Napredni WPF #3: *Model View ViewModel Design Pattern*

Nakon što su predstavljeni osnovni koncepti i ideje **Model View ViewModel (MVVM)** *Design Pattern*-a (projektnog obrasca), implementacija se dalje proširuje dodavanjem koncepta validacije i radom sa WPF događajima. Kao što je već poznato, izuzeci (*Exception*) predstavljaju indikaciju grešaka u izvršavanju programa.

Posmatrajući domen interakcije čoveka i računara, izuzeci igraju jako važnu ulogu, jer je potrebo korisniku na adekvatan način skrenuti pažnju da je napravio grešku i pomoći mu da je otkloni. Validacija se u okviru **MVVM** standarda implementira drugačije i ono što je promena je to da se validacija dodaje u samoj klasi koja modeluje objekte koji se koriste u aplikaciji.

Validacija se može izraziti na više načina:

- „bacanjem“ *Exception*-a nad metodom set nekog property-ja
- implementacijom interfejsa IDataErrorInfo
- implementacijom interfejsa InotifyDataErrorInfo (omogućava asinhronu validaciju – više poruka greške za isto polje, uvedeno sa .NET-om 4.5)
- WPF pravila validacije

Dobra praksa za **MVVM** implementaciju validacije je ubacivanje podrške validacije u neku zajedničku baznu klasu (kasnije **ValidationBase**), što omogućava da se ona nasledi iz različitih scenarija.

Jedan od glavnih aspekata interakcije kada je u pitanju validacija na formama za unos podataka je ispis poruka o napravljenim greškama. U okviru MVVM aplikacije se formira posebna klasa, koja će modelovati poruke greške, imena **ValidationErrors**.

U okviru nje, definiše se **Dictionary** koji sadrži par ključ-vrednost tipa *string*. Ključ će biti ime svojstva za čiji unos se ispisuje poruka greške, dok će vrednost biti sama poruka greške za dato svojstvo. U okviru ove klase definisaće se property koji je indikacija da li je objekat za koji se proverava validnost (odnosno da li ima grešaka ili ne). Ova implementacija je navedena u listingu koda ispod.

```
public bool IsValid
{
     get
     {
         return this.validationErrors.Count < 1;
     }
}
```

Listing 1. Property koji prikazuje da li je objekat za koji se unose vrednosti property-ja validan

Pored toga, definiše se način na koji se klasa može koristiti kao svoj property, kako bi se moglo pristupiti samom **Dictionary**-ju. Zavisno od prosleđenog ključa, pomoću ovog property-ja će se, upotrebom *get* metode pročitati koja je poruka greške za dati property klase za koju je napravljena greška (naravno, ako sama poruka greške postoji), dok se sa *set* metodom dodeljuje poruka o grešci za dato polje.

Zavisno od toga da li već postoji definisana poruka greške, ako je greška ispravljena, par ključ-vrednost se briše, dok u slučaju da je u pitanju nova poruka o grešci, sama poruka se dodeljuje na poziciju gde je ključ ime datog property-ja klase čija se polja validiraju. U suprotnom da greška za dato polje nije definisana, a poruka se definiše, biće dodat novi par ključ-vrednost sa imenom polja i porukom o grešci.

Na kraju potrebno je da se notifikuju svi property-ji koji zavise od ove vrednosti pomoću implementacije **INotifyPropertyChanged** interfejsa. Implementacija ove funkcionalnosti je navedena u listingu koda ispod.

```
public string this[string fieldName]
{
    get
    {
         return this.validationErrors.ContainsKey(fieldName) ?
                this.validationErrors[fieldName] : string.Empty;
    }

    set
    {
          if (this.validationErrors.ContainsKey(fieldName))
          {
               if (string.IsNullOrWhiteSpace(value))
               {
                   this.validationErrors.Remove(fieldName);
               }
               else
               {
                   this.validationErrors[fieldName] = value;
               }
          }
          else
          {
                if (!string.IsNullOrWhiteSpace(value))
                {
                    this.validationErrors.Add(fieldName, value);
                }
          }
          this.OnPropertyChanged("IsValid");
    }
}
```

Listing 2. Property za pristup i definisanje vrednosti listi grešaka

Uz **ValidationErrors** klasu, koja modeluje poruke o greškama, definiše se i **ValidationBase** klasa, bazna klasa, koju će naslediti klasa čija polja se validiraju. Sadrži svojstvo tipa klase **ValidationErrors**, kao i svojstvo koji označava da li su polja objekta validno popunjena. Konstruktor klase instancira **ValidationErrors** objekat, kako bi postojala lista grešaka.

Ključan deo je metoda *ValidateSelf*, koja je abstraktna i sa modifikatorom pristupa *protected*. Nju će morati da redefiniše klasa koja nasleđuje **ValidationBase** i čija se polja proveravaju sa konkretnom implementacijom provere. Tu je i metoda *Validate*, čiji je zadatak da izvrši samu funkcionalnost validacije - isprazni se lista grešaka, poziva se *ValidateSelf*, onda se proverava da li je objekat validan (nema definisane poruke o greškama) i na kraju se notifikuje promena property-ja. Implementacija ove funkcionalnosti je navedena u listingu koda ispod.

```
protected abstract void ValidateSelf();

public void Validate()
{
     this.ValidationErrors.Clear();
     this.ValidateSelf();
     this.IsValid = this.ValidationErrors.IsValid;
     this.OnPropertyChanged("IsValid");
     this.OnPropertyChanged("ValidationErrors");
```

```
}
```

Listing 3. Metode ValidationBase klase

U okviru klase čija će se polja validirati, nasleđuje se klasa **ValidationBase** i redefiniše se metoda *ValidateSelf*, koja će proveriti validnost svih polja klase, gde se definišu poruke greške za data polja. Implementacija ove funkcionalnosti je navedena u listingu koda ispod.

```
protected override void ValidateSelf()
{
     if(string.IsNullOrWhiteSpace(this.title))
     {
         this.ValidationErrors["Title"] = "Title is required.";
     }
     if (string.IsNullOrWhiteSpace(this.description))
     {
         this.ValidationErrors["Description"] = "Description cannot be empty.";
     }
}
```

Listing 4. Redefinisana metoda ValidateSelf

U **ViewModel** klasi se definiše property objekat koji će sadržati informacije o unetim podacima sa forme (interfejsa) i koji se validira. Prilikom kreiranja objekta on se validira kroz poziv *Validate* metode. Rezultat metode je postavljanje vrednosti svojstva ***IsValid***, koje pokazuje da li ima poruka o greškama i sam ishod validacije, što (ne)dopušta akciju kreiranja. Implementacija ove funkcionalnosti je navedena u listingu koda ispod.

```
public void OnAdd()
{
     CurrentNote.Validate();
     if(CurrentNote.IsValid)
     {
         Notes.Add(new Note()
         {
              Title = CurrentNote.Title,
              Description = CurrentNote.Description
         });
     }
}
```

Listing 5. Korišćenje validacije nad konkretnim objektom

Ispis poruke o greškama se takođe povezuje sa interfejsom putem *DataBinding*-a, kao sadržaj **TextBlock** kontrole. Poruke greške se takođe sadrže u property objektu u koji se unose vrednosti polja. XAML kod koji to opisuje je naveden u listingu ispod.

```
<TextBlock Text="{Binding CurrentNote.ValidationErrors[Title]}"
           Foreground="{StaticResource UITertiaryColor}"
           FontSize="16"/>
```

Listing 6. Ispis poruke greške sa strane interfejsa.

Kako se u **MVVM** rešenjima implementacija ne nalazi u *xaml.cs* datotekama, realizacija **WPF** događaja se dobija pomoću *NuGet* paketa **WPF.Interactivity**. Može se uključiti u projekat pomoću **NuGet Package Manager**-a, ili preko komandne linije **Visual Studio**-ja pomoću instrukcije *Install-Package System.Windows.Interactivity.WPF*. Da bi se paket primenio na XAML kod, navodi se kao imenski prostor u tagu prozora ili UserControl-e na način navedem u listingu ispod.

```
xmlns:i="clr-
namespace:System.Windows.Interactivity;assembly=System.Windows.Interactivity"
```

Listing 7. Navođenje **Interactivity** imenskog prostora u tagu prozora ili **UserControl**-e

Način korišćenja ovog **NuGet** paketa je naveden u **Listingu 8**. Nad kontrolom se navodi tačno ime WPF događaja koji treba da se poveže sa datom komandom.

```
<Kontrola gde se poziva događaj>
       <i:Interaction.Triggers>
              <i:EventTrigger EventName="(TAČNO ime WPF XAML događaja)" >
                     <i:InvokeCommandAction
                     Command="{Binding (ime komande)}"/>
              </i:EventTrigger>
       </i:Interaction.Triggers>
</Kontrola gde se poziva događaj>
```

Listing 8. Korišćenje **Interactivity** paketa

Kao u slučaju generalne implementacije komandi i ovde se može proslediti parametar, a parametar takođe može biti i neka kontrola iz **XAML** hijerarhije aplikacije (prosleđuje se vrednost dodeljena toj kontroli za svojstvo *Name*). Primena ovoga je u **Listingu 9**.

```
<i:InvokeCommandAction Command="{Binding (ime komande)}"
CommandParameter="{Binding ElementName=(ime neke instance kontrole)}"/>
```

Listing 9. Dodela parametra komandi

Kada je potrebno omogućiti komunikaciju, odnosno razmenu podataka između **ViewModel**-a, koristi se varijanta projektnog obrasca **Publisher-Subscriber**, koja se zove **MVVM Light Messenger**. On se takođe se dodaje u projekat kao **NuGet** paket. Funkcija **Register** će prihvatiti slanje tipa podataka, koje se iz drugog ViewModel-a šalje pomoću funkcije **Send**. Primeri upotrebe ovog paketa su navedeni u listingu 10.

```
//Subscribe
Messenger.Default.Register<(tip podatka koji se šalje)>(metoda kojom se reaguje na
prijem podatka);

// Broadcast
Messenger.Default.Send<(tip podatka koji se šalje)>(podatak datog tipa koji se
šalje)
```

Listing 10. Upotreba paketa **MVVM Light Messenger**-a

Prilikom dizajniranja korisničkog interfejsa, dobra praksa je primeniti tzv. "pravilo 60-30-10". Ono se manifestuje tako što se prilikom dizajniranja interfejsa bazna, neutralna boja najviše javlja (60% interfejsa), dok se primarna i sekundarna („akcenat boja“) koriste u 30 i 10% interfejsa. Naravno, ovo pravilo se uglavnom koristi kao smernica. Može se upotrebiti i u kontra smeru, tako da upadljivija boja bude zastupljena 60%, a ostale 30 i 10.

Kada se odrede boje, one se mogu dodeliti kao resursi aplikacije u okviru datoteke **App.xaml**. Odatle se onda mogu koristiti u celoj aplikaciji umesto ponovnog navođenja boja. Kreiranje resursa boja je dato u **Listingu 11**.

```
<Application x:Class="MVVM3.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:MVVM3"
             StartupUri="MainWindow.xaml">
    <Application.Resources>
        <SolidColorBrush x:Key="UIPrimaryColor"
                         Color="#1C1D31"/>
```

```
        <SolidColorBrush x:Key="UISecondaryColor"
                         Color="#B08D00"/>
        <SolidColorBrush x:Key="UITertiaryColor"
                         Color="#5C4400"/>
        <SolidColorBrush x:Key="UIAccentColor"
                         Color="#AAA9C3"/>
        <SolidColorBrush x:Key="UIWhiteColor"
                         Color="White"/>
    </Application.Resources>
</Application>
```

Listing 11. Kreiranje resursa boja u datoteci **App.xaml**

Sve ove funkcionalnosti su iskorišćene da se kreira primer za devetu nedelju vežbi koji sadrži mogućnost dodavanja novog objekta tipa Note u prikaz beleški, sa validacijom unosa njegovih atributa prilikom dodavanja objekta. Implementacija je realizovana primenom MVVM projektnog obrasca.