Interakcija čovek - računar

Vežbe 3

Fakultet tehničkih nauka
Univerzitet u Novom Sadu

Uvod u WPF

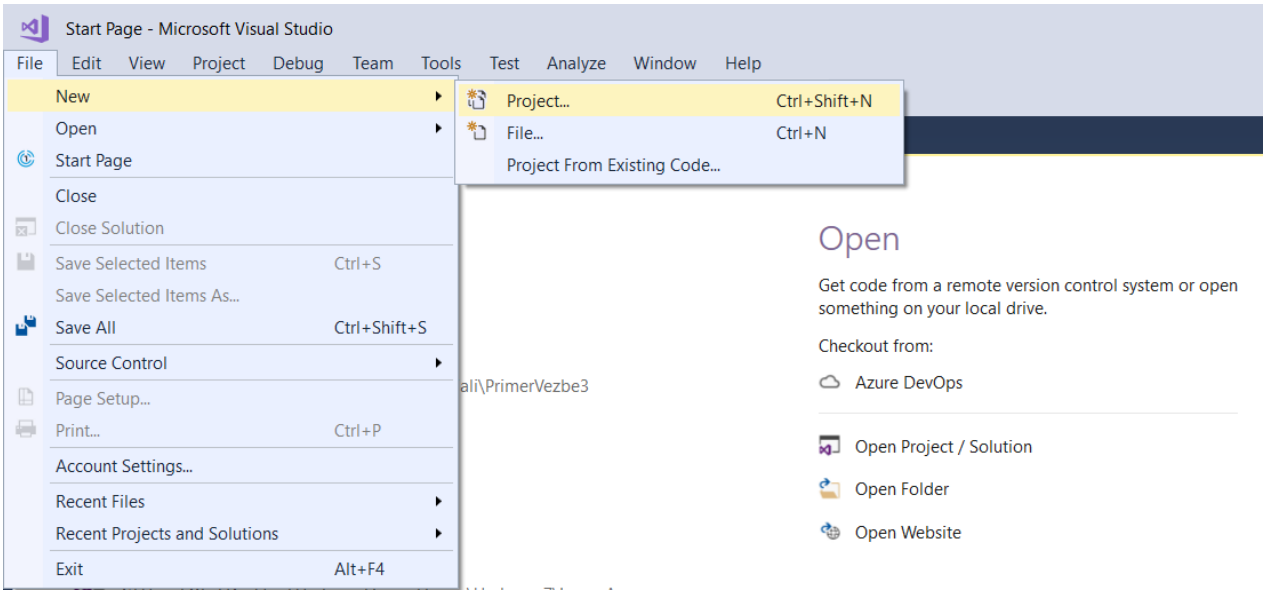
Windows Presentation Foundation (WPF) je GUI (*Graphical User Interface*) *framework* za kreiranje Windows desktop aplikacija.

GUI *framework* omogućava da kreiramo aplikacije uz pomoć velikog opsega GUI elemenata kao što su labele (eng. *labels*), polja za unos teksta (eng. *textboxes*). Bez GUI *framework*-a, morali bi da crtamo ove elemente ručno i da upravljamo sa svim korisničkim interakcijama, kao što je unos teksta ili pomeranje miša.

WPF sadrži set *application-development* alata (eng. *features*): *Application Markup Language* (XAML), kontrole, *Data binding*, *layouts*, 2D i 3D grafika, animacije, stilovi, *media* i slično.

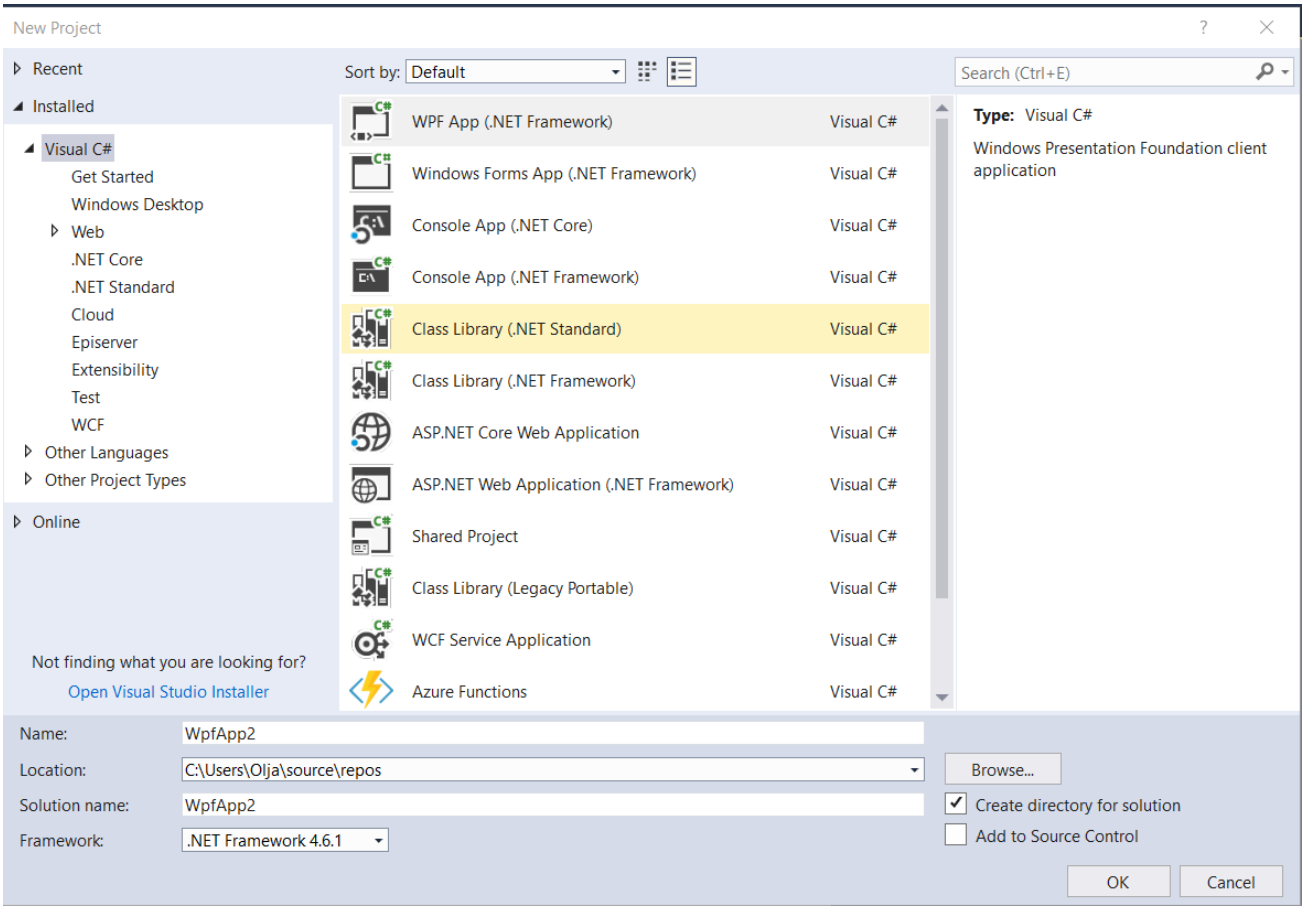
Kreiranje WPF projekta

Jedan od načina kreiranja novog WPF projekta je odabirom File/New/Project. (Slika 1)



Slika 1. Kreiranje novog projekta

Potrebno je odabrati WPF App (.NET Framework) i popuniti ***Name*** i ***Location*** polja. (Slika 2)



Slika 2. Odabir WPF projekta

Markup i code-behind

WPF omogućava da se razvija aplikacija koristeći *markup* i *code-behind*.

XAML *markup* se obično koristi za implementaciju izgleda aplikacije dok se *code behind* koristi za implementaciju ponašanja tih komponenti.

Ovo razdvajanje na izgled i ponašanje povlači neke od prednosti kao što su:

1. Cena razvoja i održavanja aplikacije je smanjena jer je znatno lakše upravljati sa aplikacijom kod koje izgled i ponašanje nisu usko vezani (eng. *tightly coupled*).
2. Razvoj aplikacije je ubrzan i efikasan jer dizajneri mogu implementirati izgled aplikacije nezavisno i paralelno od programera koji implementiraju ponašanje aplikacije.
3. Globalizacija i lokalizacija za WPF aplikacije je pojednostavljena.

Na slici 3 se može videti otvoren prazan WPF projekat i brojevi kojima su označeni najbitniji segmenti za razvoj WPF aplikacije.

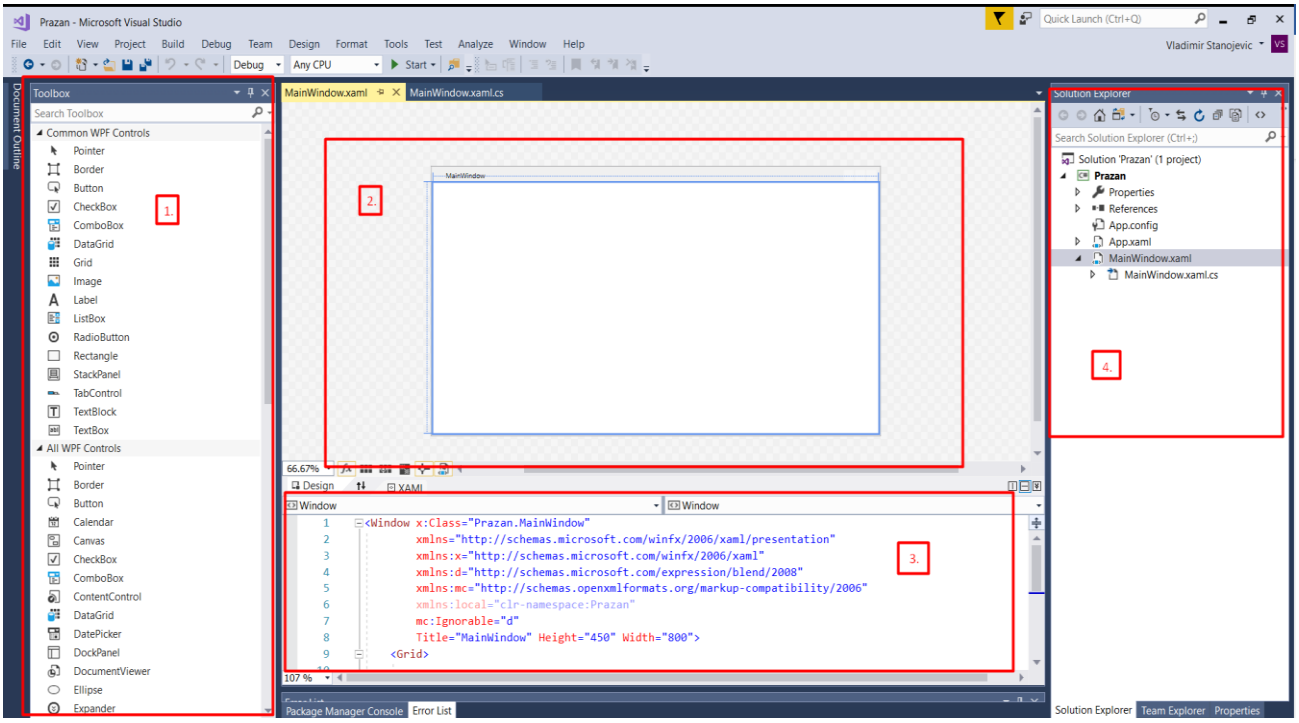
Sa (1) označen je *Toolbox* u kom se nalaze sve WPF kontrole koje se mogu iskoristiti za kreiranje grafičkog korisničkog interfejsa. Sa brojem 2 označen je dizajner u kom se može pratiti izgled aplikacije u toku njenog kreiranja.

Dodavanjem komponente iz *Toolbox*-a na prozor može se vršiti prevlačenjem (*drag&drop*).

Sa (3) označen je XAML kod (*markup*). Kada se kontrola prevuče na prozor (sekcija 2), u XAML kodu (sekcija 3) pojaviće se odgovarajući tag. Takođe, dodavanjem taga u XAML kod (sekcija 3), promene će se manifestovati u prozoru (sekcija 2).

Sekcija 4 predstavlja *Solution Explorer* gde se može vršiti pregled fajlova u projektu.

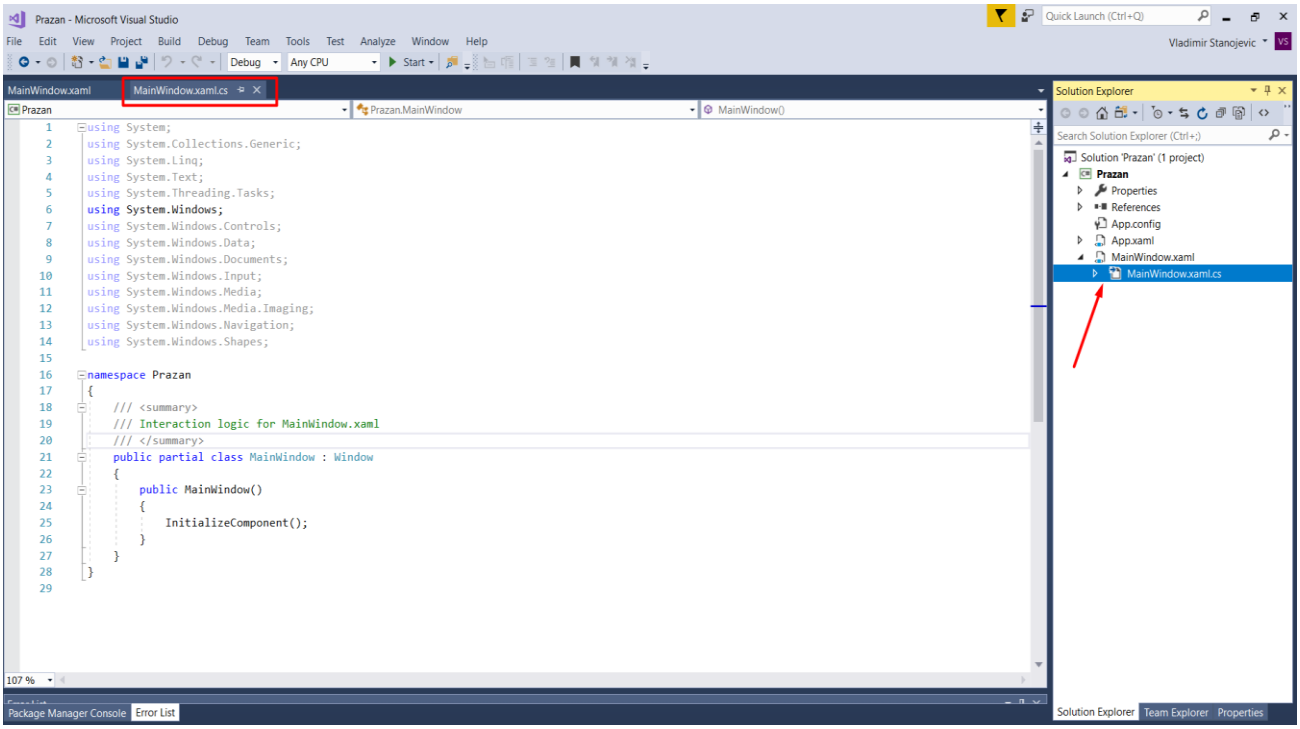
Na slici 3 možemo primetiti da je otvoren MainWindow.xaml fajl što zapravo predstavlja XAML kod ovog projekta.



Slika 3. Okruženje i XAML kod

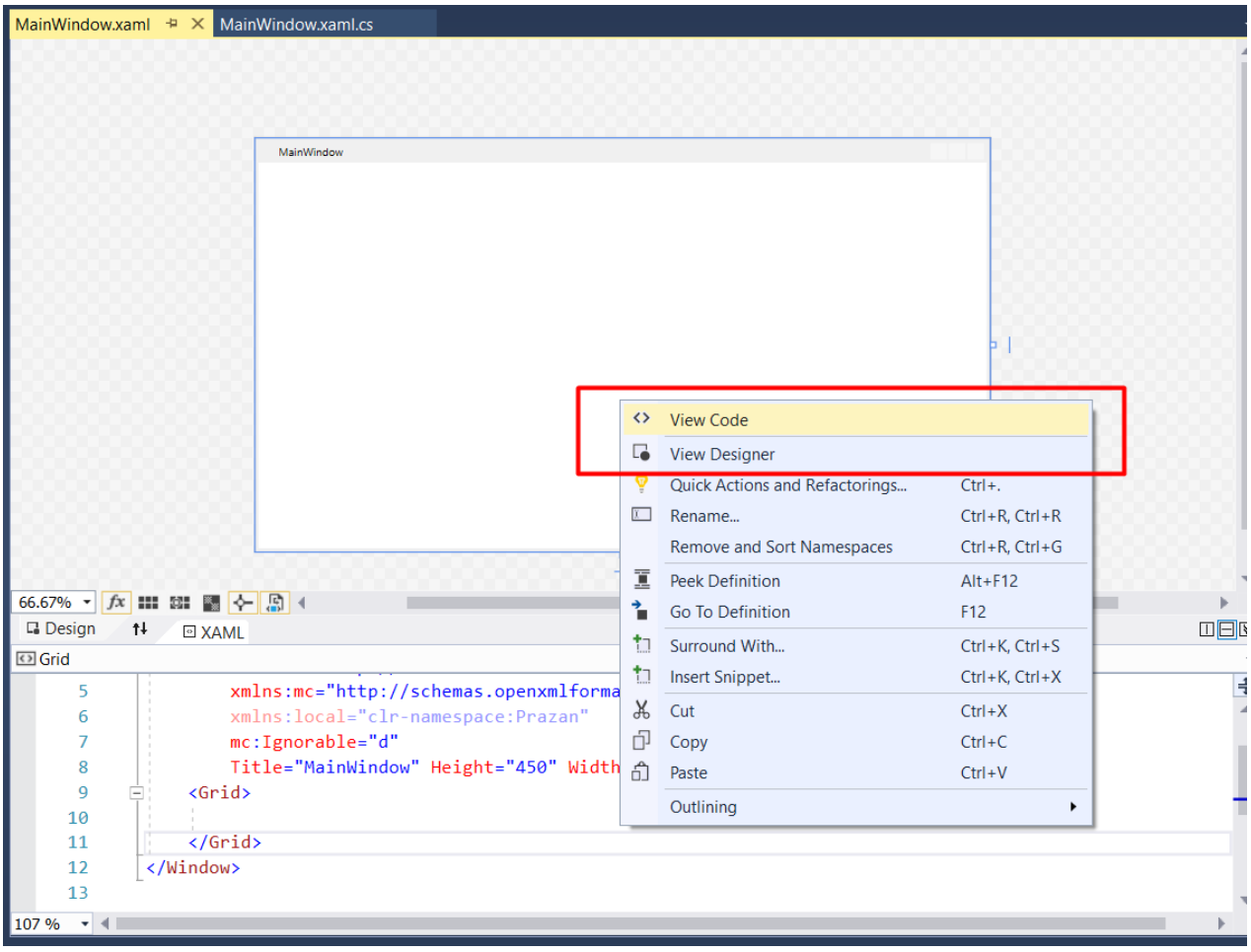
Ako se u *Solution Exploreru* (sekcija 4) otvori MainWindow.xaml.cs dobija se *code behind. (Slika 4)*

U nastavu će biti objašnjena razlika između XAML koda i *code behind*.



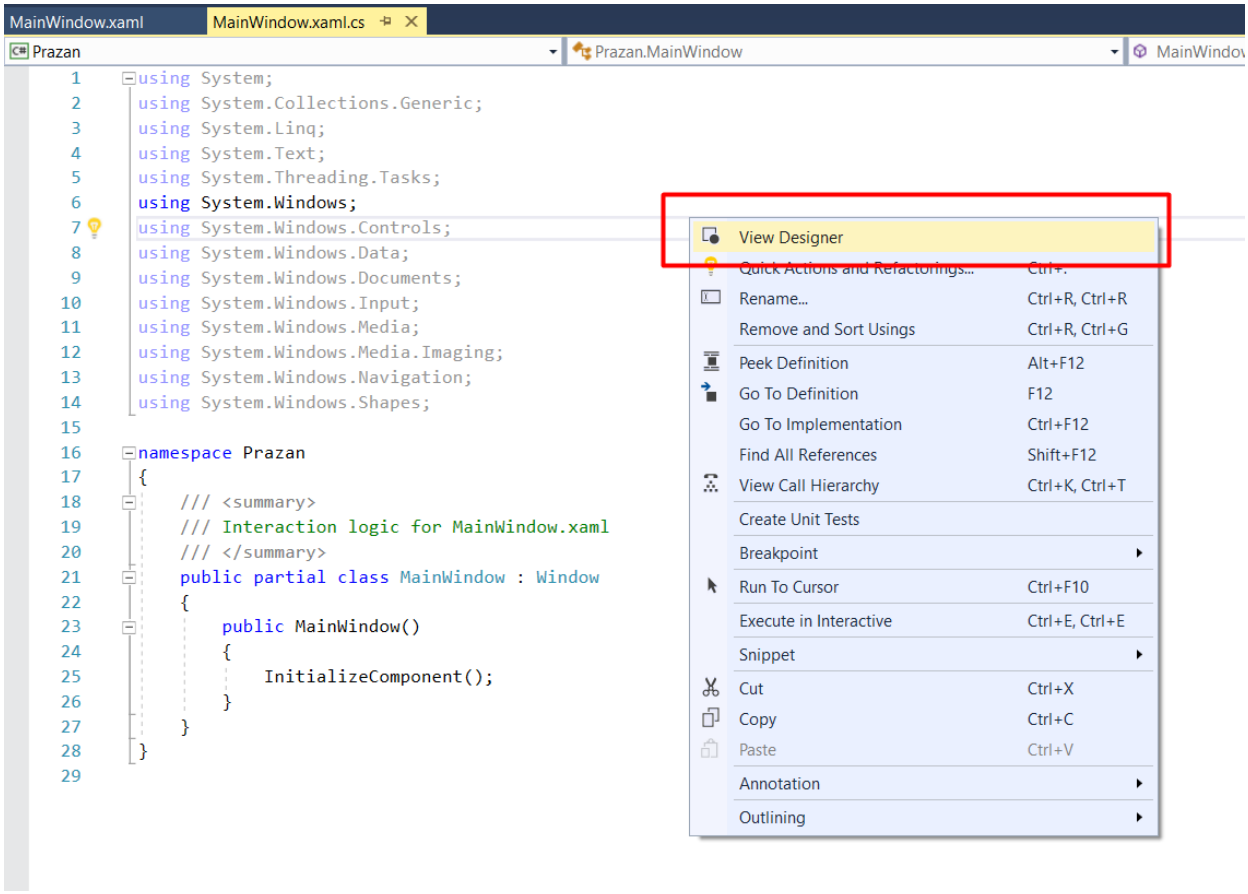
Slika 4. Code behind

Osim odabirom iz *Solution Explorera*, jednostavan način da se iz XAML koda pređe na .cs fajl je desni klik i odabir *View Code* opcije iz padajućeg menija. (Slika 5)



Slika 5. Prelazak sa XAML koda na Code behind

Prelazak iz .cs fajla na dizajner (XAML) moguć je pomoću desni klik pa na *View Designer* opciju. (Slika 6)



Slika 6. Prelazak sa Code behind na XAML kod

Markup (XAML)

XAML je XML-bazirani *markup* jezik koji omogućava implementiranje izgleda aplikacije (eng. *application's appearance*). Koristi se za kreiranje prozora, jednostavnih dijaloga i kontrola a navedeno dalje možemo popunjavati sa novim kontrolama, oblicima i grafičkim elementima.

Primer sa slike 7 prikazuje XAML kod jednostavnog prozora (tag <*Window*>) koji sadrži samo jedno dugme. (tag <*Button*>)

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    Title="Window with Button"
    Width="250" Height="100">

  <!-- Add button to window -->
  <Button Name="button">Click Me!</Button>

</Window>
```

Slika 7. Jednostavan prozor sa jednim dugmetom

Svaki element može da sadrži određene atribute, slično kao u XML-u. Na slici 7 se vidi da tag *<Window>* sadrži *Title* atribut koji je zapravo *title-bar* tekst.

U primeru sa slike 8 je prikazano kako se uz pomoć *FontWeight property* može postaviti boldovan tekst sa sadržajem *"A button"*.

```
<Button FontWeight="Bold" Content="A button" />
```

Slika 8. Atributi dugmeta - inline

Još jedan način definisanja atributa je prikazan na slici 9. Ovaj način prikazuje atribute kao *child* tagove glavnog taga, a u ovom slučaju to je *<Button>.*

```
<Button>
    <Button.FontWeight>Bold</Button.FontWeight>
    <Button.Content>A button</Button.Content>
</Button>
```

Slika 9. Atributi dugmeta - child komponente

S obzirom da je XAML XML-bazirani jezik, UI koji se kreira predstavlja hijerarhiju ugnježdenih elementa poznatih pod nazivom *element tree.*

Dat je primer gde je u okviru jednog dugmeta kreirano više tekst blokova koristeći više *<TextBlock>* tagova u *<Button>* tagu. (Slika 10)

Tekst u tekst blokovima je obojen različitim bojama koristeći atribut *Foreground*.

```
<Button>
    <Button.FontWeight>Bold</Button.FontWeight>
    <Button.Content>
        <WrapPanel>
            <TextBlock Foreground="Blue">Multi</TextBlock>
            <TextBlock Foreground="Red">Color</TextBlock>
            <TextBlock>Button</TextBlock>
        </WrapPanel>
    </Button.Content>
</Button>
```

Slika 10. Složeni oblik dugmeta - prvi način

Content property dozvoljava samo jedan *child* element pa je bilo potrebno obuhvatiti *child* komeponente u *WrapPanel*. Paneli, kao što je *WrapPanel*, igraju veoma važnu ulogu u WPF i predstavljaju kontejnere za druge kontrole. O panelima će biti više govora u daljem tekstu.

Isti rezultat se može postići na sledeći način. Ovaj način je jednostavniji i češće je u upotrebi. (Slika 11)

```
<Button FontWeight="Bold">
    <WrapPanel>
        <TextBlock Foreground="Blue">Multi</TextBlock>
        <TextBlock Foreground="Red">Color</TextBlock>
        <TextBlock>Button</TextBlock>
    </WrapPanel>
</Button>
```

Slika 11. Složeni oblik dugmeta - drugi način

Code-behind

Osnovna odlika svake aplikacije je da implementira funkcionalnosti, uključujući obradu događaja (eng. *handling events*) kao što je korisnički klik. U WPF-u, ponašanje nakon okidanja dođaja se implementira u kodu koji je povezan sa *markup*-om (XAML). Ovaj tip koda se naziva *code-behind*. Primer na slici 12 prikazuje proširen *markup* kod (XAML) a slika 13 *code-behind* koji odgovara tom kodu.

Sličan primer videli smo pri kreiranju praznog WPF projekta.

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.AWindow"
    Title="Window with Button"
    Width="250" Height="100">

  <!-- Add button to window -->
  <Button Name="button" Click="button_Click">Click Me!</Button>

</Window>
```

Slika 12. Proširen markup kod sa event handler-om

```
using System.Windows; // Window, RoutedEventArgs, MessageBox

namespace SDKSample
{
    public partial class AWindow : Window
    {
        public AWindow()
        {
            // InitializeComponent call is required to merge the UI
            // that is defined in markup with this class, including
            // setting properties and registering event handlers
            InitializeComponent();
        }

        void button_Click(object sender, RoutedEventArgs e)
        {
            // Show message box when button is clicked.
            MessageBox.Show("Hello, Windows Presentation Foundation!");
        }
    }
}
```

Slika 13. Code behind sa event handler-om

InitializeComponent metoda se poziva iz *code-behind-a*, iz konstruktora klase. Ova metoda zadužena je da spoji (eng. *merge*) UI koji se definisan u *markup* kodu sa *code-behind* klasom.

Napomena : *InitializeComponent* se generiše za vas kada se kreira aplikacija i ne treba je implementirati ručno.

Na slici 13 se takođe može videti kako je implementiran *event handler* za *Click* događaj. Ovaj *Click* događaj je zakačen na dugme sa slike 12. Kada se klikne, *event handler* prikazuje *message box* pozivajući System.Windows.MessageBox.Show metodu.

O događajima i obradi događaja biće više govora u nastavku.

Događaji i obrada događaja (eng. Events and handling events)

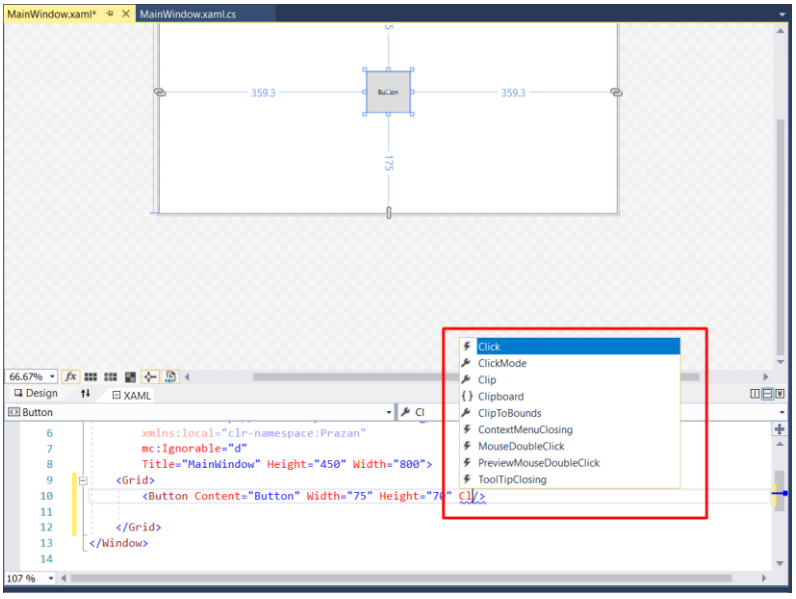
Većina UI framework-a su *event driven*. Sve kontrole, uključujući *Window* kontrolu, izlažu događaje na koje se može odreagovati. Možete se prijaviti (eng. *subscribe*) na ove događaje, što znači da će vaša aplikacija biti obaveštena kada se ovaj događaj desi i tada je moguće odreagovati na određeni način. Reakcija na događaj se zapravo definiše u *event handler*-u.

Postoji više tipova događaja ali oni koji se najčešće koriste su vezani za interakciju sa mišem ili tastaturom.

Na većini kontrola mogu se definisati događaji kao što su: *KeyDown*, *KeyUp*, *MouseDown*, *MouseEnter*, *MouseLeave*, *MouseUp*.

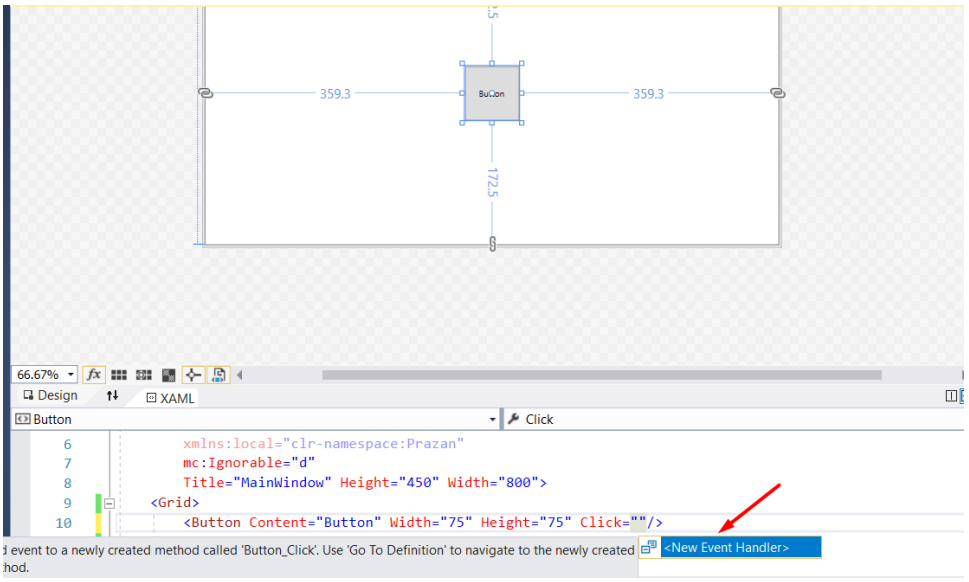
Postoji više načina za dodavanje događaja.

Moguće je ručno dodavanje događaja u okviru kontrole za koju se želi zakačiti. (Slika 14)



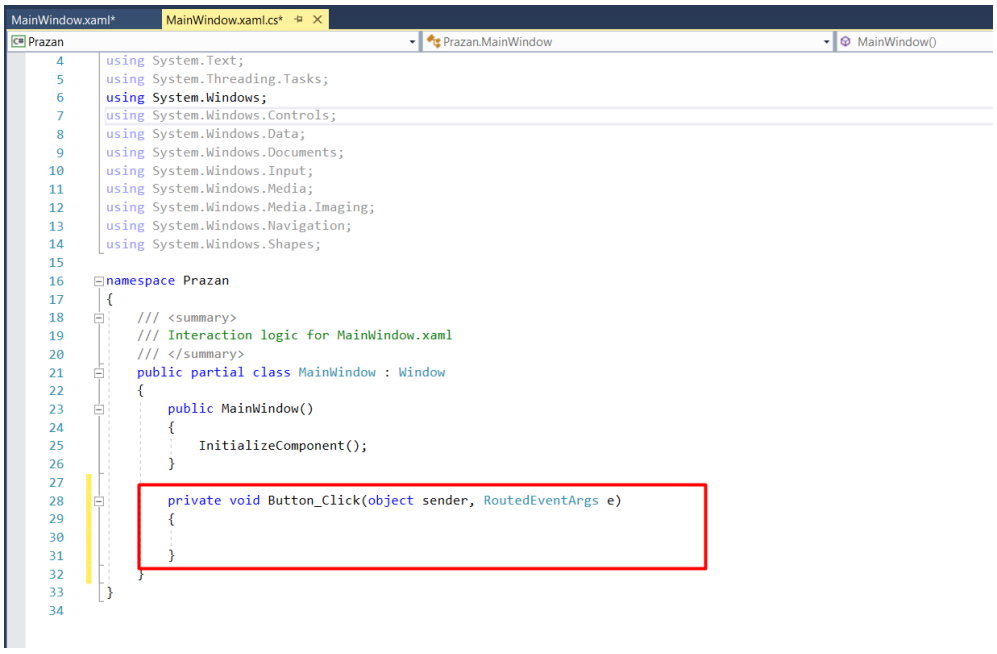
Slika 14. Dodavanje događaja

Odabirom *Click* događaja, pojaviće se opcija ***New Event Handler*** na čiji odabir će okruženje kreirati za vas prazan *event handler* u *code behind*. (Slika 15)



Slika 15. Automatsko reiranje event handler-a za događaj

Kreirana metoda (*event handler*) u *code behind* prikazana je na slici 16.



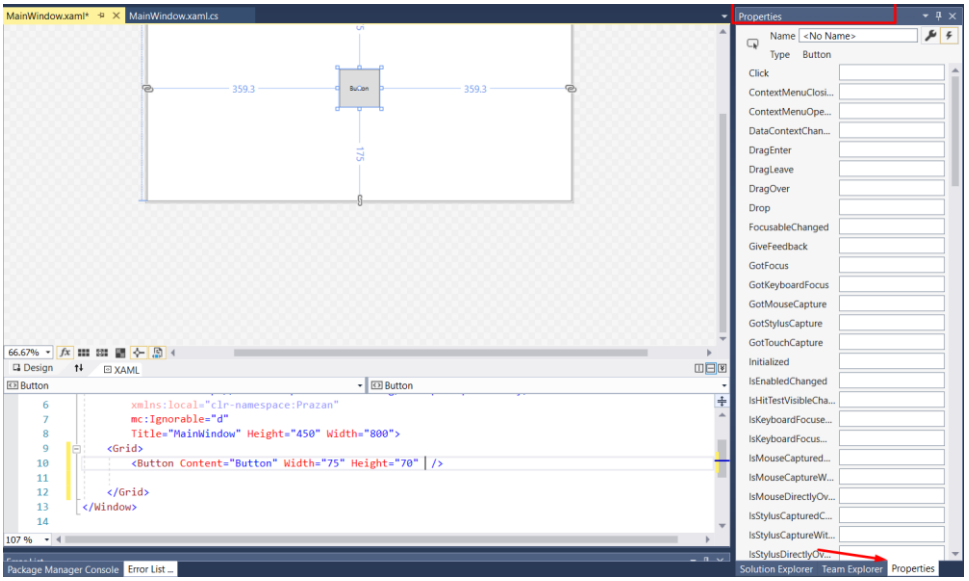
Slika 16. Automatski kreiran event handler u code behind-u

Nakon kreiranja *event handler*-a, dugme u code behind-u izgleda kao na slici 17.

```
        xmlns:local="clr-namespace:Prazan"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <Button Content="Button" Width="75" Height="75" Click="Button_Click"/>
```

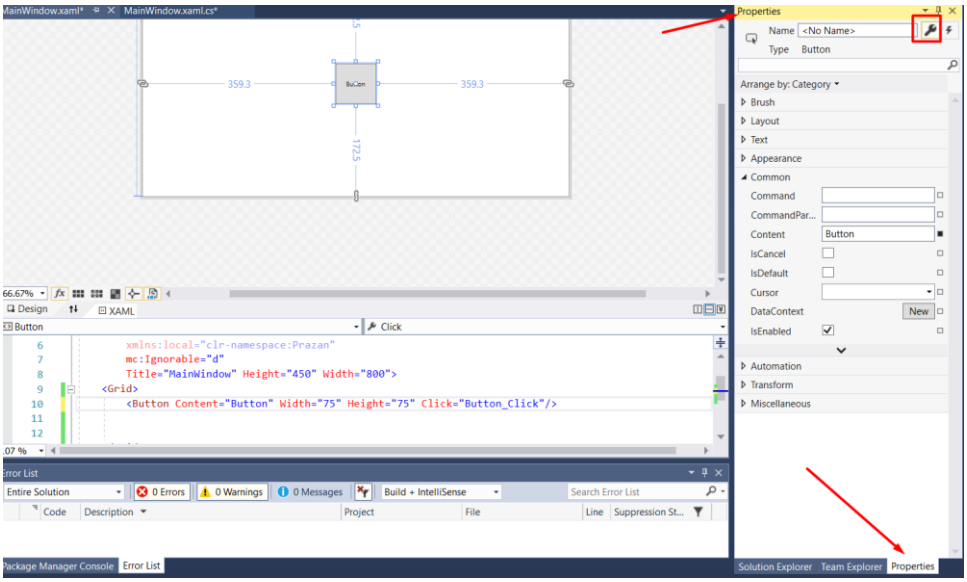
Slika 17. Izgled dugmeta u XAML kodu nakon dodavanja event handler-a

Drugi način je preko *Properties tool*-a.(Slika 18) Na ovom mestu mogu se pronaći svi događaji koji mogu biti zakačeni za selektovanu kontrolu.



Slika 18. Dodavanje event-a i event handler-a preko Properties tool-a

Odabirom naznačene opcije u *Properties tool*-u mogu se pregledati i svi atributi koji se mogu dodati na kontrolu. (Slika 19)



Slika 19. Dodavanje atributa preko Properties tool-a

Paneli (eng. Panels)

Panel, kao jedna od najvažnijih kontrola, predstavlja kontejner za druge kontrole i upravlja sa rasporedom istih na prozoru. S obzirom da *Window* kontrola može da ima samo jednu *child* kontrolu, obično se koristi panel da podeli prostor na regije gde svaka regija može ponovo biti panel.

U nastavku će biti navedeni i objašnjeni neki od panela koji se koriste u WPF-u.

Wrap panel

WrapPanel pozicionira svaku od svojih *child* kontrola jednu do druge, horizontalno (*default* varijanta) ili vertikalno, dokle god ima dovoljno prostora. Kada ne postoji dovoljno prostora, *child* elementi se *wrap*-uju u novi red i nastavlja se njihovoređanje.

WrapPanel treba koristiti kada imamo listu kontrola koje želimo da rasporedimo vertikalno ili horizontalno.

Kada *WrapPanel* koristi horizontalnu orijentaciju, *child* kontrole će imati istu visinu, zasnovanu na visini najvišeg elementa. Kada je *WrapPanel* vertikalno orijentisan, *child* kontrole će imati istu širinu kao najširi element.

Na slici 20 je dat primer *WrapLayout*-a u okviru kog se nalazi lista dugmića. Treba primetiti da *'Test button 4'* i *'Test button 10'* za atribut *Height* imaju vrednost 40. Ova promena prouzrokuje da sva dugmad iz istog reda dobijaju istu visinu, visinu 40.

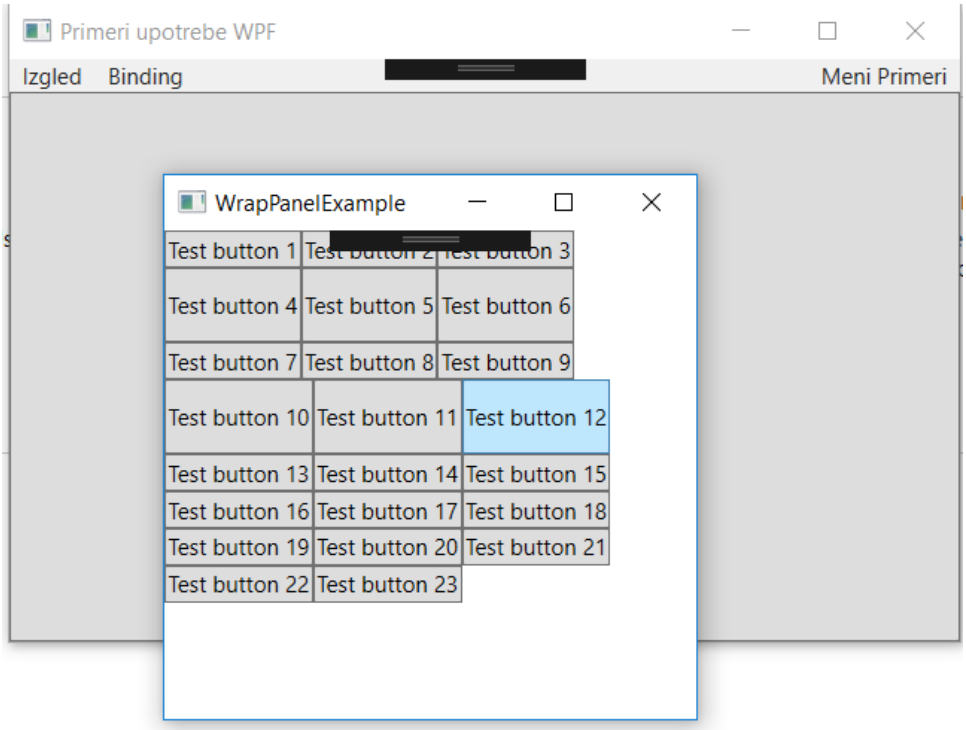
Takođe se može primetiti da *WrapPanel,* kao što mu ime kaže, radi *wrap* elemenata u novi red, kada nema dovoljno prostora.

```
<Window x:Class="PrimerCas4.Layouts.WrapPanelExample"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="WrapPanelExample" Height="300" Width="300">
    <WrapPanel Orientation="Horizontal">
        <Button>Test button 1</Button>
        <Button>Test button 2</Button>
        <Button>Test button 3</Button>
        <Button Height="40">Test button 4</Button>
        <Button>Test button 5</Button>
        <Button>Test button 6</Button>
        <Button>Test button 7</Button>
        <Button>Test button 8</Button>
        <Button>Test button 9</Button>
        <Button Height="40">Test button 10</Button>
        <Button>Test button 11</Button>
        <Button>Test button 12</Button>
        <Button>Test button 13</Button>
        <Button>Test button 14</Button>
        <Button>Test button 15</Button>
        <Button>Test button 16</Button>
        <Button>Test button 17</Button>
        <Button>Test button 18</Button>
        <Button>Test button 19</Button>
        <Button>Test button 20</Button>
        <Button>Test button 21</Button>
        <Button>Test button 22</Button>
        <Button>Test button 23</Button>


    </WrapPanel>
</Window>
```

Slika 20. PrimerVezbe3 - WrapPanel

Na slici 21 prikazan je raspored elemenata u okviru *Wrap* panela.



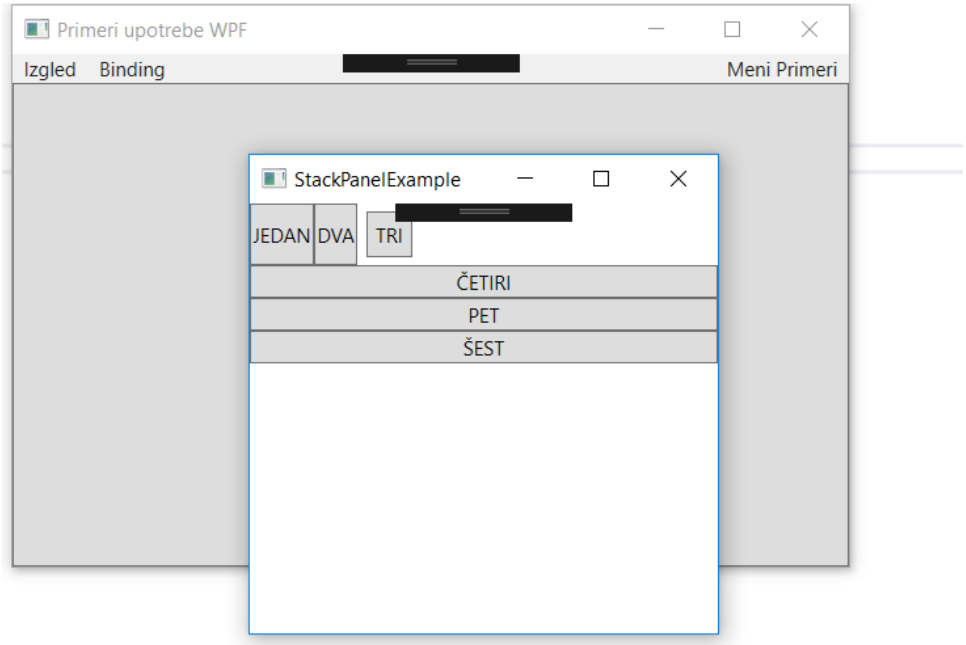
Slika 21. PrimerVezbe3 – Izgled WrapPanel-a nakon pokretanja aplikacije

Stack panel

StackPanel je veoma sličan *Wrap* panelu ali sa jednom važnom razlikom, ne *wrap*-uje sadržaj. Omogućava da se elementi ređaju jedan na drugi, u određenom smeru. (Slika 23)

```
<Window x:Class="PrimerCas4.Layouts.StackPanelExample"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="StackPanelExample" Height="300" Width="300">
    <StackPanel Orientation="Vertical">
        <StackPanel Orientation="Horizontal">
            <Button>JEDAN</Button>
            <Button>DVA</Button>
            <Button Margin="5" Padding="5">TRI</Button>
        </StackPanel>
        <Button>ČETIRI</Button>
        <Button>PET</Button>
        <Button>ŠEST</Button>
    </StackPanel>
</Window>
```

Slika 22. PrimerVezbe3 – StackPanel



Slika 23. PrimerVezbe3 - Izgled StackPanel-a nakon pokretanja aplikacije

Treba primetiti da je *default* ponašanje Stack panela da razvlači (eng. *Stretch)* svoje *chid* kontrole.

StackPanel koji ima vertikalnu orijentaciju kao na primeru sa slike 22 , sve svoje *child* kontrole rasteže horizontalno. Kada je u pitanju StackPanel sa horizontalnom orijentacijom, sve *child* kontrole se rastežu vertikalno.

StackPanel omogućava to zato što su *HorizontalAlignment* ili *VerticalAlignment property* na njegovim *child* kontrolama postavljeni na *Stretch*. Ovo ponašanje se može promeniti na način kao što je urađeno sa dugmićem 3.

Takođe treba uočiti da je *default* orijentacija za *StackPanel Vertical*, za razliku od *WrapPanel* kod kog je *Horizontal*.

Grid panel

Grid može da sadrži više redova i kolona. Moguće je definisati visinu za svaki od redova ili širinu za svaku kolonu. Visina ili širina se mogu definisati na više načina, kao broj piksela, procentima u zavisnosti od slobodnog prostora ili da se vrednost prilagodi veličini sadržaja. O tome će biti više reči u nekim od narednih vežbi.

Uz pomoć *RowDefinition* atributa odredili smo da će *Grid* panel imati 3 reda. Za prva dva reda visina je postavljena na *Auto*. (Slika 24)

Kako smo odredili koliko će *Grid* imati redova, možemo da odredimo i koliko kolona želimo u *Grid*-u. Ovo ćemo uraditi uz pomoć *ColumnDefinition* atributa. Za prve dve kolone širina je postavljena na *Auto* dok je za treću uzeta *default* vrednost. (Slika 24)

Default vrednost za širinu je GridLength i predstavlja "1*" veličinu (takođe više u nastavku).

```
<Window x:Class="PrimerCas4.Layouts.GridExample"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="GridExample" Height="300" Width="300">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Button Grid.Column="0" Grid.Row="0" Background="Red">JEDAN</Button>
        <Button Grid.Column="0" Grid.Row="1" Background="Blue">DVA</Button>
        <Button Grid.Column="0" Grid.Row="2" Grid.ColumnSpan="3" Background="Green">TRI</Button>
        <Button Grid.Column="1" Grid.Row="0" Grid.RowSpan="2" Background="Yellow">CETIRI</Button>
        <Button Grid.Column="2" Grid.Row="0" Background="Pink">PET</Button>
        <Button Grid.Column="2" Grid.Row="1" Background="Violet">SEST</Button>
    </Grid>
</Window>
```

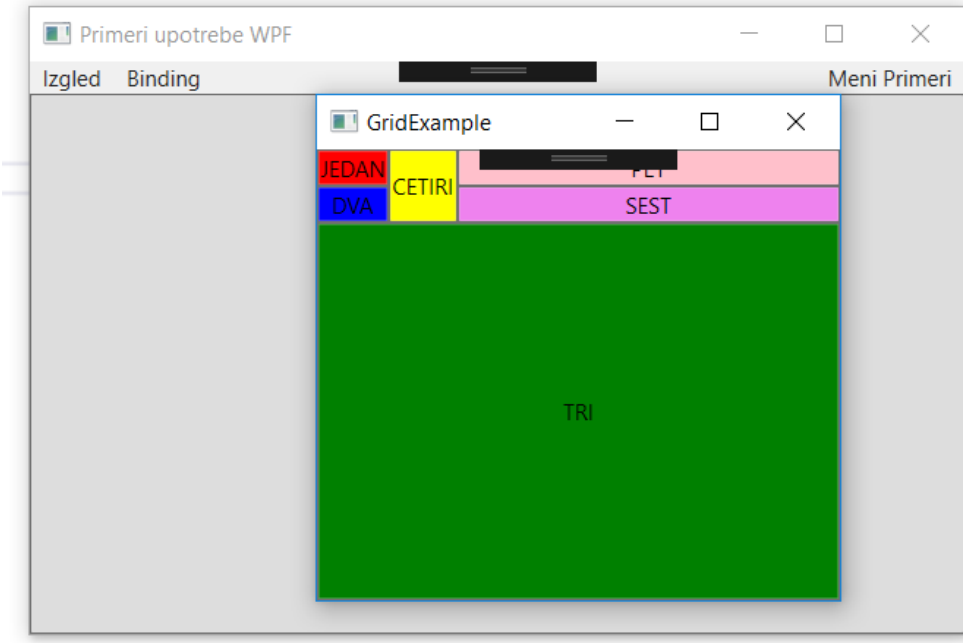
Slika 24. PrimerVezbe3 - GridPanel

Na slici 24 u XAML kodu možemo primetiti da je dodato 6 dugmića i da je za svaki definisano u kojoj koloni i redu se nalazi. Ovakvo ponašanje omogućavaju *Grid.Column* i *Grid.Row* atributi.

Osim toga, dodati su atributi koji određuju boju pozadine, kao i *ColumnSpan* i *RowSpan*.

Za dugme 3, definisanjem *ColumnSpan = 3* omogućeno je da dugme 3 zauzme širinu sve 3 kolone u tabeli.

Slično tome, za dugme 4, uz pomoć *RowSpan = 2* omogućeno je da dugme 4 zauzme 2 reda u tabeli. (Slika 25)



Slika 25. PrimerVezbe3 - Izgled GridPanel-a nakon pokretanja aplikacije

Dock panel

DockPanel olakšava raspoređivanje sadržaja u 4 smera (*top, bottom, left, right*). Predstavlja najbolji izbor kada je potrebno podeliti prozor u specifične sekcije, naročito zbog toga što poslednji element u *DockPanelu*, po *default*-u, zauzima preostali prostor (*center*). Na taj način se sav prostor u prozoru zauzima.

DockPanel.Dock property omogućava da odredimo smer gde će kontrola biti postavljena.

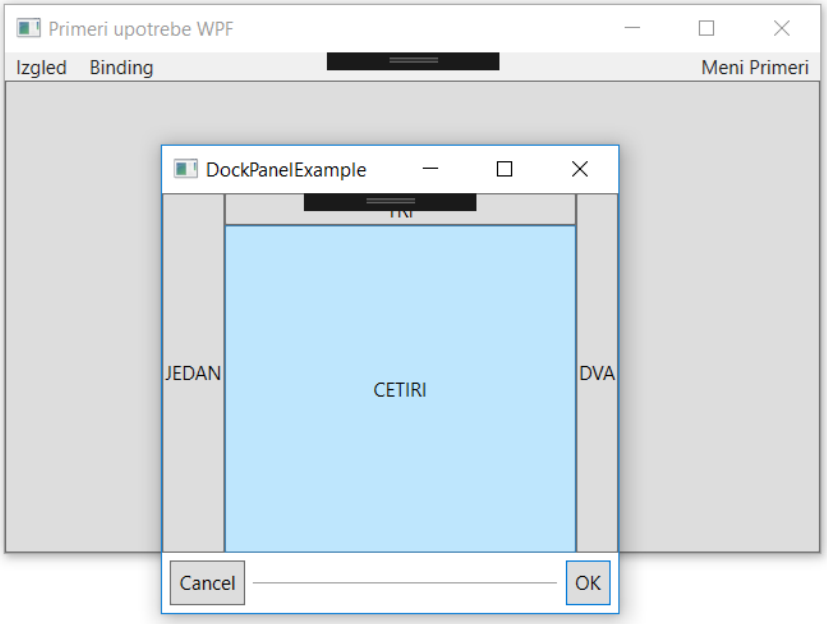
Na primeru sa slike 26 se vidi da su u okviru jednog *DockPanel*-a smeštena druga dva *DockPanel*-a.

Prvi od njih objedinjuje dva dugmića i separator a drugi u sebi sadrži 4 dugmića. Prvi panel je postavljen na dno uz pomoć *DockPanel.Dock = "Bottom"* a drugi na vrh sa *DockPanel.Dock = "Top"*. Osim toga, za drugi panel je postavljeno *VerticalAlignment = "Stretch"* pa shodno tome zauzima ostatak prozora.

```
<Window x:Class="PrimerCas3.Layouts.DockPanelExample"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="DockPanelExample" Height="300" Width="300">
    <DockPanel>
        <DockPanel DockPanel.Dock="Bottom">
            <Button DockPanel.Dock="Left" Padding="5" Margin="5" IsCancel="True">Cancel</Button>
            <Button DockPanel.Dock="Right" Padding="5" Margin="5" IsDefault="True">OK</Button>
            <Separator DockPanel.Dock="Bottom"></Separator>
        </DockPanel>
        <DockPanel DockPanel.Dock="Top" VerticalAlignment="Stretch">
            <Button DockPanel.Dock="Left">JEDAN</Button>
            <Button DockPanel.Dock="Right">DVA</Button>
            <Button DockPanel.Dock="Top">TRI</Button>
            <Button DockPanel.Dock="Bottom">CETIRI</Button>
        </DockPanel>
    </DockPanel>
</Window>
```

Slika 26. PrimerVezbe3 – DockPanel

Dugmići u drugom panelu su raspoređeni u različitim smerovima a dugme 'CETIRI' osim toga zauzima i centralni deo jer je poslednji naveden. (Slika 27)



Slika 27. PrimerVezbe3 - Izgled DockPanel-a nakon pokretanja aplikacije

Primer kombinacije više panela – Complex Layout

U primeru sa slike 28 iskorišćen je *DockPanel* da bi se prostor podelio na sekcije.

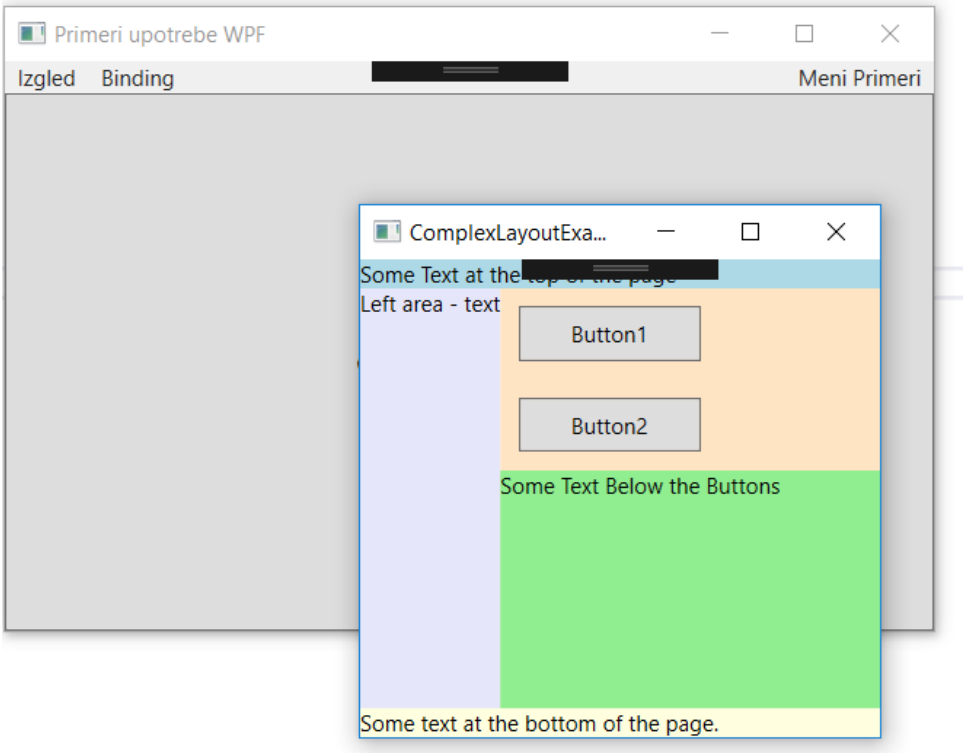
Na vrhu (*DockPanel.Dock = "Top"*) postavljen je jedan *TextBlock*. Ista stvar urađena je i na dnu prozora.

Ostatak prostora popunjen je sa novim *DockPanelom* u okviru kog je kreiran *StackPanel* i dodat jedan *TextBlock*. *StackPanel* u sebi objedinjuje dva dugmeta a svaki od njih određen je atributima kao što su *Height, Width i Margin*.

Ovim primerom pokazano je kako se kompozicijom više panela može napraviti željeni raspored komponenti. (Slika 29)

```
<Window x:Class="PrimerCas4.Layouts.ComplexLayoutExample"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="ComplexLayoutExample" Height="300" Width="300">
    <DockPanel>
        <TextBlock Background="LightBlue"
                DockPanel.Dock="Top">Some Text at the top of the page</TextBlock>
        <TextBlock DockPanel.Dock="Bottom"
                Background="LightYellow">Some text at the bottom of the page.</TextBlock>
        <TextBlock DockPanel.Dock="Left"
                Background="Lavender">Left area - text </TextBlock>
        <DockPanel Background="Bisque">
            <StackPanel DockPanel.Dock="Top">
                <Button HorizontalAlignment="Left"
                Height="30px"
                Width="100px"
                Margin="10,10,10,10">Button1</Button>
                <Button HorizontalAlignment="Left"
                Height="30px"
                Width="100px"
                Margin="10,10,10,10">Button2</Button>
            </StackPanel>
            <TextBlock Background="LightGreen">Some Text Below the Buttons</TextBlock>
        </DockPanel>
    </DockPanel>
</Window>
```

Slika 28. PrimerVezbe3 - ComplexLayout



Slika 29. PrimerVezbe3 - Izgled ComplexLayout-a nakon pokretanja aplikacije

Data binding

Većina aplikacija omogućava korisniku da manipuliše sa nekim podacima. Kada su podaci učitani u aplikaciju (npr. iz neke baze podataka) potrebno ih je prikazati korisniku. Osim toga, korisniku se često omogućava da menja te podatke pa se mora voditi računa da bude tačno stanje podataka u bazi.

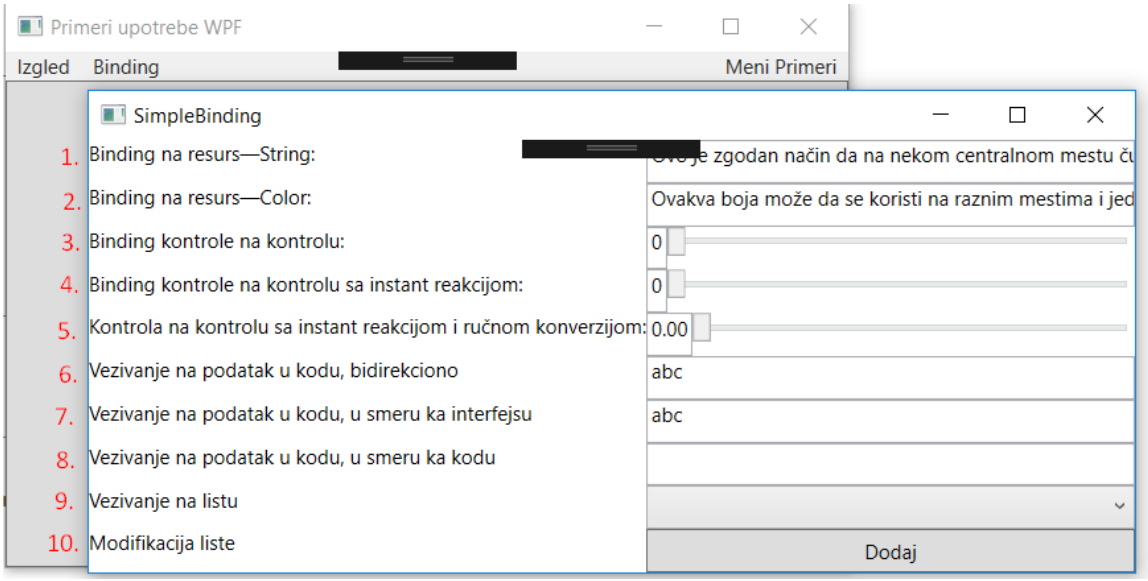
Da bi olakšao prikaz i interakciju sa podacima, WPF uvodi *Data binding*. *Data binding* predstavlja mehanizam koji omogućava komunikaciju UI elemenata sa podacima. Kada je *binding* uspostavljen i kada se podaci promene, moguće je automatski promeniti i UI elemente koji zavise od tih podataka. Obrnuto je takođe moguće.

Moguće je vršiti *binding* između dve kontrole kao i *binding* na resurs.

Da bi se mogle detektovati promene na *source*-u (primenjivo na *OneWay* and *TwoWay bindings*) *source* mora implementirati odgovarajući *Property change notification mechanism* kao što je INotifyPropertyChanged.

O ovom interfejsu će više govora biti u nastavku kursa.

Na primeru sa slike 30 biće objašnjeni različiti načini upotrebe *binding*-a.



Slika 30. PrimerVezbe3 – Binding

Binding na resurs – string (1)

Na slici 31 je prikazano kako se može dodati resurs. U ovom slučaju, resursi su vezani za prozor tako da ih mogu koristiti sve kontrole iz tog prozora. Uz pomoć atributa ***x:Key*** se jedinstveno identifikuje i na taj način će mu se pristupati iz ostatka koda.

```
<Window x:Class="PrimerCas4.Binding.SimpleBinding"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:sys="clr-namespace:System;assembly=mscorlib"
        xmlns:cvt="clr-namespace:PrimerCas4.Binding"
        Title="SimpleBinding" Height="300" Width="648">
    <Window.Resources>
        <sys:String x:Key="strTestBinding">Ovo je zgodan način da na nekom centralnom mestu čuvamo podatke koji se ne menjaju</sys:String>
        <SolidColorBrush x:Key="clrSpecialColor" Color="#FFF2F254"></SolidColorBrush>
    </Window.Resources>
```

Slika 31. PrimerVezbe3 – Resursi zakačeni na Window tag

Ako želimo da *TextBox* kao svoj sadržaj ima tekst iz resursa, možemo to uraditi tako što ćemo ***Text*** *property* ove kontrole *bindovati* na resurs – string.

Sa ključnom reči ***StaticResource*** označavamo da će se koristi statički resurs a kao vrednost ***ResourceKey*** navodimo jedinstveni identifikator resursa kog želimo da koristimo.

S obzirom da smo definisali vrednost ***x:Key*** kao ***"strTestBinding"***, na ovaj način ćemo pozivati resurs. (Slika 32)

```
<TextBlock Grid.Column="0" Grid.Row="6">Vezivanje na podatak u kodu, u smeru ka interfejsu</TextBlock>
<TextBlock Grid.Column="0" Grid.Row="7">Vezivanje na podatak u kodu, u smeru ka kodu</TextBlock>
<TextBlock Grid.Column="0" Grid.Row="8">Vezivanje na listu</TextBlock>
<TextBlock Grid.Column="0" Grid.Row="9">Modifikacija liste</TextBlock>

<TextBox Grid.Column="1" Grid.Row="0" Text="{StaticResource ResourceKey=strTestBinding}"></TextBox>
<TextBox Grid.Column="1" Grid.Row="1" SelectionBrush="{StaticResource ResourceKey=clrSpecialColor}">Ovakva boja može da se
<DockPanel Grid.Column="1" Grid.Row="2">
    <TextBox Text="{Binding ElementName=testSlider1,Path=
    <Slider x:Name="testSlider1" Maximum="100"></Slider>
</DockPanel>
<DockPanel Grid.Column="1" Grid.Row="3">
```

StaticResourceExtension

Slika 32. PrimerVezbe3 – Binding na string

Binding na resurs – boja (2)

Na slici 31 možemo videti još jedan resurs. Ovaj resurs definisan je kao ***SolidColorBrush*** I koristiće se kao boja za selektovani tekst.

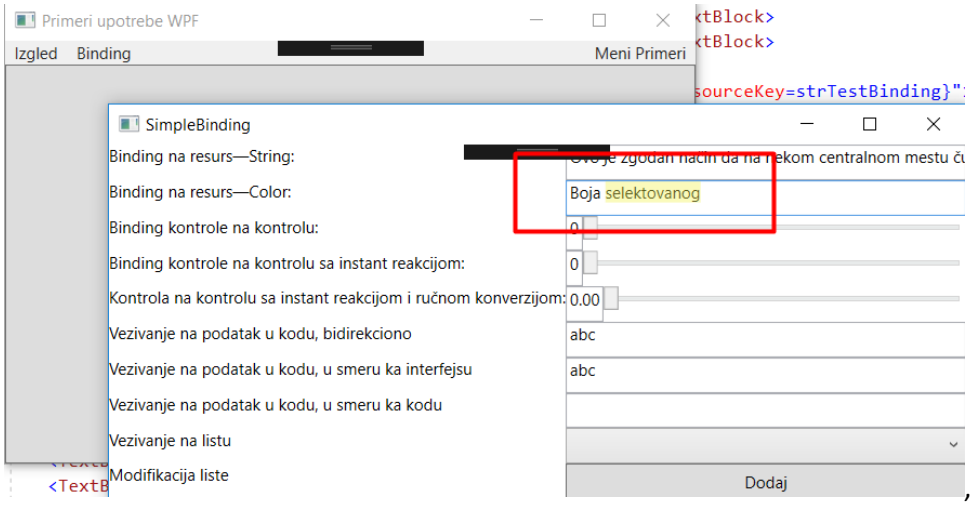
Ovom resursu smo dodelili jedinstveni identifikator "***clrSpecialColor"***.

TextBox nudi ***SelectionBrush*** *i* na ovom mestu možemo odrediti boju koja se koristi kada se selektuje text iz ovog *TextBox*-a. Tako je na ovom mestu odabran resurs i *bindovan* je na ***SelectionBrush***. (Slika 33)

```
<TextBlock Grid.Column="0" Grid.Row="6">Vezivanje na podatak u kodu, u smeru ka interfejsu</TextBlock>
<TextBlock Grid.Column="0" Grid.Row="7">Vezivanje na podatak u kodu, u smeru ka kodu</TextBlock>
<TextBlock Grid.Column="0" Grid.Row="8">Vezivanje na listu</TextBlock>
<TextBlock Grid.Column="0" Grid.Row="9">Modifikacija liste</TextBlock>

<TextBox Grid.Column="1" Grid.Row="0" Text="{StaticResource ResourceKey=strTestBinding}"></TextBox>
<TextBox Grid.Column="1" Grid.Row="1" SelectionBrush="{StaticResource ResourceKey=clrSpecialColor}">Boja selektovanog</TextBox>
<DockPanel Grid.Column="1" Grid.Row="2">
    <TextBox Text="{Binding ElementName=testSlider1,Path=Value}"></TextBox>
    <Slider x:Name="testSlider1" Maximum="100"></Slider>
</DockPanel>
<DockPanel Grid.Column="1" Grid.Row="3">
    <TextBox Text="{Binding ElementName=testSlider2,Path=Value,UpdateSourceTrigger=PropertyChanged}"></TextBox>
    <Slider x:Name="testSlider2" Maximum="100"></Slider>
</DockPanel>
```

Slika 33. PrimerVezbe3 – Binding na boju



Slika 34. PrimerVezbe3 – Binding na boju - Rezultat selektovanja teksta

One way – ka interfejsu (7)

OneWay binding – promene na *source property* (model) automatski osvežavaju *target property* (UI kontrola) ali promene na *target property* (UI kontrola) ne emituju svoje promene na *source property* (model).

Na kratko ćemo zanemariti *UpdateSourceTrigger*, objašnjenje sledi u nastavku.

```
<TextBox Grid.Column="1" Grid.Row="5" Text="{Binding Path=Test1,UpdateSourceTrigger=PropertyChanged}"></TextBox>
<TextBox Grid.Column="1" Grid.Row="6" Text="{Binding Path=Test1,Mode=OneWay,UpdateSourceTrigger=PropertyChanged}"></TextBox>
<TextBox Grid.Column="1" Grid.Row="7" Text="{Binding Path=Test2,Mode=OneWayToSource,UpdateSourceTrigger=PropertyChanged}"></TextBox>
<ComboBox Grid.Column="1" Grid.Row="8" IsEditable="False" ItemsSource="{Binding Path=Test3}"/>
```

Slika 35. PrimerVezbe3 – One way binding

U ovom primeru (Slika 35) biće navedeno kako bindovati *"Test1"* na *Text property TextBox* kontrole. *"Test1"* se nalazi u klasi *SimpleBinding.xaml.cs*. (Slika 36)

Da bi mogli da izvršimo *binding*, *property* iz *code-behind*-a sa nekom kontrolom (u ovom slučaju *TextBox*) u XAML kodu, potrebno je da popunimo DataContext. (Slika 37)

DataContext property je *default source* za naš *binding*, osim ako se ne specificira drugačije.

Ne postoji *default source* za *DataContext property* (on je null na početku), ali s obzirom da se DataContext nasleđuje dalje u hijerarhijI kontrola , često je dovoljno postaviti *DataContext* na celi prozor i dalje koristiti za sve *child* kontrole.

```
public partial class SimpleBinding : Window, INotifyPropertyChanged
{
    public ObservableCollection<string> Test3
    {
        get;
        set;
    }

    #region NotifyProperties
    private string _test1;
    private string _test2;
    public string Test1
    {
        get
        {
            return _test1;
        }
        set
        {
            if (value != _test1)
            {
                _test1 = value;
                OnPropertyChanged("Test1");
            }
        }
    }
```

Slika 36. PrimerVezbe3 – Code behind – Test1 property

Nakon poziva *InitalizeComponent()* metode, postavljamo "this" referencu kao *DataContext*, što u suštini govori *Window* komponenti da želimo da on sam bude *Data context* u *bindingu*. (Slika 37)

```
    public SimpleBinding()
    {
        InitializeComponent();
        this.DataContext = this;
        Test1 = "abc";
        Test2 = "def";
        Test3 = new ObservableCollection<string>();
        Test3.Add("A");
        Test3.Add("B");
        Test3.Add("C");
    }
```

Slika 37. PrimerVezbe3 – Code behind – Konstruktor

Napomena: Obratiti pažnju na način kako funkcioniše!

Two way – bidirekciono (6)

TwoWay binding – promene na *source property* (model) ili *target property* (UI kontrola) automatski osvežavaju promene na drugom. (Slika 38) Ovaj tip *binding*-a odgovara npr. situaciji kada se koriste forme za editovanje podataka.

Na kratko ćemo zanemariti *UpdateSourceTrigger*, biće objašnjeno u nastavku.

Napomena: Obratiti pažnju na način kako funkcioniše!

```
<TextBox Grid.Column="1" Grid.Row="5" Text="{Binding Path=Test1,UpdateSourceTrigger=PropertyChanged}"></TextBox>
<TextBox Grid.Column="1" Grid.Row="6" Text="{Binding Path=Test1,Mode=OneWay,UpdateSourceTrigger=PropertyChanged}"></TextBox>
<TextBox Grid.Column="1" Grid.Row="7" Text="{Binding Path=Test2,Mode=OneWayToSource,UpdateSourceTrigger=PropertyChanged}"></TextBox>
<ComboBox Grid.Column="1" Grid.Row="8" IsEditable="False" ItemsSource="{Binding Path=Test3}"/>
```

Slika 38. PrimerVezbe3 – Two way binding

One way to source – ka kodu (8)

OneWayToSource binding – suprotno od *OneWay binding*-a. Osvežava *source property* (model) kada dođe do promena na *target property*. (UI element) (Slika 39)

Napomena: Obratiti pažnju na način kako funkcioniše!

```
<TextBox Grid.Column="1" Grid.Row="5" Text="{Binding Path=Test1,UpdateSourceTrigger=PropertyChanged}"></TextBox>
<TextBox Grid.Column="1" Grid.Row="6" Text="{Binding Path=Test1,Mode=OneWay,UpdateSourceTrigger=PropertyChanged}"></TextBox>
<TextBox Grid.Column="1" Grid.Row="7" Text="{Binding Path=Test2,Mode=OneWayToSource,UpdateSourceTrigger=PropertyChanged}"></TextBox>
<ComboBox Grid.Column="1" Grid.Row="8" IsEditable="False" ItemsSource="{Binding Path=Test3}"/>
```

Slika 39. PrimerVezbe3 – One way to source binding

Binding kontrole na kontrolu (3)

Osim *binding*-a na resurs ili na *property* iz *code-behind*-a, moguće je bindovati kontrolu na kontrolu. Ovaj proces podrazumeva da ponašanje jedne kontrole zavisi od ponašanja druge kontrole.

U primeru sa slike 40 možemo videti kako vrednost *TextBox* kontrole zavisi od vrednosti *Slider* kontrole. Postiže se tako što *Text property TextBox*-a bindujemo na *Value property Slider*-a.

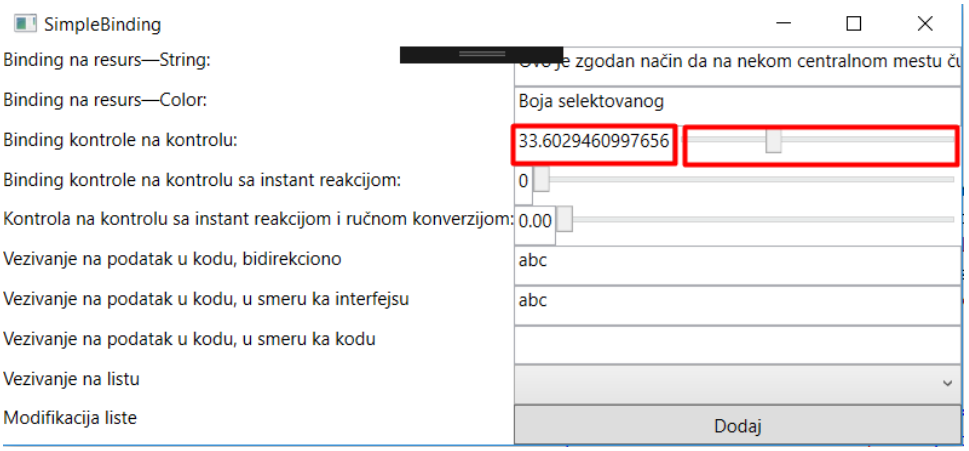
Generička sintaksa za ovakav *binding* je

{Binding ElementName=keyName, Path=propertyName}

```
<TextBlock Grid.Column="0" Grid.Row="5">Vezivanje na podatak u kodu, bidirekciono</TextBlock>
<TextBlock Grid.Column="0" Grid.Row="6">Vezivanje na podatak u kodu, u smeru ka interfejsu</TextBlock>
<TextBlock Grid.Column="0" Grid.Row="7">Vezivanje na podatak u kodu, u smeru ka kodu</TextBlock>
<TextBlock Grid.Column="0" Grid.Row="8">Vezivanje na listu</TextBlock>
<TextBlock Grid.Column="0" Grid.Row="9">Modifikacija liste</TextBlock>

<TextBox Grid.Column="1" Grid.Row="0" Text="{StaticResource ResourceKey=strTestBinding}"></TextBox>
<TextBox Grid.Column="1" Grid.Row="1" SelectionBrush="{StaticResource ResourceKey=clrSpecialColor}">Boja selektovanog</TextBox>
<DockPanel Grid.Column="1" Grid.Row="1" HorizontalAlignment="Right" Margin="0,26.8,-0.4,26.8" Grid.RowSpan="3" Width="304">
    <TextBox Text="{Binding ElementName=testSlider1,Path=Value}"></TextBox>
    <Slider x:Name="testSlider1" Maximum="100"></Slider>
</DockPanel>
<DockPanel Grid.Column="1" Grid.Row="3">
    <TextBox Text="{Binding ElementName=testSlider2,Path=Value,UpdateSourceTrigger=PropertyChanged}"></TextBox>
    <Slider x:Name="testSlider2" Maximum="100"></Slider>
</DockPanel>
```

Slika 40. PrimerVezbe3 – Binding kontrole na kontrolu



Slika 41. PrimerVezbe3 – Binding kontrole na kontrolu – Prikaz nakon pokretanja aplikacije

Binding kontrole na kontrolu - real time (4)

Binding kao što su *TwoWay* ili *OneWayToSource* osluškuju promene na *target property* (UI element) i propagiraju do *source* (model) I ovo je poznato kao *"updating the source"*. Npr. kada se edituje polje (*TextBox*) treba promeniti *source* value. Ova promena može da bude istog momenta (eng. *real-time*) ili kada se izgubi fokus. *Binding.UpdateSourceTrigger property* određuje kada će se ova promena zaista desiti.

Ako je vrednost za *UpdateSourceTrigger* postavljena na *LostFocus*, onda će se vrednost *source* -a promeniti na novu vrednost tek nakon što polje u koje ste unosili vrednost izgubi fokus.

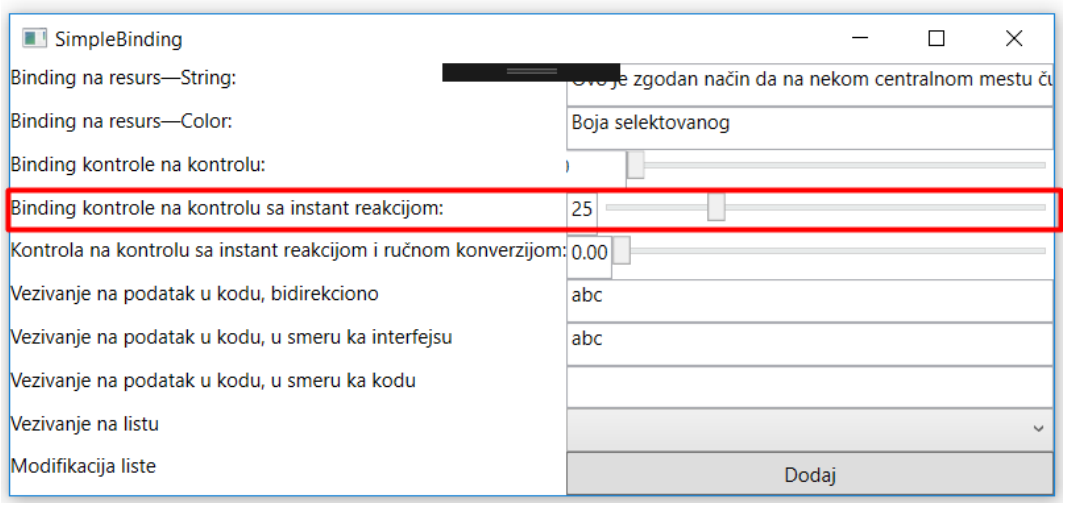
Default vrednost *UpdateSourceTrigger* za većinu je *PropertyChanged*, dok je za *TextBox.Text property LostFocus*.

U prethodnom primeru bindovanja kontrole na kontrolu (slika 40) uzeta je *default* vrednost i zbog toga nemamo *real time* promene nego na *LostFocus*. U ovom slučaju (slika 41) promene se odmah reflektuju jer je eksplicitno navedeno *UpdateSourceTrigger = PropertyChanged*.

```
<TextBox Grid.Column="1" Grid.Row="0" Text="{StaticResource ResourceKey=strTestBinding}"></TextBox>
<TextBox Grid.Column="1" Grid.Row="1" SelectionBrush="{StaticResource ResourceKey=clrSpecialColor}">Boja selektovanog</TextBox>
<DockPanel Grid.Column="1" Grid.Row="1" HorizontalAlignment="Right" Margin="0,26.8,-0.4,26.8" Grid.RowSpan="3" Width="304">
    <TextBox Text="{Binding ElementName=testSlider1,Path=Value}" Width="44"></TextBox>
    <Slider x:Name="testSlider1" Maximum="100"></Slider>
</DockPanel>
<DockPanel Grid.Column="1" Grid.Row="3">
    <TextBox Text="{Binding ElementName=testSlider2,Path=Value,UpdateSourceTrigger=PropertyChanged}"></TextBox>
    <Slider x:Name="testSlider2" Maximum="100"></Slider>
</DockPanel>
```

Binding.UpdateSourceTrigger

Slika 42. PrimerVezbe3 - Real time binding kontrole na kontrolu



Slika 43. PrimerVezbe3 - Real time binding kontrole na kontrolu - Prikaz nakon pokretanja aplikacije

Binding kontrole na kontrolu - real time + konverzija (5)

Ako se u procesu *binding*-a želi iskoristi konverzija to se može lako uvesti uz pomoć *Converter* atributa. Dovoljno je navesti koji konverter će se koristiti. Da bi se kreirao konverter dovoljno je implementirati *interface* (Slika 46) i implementirati telo *Convert* i *ConvertBack* metode.

```
<DockPanel Grid.Column="1" Grid.Row="3">
    <TextBox Text="{Binding ElementName=testSlider2,Path=Value,UpdateSourceTrigger=PropertyChanged}"></TextBox>
    <Slider x:Name="testSlider2" Maximum="100"></Slider>
</DockPanel>

<DockPanel Grid.Column="1" Grid.Row="4">
    <TextBox Text="{Binding ElementName=testSlider3,Path=Value,UpdateSourceTrigger=PropertyChanged,Converter={cvt:DoubleToTextConverter}}"></TextBox>
    <Slider x:Name="testSlider3" Maximum="100"></Slider>
</DockPanel>
<TextBox Grid.Column="1" Grid.Row="5" Text="{Binding Path=Test1,UpdateSourceTrigger=PropertyChanged}"></TextBox>
<TextBox Grid.Column="1" Grid.Row="6" Text="{Binding Path=Test1,Mode=OneWay,UpdateSourceTrigger=PropertyChanged}"></TextBox>
<TextBox Grid.Column="1" Grid.Row="7" Text="{Binding Path=Test2,Mode=OneWayToSource,UpdateSourceTrigger=PropertyChanged}"></TextBox>
```

Slika 44. PrimerVezbe3 - Real time binding kontrole na kontrolu i konverzija

Da bi WPF mogao prepoznati i koristiti *DoubleToTextConverter* navodi se putanja do klase *DoubleToTextConverter* (Slika 45)

```
<Window x:Class="PrimerCas4.Binding.SimpleBinding"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:sys="clr-namespace:System;assembly=mscorlib"
        xmlns:cvt="clr-namespace:PrimerCas4.Binding"
        Title="SimpleBinding" Height="300" Width="648">
    <Window.Resources>
        <sys:String x:Key="strTestBinding">Ovo je zgodan način da na nekom centralnom mestu čuvamo podatke koji se ne menjaju</sys:String>
        <SolidColorBrush x:Key="clrSpecialColor" Color="#FFF2F254"></SolidColorBrush>
    </Window.Resources>
```

Slika 45. PrimerVezbe3 - Real time binding kontrole na kontrolu i konverzija - Prikaz nakon pokretanja aplikacije

```
namespace PrimerCas4.Binding
{
    public class DoubleToTextConverter : MarkupExtension, IValueConverter
    {
        public DoubleToTextConverter()
        {
        }

        public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
        {
            var v = (double)value;
            return String.Format("{0:0.00}", v);
        }

        public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
        {
            var v = value as string;
            double ret = 0;
            if (double.TryParse(v, out ret))
            {
                return ret;
            }
            else
            {
                return value;
            }
        }

        public override object ProvideValue(IServiceProvider serviceProvider)
        {
            return this;
        }
    }
}
```

Slika 46. PrimerVezbe3 – Konverter klasa

Napomena: Obavezno pogledati primer u kodu!

Binding na listu (9)

U ovom primeru biće navedeno kako *bindovati ComboBox* vrednosti na vrednosti neke liste.

'Test3' predstavlja *ObservableCollecton* u klasi *SimpleBinding* i sadržaće niz string vrednosti koje želimo prikazati u *ComboBox* kontroli. (Slika 47) Da bi mogli da povežemo ovu kolekciju iz *SimpleBinding* klase (*code-behind*) sa nekom kontrolom u XAML kodu ,potrebno je da popunimo *DataContext*. (Slika 48)

DataContext property je *default source* za naš binding, osim ako se ne specificira drugačije *source*, kao što je urađeno u primeru Binding kontrole na kontrolu, kada smo koristili *ElementName* property da bi definisali šta je *source* za taj binding. Upotreba *DataContext* objašnjena je u primeru sa *One-way binding*-om.

Napomena: Obavezno pogledati primer u kodu!

```
<TextBox Grid.Column="1" Grid.Row="5" Text="{Binding Path=Test1,UpdateSourceTrigger=PropertyChanged}"></TextBox>
<TextBox Grid.Column="1" Grid.Row="6" Text="{Binding Path=Test1,Mode=OneWay,UpdateSourceTrigger=PropertyChanged}"></TextBox>
<TextBox Grid.Column="1" Grid.Row="7" Text="{Binding Path=Test2,Mode=OneWayToSource,UpdateSourceTrigger=PropertyChanged}"></TextBox>
<ComboBox Grid.Column="1" Grid.Row="8" IsEditable="False" ItemsSource="{Binding Path=Test3}"/>
<Button Grid.Column="1" Grid.Row="9" Click="Button_Click">Dodaj</Button>
```

Slika 47. PrimerVezbe3 – Binding na listu vrednosti

```
namespace PrimerCas4.Binding
{
    /// <summary>
    /// Interaction logic for SimpleBinding.xaml
    /// </summary>
    public partial class SimpleBinding : Window, INotifyPropertyChanged
    {
        public ObservableCollection<string> Test3
        {
            get;
            set;
        }

        #region NotifyProperties
        private string _test1;
        private string _test2;
        public string Test1...
        public string Test2...
        #endregion
        PropertyChangedNotifier

        public SimpleBinding()
        {
            InitializeComponent();
            this.DataContext = this;
            Test1 = "abc";
            Test2 = "def";
            Test3 = new ObservableCollection<string>();
            Test3.Add("A");
            Test3.Add("B");
            Test3.Add("C");
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            Test3.Add("D");
            Test3.Add("E");
            Test3.Add("F");
        }

    }
}
```

Slika 48. PrimerVezbe3 – Code behind – Kolekcija

Klikom na dugme sa slike 30 označeno sa (10) poziva se metoda *Button_Click* u kojoj se vrši dodavanje elemenata u kolekciju. (Slika 48)

Napomena: Obavezno pogledati da li će se promena (klik na 10) manifestovati na *ComboBox*!

DODATAK:

Kontrole podeljene po funkciji, preuzeto sa docs.microsoft.com.

- **Buttons**: Button and RepeatButton.
- **Data Display**: DataGrid, ListView, and TreeView.
- **Date Display and Selection**: Calendar and DatePicker.
- **Dialog Boxes**: OpenFileDialog, PrintDialog, and SaveFileDialog.
- **Digital Ink**: InkCanvas and InkPresenter.
- **Documents**: DocumentViewer, FlowDocumentPageViewer, FlowDocumentReader, FlowDocumentScrollViewer, and StickyNoteControl.
- **Input**: TextBox, RichTextBox, and PasswordBox.
- **Layout**: Border, BulletDecorator, Canvas, DockPanel, Expander, Grid, GridView, GridSplitter, GroupBox, Panel, ResizeGrip, Separator, ScrollBar, ScrollViewer, StackPanel, Thumb, Viewbox, VirtualizingStackPanel, Window, and WrapPanel.
- **Media**: Image, MediaElement, and SoundPlayerAction.
- **Menus**: ContextMenu, Menu, and ToolBar.
- **Navigation**: Frame, Hyperlink, Page, NavigationWindow, and TabControl.
- **Selection**: CheckBox, ComboBox, ListBox, RadioButton, and Slider.
- **User Information**: AccessText, Label, Popup, ProgressBar, StatusBar, TextBlock, and ToolTip.

Često korišćeni *layout*-i, preuzeto sa docs.microsoft.com.

- Canvas: Child controls provide their own layout.
- DockPanel: Child controls are aligned to the edges of the panel.
- Grid: Child controls are positioned by rows and columns.
- StackPanel: Child controls are stacked either vertically or horizontally.
- VirtualizingStackPanel: Child controls are virtualized and arranged on a single line that is either horizontally or vertically oriented.
- WrapPanel: Child controls are positioned in left-to-right order and wrapped to the next line when there are more controls on the current line than space allows.

KORISNI LINKOVI:

https://docs.microsoft.com/en-us/dotnet/desktop-wpf/data/data-binding-overview