Contents

CAPITOLUL 5

Windows Presentation Foundation

1. Introducere

Windows Presentation Foundation (WPF), este proiectat de Microsoft, fiind un framework prietenos. Inițial, a fost parte a platformei .NETFramework 3.0. WPF utilizează limbajul extensibil de marcare a aplicațiilor (XAML), un derivat al limbajului extensibil de marcare (XML) pentru inițializarea valorilor structurii și a obiectelor. Oferă posibilitatea de a construi UI, grafică 2D, grafică 3D, documente și aplicații multimedia. WPF face mult mai mult decât ceea ce face Windows Form deoarece este o soluție unificată care conține tehnologii diferite. De exemplu, dacă doriți să dezvoltați o aplicație Windows Form care afișează video sau audio, trebuie să utilizați și Windows Media Player. Sau, dacă doriți să afișați documente pe o interfață, puteți utiliza Windows Forms, dar pentru fișiere trebuie să utilizați și PDF-ul Adobe. WPF are suport pentru toate acestea și chiar mai mult. Scopul WPF nu este de a înlocui Windows Forms. Exista numeroase aplicații care conectează atat Windows Form cât și WPF.

WPF poate crea interfețe moderne într-o manieră destul de simplă. Comparat cu aplicațiile Windows Form, WPF-urile sunt împărțite în două părți:

- aspectul unei interfețe de utilizator specificate în XAML,
- comportamentul implementat utilizând C# sau Visual Basic.

1.1 Prima aplicatie de tip Windows Presentation Foundation

O aplicație WPF poate fi implementată utilizând Visual Studio. O aplicație WPF simplă poate fi creată selectând File → New Project → Visual C# → WPF Application. Se va afișa o fereastră ca în figura 1. Proiectul poate fi redenumit și locația sa poate fi schimbată.

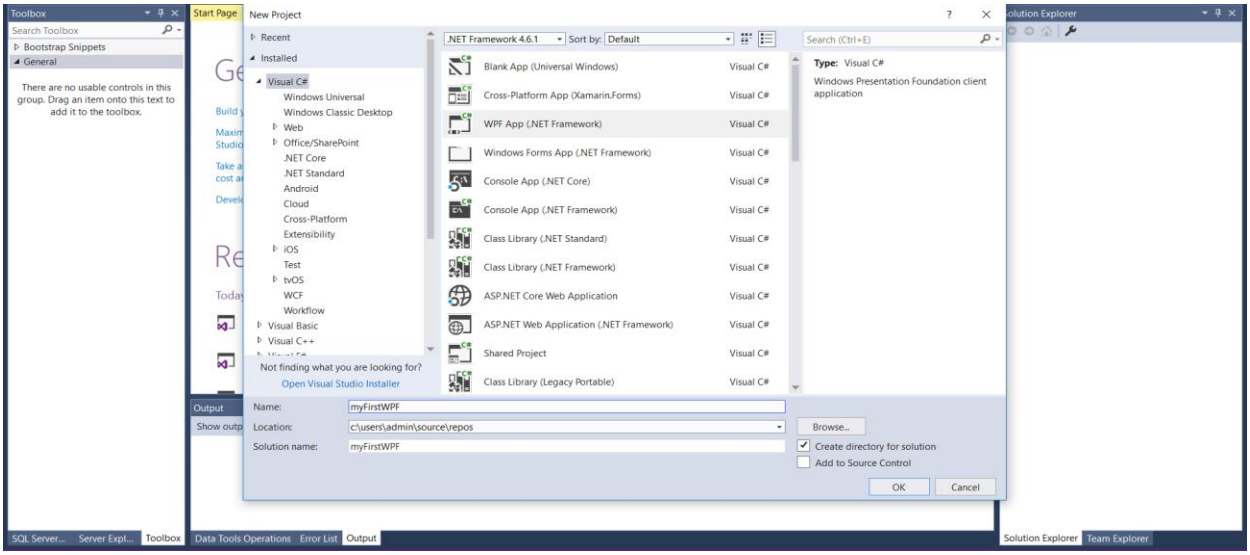

Fig. 1 Crearea unei aplicatii WPF

Proiectul împreună cu fișierele MainWindow.xaml și App.xaml sunt create automat. Are aproape aceeași structură ca Windows Form, cu excepția faptului că fișierul Window1.designer.cs este acum declarat în XAML ca MainWindow.xaml.

1.2 Controale

Bara de instrumente Toolbox (Ctrl+W, X) conține un set bogat de controale integrate. Acestea pot fi utilizate cu ”drag and dropped” din toolbox, sau pot fi implementate utilizand cod XAML. Figura de mai jos prezintă codul XAML generat automat la crearea proiectuui. Acesta oferă posibilitatea de a crea, contura, afișa și gestiona durata de viață a ferestrelor și a casetelor de dialog.

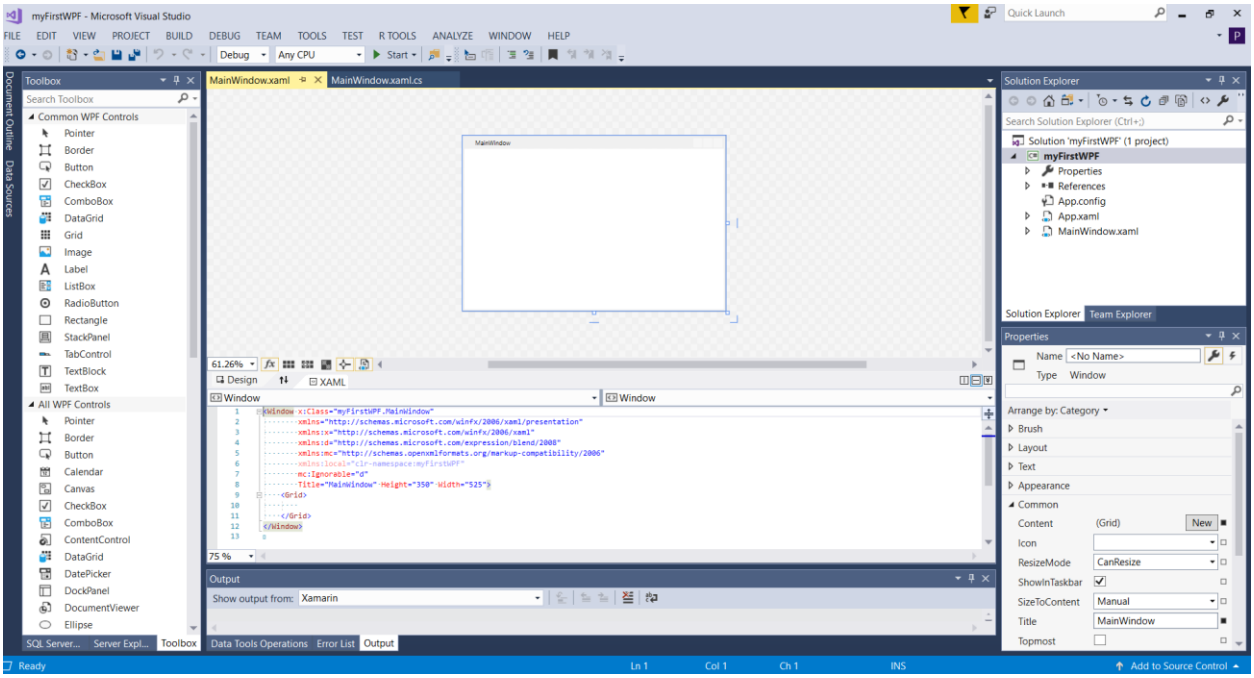

Fig. 2 Codul XAML generat automat

1.2.1 Butoane

Primul control prezentat este ***butonul*** bine cunoscut (Figura 3), care are două proprietăți principale: IsDefault și IsCancel.

Exemplu: Folosind aplicația deja creată, adăugați un buton, numit MyButton. Apoi, în fereastra de prosperitate, după ce ați vizionat apariția, faceți dublu clic pe evenimentul "Click". În acest fel, metoda button1_Click este creată automat și va fi executată la apăsarea butonului. Acest lucru se poate face numai dacă este instalat Service Pack 1 pentru Visual Studio. Metoda creată în fișierul din spatele codului este aceeași ca în Windows Form. Adăugați în interiorul metodei codul: MessageBox.Show ("Hi!"). Apoi, testați aplicația care este de asemenea prezentată în figura 4.

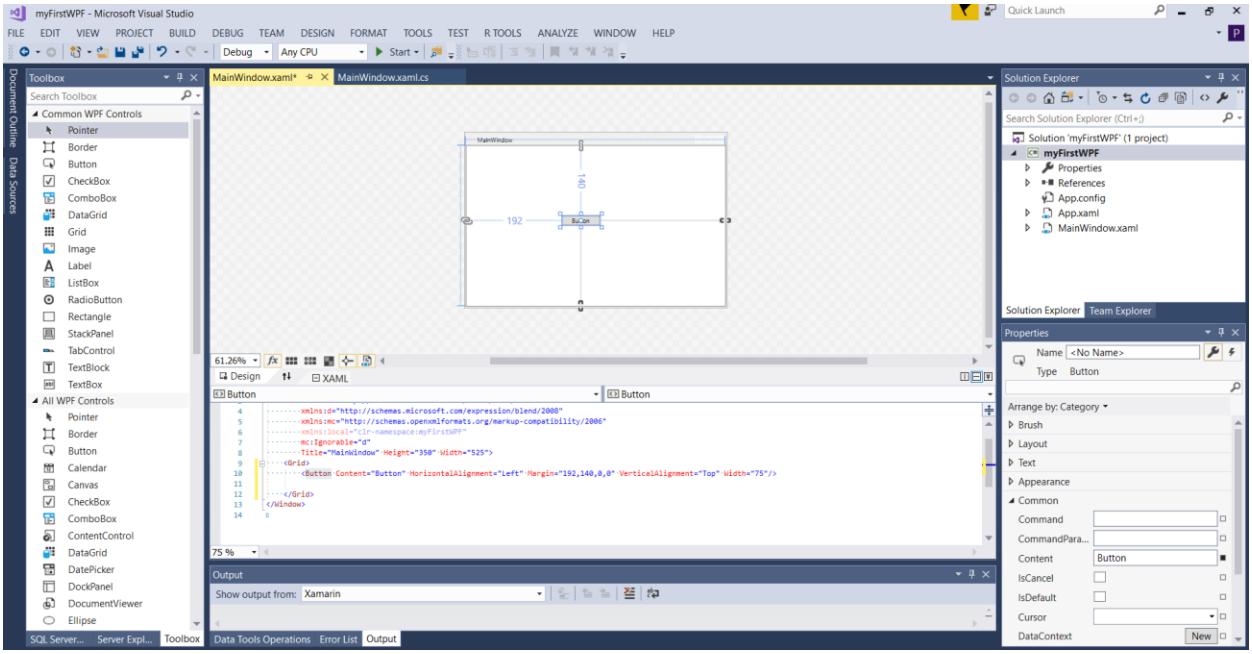

Fig. 3 Adăugare buton

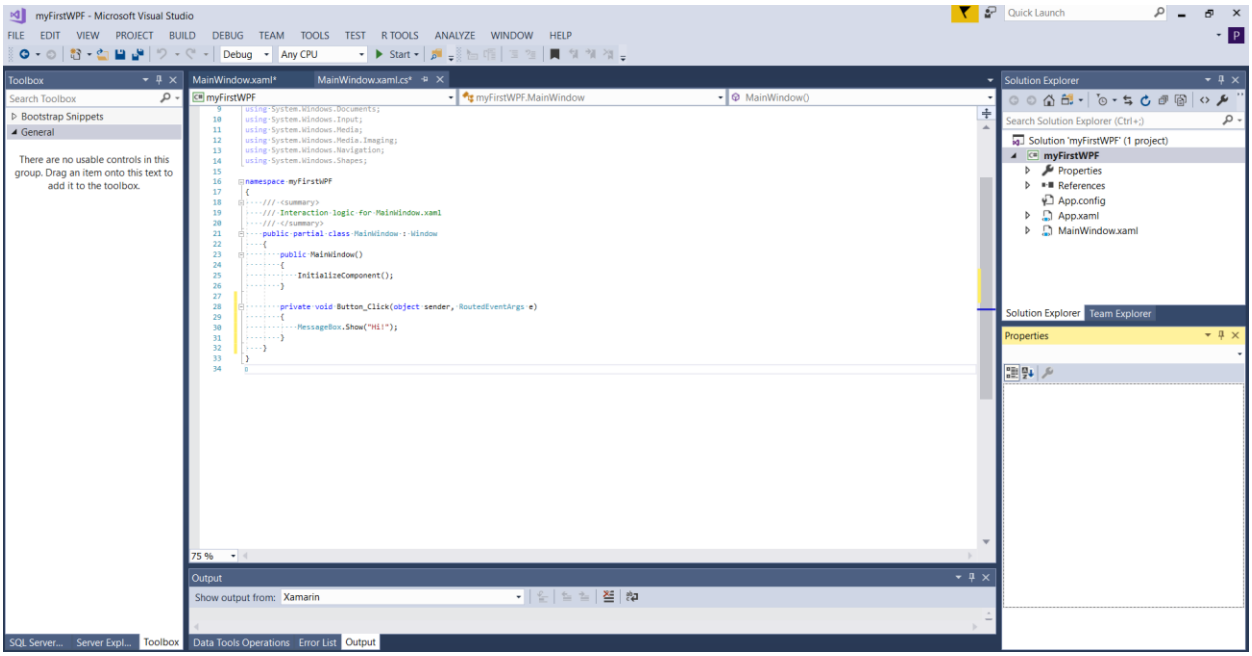

Fig. 4 Exemplu myFirstWPF

De asemenea în figura de mai jos putem vedea codul butonului din formatul XML.

```
<Window x:Class="myFirstWPF.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
```

```
        xmlns:local="clr-namespace:myFirstWPF"
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="525">
    <Grid Height="187"  Width="365">
        <Grid.RowDefinitions>
            <RowDefinition Height="184*" />
            <RowDefinition Height="3*"/>
        </Grid.RowDefinitions>
            <Button Content="Button" HorizontalAlignment="Left" Margin="152,83,0,0"
VerticalAlignment="Top" Width="75"/>
    </Grid>
</Window>
```

Fig 5 Adăugare buton – cod XAML

1.2.3. TextBoxes si RichTextBoxes

Un textBox poate fi utilizat pentru afișarea textului neformatat sau pentru a da posibilitatea introducerii unui text. Aceasta aduce ocazia de auto-corectare a cuvintelor folosind funcția de verificare a ortografiei. Limbile disponibile sunt engleză, franceză, germană și spaniolă. Codul pentru activarea funcției de verificare a ortografiei este prezentat mai jos. Acest lucru se poate face prin setarea proprietății SpellCheck.Is Enabled la true.

```
1|<TextBox SpellCheck.IsEnabled='True' Language='en-US'/>
```

Urmatorul tabel prezinta câteva controale WPF:

Label	displays text
ListBox	displays a list of items
CheckBox	displays item that can be selected or cleared
ComboBox	items control where only one item can be visible at a time
RadioButton	displays items that can be selected but not cleared
TabControl	displays multiple items that share the same space in a form
Image	displays an image

Tab.1 Controale WPF

1.2.4. Layouts

În WPF, machetele sunt folosite pentru aranjarea comenzilor în medii complexe. Tabelul 2 prezintă cele mai populare panouri de layout-uri.

Stack Panel	places the controls beside each other or below
Dock Panel	arranges the elements in the right, left, bottom, top or center of the panel
Wrap Panel	as StackPanel, except that, when the line finishes, the child elements can be wrapped to new lines
Grid Panel	organizes the child in a tabular structure
Canvas Panel	the child's coordinates are explicitly given

Tab.2 Layouts

2. Styling și Data Binding

2.1 WPF vs Windows Form

Diferența principală dintre aplicațiile Windows Forms și WPF este modul în care sunt afișate și populate elementele în Interfața Utilizator (UI). WPF introduce ideea de styling element și de legare a datelor pentru a simplifica procesul de producție. Să luăm în considerare următoarea problemă: o fereastră principală cu aproximativ 10 elemente Slider. Toate glisoarele ar trebui să se încadreze în intervalul 128 până la 200 și să aibă o lățime de 250 px și o înălțime de 30 px. De asemenea, ar trebui adăugat un afișaj numeric, sub forma unui control TextBox, pentru a afișa valoarea curentă selectată și a modifica valoarea cursorului. Toate datele din glisoare trebuie salvate ca proprietăți ale obiectului. Pentru a rezolva problema în Windows Forms, designerul ar trebui să stabilească manual proprietățile Minimum, Maximum, Width și Height pe fiecare cursor pentru a seta aspectul, făcând modificări ulterioare foarte grele. La un nivel funcțional, trebuie să adăugăm manipulatoare de evenimente la glisoarele în care proprietatea TextBox.Text ar fi actualizată și, de asemenea, să atribuiți fiecărei valori glisante proprietățiile obiectului respectiv. WPF aduce două concepte noi pentru a rezolva aceste probleme: Styling și legarea datelor.

2.2 Stiluri în WPF

Conceptul de stiluri vă permite să eliminați toate valorile proprietăților din elementele interfeței utilizator individuale și să le combinați într-un stil. Un stil constă dintr-o listă de setters. Dacă aplicați acest stil unui element, acesta setează toate proprietățile cu valorile specificate. Ideea este destul de similară cu CSS (Cascading Styles Sheets) pe care o cunoaștem din dezvoltarea web. Pentru a face stilul accesibil controalelor dvs., trebuie să îl adăugați la resurse. Orice control din WPF are o listă de resurse care este moștenită pentru toate controalele aflate sub arborele vizual. Acesta este motivul pentru care trebuie să specificăm o proprietate x: Key = "myStyle" care definește un identificator de resurse unic. În acest mod, modificările la toate comenzile pot fi făcute într-un singur loc. De exemplu, setarea proprietăților la comenzile cursorului poate fi făcută după cum urmează. La începutul etichetei <Window>, trebuie să se adauge definiția stilului.

```
<Window.Resources>
    <Style x:Key="SliderStyle1" TargetType="{x:Type Slider}">
        <Setter Property="Width" Value="150"/>
        <Setter Property="Height" Value="30"/>
        <Setter Property="Minimum" Value="128"/>
        <Setter Property="Maximum" Value="200"/>
    </Style>
</Window.Resources>
```

Aceasta definește stilul SliderStyle1 care poate fi aplicat apoi comenzilor de tipul Slider. Atribuirea unui stil unui control se face adăugând proprietatea Style la definiția de control:

```
<Slider Name="slider1" Style="{StaticResource SliderStyle1}" />
```

Cursoare multiple pot avea același stil atribuit. Aceasta rezolvă problema modificării unei proprietăți în mai multe controale.

```
<Window x:Class="LabWPF2.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
            presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="195" Width="347">
    <Window.Resources>
        <Style x:Key="SliderStyle1" TargetType="{x:Type Slider}">
            <Setter Property="HorizontalAlignment" Value="Left"/>
            <Setter Property="Minimum" Value="128"/>
            <Setter Property="Maximum" Value="200"/>
            <Setter Property="Width" Value="150"/>
            <Setter Property="Height" Value="30"/>
        </Style>
        <Style x:Key="TextBoxStyle1" TargetType="{x:Type TextBox}">
            <Setter Property="HorizontalAlignment" Value="Left"/>
            <Setter Property="Width" Value="50"/>
            <Setter Property="Height" Value="30"/>
        </Style>
    </Window.Resources>

    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>
        <StackPanel Grid.Column="0">
            <Slider Name="slider1" Style="{StaticResource
                SliderStyle1}" />
            <Slider Name="slider2" Style="{StaticResource
                SliderStyle1}" />
            <Slider Name="slider3" Style="{StaticResource
                SliderStyle1}" />
            <Slider Name="slider4" Style="{StaticResource
                SliderStyle1}" />
            <Slider Name="slider5" Style="{StaticResource
                SliderStyle1}" />
        </StackPanel>
        <StackPanel Grid.Column="1">
            <TextBox Style="{StaticResource TextBoxStyle1}" Text="
                {Binding ElementName=slider1, Path=Value,
                UpdateSourceTrigger=PropertyChanged, Mode=TwoWay}"
                Name="textbox1"/>
            <TextBox Style="{StaticResource TextBoxStyle1}" Text="
                {Binding ElementName=slider2, Path=Value,
                UpdateSourceTrigger=PropertyChanged, Mode=TwoWay}"
                Name="textbox2"/>
            <TextBox Style="{StaticResource TextBoxStyle1}" Text="
                {Binding ElementName=slider3, Path=Value,
                UpdateSourceTrigger=PropertyChanged, Mode=TwoWay}"
```

```
            Name='textbox3'/>
        <TextBox Style='{StaticResource TextBoxStyle1}' Text='
            {Binding ElementName=slider4, Path=Value,
            UpdateSourceTrigger=PropertyChanged, Mode=TwoWay}'
            Name='textbox4'/>
        <TextBox Style='{StaticResource TextBoxStyle1}' Text='
            {Binding ElementName=slider5, Path=Value,
            UpdateSourceTrigger=PropertyChanged, Mode=TwoWay}'
            Name='textbox5'/>
    </StackPanel>
  </Grid>
</Window>
```

Fig.6 Exemplu cursoare multiple

2.3 Data Binding

WPF oferă o modalitate simplă și puternică de actualizare automată a datelor între modelul de bussines și interfața cu utilizatorul. Acest mecanism se numește legarea datelor. De fiecare dată când datele din modelul dvs. de afaceri se modifică, acesta reflectă în mod automat actualizările interfeței utilizator și invers. Aceasta este metoda preferată din WPF pentru aducerea datelor în interfața cu utilizatorul. Legarea datelor poate fi unidirecțională (sursă → țintă sau țintă ← sursă) sau bidirecțională (sursă ↔ țintă).

2.3.1 Data Binding dintre elementele UI (User Interface)

Problema de a afișa valoarea cursorului într-o casetă de text și de a putea modifica datele în ambele direcții este mult mai simplu de rezolvat în WPF. Pentru a lega un control TextBox la un cursor, proprietatea TextBox.Text trebuie legată de proprietatea Slider.Value.

```
<TextBox Height='23' Text='{Binding ElementName=slider1, Path=Value
    , UpdateSourceTrigger=PropertyChanged, Mode=TwoWay}' Name='
    textbox1'/>
```

Pentru a implementa acest lucru, proprietatea Text conține o definiție obligatorie:

- ElementName se referă la controlul legat de TextBox.
- Valoarea Căi reprezintă proprietatea obiectului țintă care urmează să fie afișată.
- UpdateSourceTrigger reprezintă evenimentul care declanșează schimbul de date între comenzi.

Modul stabilește direcția în care trebuie schimbate datele, în acest caz dorim ca cursorul să modifice valoarea TextBox și invers. Mecanismul de legare permite de asemenea formatarea șirului afișat în tehnicile TextBox și de conversie a datelor, care sunt în afara acestui domeniu.

4 Animații

În continuare este prezentată o aplicație ce utilizează animații. În prima figură avem codul XML, iar in cea de-a doua figură avem codul C#.

```
<Window x:Class="Animation.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
            presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Border BorderBrush="Gray" BorderThickness="1" Margin="4"
        Height="374" Width="523">
        <Canvas Height="294" Width="472">
            <Rectangle Fill="#D400CE51" Width="75" Height="69"
                Canvas.Top="100" Canvas.Left="396" Stroke="Brown" /
                    >
            <Ellipse x:Name="ball" Width="50" Height="50" Stroke="
                Brown" Canvas.Top="107" Canvas.Left="90">
                <Ellipse.Fill>
                    <LinearGradientBrush>
                        <GradientStop Color="Pink" Offset="0" />
                        <GradientStop Color="#D400CE51" Offset="
                            0.628" />
                    </LinearGradientBrush>
                </Ellipse.Fill>
                <Ellipse.RenderTransform>
                    <RotateTransform x:Name="ballRotate"
                    CenterX="25" CenterY="25" />
                </Ellipse.RenderTransform>
            </Ellipse>
        </Canvas>
    </Border>
</Window>
```

Fig. 7 Cod XML – exemplu animație

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace Animation
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            CompositionTarget.Rendering += Move;

        }

        Int32 counter = 0;
        void Move(object sender, EventArgs e)
        {
            counter = counter + 2;
            Canvas.SetLeft(ball, counter);
        }
    }
}
```

Fig.8 Cod C# – exemplu animație

Exerciții

1. Creați o fereastră de conectare care deschide un alt formular (utilizatorul și parola sunt citite dintr-un fișier). Al doilea formular va conține o etichetă "Hello + numele utilizatorului!" și un buton de Ieșire.

2. Creați o aplicație care modifică culoarea unui obiect în funcție de anumiți parametri RGB.