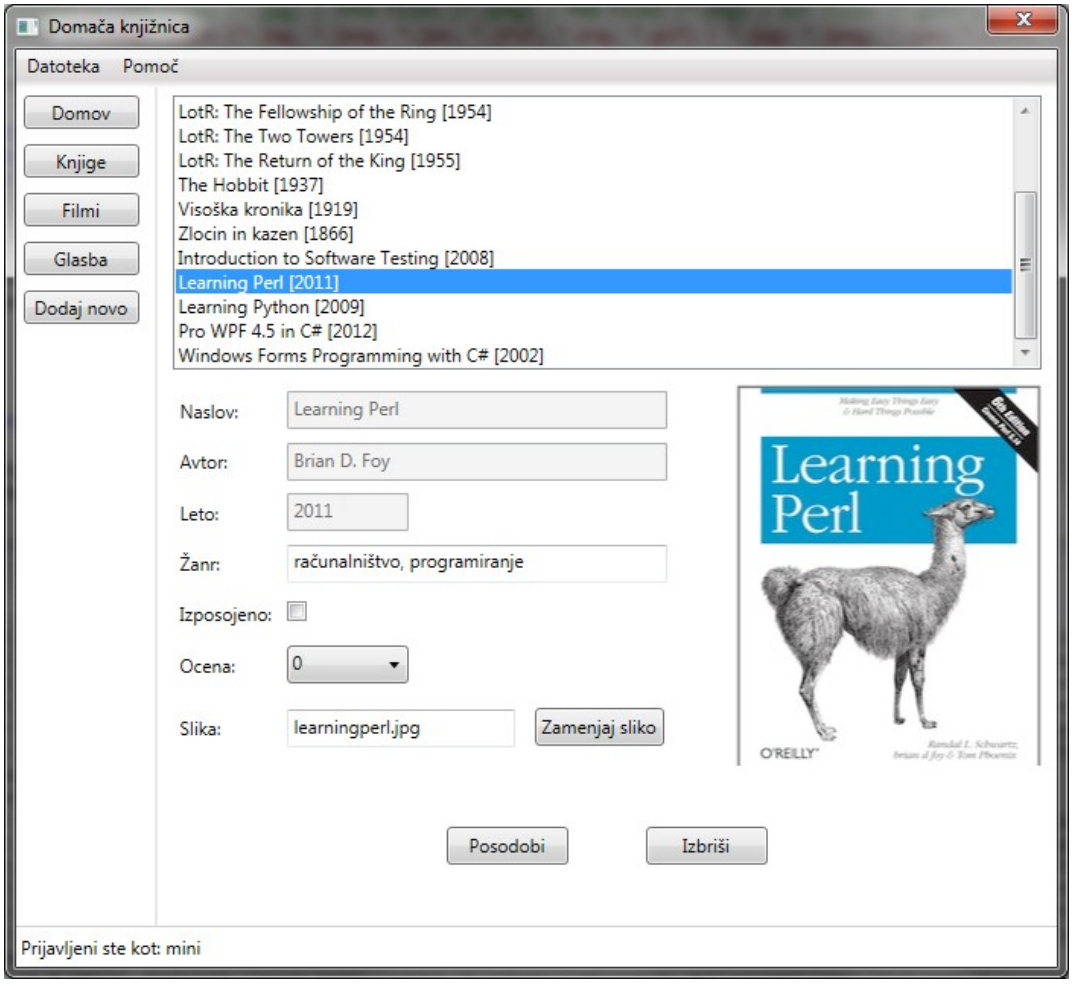
Seznam na zaslonski maski nato izpiše z nizom objekte, ki so shranjeni v seznamu objektov. V seznamu so opisani le z naslovom in letnico dela.



Slika 3.6: Izpisani podatki o izbrani knjigi.

S klikom na posamezno delo se na zaslonski maski izrišejo gradniki za predstavitev vseh podatkov, ki so shranjeni v objektu, kot je to prikazano na sliki 3.6. Niz, ki predstavlja sliko se pojavi dvakrat: enkrat kot niz, kakor je tudi shranjen v objektu (in podatkovni bazi), drugič pa predstavlja pot slikovne datoteke in se torej izriše slika. Če slike v podatkovni bazi ni ali pa datoteka ne obstaja, se izriše privzeta slika.

Kot je razvidno iz slike 3.7, ima uporabnik omogočeno popravljanje nekaterih polj, če je prijavljen. Lahko popravi oceno, žanr knjige, filma ali glasbenega dela, lahko odkljuka ali je izdelek izposojen ali ne in spremeni sliko. Celoten vnos lahko tudi izbriše. Neprijavljenim uporabnikom so gumbi za izbiro slike, brisanje in popravljanje vnosov onemogočeni.

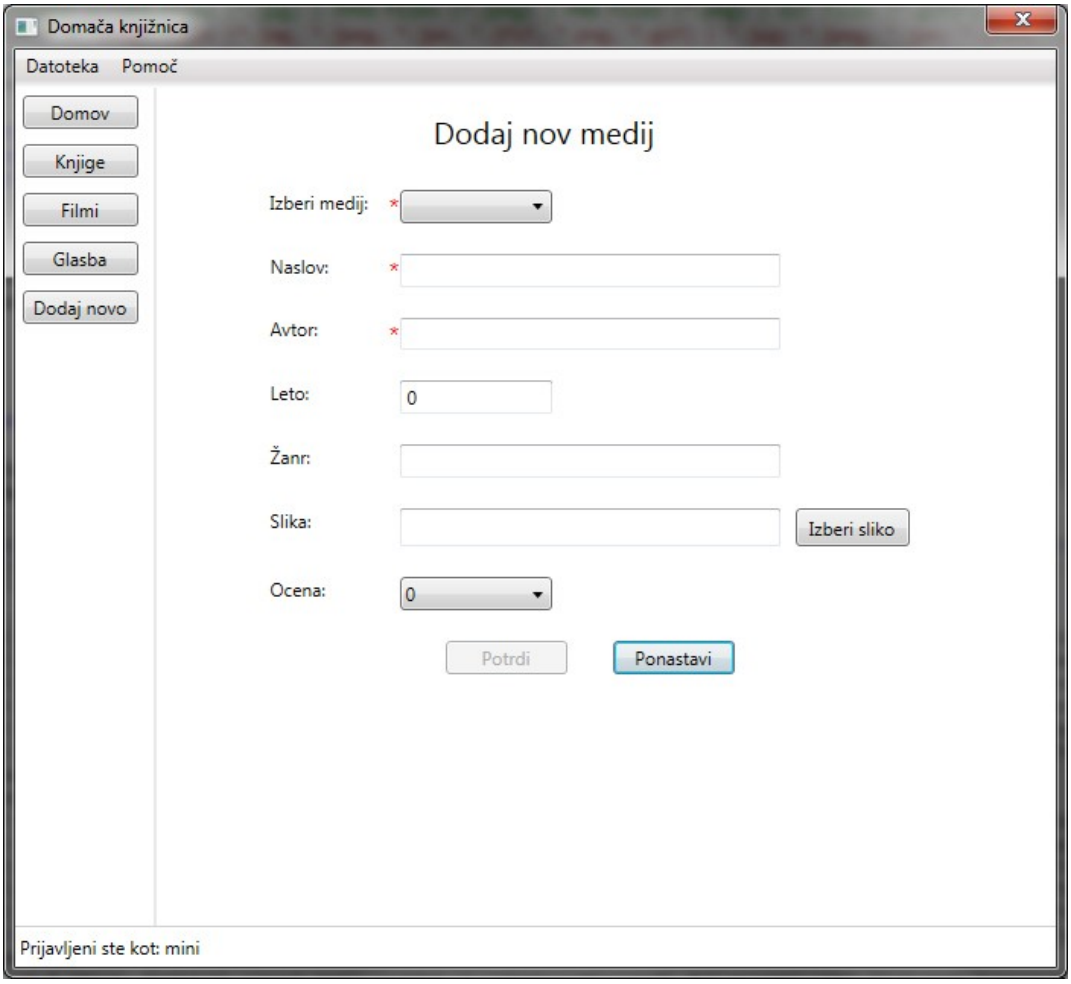


Slika 3.7: Možnost urejanja vnosov.

3.2.4 Dodajanje novega vnosa

Dodajanje novih vnosov v domačo knjižnico je pravzaprav najpomembnejši del aplikacije, saj uporabnik začne s prazno podatkovno bazo. Privilegij dodajanja novih del imajo zgolj prijavljeni uporabniki, sicer je onemogočeno.

Pri dodajanju si mora uporabnik najprej izbrati tip medija, ki ga bo dodal. Izbira v kombiniranem seznamu določi, v katero tabelo v podatkovni bazi bomo vnos shranili. Polje je zato obvezno.



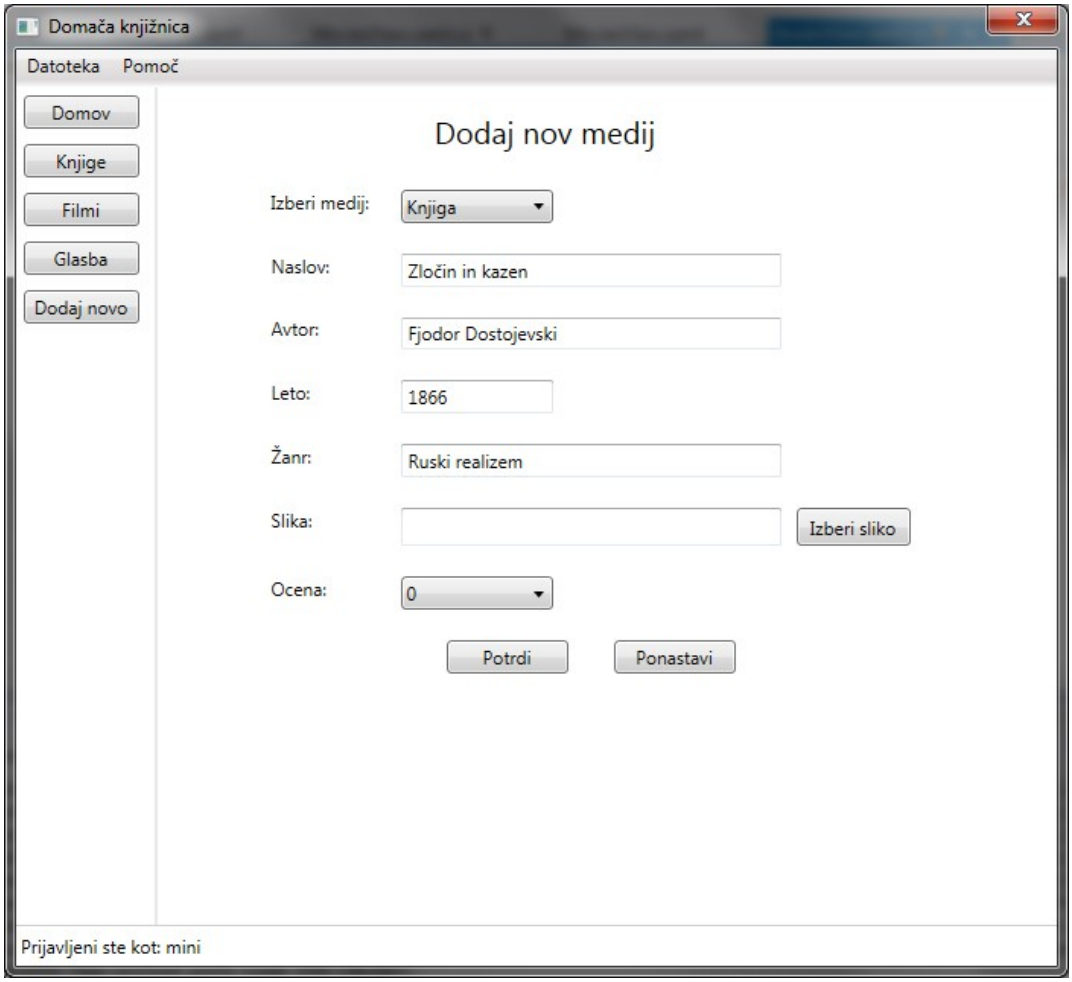
Slika 3.8: Dodajanje novega medija.

Preden podatke shranimo v podatkovno bazoje obvezna verifikacija, da ne pride do napake pri zapisovanju. Polja kot so omenjeni tip medija, naslov, avtor in letnica so obvezna in morajo biti izpolnjena, preden je uporabniku omogočen gumb `Potrdi`, s katerim se v podatkovno bazo shrani vnos. Iz slike 3.8 je razvidno, da je gumb `Potrdi` onemogočen, saj so zahtevana polja prazna.

Dokler vnosna polja niso izpolnjena, so označena z rdečim simbolom. Ta vnosna polja generirajo povratno informacijo validacije. Po vzorcu MVVM lahko za validacijo uporabimo vmesnik `ICommand`. To je vmesnik, ki zgolj definira nek ukaz. Ima dve metodi: `Execute()` in `CanExecute()`. Metoda `Execute()` je poklicana, ko se izvrši ukaz, `CanExecute()` pa določi, če se v trenutnem stanju ukaz sploh lahko izvrši.

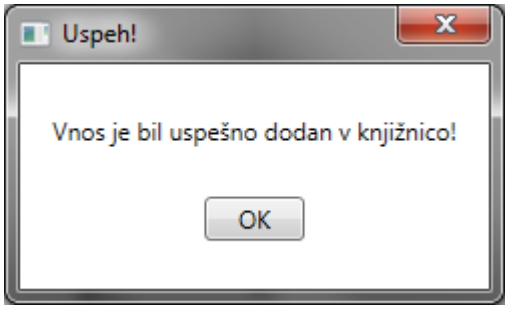
V aplikaciji imamo en pomožni razred `RelayCommand.cs`, ki implementira vmesnik `ICommand`. Ta v konstruktorju kliče obe metodi vmesnika in sproži ustrezno akcijo. Gumb

`Potrdi` je vezan na ukaz `SaveCommand`, ki implementira pomožni razred `RelayCommand`. Ima definirani metodi `Save()`, ki skrbita za izvrševanje ukaza `SaveCommand()` ter `CanSave()` in preverja, ali je ukaz sploh izvršljiv.



Slika 3.9: Izpolnjen obrazec za dodajanje del.

Metoda `CanSave()` preverja, ali so obvezna vnosna polja izpolnjena in vrača vrednost false, dokler niso. Na sliki 3.9 so obvezna polja izpolnjena, torej je metoda `CanSave()` vrnila vrednost true. Sedaj se s klikom na gumb `Potrdi` sproži metoda `Save()` v kateri se odpre povezava s podatkovno bazo in izvede stavek SQL za dodajanje vrstic `INSERT`. Ob uspešni izvedbi stavka se odpre sporočilno okno, ki uporabniku potrdi, da je bil vnos dodan v podatkovno bazo, kot je razvidno iz slike 3.10.



Slika 3.10: Uspešen vnos v tabelo.

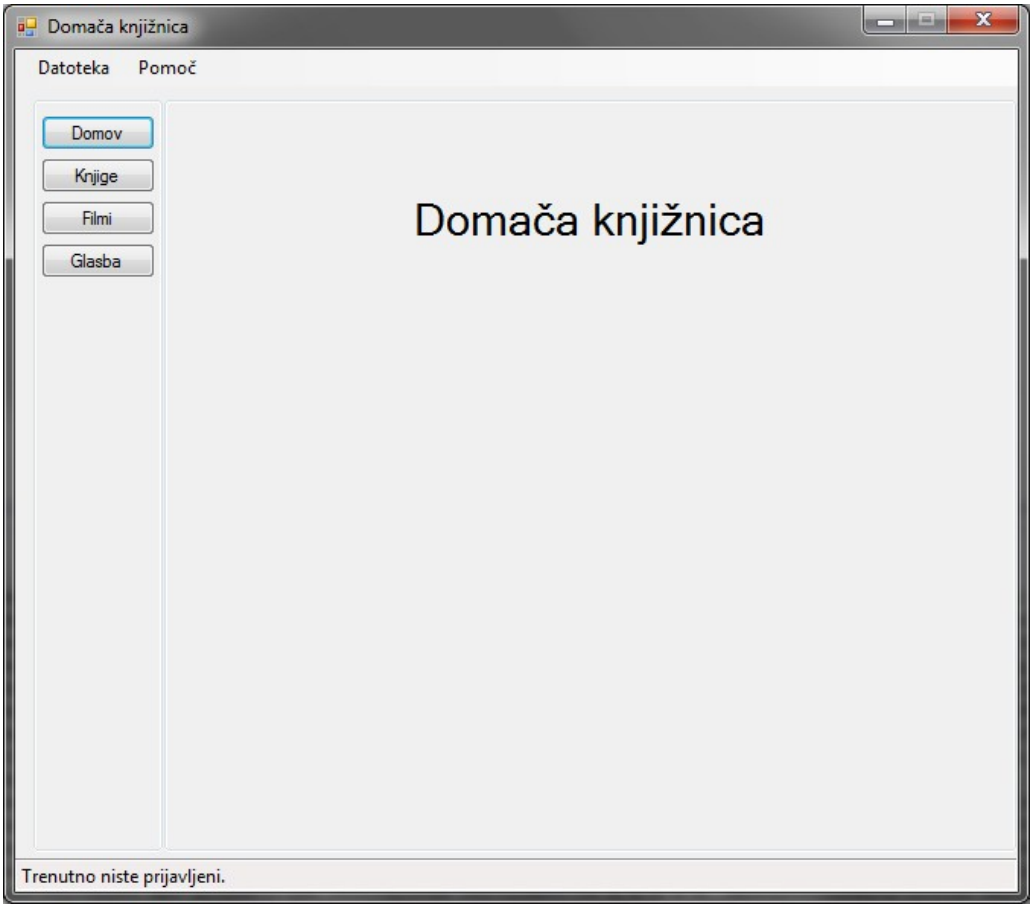
3.3 Izdelava aplikacije z ogrodjem Windows Forms

Za izdelavo vmesnika z ogrodjem Windows Forms smo v Visual Studiu ustvarili nov projekt in izbrali ustrezno programsko okolje.

Oblikovanje grafičnega vmesnika z ogrodjem Windows Forms poteka z vlečenjem gradnikov in kontrol iz nabora orodij na delovno površino oziroma zaslonski obrazec. Vsakemu gradniku, ki ga dodamo na zaslonski obrazec, lahko priredimo lastnosti in dogodke. Vsaka maska je sestavljena iz dveh datotek in dveh delnih razredov (angl. partial class), vendar je eden izmed njih avtomatično generirana koda, ki se v razred dodaja, ko postavljamo gradnike na zaslonski obrazec. Drugi delni razred vsebuje vso kodo v ozadju, od metod, ki se sprožijo ob interakciji z zaslonsko masko, do dostopanja do baze in ostale logike, ki jo aplikacija za delovanje potencialno potrebuje.

3.3.1 Osrednje okna vmesnika in prijavno okno

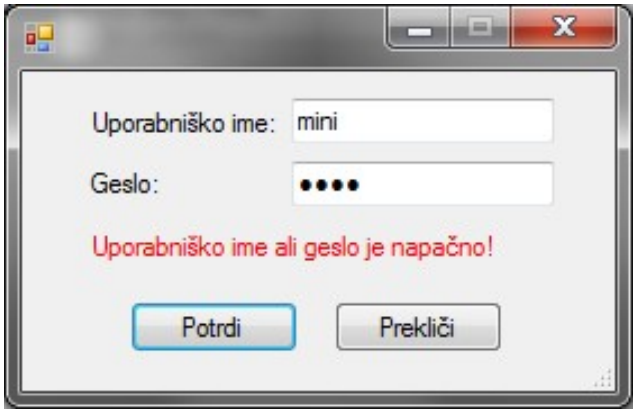
Ko se aplikacija odpre, se uporabniku prikaže osrednja zaslonska maska kot je prikazano na sliki 3.11. Uporabnik sprva ni prijavljen, kar je zavedeno v statusni vrstici. Na vrhu je orodna vrstica, kjer se uporabnik lahko vpiše, zapre program, ali odpre navodila za pomoč. Na desni strani so gumbi za navigacijo, levo pa je plošča, kjer se prikazuje in spreminja vsebina glede na pritisnjene gumbe.



Slika 3.11: Osrednja zaslonska maska aplikacije.

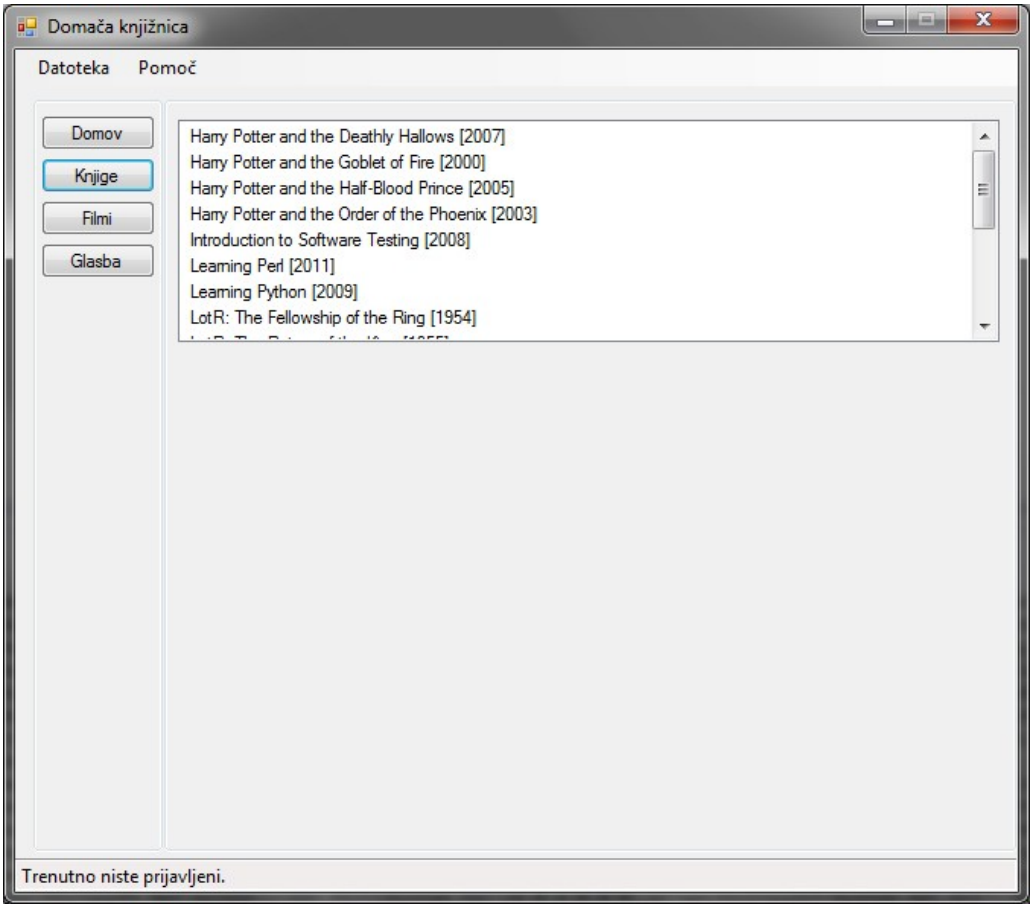
Gumbi za navigacijo so razporejeni znotraj gradnika za grupiranje (angl. GroupBox), da so vizualno ločeni od vsebinskega dela zaslonske maske z robom. Tudi vsebina je znotraj svojega gradnika za grupiranje, le da je znotraj tega le plošča (angl. panel), na kateri se izrisujejo maske, ki jih sprožijo pritisnjeni gumbi.

S klikom na gumb `Datoteka` v orodni vrstici se uporabniku odpre meni, na katerem je tudi opcija za vpis v aplikacijo. S klikom na izbiro se odpre prijavno okno. Kot je prikazano na sliki 3.12 je prijavno okno preprost dialog, ki uporabnika vpraša po uporabniškem imenu in geslu. Če je eno ali drugo napačno (ali pa v podatkovni bazi ne obstaja), se na oknu prikaže opozorilo. V nasprotnem primeru se okno uspešno zapre in na statusni vrstici uporabniku sporoči, da je vpisan.



Slika 3.12: Prijavno okno z opozorilom.

3.3.2 Navigacija



Slika 3.13: Seznam knjig.

Zaslonske maske so običajno narejene tako, da se vse kontrole in gradniki izrišejo ob inicializaciji [10]. Plošča, na kateri se prikazuje vsebina s slike 3.13 pa se osvežuje sproti

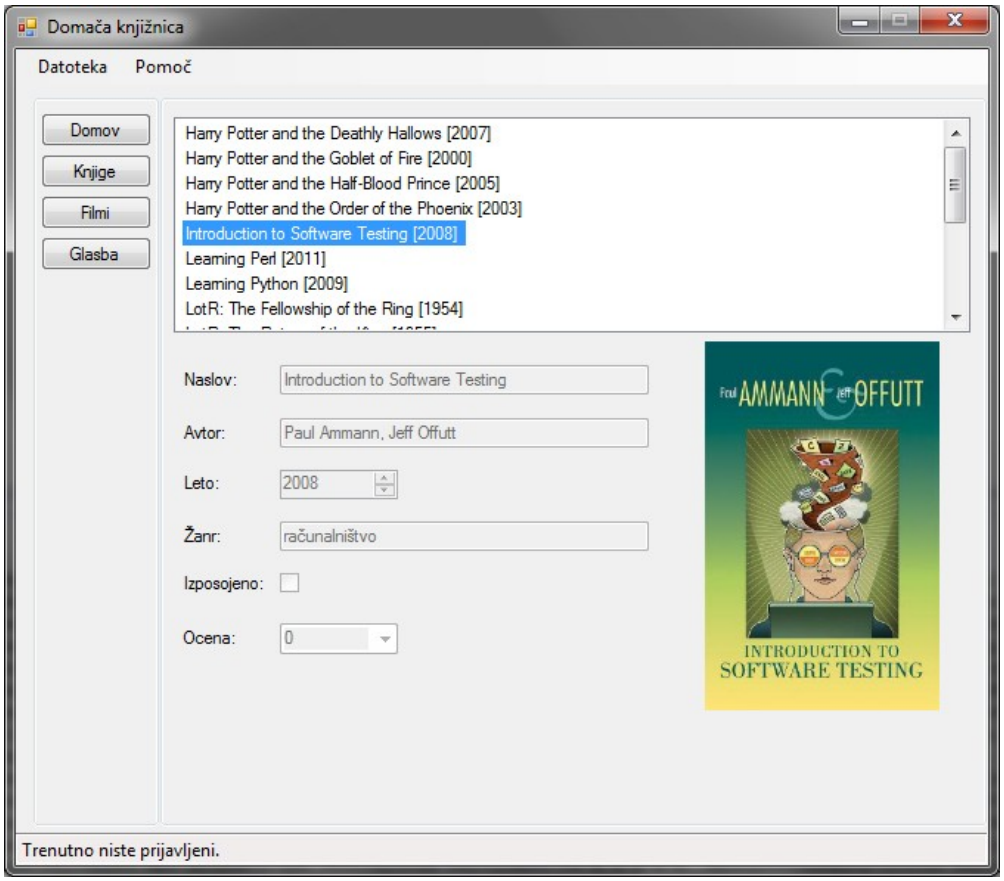
glede na to, kateri gumb na meniju je uporabnik pritisnil. To je rešeno tako, da klik na gumb izprazni seznam trenutnih kontrol na plošči in doda tisto, ki jo uporabnik želi videti. Kar je na končnem uporabniškem vmesniku videti zgolj kot spreminjanje vsebine, je v ozadju ves čas brisanje in ponovno izrisovanje kontrol, kot je prikazano na sliki 3.14.

```
private void btAddClick(object sender, EventArgs e)
{
    panelContent.Controls.Clear();
    panelContent.Controls.Add(new CustomForms.AddNew());
}

private void btHome_Click(object sender, EventArgs e)
{
    panelContent.Controls.Clear();
    panelContent.Controls.Add(new CustomForms.Splash());
}
```

Slika 3.14: Izrisovanje zaslonskih mask.

3.3.3 Izpis podatkov na zaslonski maski

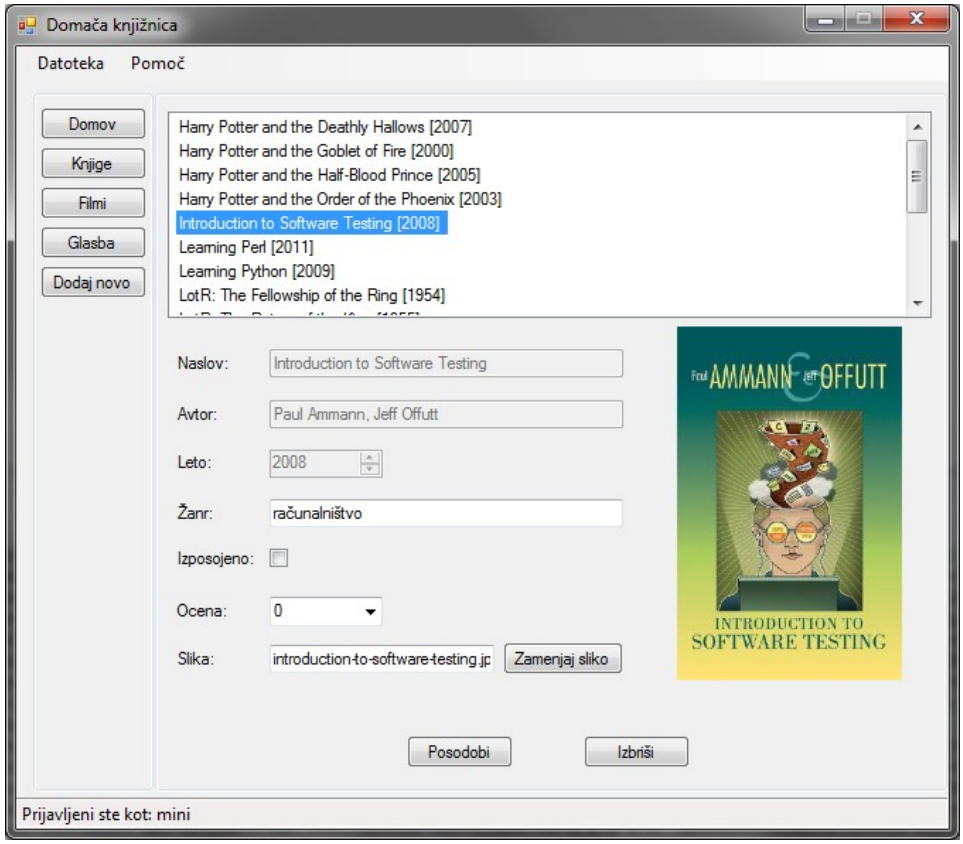


Slika 3.15: Prikazane informacije o izbranem delu.

Kot je prikazano na sliki 3.15 se neprijavljen uporabnik po seznamu knjig lahko le sprehaja, ne more pa spreminjati in brisati vnosov, kot tudi ne dodajati novih.

Namesto zgolj enega dostopa do podatkovne baze sta potrebna za izpis vseh podatkov na zaslonsko masko dva: gradnik seznama, na katerem je seznam vseh del nekega tipa, izpiše vse vrstice iz podatkovne baze. Metoda `get()`, ki je poklicana ob inicializaciji maske naredi povezavo s podatkovno bazo in izvede stavek `SELECT`, ki iz klicane tabele izpiše naslov, letnico in identifikacijsko številko dela. Slednja sicer na gradniku seznam ni vidna, je pa pomembna za kasnejšo obdelavo podatkov. Povezava z bazo se zapre, ko so podatki izpisani na seznamu.

Razen naslova, letnice in identifikacijske številke sedaj nimamo ostalih podatkov, saj jih ne hranimo v objektu. Ko uporabnik klikne na posamezno delo, da izve ostale informacije, se v ozadju izvede metoda `getOne(int id)`. Iz seznama dobi parameter `id`, (identifikacijska številka dela), ki je edino enolično polje pri posameznem delu. V metodi se ponovno odpre povezava s podatkovno bazo in izvede poizvedba s stavkom `SELECT`, ki iz zahtevane tabele izpiše vse podatke v vrstici s tisto identifikacijsko številko, ko smo jo dobili kot parameter iz seznama del. Podatke izpiše v gradnike na plošči.

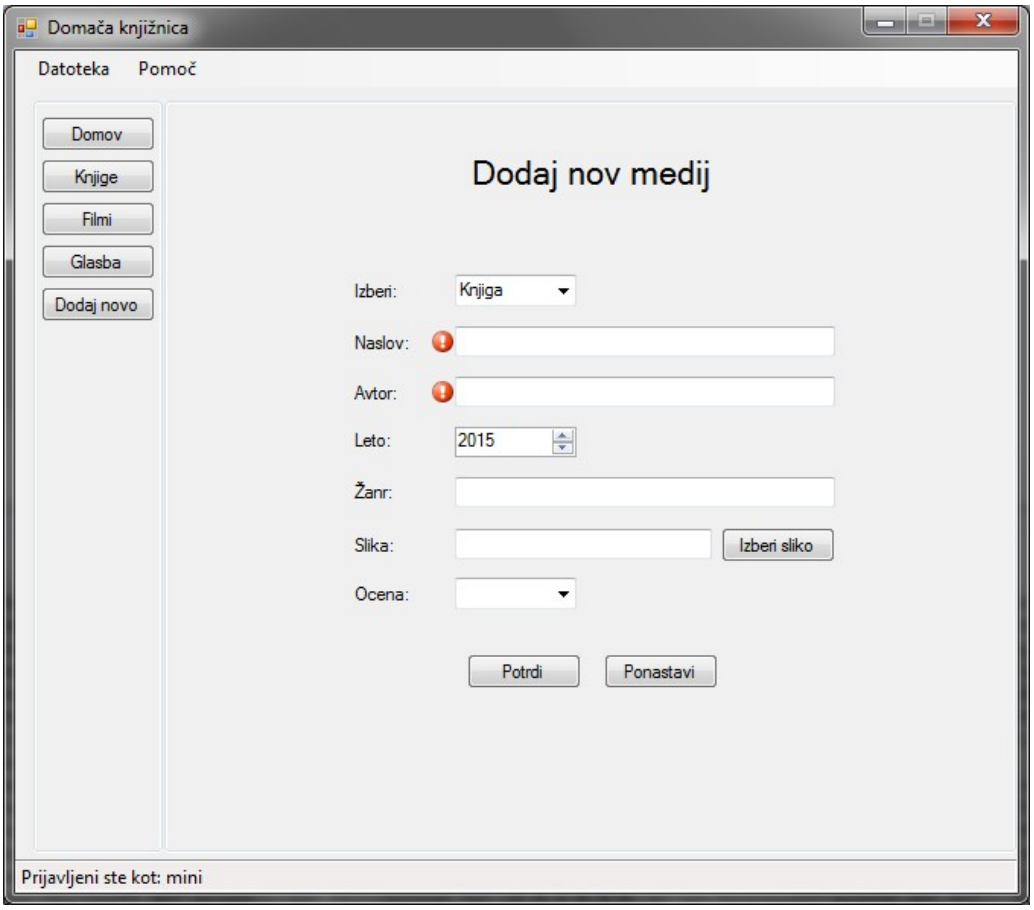


Slika 3.16: Prikazani podatki o izbranem delu in možnost popravljanja podatkov.

Prijavljen uporabnik lahko popravlja in briše vnose kot je razvidno iz slike 3.16. V primeru, da uporabnik ni vpisan, gumbov za urejanje, brisanje in menjavo slik sploh ne vidi, saj so kontrole nevidne.

3.3.4 Dodajanje novega vnosa

Kot je razvidno iz slik 3.15 in 3.16 je gumb za dodajanje novega vnosa v aplikacijo skrit ali viden glede na uporabnikov status. Namesto onemogočenih kontrol znotraj maske za dodajanje vsebin, je onemogočen celoten dostop do zaslonske maske dodajanja.



Slika 3.17: Dodajanje novega medija.

Ko je uporabnik prijavljen, lahko v svojo knjižnico doda nov vnos. Mora si izbrati tip medija, saj to vpliva na parametre, ki jih podamo v stavku `INSERT` pri dodajanju v podatkovno bazo. Poleg tega so obvezna tudi druga polja (kot so naslov, avtor in leto), ki ravno tako ne smejo ostati prazna. Če se to zgodi (kot je prikazano na sliki 3.17), se pojavi na zaslonski maski opozorilo pri vnosnem polju, ki je ostalo neizpolnjeno. Pri opozarjanju na napake je uporabljen razred `ErrorProvider`, ki vizualno prikaže napako na kontroli, na katero je

vezan. Če uporabnik kljub opozorilu pritisne na gumb `Potrdi` in želi shraniti vnos, se odpre dialog, ki ga opozori, da so polja ostala prazna, povezava s podatkovno bazo pa se ne odpre. V primeru uspešnega vnosa se odpre dialog, ki potrdi uspešen vnos podatkov v bazo.

Poglavje 4 Primerjava ogrodij WPF in Windows Forms

Oblikovanje uporabniških vmesnikov v ogrodju WPF se znatno razlikuje od oblikovanja vmesnikov v ogrodju Windows Forms. Kljub temu, da je končni rezultat na videz enak, sta bila postopka izdelave vmesnikov drugačna. V prejšnjem poglavju sta bila opisana oba postopka izdelave, v tem poglavju pa bomo podrobneje opisali razlike med posameznimi vidiki razvoja uporabniškega vmesnika. Ogrodji smo v vsaki kategoriji tudi ocenili z oceno od 1 do 5, kjer ocena 1 predstavlja najslabši, ocena 5 pa najboljši rezultat.

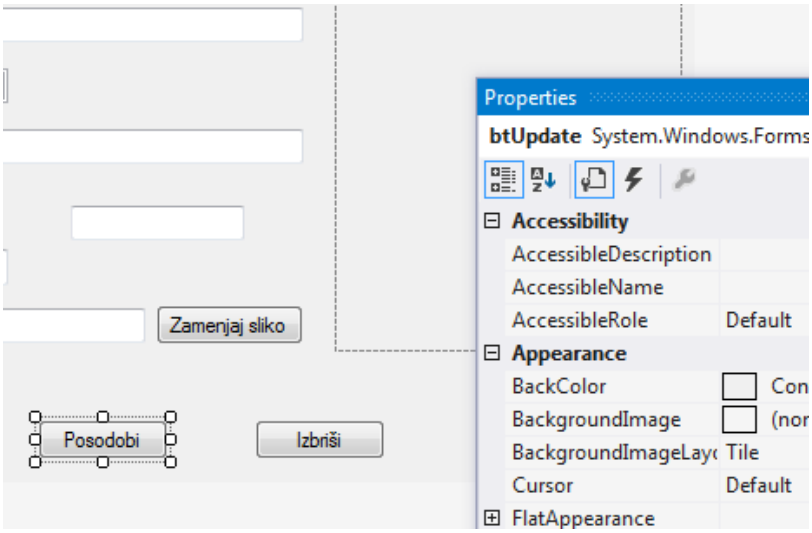
4.1 Dokumentacija

Podjetje Microsoft ponuja ustrezno dokumentacijo tako za ogrodje WPF [15] kot tudi za ogrodje Windows Forms [14]. Dokumentacija obsega vse od vadnic za začetnike, do primerov uporabe posameznih razredov in gradnikov ter bolj zahtevnih in posebnih primerov izvedbe ogrodja. Pomanjkanja bogate uradne dokumentacije torej ni, ima pa Windows Forms večjo podporo na uradnih (in neuradnih) forumih, čeprav je to zgolj zaradi starosti ogrodja v primerjavi z WPF.

Tako WPF, kot Windows Forms sta torej ustrezno podkrepljena z dokumentacijo in si zaslužita oceno 5.

4.2 Izdelava zaslonskih mask

Izdelava zaslonskih mask v ogrodju Windows Forms je bila nezahtevna. Iz zbirke komponent smo na obrazec povlekli želene komponente, jih postavili na poljubna mesta na obrazcu in jim dodelili lastnosti in ukaze, kar je razvidno iz slike 4.1.

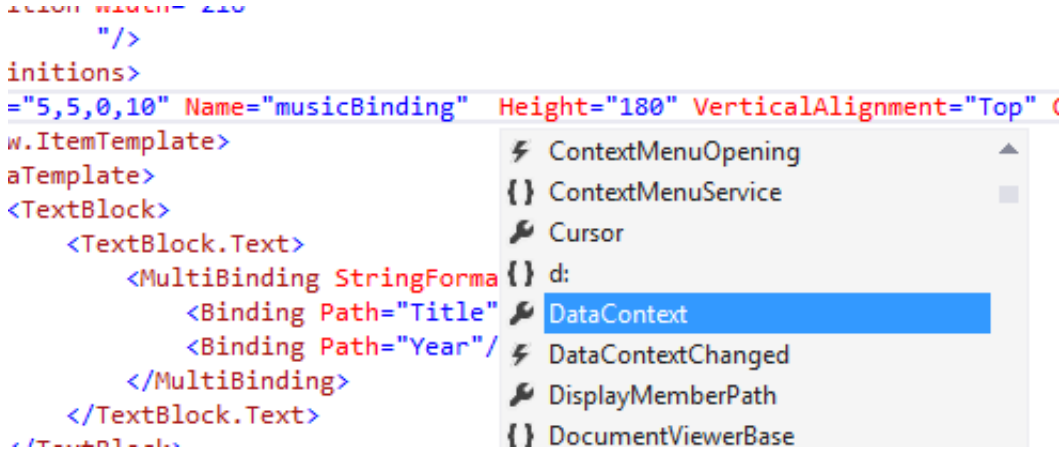


Slika 4.1: Oblikovanje uporabiškega vmesnika.

Potrebovali smo več zaslonskih mask in oken, pri vsaki pa je bil postopek enak. Aplikacija je zasnovana tako, da se na glavni zaslonski maski v osrednjem delu odpira različna vsebina glede na uporabnikove ukaze in klike gumbov. Tudi to je bilo v ogrodju Windows Forms nezahtevno, saj se je s klikom na gumb v meniju na osrednji del aplikacije izrisovala primerna zaslonska maska, kot je predstavljeno v poglavju 3.3.2.

Kljub temu, da v programu še nismo imeli povezave s podatkovno bazo in v poljih na zaslonskih maskah ni bilo izpisane vsebine, je program vseboval že vso potrebno funkcionalnost. Gumbi so se odzivali na uporabnikove ukaze in spreminjali vsebino, v orodni vrstici je uporabnik lahko odprl okno za prijavo (kjer se na tej točki sicer ni zgodilo nič, ravno zaradi povezave z bazo, ki še ni implementirana) ali poiskal pomoč in podobno. Zunanja podoba aplikacije je bila izdelana.

Izdelava zaslonskih mask z ogrodjem WPF je potekala drugače. Vizualno podobo aplikacije smo oblikovali z označevalnim jezikom XAML. To nam je dalo večji nadzor nad obliko in postavitvijo komponent na posamezni maski. Zahteva več poznavanja komponent in njihovih lastnosti, čeprav pri tem pomaga sistem IntelliSense [16] znotraj programa Visual Studio, ki nam predlaga možne lastnosti in ukaze in je prikazan na sliki 4.2.



Slika 4.2: Sistem IntelliSense.

Oblikovanje zaslonske maske s pisanjem v jeziku XAML ima svoje prednosti, predvsem kadar želimo na masko postaviti več enakih komponent (na primer več gumbov), saj jih v kodi lahko le kopiramo in prilepimo in jim popravimo lastnosti.

Na prvo težavo smo naleteli, ko smo želeli na glavni maski prikazati druge. Po vzorcu MVVM se pogledi (zaslonske maske) med seboj ne morejo sklicevati, torej je bilo treba najti drugačno rešitev. Glavni maski smo dodali model pogleda, v katerem je shranjen podatek o trenutno prikazani zaslonski maski znotraj osrednjega dela, kjer se prikazuje vsebina. Kot je razloženo v poglavju 3.2.2, smo v modelu pogleda glavne maske morali vpeljati vmesnik `INotifyPropertyChanged`, ki skrbi za obveščevanje o proženju sprememb. Gumbov za navigacijo tako nismo definirali v pogledu, temveč so definirani v modelu pogleda in se nanje v pogledu sklicujemo z lastnostjo `DataContext`. Gumbi, ki se pojavijo ob inicializaciji zaslonske maske, so že vezani na svoje poglede (in modele pogledov), kot smo to definirali modelu pogleda. Vse zaslonske maske razen glavne imajo implementiran vmesnik `PageViewModel`, ki jim dodeli skupno lastnost `Ime`. Vsebina, ki se izpiše na gumbih v navigaciji glavne zaslonske maske, je vezana na to lastnost, torej se izpiše na gumbu tako, kot je v modelu pogleda definirana.

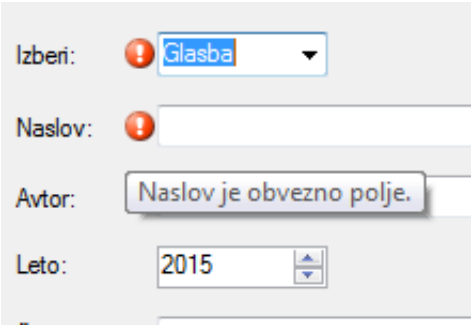
Ko smo vpeljali vmesnike in metode za navigacijo med zaslonskimi maskami, je bil skelet aplikacije narejen.

4.2.1 Manjše razlike med v izdelavi zaslonskih mask

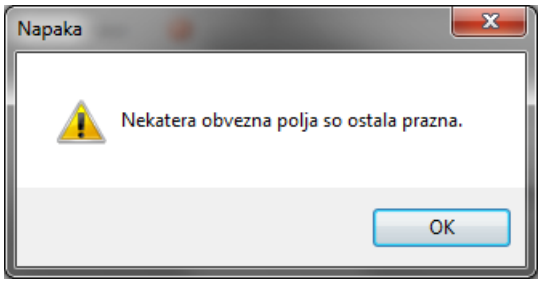
Poleg navigacije, ki je v ogrodju WPF predstavljala večjo oviro, se je v postopku oblikovanja zaslonskih mask pojavilo še nekaj razlik med ogrodjema WinForms in WPF, predvsem pri izbiri komponent, ki ponekod nimajo svojih enakovrednih elementov v obeh ogrodjih.

4.2.1.1 Preverjanje napak

V ogrodju WinForms obstaja metoda `ErrorProvider`, ki jo lahko nastavimo poljubni komponenti na zaslonski maski, metoda nato vizualno prikaže napako na kontroli. Kdaj pokaže napako je seveda odvisno od logike, ki je v ozadju komponent. V tem primeru smo potrebovali validacijo vnosnih polj pri dodajanju novih vnosov, da obvezna polja ne bi ostala prazna. Iz slik 4.3 in 4.4 je razviden način obveščanja uporabnika o napakah. Na sliki 4.3 so prikazana opozorila, ki jih izpiše metoda `ErrorProvider`, na sliki 4.4 pa sporočilno okno, ki se odpre, če uporabnik opozorila ignorira.



Slika 4.3: Obvestilo o obveznem polju.



Slika 4.4: Sporočilno okno.

Ogrodje WPF metode `ErrorProvider` nima in je bilo za validacijo treba najti drugačno rešitev. Zaslonska maska, ki zahteva validacijo (kot npr. pogled za dodajanje novega medija) implementira vmesnik `IDataErrorInfo`, ki preverja napake na komponentah, na katere je vezan. Komponenti, kjer preverjamo napake, moramo nastaviti poleg povezave na lastnost v modelu pogleda tudi lastnost, ki pove, da je komponenta preverljiva, torej ob napaki uporabnika o tem opozori. Metoda `ErrorProvider` v ogrodju WinForms poskrbi tudi za vizualno podobe napake na zaslonski maski, v ogrodju WPF pa moramo v pogledu nastaviti predlogo, ki bo v programu predstavljala napako, kot je prikazano na sliki 3.5. Spustni seznam `cbSelection` ima nastavljeno lastnost, da preverja napake. Predloga, ki se ob validaciji pojavi je definirana na sliki 4.5.

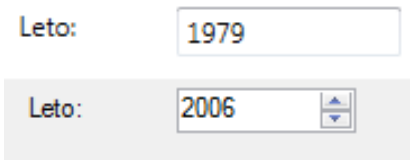
```
<UserControl.Resources>
    <ControlTemplate x:Key="validate">
        <DockPanel>
            <TextBlock Foreground="Red" FontSize="16" Text="*"/>
            <AdornedElementPlaceholder x:Name="ErrorAdorner"/>
        </DockPanel>
    </ControlTemplate>
</UserControl.Resources>

<ComboBox Name="cbSelection" Grid.Row="1" Grid.Column="2"
          SelectedItem="{Binding Path=Choice, ValidatesOnDataEr..
          HorizontalAlignment="Left" Margin="10" Width="100"
          Validation.ErrorTemplate="{StaticResource validate}">
</ComboBox>
```

Slika 4.5: Spustni seznam vsebuje predlogo napake.

4.2.1.2 Komponenta `numericUpDown`

Komponenta `numericUpDown` oziroma `Spinner` je namenjena nastavljanju neke številske vrednosti v vnosnem polju s puščicami za manjšanje ali večanje vrednosti. Dopušča tudi ročno vnašanje vrednosti. Komponenta lahko zmanjša možnost napak v programu, saj je možno nastaviti dovoljene vrednosti, poleg tega pa ponavadi ne sprejema neštevilskih vrednosti. Aplikacija v ogrodju WinForms ima na zaslonski maski za dodajanje in predstavitev del polje, ki predstavlja leto. To polje je predstavljeno s komponento `numericUpDown`.



Slika 4.6: Razlike med komponentami.

Ogrodje WPF enakovredne komponente nima. Možno bi bilo izdelati svojo s pomočjo gumbov, vnosnega polja in logike za spreminjanje vrednosti ali pa uporabiti obstoječe predloge iz različnih kompletov programskih orodij. Za poudarek, da ima ogrodje WPF osnovne pomanjkljivosti, smo v aplikaciji pustili polje leto kot navadno vnosno polje in zgolj preverjamo, ali je vrednost število ali ne, kot je prikazano na zgornjem delu slike 4.6. Spodnja komponenta na sliki je `numericUpDown` v ogrodju WinForms.

Izdelava zaslonskih mask je v ogrodju Windows Forms preprostejša, poleg tega pa smo imeli na voljo tudi bolj specifične metode in komponente, ki jih v ogrodju WPF ni več ali pa je njihova izvedba bolj zapletena, saj je treba vpeljevati vmesnike. Windows Forms dobi oceno 5, WPF pa le 2.

4.3 Podatkovna baza in povezovanje podatkov

4.3.1 Podatkovna baza

Povezovanje podatkovne baze z aplikacijo je bilo v obeh primerih enako. V projekt je bilo treba dodati vir podatkov, ki predstavlja lokalno podatkovno bazo.

Povezavo smo vzpostavili z metodo `SqlConnection`, ki smo ji podali želene poizvedbe za vstavljanje, izpisovanje, urejanje in brisanje podatkov z metodo `SqlCommand`. Kadar smo uporabili stavek `SELECT`, smo se po vrnjenih podatkih sprehodili z bralcem (metoda `SqlDataReader`), ki podatke bere po vrsticah. Pri urejanju in dodajanju novih vnosov, smo metodi `SqlCommand` podali še parametre. Po končani poizvedbi smo povezavo zaprli.

4.3.2 Povezovanje podatkov

Povezovanje podatkov je v ogrodjih drugačno že zaradi različnih vzorcev načrtovanja. Ogrodje WPF sledi vzorcu MVVM, kar predpostavlja, da pogledi oziroma zaslonske maske praviloma nimajo neposrednega dostopa do podatkov v podatkovni bazi oziroma modelu, temveč povezovanje podatkov poteka prek modela pogleda. Aplikacija WinForms takemu vzorcu ne sledi, zato je vsa koda povezovanja z bazo v ozadju zaslonske maske in lahko komponente neposredno dostopajo do podatkov.

Primer različnega delovanja v ozadju je najbolj očiten v maski za prikazovanje vsebine. V aplikaciji WPF v objektih hranimo podatke, ki so pridobljeni iz tabele. Objekti predstavljajo model v vzorcu MVVM. Ob inicializaciji zaslonske maske pokličemo metodo `Get()`, ki seznam za izpisovanje del napolni z objekti. Ob izbiri ene vrstice v seznamu se v preostalih komponentah na zaslonski maski izpišejo podrobnejši podatki posameznega objekta oziroma dela. Namesto, da se ob izbiri posamezne vrstice ponovno povežemo s podatkovno bazo, podatke enostavno povežemo z objektom iz seznama. Vsebina, ki se prikaže v komponentah na maski je povezana na objekte v seznamu (vsi objekti imajo lastnosti, kot so naslov, avtor, letnica ipd.), torej se vsebina vnosnih polj spreminja glede na zbran objekt v seznamu. To je prikazano na sliki 4.7. Seznam se imenuje `musicBinding`. V vnosnem polju `tbTitle` se izpiše besedilo, ki je vezano na lastnost `Title` iz seznama objektov.

```
<ListView Margin="5,5,0,10" Name="musicBinding" Height="180" Vertical...
     Grid.ColumnSpan="3" SelectionChanged="musicBinding_SelectionChanged">
  <ListView.ItemTemplate...>
</ListView>
<TextBlock Grid.Row="1" Margin="10,0,0,0" VerticalAlignment="Center" ...
<TextBox Name="tbTitle" HorizontalAlignment="Left" Height="25"
         Grid.Row="1" Grid.Column="1" Width="250" IsEnabled="False"
         Text="{Binding SelectionItem.Title, ElementName=musicBinding}"/>
```

Slika 4.7: Povezanost seznama in vnosnega polja.

Maska za prikazovanje vsebine ima onemogočeno upravljanje in brisanje del, če uporabnik ni prijavljen. Sama koda v ozadju pogleda za onemogočanje gumbov ne skrbi, temveč je v model pogleda implementiran vmesnik `ICommand`. Sestavlja ga metoda, ki preveri, ali je neki ukaz izvršljiv in metoda, ki ta ukaz izvede. V tem primeru mora vmesnik preveriti, če je uporabnik vpisan. Če je, potem sta gumba za posodabljanje in brisanje knjig omogočena. V primeru, da se uporabnik izpiše, vmesnik `ICommand` ujame spremembo in gumba ponovno onemogoči.

V aplikaciji WinForms je celotna logika maske za prikazovanje vsebine v ozadju obrazca. Ker nimamo modela pogleda, ne potrebujemo niti modela in podatkov iz podatkovne baze ne hranimo v objektih. Obrazec ima neposredni dostop do podatkov, ki jih pridobimo iz baze. Zaradi drugačnega pristopa se tu dvakrat povežemo s podatkovno bazo: prvič, ko želimo izpisati podatke v seznam in naredimo poizvedbo zgolj po naslovih in letnicah del in drugič, ko želimo pridobiti bolj podrobne podatke o posameznem delu. Podatke ob poizvedbi neposredno zapišemo v komponente na zaslonski maski.

Kot v aplikaciji WPF je tudi tu posodabljanje in brisanje del onemogočeno, če uporabnik ni prijavljen. Tu ni vmesnika `ICommand`, ki bi preverjal stanje uporabnika in na podlagi tega omogočal gumbe. V ozadju maske so lastne metode, ki komponente zaklenejo ali odklenejo glede na status uporabnika.

Pri obeh ogrodjih je bilo povezovanje s podatkovno bazo enostavno za vpeljavo. Za prikazovanje vsebine smo sicer uporabili različne prijeme, vendar smo v primeru ogrodja WPF le izkoristili moč povezovanja na objekte. Če bi imeli zelo veliko število podatkov, bi bilo verjetno bolje podatke iz baze pridobivati sproti, kot je to vpeljano v ogrodju Windows Forms.

Rešitev onemogočanja podatkov glede na to, ali je uporabnik prijavljen, pa je bolje vpeljana v primeru ogrodja WPF. Za onemogočanje skrbi poseben vmesnik. Pri Windows Forms je potrebno onemogočiti vsako komponento posebej. WPF dobi pri povezovanju podatkov skupno oceno 4, Windows Forms pa 3.

4.4 Enostavnost uporabe

Ogrodje Windows Forms ponuja bolj preprost način razvijanja uporabniških vmesnikov in je navidezno bolj preprosto orodje. Poljubne gradnike povlečemo na zaslonsko masko in jim dodamo lastnosti in ukaze. To ne zahteva predznanja posameznih metod ali dodatnega znanja opisnega jezika XAML, kot je to potrebno pri WPF. Kljub temu pa lahko tudi preproste aplikacije hitro postanejo slabše obvladljive, saj je vsa koda shranjena v eni datoteki - od kode, ki je vezana na posamezne komponente na zaslonski maski do kode, ki predstavlja logiko v ozadju.

Ogrodje WPF je težje za začetnika in ga ni preprosto osvojiti, je pa kasneje zato lažje obvladljivo. Enostavneje je popravljati zaslonske maske, saj so te neodvisne od logike v ozadju in ni treba biti pozoren na oboje hkrati. Ravno zaradi te organizacije podatkov postane tudi WPF enostavno za uporabo, kljub temu da sprva tako ne zgleda.

V kategoriji enostavnosti uporabe si obe ogrodji zaslužita oceno 3. Kljub navidezni enostavnosti lahko ogrodje Windows Forms hitro postane težko obvladljivo, poleg tega pa je pri velikem številu komponent tudi zelo težko spreminjati nastavitve. To je pri ogrodju WPF lažje, saj nastavitve lahko pišemo z jezikom XAML. WPF pa je ogrodje, ki zahteva več znanja in vaje, preden ga razvijalec res osvoji. Šele takrat postane enostavno.

4.5 Ugotovitve

Zaradi vzorca MVVM, ki je tesno povezan z ogrodjem WPF, je razvijalec prisiljen v logičen red, ki aplikacijo razdeli na dva izrazita dela - grafični del in kodo. Kljub temu, da to pomeni več datotek v ozadju ene zaslonske maske, v resnici prinaša boljšo organiziranost, saj vsebinsko loči kodo grafičnega vmesnika od predstavitvene logike.

Datoteka s kodo zaslonske maske v aplikaciji WinForms je hitro postala neobvladljiva in natlačena, saj vsebuje tako povezovanje na podatkovno bazo, izpisovanje podatkov v komponente, kot metode, ki komponente spreminjajo glede na status uporabnika in vse pomožne metode, ki jih potrebujemo.

Zaslonska maska z enako vsebino v aplikaciji WPF ima bolje organizirano strukturo. V pogledu je z jezikom XAML definirana vizualna podoba, v kodi v ozadju pogleda pa le metode in funkcije, ki se tičejo le pogleda. Primer je metoda v maski za prikazovanje v ozadju pogleda, ki odpre okno za izbiro nove slike. Ime slikovne datoteke se izpiše v pripadajočo komponento, ta komponenta pa je vezana na lastnost v modelu pogleda. Samega okna za

izbiro slike ne potrebujemo v nobenem drugem primeru in ne vpliva na spreminjanje lastnosti, zato je smiselno kodo pustiti v ozadju pogleda in ne v modelu pogleda. Tam je shranjena logika povezovanja s podatkovno bazo, metode za posodabljanje in brisanje podatkov.

Skupna povprečna ocena kategorij primerjave za WPF je 3.5, za ogrodje Windows Forms pa 4. Boljša končna ocena ne pomeni, da je ogrodje na splošno boljše, saj so merila primerjave nastavljeni glede na naše zahteve in veljajo le na primeru razvite aplikacije. Iz tega sklepamo, da se je v našem primeru bolj obneslo ogrodje Windows Forms. Če bi primerjali po drugih merilih, kot je na primer večji poudarek na oblikovanju in bogatem vmesniku, bi rezultati bili drugačni.

Poglavje 5 Sklepne ugotovitve

V diplomski nalogi smo primerjali ogrodji Windows Forms in Windows Presentation Foundation, ki sta namenjeni izdelavi uporabniških vmesnikov. Poleg teoretične primerjave nas je zanimalo, kako praktično izgleda postopek oblikovanja uporabniškega vmesnika. V ta namen smo razvili preprosto aplikacijo, v katero lahko uporabnik dodaja podatke, jih ureja in briše. Podatki so shranjeni v podatkovni bazi. Najprej smo se lotili izdelave aplikacije v ogrodju WPF, nato pa smo uporabniški vmesnik z enako funkcionalnostjo izdelali še v ogrodju Windows Forms. To sicer pomeni, da smo nekatere probleme razrešili, ko smo se prvič lotili razvoja v enem ogrodju in v drugem ogrodju to ni zahtevalo veliko dodatnega dela.

Kljub temu, da ima končni izdelek enako funkcionalnost, zelo podoben uporabniški vmesnik in končni uporabnik ne vidi razlike, je bil postopek izdelave popolnoma drugačen.

Ogrodje Windows Forms ponuja preprostejši način ustvarjanja uporabniških vmesnikov. Razvijalec oblikuje zaslonsko masko z želenimi komponentami in nato doda maski funkcionalnost, v katero je vključena tudi logika dostopanja do podatkovnih objektov. Tehnologija je starejša in ima (trenutno) tudi več dokumentacije in podpore.

Ogrodje WPF ni tako preprosto. Je bolj fleksibilno, prilagodljivo in močno orodje za razvoj raznovrstnih aplikacij, vendar je lahko za začetnika videti bolj okorno. Veliko funkcionalnosti je odvisnih od razvijalca in ni dostopnih že od samega začetka. Razvijalec se lahko bolj posveti logiki dostopa do podatkovnih objektov, nato pa oblikuje zaslonsko masko s komponentami, ki podatkovne objekte poslušajo. Moč ogrodja WPF je predvsem v delitvi oblikovne plasti aplikacije od programske. To pomeni, da razvijalec lahko zaslonske maske in podobo uporabniškega vmesnika oblikuje neodvisno od razvoja programske logike v ozadju.

Praktična poglobitev v ogrodje Windows Forms in ogrodje WPF nas je pripeljala do zaključka, da sta v današnjem svetu obe ogrodji razvoja uporabniških vmesnikov še vedno zelo uporabni. Novejše ogrodje WPF ni zamenjava za Windows Forms, saj bodo aplikacije, ki ne potrebujejo veliko funkcionalnosti in prilagodljivega oblikovanja uporabniškega vmesnika hitreje in lažje razvite z ogrodjem Windows Forms. To sicer ne pomeni, da je ogrodje WPF uporabno samo takrat, kadar je potrebno oblikovati bogat in interaktiven vmesnik. Prednost

ogrodja WPF je tudi v drugačnem načinu povezovanja podatkov iz modela (podatkovne baze), ki ni odvisno od same zaslonske maske in njenih funkcionalnosti.

Izbira ogrodja za izdelavo uporabniškega vmesnika je tako prepuščena posameznim razvijalcem, ki se na podlagi zahtev in funkcionalnosti aplikacije odločijo katero ogrodje je zanje bolj primerno. Kot izhodiščno točko pa jim lahko pri tej izbiri služijo rezultati naše raziskave. V primeru naše aplikacije je bilo ogrodje Windows Forms bolj primerno, saj je ponujalo bolj učinkovite rešitve številnim težavam, s katerimi smo se soočili. V primeru, da bi aplikacijo razširili in jo obogatili tako vizualno kot vsebinsko, bi bilo za rešitev bolj primerno ogrodje WPF.

Primerjavo ogrodij WPF in Windows Forms smo opravili na preprosti aplikaciji z majhno podatkovno bazo. V nadaljevanju bi lahko aplikacijo razširili in ji dodali dodatne funkcionalnosti, kot je komentiranje posameznih del in iskanje po knjižnici. Lahko bi razširili tabele v podatkovni bazi, da bi vsebovale bolj natančne podatke po katerih bi uporabnik iskal in tudi oblikovali vizualno bolj zanimiv vmesnik.

Literatura

[1] R. Garofalo, Applied WPF 4 in Context, Apress, 2011.

[2] J. Gossman. Introduction to Model/View/ViewModel pattern for building WPF apps. (5 Oktober 2005). [Online]. Dosegljivo: http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx. [Dostopano: januar 2015].

[3] H. Huynh. (2012, april) Encrypt With MD5, SHA Or SHA1 In MS SQL Server. [Online]. Dosegljivo: http://4rapiddev.com/sql-server/encrypt-with-md5-sha-or-sha1-in-ms-sql-server/. [Dostopano: januar 2015].

[4] P. Jones. (2001, september) US Secure Hash Algorithm 1 (SHA1). [Online]. Dosegljivo: https://tools.ietf.org/html/rfc3174. [Dostopano: januar 2015].

[5] R. Lim. Rachel Lim's Blog. (8 Maj 2011). [Online]. Dosegljivo: https://rachel53461.wordpress.com/2011/05/08/simplemvvmexample/. [Dostopano: januar 2015].

[6] M. MacDonald, Pro WPF 4.5 in C#, Apress, 2012.

[7] Microsoft. Developing Client Applications with the .NET Framework. [Online]. Dosegljivo: https://msdn.microsoft.com/en-us/library/54xbah2z%28v=vs.110%29.aspx. [Dostopano: januar 2015].

[8] Microsoft. Development Platform Support. [Online]. Dosegljivo: http://www.visualstudio.com/explore/development-platform-support-vs. [Dostopano: januar 2015].

[9] Microsoft. Download the software: Microsoft SQL Server 2014 Express. [Online]. Dosegljivo: https://msdn.microsoft.com/en-us/evalcenter/dn434042.aspx. [Dostopano: januar 2015].

[10] Microsoft. How to: Add Controls to an ASP.NET Web Page Programmatically. [Online]. Dosegljivo: https://msdn.microsoft.com/en-us/library/kyt0fzt1%28v=vs.140%29.aspx. [Dostopano: januar 2015].

[11] Microsoft. Implementing the MVVM Pattern Using the Prism Library 5.0 for WPF. [Online]. Dosegljivo: https://msdn.microsoft.com/en-us/library/gg405484%28v=pandp.40%29.aspx. [Dostopano: januar 2015].

[12] Microsoft. Integrated Development Environment. [Online]. Dosegljivo: http://www.visualstudio.com/explore/ide-vs. [Dostopano: januar 2015].

[13] Microsoft. Introducing Visual Studio. [Online]. Dosegljivo: https://msdn.microsoft.com/en-us/library/fx6bk1f4%28v=vs.90%29.aspx. [Dostopano: januar 2015].

[14] Microsoft. Introduction to Windows Forms. [Online]. Dosegljivo: https://msdn.microsoft.com/en-us/library/aa983655(v=vs.71).aspx. [Dostopano: januar 2015].

[15] Microsoft. Introduction to WPF. [Online]. Dosegljivo: https://msdn.microsoft.com/en-us/library/aa970268.aspx. [Dostopano: januar 2015].

[16] Microsoft. Using IntelliSense. [Online]. Dosegljivo: msdn.microsoft.com/en-us/library/hcw1s69b.aspx. [Dostopano: januar 2015].

[17] Microsoft. Visual Studio Ultimate with MSDN. [Online]. Dosegljivo: https://www.visualstudio.com/products/visual-studio-ultimate-with-MSDN-vs. [Dostopano: januar 2015].

[18] Microsoft. Walkthrough: Create a simple application with Visual C# or Visual Basic. [Online]. Dosegljivo: https://msdn.microsoft.com/en-US/library/vstudio/jj153219. [Dostopano: januar 2015].

[19] Microsoft. XAML Overview (WPF). [Online]. Dosegljivo: https://msdn.microsoft.com/en-us/library/ms752059%28v=vs.110%29.aspx. [Dostopano: januar 2015].

[20] J. Smith. WPF Apps With the MVVM Design Pattern. [Online]. Dosegljivo: msdn.microsoft.com/en-us/magazine/dd419663.aspx. [Dostopano: januar 2015].