



Univerzitet u Sarajevu
Elektrotehnički fakultet
Sarajevo

Deveti projektni zadatak

Objektno orijentisana analiza i dizajn

Naziv grupe: Spotifive

Članovi grupe: Nadina Miralem 18937

Amina Hromić 19084

Nerma Kadrić 19030

Amila Kukić 19065

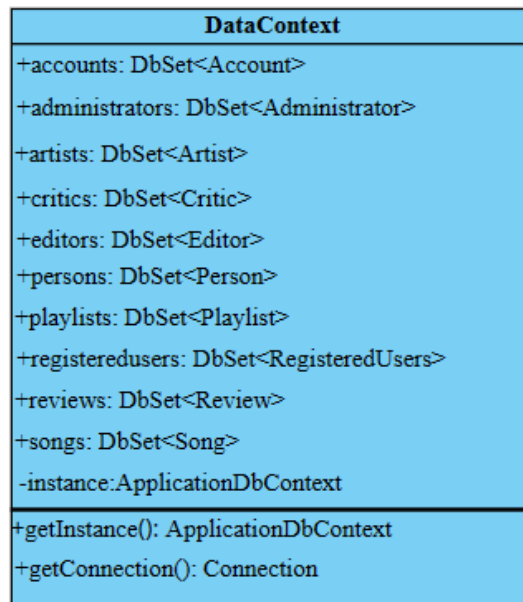
Una Hodžić 19044

Kreacijski paterni

U nastavku slijedi opis kreacijskih paterna koji će biti primijenjeni u sistemu.

Singleton

Singleton patern ograničava instanciranje klase. Singleton klase mogu imati samo jedan objekat u bilo kojem trenutku. S obzirom da naša aplikacija zahtijeva pristup bazi podataka iz različitih dijelova koda, možemo implementirati Singleton klasu koja sadrži logiku za uspostavljanje veze s bazom podataka i izvršavanje upita. Na taj način možemo osigurati da postoji samo jedna instanca veze s bazom podataka u cijeloj aplikaciji



```
private static ApplicationDbContext instance;
private ApplicationDbContext()
{
}
public static ApplicationDbContext getInstance()
{
    if (instance == null)
    {
        instance = new ApplicationDbContext();
    }
    return instance;
}
```

Prototype

Ukoliko bismo Prototype patern koristili u našem sistemu, nad klasom Person, te ukoliko bi se pojavilo više korisnika sa istim imenom i prezimenom (ili bilo kojom drugom osobinom koja je opisana atributom te klase), mogli bismo iskoristiti prvu instancu, klonirati je a zatim samo promijeniti podatke koji se razlikuju. Zbog toga dodajemo interface IPrototype koji implementira klasa Person, a koji ima metodu Clone.



```
public class Person{

    [Key] public int ID { get; set; }
    public string Name { get; set; }
    public string Surname { get; set; }
    public DateTime DateOfBirth { get; set; }
    public Gender Gender { get; set; }
    public Account Account { get; set; }
    public Person() { }

    public virtual Person Clone(){
        Person clone=(Person)this.MemberwiseClone();
        clone.Name=String.Copy(Name);
        clone.Surname=String.Copy(Surname);
        clone.Gender=Gender;
        return clone;
    }
}
```

Kao što možemo vidjeti, gore navedena klasa ima nekoliko atributa koji su zajednički za podklase RegisteredUser, Critic, Editor i Administrator, te apstraktnu metodu koja će biti implementirana od strane navedenih podklasa.

Kao primjer, u klasi RegisteredUser možemo vidjeti implementaciju spomenute apstraktne metode. Metoda Clone stvara površinsku kopiju trenutnog objekta.

```
public class RegisteredUser: Person{
    public RegisteredUser() { }
    [ForeignKey("Song")] public int SongID { get; set; }
    public Song Song { get; set; }

    public override Person Clone(){
        return (RegisteredUser) this.MemberwiseClone();
    }
}
```

```
}
```

```
}
```

U nastavku slijedi opis ostalih kreacijskih paterna, te načina na koje bi se oni mogli implementirati u našem sistemu.

Builder

U našem sistemu ovaj patern bismo mogli implementirati tako što bismo dodali interface `ISongBuilder` koji sadrži sljedeće metode: `getPopSongs`, `getRockSongs`, `getJazzSongs`, `getFolkSongs`, itd. Ove metode bi vraćale liste pjesama, te bi uzimale u obzir različite faktore (naziv izvođača, naziv albuma, datum izdavanja...). Također bismo dodali klase `PopBuilder`, `RockBuilder`, `JazzBuilder`, `FolkBuilder` i slične, koje bi imale kao atribut listu svih odgovarajućih pjesama koje se nalaze u sistemu. Ove metode bi mogle, po uzoru na već preslušane pjesme (ili neke druge faktore), korisniku preporučivati one pjesme koje su najbližnije njegovom muzičkom ukusu.

Factory method

Ovaj patern bi u našem sistemu mogao biti iskorišten prilikom pristupa žanrovima pjesama. Dodali bismo interface `ISong`, te naslijeđene klase `Pop`, `Rock`, `Jazz`, `Folk` i slične. Tako bismo izbjegli korištenje različitih metoda za kreiranje različitih žanrova.

Abstract Factory

Abstract Factory patern bi nam omogućio kreiranje više familija produkata (prema žanrovima pjesama). Mogli bismo kreirati više interface-a (npr. `IPop`, `IRock`, `IJazz`, `IFolk`...), a zatim bismo kreirali i posebne klase koje ih implementiraju.