Fakultet tehničkih nauka

Novi Sad

Dokumentacija za projekat “Disco”

Razvoj višeslojnih aplikacija

PR 83/2020 Bojana Mihajlović

PR 87/2020 Nevena Ćulibrk

PR 95/2020 Kristina Sretenović

PR 103/2020 Anabela Zonai

1. UVOD – Opis projekta

Tema ovog projekta je “Disco” klijent – server aplikacija, koja predstavlja sistem za upravljanje plejlistama.

Omogućeno je kreiranje novih plejlista, dodavanje pesama u plejlistu. Vođenje evidencije o plejlistama i pesmama se čuva u Entity Framework bazi podataka, a aplikacija se oslanja na MVVM(Model View View-Model) obrazac.

Obrazac je podeljen u nekoliko delova, a jedan od tih je Model koji predstavlja apstrakciju stvarnih modela koji čine realan sistem plejliste i koji su bitni za izgradnju ovog sistema, a tu spadaju: Plejlista i Pesma(osnovne informacije o pesmi koje su potrebne za dodavanje u plejlistu).

View omogućuje izgled korisničke aplikacije.

Pored View-a tu se nalazi i View Model – posrednik između View-a i Modela podataka, koji sadrži logiku zaduženu za prezentaciju podataka i za navigaciju kroz korisnički interfejs.

2. Komponente sistema

Gledajući sa najvišeg nivo, sistem se deli na server aplikaciju i više klijentskih aplikacija, odnosno uobičajena server-klijent arhitektura.

Naš sistem može da se podeli na sledeće komponente:

- Common – sadrži interfejse za komunikaciju između klijenta i servera, takođe tu su sadržani i modeli podataka korišćeni u razmeni između servera i klijenta. U Common-u ne postoji nikakva konkretna implementacija.

- Server – sadrži implementaciju interfejsa sadržanih u Common-u koji predstavljaju servise koje klijent može da koristi. Takođe zadužen je za pristup bazi, smeštanje podataka u bazu kao i za autentifikaciju korisnika. Ovaj deo se može podeliti na sloj pristupa bazi podataka i sloj poslovne logike.

- RVAProjekatDisco – Klijentska aplikacija sa grafičkim korisničkim interfejsom. Zadužena da korisniku omogući interakciju sa serverom. U projektovanju ovog dela je praćen MVVM šablon.

3. Tehnologije korištene u razvoju sistema

Pored osnovnih funkcionalnosti .NET framework-a korišćene su još dodatne tehnologije koje omogućavaju brz i efikasan razvoj pojedinih slojeva aplikacije:

- Windows Communication Foundation (WCF) – tehnologija korišćena za komunikaciju između klijenta i servera. Omogućuje lako i brzo pozivanje udaljenih procedura servera od strane klijenta i serijalizaciju podataka koji se koriste u komunikaciji. Tehnologija je korišćena u klijentskoj i serverskoj aplikaciji, a u delu „Common“ se nalaze zajednički interfejsi i modeli podataka korišćeni u komunikaciji.

- Windows Presentation Foundation (WPF) – tehnologija korišćena u klijentskoj aplikaciji za izradu korisničkog interfejsa, poštujući MVVM šablon pri projektovanju i razvoju. Omogućuje izradu interfejsa koji je prenosiv između različitih platformi, a MVVM šablon pomaže u jasnom razdvajanju „pogleda - View“ koji prikazuje podatke i interaktuje sa korisnikom i „pogled-modela – View Model“ koji čuva i obrađuje podatke

- Entity framework – tehnologija korišćena u serverskoj aplikaciji koja omogućuje brz, i siguran pristup bazi podataka. Omogućuje upravljanje bazom kao sa uobičajenim kolekcijama.

4. Korišćeni obrasci

U dizajnu i razvoju aplikacije korišteni su sledeći obrasci:

- Singleton
- Factory
- Prototype
- Command
- Facade
- Observer
- Proxy uz WCF

Singleton

Obezbeđuje da klasa ima samo jednu instancu i daje globalni pristup toj instanci. Odgovorna za kreiranje i rad sa svojom sopstvenom jedinstvenom instancom.

U izradi projekta singleton obrazac se koristio u klasama DbManager i SesijaManager smeštenih u folderu “PristupBaziPodataka". Klasa SesijaManager je zadužena za praćenje i upravljanje sesijom ulogovanih korisnika.

DbManager je klasa koja je zadužena za pristup bazi i preko njenih metoda svi ostali servisi pristupaju podacima iz baze. Ovde je izabran singleton za izradu DbManager-a zato što olakšava implementaciju i pristupačan je svima preko jedne pristupne tačke.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel;
using System.Text;
using System.Threading.Tasks;

namespace Server.PristupBaziPodataka
{
    public class DbManager
    {
        private static DbManager instance = null;

        public DiscoContext discoContext = null;

        private DbManager()
        {
            discoContext = new DiscoContext();
        }

        // Singleton pattern
        public static DbManager Instance
        {
            get
            {
                if (instance == null)
                {
                    instance = new DbManager();
                }
                return instance;
            }
        }
```

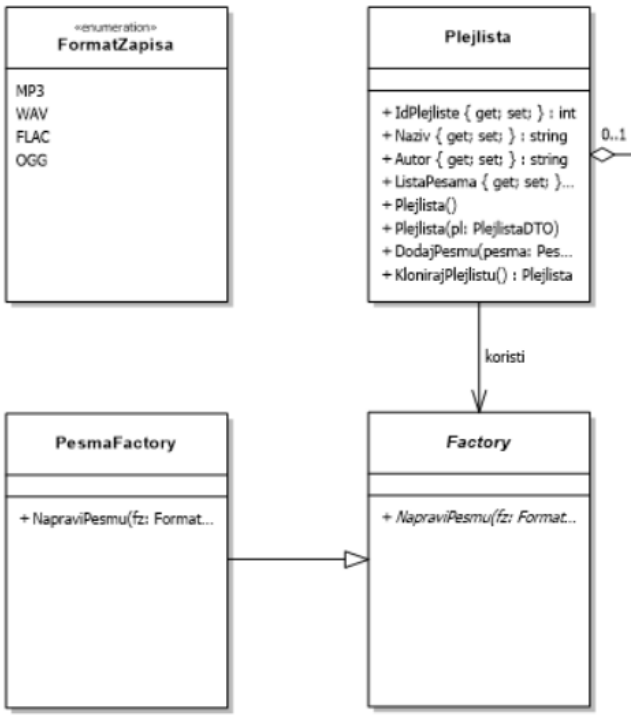
Factory

U ovom projektu, Factory obrazac se koristi za enkapsulaciju procesa kreiranja različitih formata pesama (MP3, WAV, FLAC, OGG) u aplikaciji koja upravlja muzičkim plejlistama. Cilj korišćenja ovog obrasca je omogućavanje fleksibilnog i jednostavnog proširenja aplikacije dodavanjem novih formata pesama bez potrebe za značajnim promenama u postojećem kodu.

Factory obrazac je implementiran kroz apstraktnu klasu Factory i konkretnu klasu PesmaFactory. Ove klase omogućavaju kreiranje različitih formata pesama bez direktnog instanciranja konkretnih klasa unutar koda koji koristi te instance.

Metoda NapraviPesmu u klasi PesmaFactory vraća instancu odgovarajuće klase pesme na osnovu prosleđenog parametra FormatZapisa.

Metoda DodajPesmu u klasi Plejlista koristi PesmaFactory za kreiranje nove instance pesme na osnovu njenog formata.



Prototype

Definiše mehanizam kako da se pravljenje objekta-duplikata određene klase poveri posebnom objektu date klase, koji predstavlja prototipski objekat te klase i koji se može

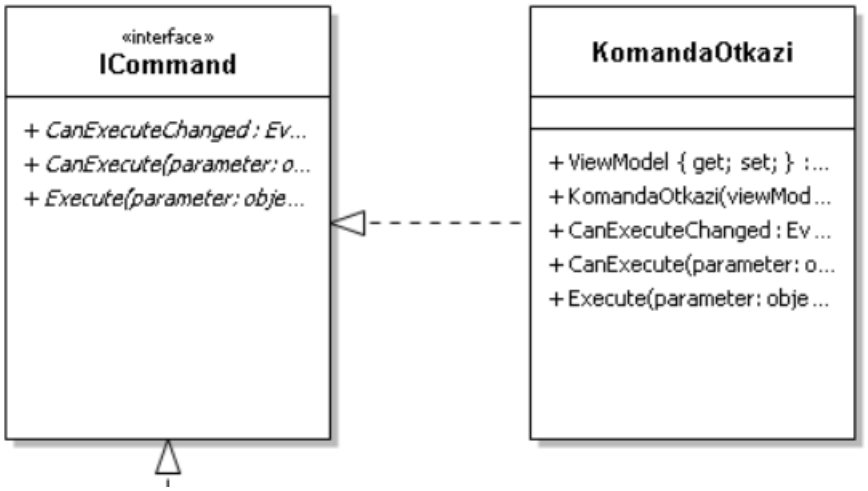
klonirati. Ukratko, govori kako klonirati određenu instancu objekta. U projektu prvenstveno je korišten zbog potrebne funkcionalnosti da se plejlista može klonirati u bazi. Implementiraju ga klase Plejlista i Pesma.

```
// Kloniranje plejliste sa svim njenim pesmama
public Plejlista KlonirajPlejlistu()
{
    Plejlista kopija = new Plejlista()
    {
        IdPlejliste = this.IdPlejliste,
        Autor = this.Autor,
        Naziv = this.Naziv
    };

    kopija.ListaPesama = new List<Pesma>();
    if (this.ListaPesama.Count > 0)
    {
        foreach (var item in this.ListaPesama)
        {
            kopija.ListaPesama.Add(item.KlonirajPesmu());
        }
    }

    return kopija;
}
```

Command

Enkapsulira zahtev za izvršenjem određene operacije u jedan objekat. Umesto da se direktno izvrši određena operacija, kreira se objekat-komanda, koji se potom prosljeđuje na izvršenje.

Ovaj obrazac kao prirodan i on je u tom vidu često korišten implementirajući komande korisničkog interfejsa kroz ICommand interfejs.



Da bi se izbeglo da svaka klasa implementira ICommand za svako dugme na korisničkom interfejsu, korišćen je RelayCommand koji implementira ICommand, a zahteve prosljeđuje kao delegat funkcijama koje je dobio u parametrima konstruktora.

```
DodajPesmuKomanda = new RelayCommand(DodajPesmuUTabelu);
ObrisiPesmuKomanda = new RelayCommand(ObrisiPesmu, SelektovanaPesma);
DodajPlejlistuKomanda = new RelayCommand(SacuvajPlejlistu, ValidacijaSacuvajPlejlistu);

UndoKomanda = new RelayCommand(commandExecutor.Undo, commandExecutor.ValidacijaUndo);
RedoKomanda = new RelayCommand(commandExecutor.Redo, commandExecutor.ValidacijaRedo);

OtkaziKomanda = new RelayCommand(ZatvoriProzor);
```

Primer upotrebe RelayCommand-a u klasi DodajPlejlistuVM.cs

Command šablon se koristio i pri implementaciji Undo/Redo funkcionalnosti nad listom Pesama. Postoji intefejs IUndoRedo koji definiše metode za realizaciju Undo/Redo komandi. Klasa CommandExecutor je zadužena za upravljanje nad komandama i omogućavanje Undo i Redo funkcionalnosti.

Facade

KomunikacijaWCF klasa implementira Facade obrazac kako bi pojednostavila interakciju sa različitim WCF servisima (ILogovanjeServis, IKorisnikServis, IDataServis). Umesto da klijenti direktno komuniciraju sa više različitih servisa, oni koriste ovu klasu kao centralnu tačku interakcije, čime se smanjuje složenost i poboljšava održavanje koda.

Fasada je korišćena u cilju da sakrije od korisnika rukovanje sa 3 tabele koje se nalaze u bazi. To su: Plejliste, Pesme i Korisnici.

```
public class KomunikacijaWCF
{
    private static readonly ILog log = LogManager.GetLogger(typeof(KomunikacijaWCF));

    private ChannelFactory<ILogovanjeServis> logovanjeServisFactory;
    private ChannelFactory<IKorisnikServis> korisnikServisFactory;
    private ChannelFactory<IDataServis> dataServisFactory;

    private ILogovanjeServis logovanjeServisProxy;
    private IKorisnikServis korisnikServisProxy;
    private IDataServis dataServisProxy;

    private Sesija sesija = null;

    public KomunikacijaWCF()
    {
        logovanjeServisFactory = new ChannelFactory<ILogovanjeServis>(typeof(ILogovanjeServis).ToString());
        korisnikServisFactory = new ChannelFactory<IKorisnikServis>(typeof(IKorisnikServis).ToString());
        dataServisFactory = new ChannelFactory<IDataServis>(typeof(IDataServis).ToString());

        logovanjeServisProxy = logovanjeServisFactory.CreateChannel();
        korisnikServisProxy = korisnikServisFactory.CreateChannel();
        dataServisProxy = dataServisFactory.CreateChannel();
    }
```

Observer

U .NET frameworku, INotifyPropertyChanged interfejs omogućava implementaciju ovog obrasca za objekte koji koriste svojstva, posebno u kontekstu MVVM arhitekture u WPF aplikacijama.

Klasa LogovanjeVM implementira interfejs INotifyPropertyChanged kako bi omogućila automatsko obaveštavanje korisničkog interfejsa o promenama svojstava.

```
public class LogovanjeVM : ProzorManagingVM, INotifyPropertyChanged, ICommand
{
    private static readonly ILog log = LogManager.GetLogger(typeof(LogovanjeVM));

    public KorisnikZaLogovanje KorisnikZaLog { get; set; }
    public ICommand LogovanjeKomanda { get; set; }

    public event PropertyChangedEventHandler PropertyChanged;
    public event EventHandler CanExecuteChanged;

    protected void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }

    public string KorisnickoIme
    {
        get { return KorisnikZaLog.KorisnickoIme; }
        set
        {
            KorisnikZaLog.KorisnickoIme = value;
            OnPropertyChanged(nameof(KorisnickoIme));
        }
    }

    private string lozinka;
    public string Lozinka
    {
        get { return lozinka; }
        set
        {
            lozinka = value;
            KorisnikZaLog.Lozinka = value;
            OnPropertyChanged(nameof(Lozinka));
        }
    }
```

Primer upotrebe Observer obrazca u klasi LogovanjeVM.cs