

|  |
| --- |
|  |
| Vrei sa fii milionar! |
| Windows presentation foundation |
|  |

Cuprins

[Inițializare 1](#_Toc58505735)

[Informații generale despre jocul “VREI SĂ FII MILIONAR” 1](#_Toc58505736)

[Proiectarea la nivel de concept al aplicației 2](#_Toc58505737)

[Pattern-ul MVVM 2](#_Toc58505738)

[Descrierea aspectelor cheie 3](#_Toc58505739)

[Dezvoltarea aplicației 7](#_Toc58505740)

[Crearea proiectului 7](#_Toc58505741)

[Pregătirea terenului 7](#_Toc58505742)

[Crearea stilurilor 9](#_Toc58505743)

[Conținutul fișierului Styles.xaml 11](#_Toc58505744)

[Modelele aplicației 18](#_Toc58505745)

[Conținutul Game.cs 19](#_Toc58505746)

[Conținutul Question.cs 20](#_Toc58505747)

[Conținutul Result.cs 21](#_Toc58505748)

[Implementarea serviciilor 22](#_Toc58505749)

[Serviciul AudioService 23](#_Toc58505750)

[Conținutul AudioService.cs 23](#_Toc58505751)

[Serviciul DBSERVICE 26](#_Toc58505752)

[Conținutul DBService.cs 26](#_Toc58505753)

[Serviciul gameservice 30](#_Toc58505754)

[Conținutul GameService.cs 30](#_Toc58505755)

[Serviciul Lifelines service 34](#_Toc58505756)

[Conținutul LifelineService.cs 35](#_Toc58505757)

[Ferestrele aplicației 42](#_Toc58505758)

[Adăugarea ferestrelor 42](#_Toc58505759)

[Conținutul baseviewmodel 44](#_Toc58505760)

[Fereastra mainwindow 45](#_Toc58505761)

[Conținut MainWindow.xaml 45](#_Toc58505762)

[Code behind (Mainwindow.xaml.cs) 47](#_Toc58505763)

[fereastra startview 49](#_Toc58505764)

[Conținutul startview.xaml 49](#_Toc58505765)

[Conținutul startview.xaml.cs 51](#_Toc58505766)

[Fereastra rulesview 52](#_Toc58505767)

[Conținutul rulesview.xaml 52](#_Toc58505768)

[Conținutul rulesview.xaml.cs 54](#_Toc58505769)

[Fereastra Gameview 55](#_Toc58505770)

[Conținutul gameview.xaml 56](#_Toc58505771)

[Conținutul gameview.xaml.cs 60](#_Toc58505772)

[Conținutul GAMEVIEWMODEL.CS 61](#_Toc58505773)

[ResultsView 69](#_Toc58505774)

[Conținutul resultsview.xaml 70](#_Toc58505775)

[conținutul Resultsview.xaml.cs 71](#_Toc58505776)

[Conținutul resultsviewmodel.cs 71](#_Toc58505777)

[fereastra Adminview 73](#_Toc58505778)

[Conținutul adminview.xaml 74](#_Toc58505779)

[Conținutul adminview.xaml.cs 75](#_Toc58505780)

[conținutul adminviewmodel.cs 77](#_Toc58505781)

[Fereastra askaudienceview 80](#_Toc58505782)

[Conținutul Askaudienceview.xaml 80](#_Toc58505783)

[Conținutul askaudienceview.xaml.cs 83](#_Toc58505784)

[Conținutul askaudienceviewmodel.cs 84](#_Toc58505785)

[Fereastra phonecallview 86](#_Toc58505786)

[Conținutul phonecallview.xaml 87](#_Toc58505787)

[Conținutul phonecallview.xaml.cs 88](#_Toc58505788)

[Conținutul phonecallviewmodel.cs 88](#_Toc58505789)

[notă de final 91](#_Toc58505790)



# Inițializare

## Informații generale despre jocul “VREI SĂ FII MILIONAR”

**Vrei să fii milionar** este un concurs care oferă premii în bani în schimbul unor răspunsuri corecte la o serie de întrebări cu alegere multiplă și dificultate crescândă. Întrebările sunt cu alegere multiplă: cele 4 răspunsuri posibile sunt marcate cu literele A, B, C și D. Concurentul trebuie să îl aleagă pe cel corect.

După citirea unei întrebări, concurentul poate alege să se retragă cu banii deja câștigați și să nu ofere un răspuns. Dacă răspunde greșit la o întrebare, acesta pierde toți banii câștigați, cu excepția cazurilor în care concurentul a ajuns deja la unul din cele două praguri (de $1 000 și $32 000).

Dacă un concurent este nesigur de răspunsul la o întrebare, el poate folosi o singură dată una sau mai multe variante ajutătoare. Folosirea variantelor ajutătoare nu îl obligă să ofere un răspuns final.

* **50:50 (Fifty-Fifty; cunoscut și ca 50 la 50 în Republica Moldova):** Calculatorul elimină la întâmplare două din cele trei răspunsuri greșite.
* **Întreabă publicul:** Prezentatorul cere spectatorilor să voteze răspunsul pe care îl cred corect folosind keypad-uri. Rezultatele sunt afișate imediat pe ecran în procente.
* **Sună un prieten:** Concurentului i se permite să se consulte telefonic cu o rudă sau un prieten.

Coloana sonoră a emisiunii a fost compusă de Keith și Matthew Strachan și oferă o atmosferă tensionantă. Este prezentă pe aproape tot parcursul emisiunii. Începând cu întrebarea a șaptea, muzica urcă cu un semiton pe măsură ce concurentul evoluează.

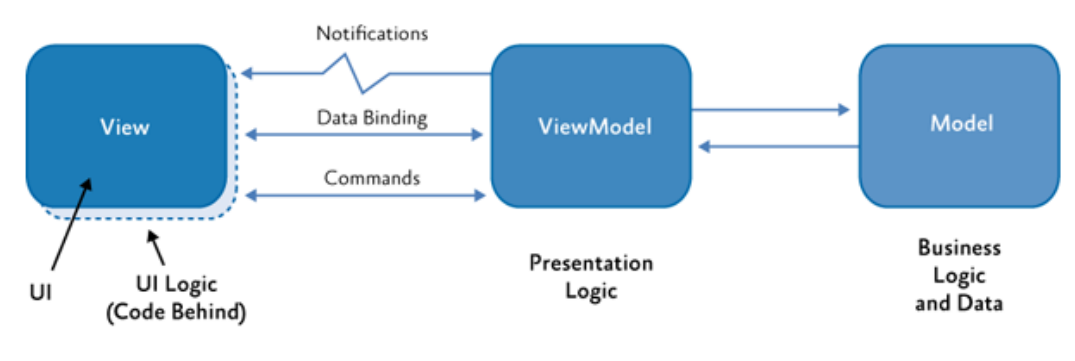
## Proiectarea la nivel de concept al aplicației

**Aspectele cheie ale aplicației:**

* Să aibă o interfață intuitivă și să ofere atmosfera jocului real.
* Să salveze întrebările pe disc și să ofere posibilitatea de a modifica, adăuga, și elimina întrebări.
* Să selecteze întrebări în ordine crescândă a dificultății acestora.
* Să cronometreze timpul – maxim 60 de secunde per întrebare.
* Să conțină coloanele sonore specifice jocului **Vrei Să Fii Milionar** și un buton pentru oprirea / pornirea sunetului.
* Să conțină un panou de administrare cu acces limitat (protejat printr-o parolă) pentru a modifica, șterge sau adăuga întrebări.
* Să conțină 3 variante funcționale de ajutor: 50/50, întreabă publicul și sună un prieten.
* Să conțină o pagină cu regulamentul jocului.
* Să salveze pe disc rezultatele fiecărui joc.
* Să ofere posibilitatea de a te retrage din joc cu suma deja câștigată.

## Pattern-ul MVVM

Pentru a avea un codebase mai ușor de întreținut, vom folosi un model simplificat al pattern-ului **MVVM – Model, View, ViewModel.**



**Acest pattern presupune crearea unei clase ViewModel pentru fiecare fereastră. Clasa respectivă va conține toate proprietățile și metodele ce țin de Presentation Logic,** adică toate datele cu care operează fereastra.

## Descrierea aspectelor cheie

Să aibă o interfață intuitivă și să ofere atmosfera jocului real.

Aplicația va conține o fereastră – MainView, în interiorul căreia se vor randa celelalte ferestre necesare jocului, acestea fiind:

* Fereastra de start
  + Aici vom plasa un input în care jucătorul își va indica numele, un buton de start pentru a lansa jocul și un buton ce ne va direcționa către pagina de vizualizare a regulamentului.
* Fereastra de vizualizare a regulamentului
  + O pagină statică ce va conține informații privind regulamentul jocului.
* Fereastra jocului propriu-zis
  + Aici vor fi plasate mai multe elemente:
    - Un bloc de text aliniat pe centru pentru afișarea textului întrebării
    - Câte un buton pentru fiecare opțiune de răspuns
    - Un contor al timpului scurs
    - Lista de premii în care premiul întrebării curente va fi evidențiat
    - Câte un buton pentru fiecare opțiune ajutătoare
    - Un buton pentru retragerea din joc
    - Un panou ce va conține explicațiile referitoare răspunsului corect la întrebarea curentă. Acest panou va fi vizibil doar în cazul în care jucătorul greșește răspunsul sau decide să părăsească jocul.
* Fereastra pentru vizualizarea rezultatelor în urma intrerogării publicului
  + Aceasta va conține 4 bare ce va afișa în mod grafic procentajul din public care a votat pentru fiecare opțiune în parte
* Fereastra pentru dialogarea cu prietenul telefonat
  + Aceasta va conține un input în care jucătorul va indica numele persoanei pe care dorește să o telefoneze, după care va fi afișat convorbirea propriu zisă.
* Fereastra ce va afișa rezultatul jocului
  + Aici se va găsi premiul câștigat, timpul total scurs și media timpului scurs pentru fiecare întrebare.
* Panoul de administrare
  + Aceasta va conține un input pentru indicarea parolei și lista de întrebări cu instrumentele necesare pentru modificarea, adăugarea sau eliminarea acestora.

**Să salveze întrebările pe disc și să ofere posibilitatea de a modifica, adăuga, și elimina întrebări.**

Fiecare intrebare va conține următoarele proprietăți:

* Codul unic de indentificare (ID-ul intrebarii este **Unix time stamp,** adică numărul total de secunde scurs din momentul începerii epocii unix, 1 Ianuarie 1970 <https://www.unixtimestamp.com>, până în momentul în care a fost creată întrebarea)
* Textul întrebării
* Nivelul de dificultate
* Opțiunile de răspuns
* ID-ul opțiunii corecte
* Explicații (Opțional)

Pentru stocarea întrebărilor s-a decis utilizarea fișierelor XML.

**Avantaje:**

* Fișierele XML sunt ușor de citit atât pentru calculator, cât și pentru oameni, ceea ce ne permite să adăugăm întrebări mai rapid direct în fișier, folosind orice editor de text.
* Informația stocată în fișiere XML poate fi ușor distribuită.

**Dezavantaje**

* Utilizarea fișierelor text pe post de bază de date este un locru foarte anevoios și ineficient, în special atunci când avem nevoie să stocâm cantități mari de date sau când este nevoie de împărțirea datelor cu care operează aplicația în diverse tabele.
* Înregistrările din fișierele XML sunt accesate secvențial, ceea ce este costisitor în materie de timp. În cadrul utilizării unei baze de date SQL, înregistrările pot fi indexate și accesate mult mai rapid.
* Asupra fișierelor XML nu pot fi create interogări complexe.

În cazul nostru ne putem permite să utilizăm fișierele XML pentru a stoca datele datorită cantității relativ mici de informație de care avem nevoie, însă pentru proiectele mai mari este contra-indicată această abordare.

Vom avea nevoie de 2 fișiere XML:

Questions.xml – pentru stocarea întrebărilor

Results.xml – pentru stocarea rezultatelor.

**Să selecteze întrebări în ordine crescândă a dificultății acestora.**

Dat fiind faptul că, despre fiecare întrebare cunoaștem nivelul acesteia de dificultate, putem extrage din fișierul XML întrebări de diferite dificultăți astfel încât primele să fie cele mai ușoare, urmând ca dificultatea acestora să crească odată ce ne apropiem de premiul cel mare.

Pentru a opera mai ușor cu nivelele de dificultate, vom crea o enumerație (**enum**) cu următoarele elemente:

* Easy – Întrebări ușoare: Primele 4 întrebări vor fi de acest tip
* Medium – Întrebări medii: Începând cu întrebarea 5 și până la a 8-a
* Hard – Întrebări grele: Acestea vor fi întâlnite pe pozițiile 9 – 13
* Einstein – Întrebări extrem de grele: Utileme 2 întrebări (pentru $500 000 și $1 000 000) vor fi de acest tip.

Să cronometreze timpul – maxim 60 de secunde per întrebare

ViewModel-ul jocului va conține un element de tip Timer (DispatcherTimer) cu tick-ul de 1 secundă. Contorul secundelor va fi afișat pe suprafața ferestrei. În cazuș în care expiră cele 60 de secunde iar jucătorul nu a oferit un răspuns, acesta va părăsi jocul (Game Over).

Să conțină coloanele sonore specifice jocului Vrei Să Fii Milionar și un buton pentru oprirea / pornirea sunetului.

În dependență de acțiunile întreprinse (A oferit un răspuns corect / greșit, a început jocul etc.) vor fi lansate coloanele audio respective. Deasmenea, în colțul ferestrei va fi amplasat un buton ce va opri sau porni sunetul aplicației.

Să conțină un panou de administrare cu acces limitat (protejat printr-o parolă) pentru a modifica, șterge sau adăuga întrebări

Lângă butonul de activare / dezactivare a sunetului va fi amplasat încă un buton pentru lansarea panoului de administrare. În momentul lansaării, utilizatorul este rugat să introducă parola ce acces. Inițial, parola va fi **admin**. Dacă a fost indicată parola validă atunci va fi creată o cerere de a extrage toate întrebările disponibile din fișierul XML.

Să conțină 3 variante funcționale de ajutor: 50/50, întreabă publicul și sună un prieten

Butonale pentru activarea acestor opțiuni vor fi afișate deasupra listei de premii, la fel ca în show-ul original.

Opțiunea 50/50 va exclude 2 opțiuni de răspuns greșite

Opțiunea Întreabă Publicul va lansa o altă fereastră în care vor fi afișate rezultatele sondajului

Opțiunea Sună un Prieten va simula o discuție dintre 2 prieteni.

Să salveze pe disc rezultatele fiecărui joc

În fișierul Results.xml vor fi stocate rezultatele fiecărui joc. Pentru fiecare joc se vor salva următoarele date:

* Numele jucătorului
* Premiul final
* Numărul de întrebări răspunse corect
* Timpul scurs (total)
* Timpul mediu pentru fiecare întrebare.



# Dezvoltarea aplicației

## Crearea proiectului

Tipul proiectului va fi **WPF APP (.NET Core) – C#.**

Numele proiectului - **WhoWantsToBeAMillionaire**

## Pregătirea terenului

Mai întâi vom face rost de resursele pe care le vom utiliza în procesul de dezvoltare.

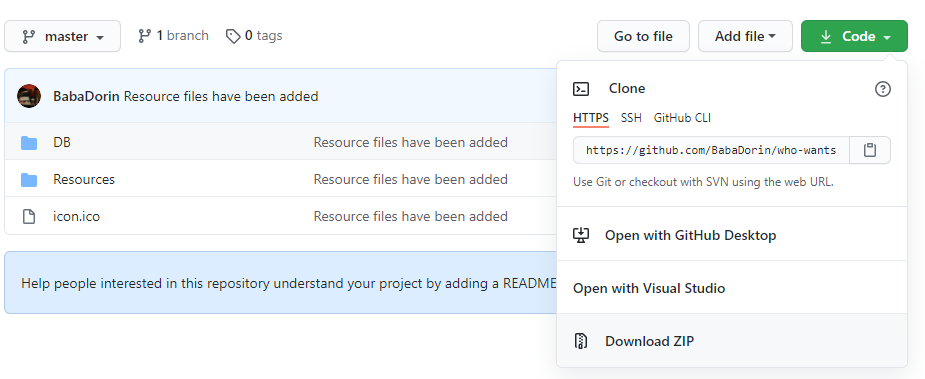
Toate resursele se află în repozitoriul

[www.github.com/babadorin/who-wants-to-millionaire-resources](http://www.github.com/babadorin/who-wants-to-millionaire-resources)

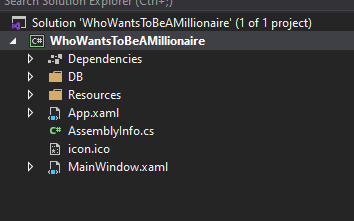
Le puteti descărca folosind git bash prin comanda

**git clone** [**https://github.com/babadorin/who-wants-to-millionaire-resources**](https://github.com/babadorin/who-wants-to-millionaire-resources)

Sau de pe platforma github.com



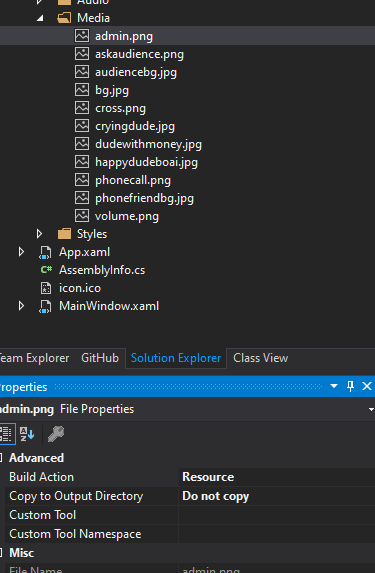
Mapele DB, Resources și iconița icon.ico vor fi plasate în directoriul proiectului nostru. Într-un final trebuie să avem aceastră structură:



În mapa DB se conțin 2 fișiere XML, unul ce conține câteva întrebări și al doilea – pentru stocarea rezultatelor.

În Mapa Resources se află alte 3 mape:

* Audio – Conține coloanele sonore pe care le vom folosi.
* Media – Imagini
* Styles – Stilurile pe care imediat le vom crea.



**IMPORTANT!**

Pentru fiecare imagine din Resources/Media, setați

Build Action**: Resource**

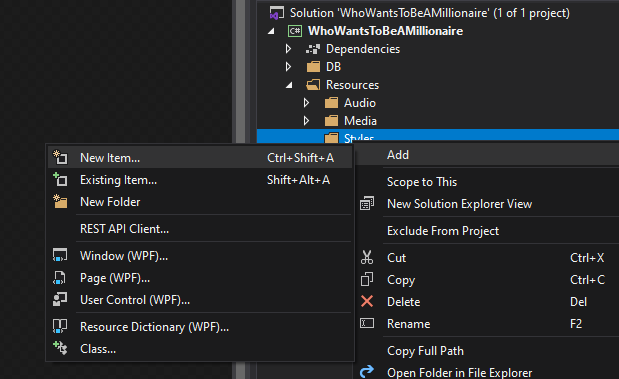
Copy to Output Directory**: Do not copy**

Repetați același lucru și pentru **icon.ico**

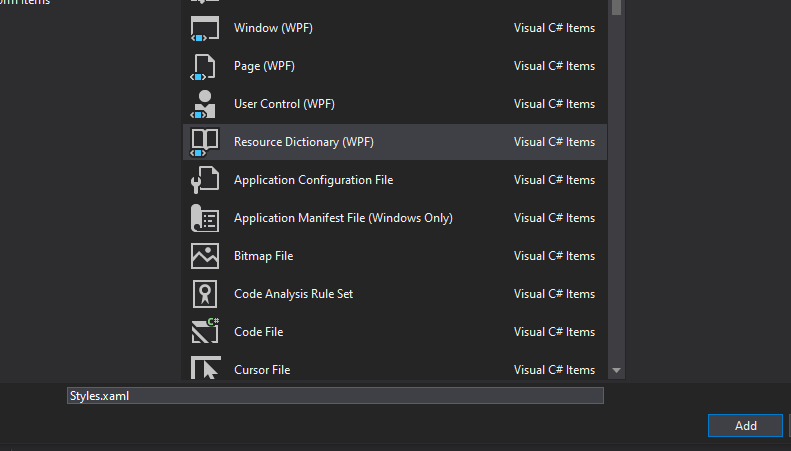
## Crearea stilurilor

Pentru a nu supra-aglomera conținutul XAML al paginilor noastre, vom crea stilurile într-un fișier aparte – ResourceDictionary.

Click dreapta pe mapa Styles – Add – New Item..



**Il denumim Styles.xaml** - Add



Urmează să mergem în App.xaml și să creăm legătura cu dicționarul nostru de resurse.

<Application x:Class="WhoWantsToBeAMillionaire.App"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="clr-namespace:WhoWantsToBeAMillionaire"

StartupUri="MainWindow.xaml">

<Application.Resources>

<ResourceDictionary Source="Resources/Styles/Styles.xaml" />

</Application.Resources>

</Application>

Toate stilurile pe care le vom crea în fișierul Styles.xaml for fi accesibile din orice pagina a aplicației.

### Conținutul fișierului Styles.xaml

<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

                    xmlns:local="clr-namespace:WhoWantsToBeAMillionaire.Styles">

    <!--TextBlock Styles-->

    <Style x:Key="BaseTextStyle" TargetType="TextBlock">

        <Setter Property="Foreground" Value="White"/>

        <Setter Property="TextWrapping" Value="Wrap"/>

        <Setter Property="FontSize" Value="15"/>

    </Style>

    <Style x:Key="Heading" TargetType="TextBlock" BasedOn="{StaticResource BaseTextStyle}">

        <Setter Property="FontSize" Value="25"/>

        <Setter Property="HorizontalAlignment" Value="Center"/>

    </Style>

    <Style x:Key="LargeText" TargetType="TextBlock" BasedOn="{StaticResource BaseTextStyle}">

        <Setter Property="FontSize" Value="20"/>

    </Style>

    <Style x:Key="NormalText" TargetType="TextBlock" BasedOn="{StaticResource BaseTextStyle}">

        <Setter Property="Width" Value="600"/>

        <Setter Property="TextAlignment" Value="Center"/>

    </Style>

    <Style x:Key="SmallText" TargetType="TextBlock" BasedOn="{StaticResource BaseTextStyle}">

        <Setter Property="FontSize" Value="12"/>

    </Style>

    <Style x:Key="WarningText" TargetType="TextBlock" BasedOn="{StaticResource BaseTextStyle}">

        <Setter Property="Foreground" Value="Red"/>

    </Style>

    <Style x:Key="SimpleTextBox" TargetType="TextBox">

        <Setter Property="Foreground" Value="White"/>

        <Setter Property="Background" Value="Black"/>

        <Setter Property="FontSize" Value="25"/>

        <Setter Property="VerticalContentAlignment" Value="Center"/>

        <Setter Property="HorizontalContentAlignment" Value="Center"/>

        <Setter Property="Width" Value="400"/>

        <Setter Property="Height" Value="60"/>

        <Setter Property="BorderBrush" Value="DarkBlue"/>

        <Setter Property="BorderThickness" Value="4"/>

    </Style>

    <Style x:Key="reply" TargetType="TextBlock">

        <Setter Property="Foreground" Value="White"/>

        <Setter Property="FontSize" Value="18"/>

        <Setter Property="DockPanel.Dock" Value="Top"/>

        <Setter Property="TextWrapping" Value="Wrap"/>

        <Setter Property="TextAlignment" Value="Center"/>

        <Style.Triggers>

            <EventTrigger RoutedEvent="Loaded">

                <BeginStoryboard>

                    <Storyboard>

                        <DoubleAnimation Storyboard.TargetProperty="Opacity"

                                         From="0" To="1"

                                         Duration="0:0:1"/>

                    </Storyboard>

                </BeginStoryboard>

            </EventTrigger>

        </Style.Triggers>

    </Style>

    <Style x:Key="friendReply" BasedOn="{StaticResource reply}" TargetType="TextBlock">

        <Setter Property="HorizontalAlignment" Value="Left"/>

        <Setter Property="Margin" Value="10 10 40 10"/>

    </Style>

    <Style x:Key="playerReply" BasedOn="{StaticResource reply}" TargetType="TextBlock">

        <Setter Property="HorizontalAlignment" Value="Right"/>

        <Setter Property="Margin" Value="40 10 10 10"/>

    </Style>

    <Style x:Key="optionText" TargetType="TextBlock">

        <Setter Property="FontSize" Value="17"/>

        <Setter Property="Foreground" Value="White"/>

        <Setter Property="TextAlignment" Value="Center"/>

        <Setter Property="FontWeight" Value="Bold"/>

        <Setter Property="TextWrapping" Value="Wrap"/>

        <Style.Triggers>

            <Trigger Property="Text" Value="A: ">

                <Setter Property="IsEnabled" Value="False"/>

            </Trigger>

            <Trigger Property="Text" Value="B: ">

                <Setter Property="IsEnabled" Value="False"/>

            </Trigger>

            <Trigger Property="Text" Value="C: ">

                <Setter Property="IsEnabled" Value="False"/>

            </Trigger>

            <Trigger Property="Text" Value="D: ">

                <Setter Property="IsEnabled" Value="False"/>

            </Trigger>

        </Style.Triggers>

    </Style>

    <!--UserControl styles-->

    <Style x:Key="NoWin" TargetType="UserControl">

        <Setter Property="Background">

            <Setter.Value>

                <ImageBrush ImageSource="pack://application:,,,/WhoWantsToBeAMillionaire;component/Resources/Media/cryingdude.jpg" Stretch="UniformToFill" Opacity="0.5"/>

            </Setter.Value>

        </Setter>

    </Style>

    <Style x:Key="Win" TargetType="UserControl">

        <Setter Property="Background">

            <Setter.Value>

                <ImageBrush ImageSource="pack://application:,,,/WhoWantsToBeAMillionaire;component/Resources/Media/happydudeboai.jpg" Stretch="UniformToFill" Opacity="0.7"/>

            </Setter.Value>

        </Setter>

    </Style>

    <Style x:Key="BigWin" TargetType="UserControl">

        <Setter Property="Background">

            <Setter.Value>

                <ImageBrush ImageSource="pack://application:,,,/WhoWantsToBeAMillionaire;component/Resources/Media/dudewithmoney.jpg" Stretch="UniformToFill" Opacity="0.5"/>

            </Setter.Value>

        </Setter>

    </Style>

    <!--Grid styles-->

    <Style x:Key="DangerGrid" TargetType="Grid">

        <Setter Property="Background">

            <Setter.Value>

                <LinearGradientBrush StartPoint="0,0" EndPoint="1,1" Opacity="0.3">

                    <GradientStop Color="Red" Offset="0.15" />

                    <GradientStop Color="Black" Offset="0.30" />

                    <GradientStop Color="Black" Offset="0.70" />

                    <GradientStop Color="Red" Offset="0.85" />

                </LinearGradientBrush>

            </Setter.Value>

        </Setter>

    </Style>

    <Style x:Key="SuccessGrid" TargetType="Grid">

        <Setter Property="Background">

            <Setter.Value>

                <LinearGradientBrush StartPoint="0,0" EndPoint="1,1"  Opacity="0.4">

                    <GradientStop Color="Green" Offset="0.15" />

                    <GradientStop Color="Black" Offset="0.30" />

                    <GradientStop Color="Black" Offset="0.70" />

                    <GradientStop Color="Green" Offset="0.85" />

                </LinearGradientBrush>

            </Setter.Value>

        </Setter>

    </Style>

    <Style x:Key="RightPanelGrid" TargetType="Grid">

        <Setter Property="Background">

            <Setter.Value>

                <SolidColorBrush Color="DarkBlue" Opacity="0.4"/>

            </Setter.Value>

        </Setter>

    </Style>

    <!--Label styles-->

    <Style x:Key="PrizeItemBaseStyle" TargetType="Label">

        <Setter Property="FontSize" Value="20"/>

        <Setter Property="FontWeight" Value="Bold"/>

    </Style>

    <Style x:Key="PrizeItemStyle" TargetType="Label" BasedOn="{StaticResource PrizeItemBaseStyle}">

        <Setter Property="Foreground" Value="#eca609"/>

    </Style>

    <Style x:Key="SafePrizeItemStyle" TargetType="Label" BasedOn="{StaticResource PrizeItemBaseStyle}">

        <Setter Property="Foreground" Value="White"/>

    </Style>

    <Style x:Key="CurrentPrize" TargetType="Label" BasedOn="{StaticResource PrizeItemBaseStyle}">

        <Setter Property="Foreground" Value="Black"/>

        <Setter Property="Background" Value="#eca609"/>

    </Style>

    <Style x:Key="timerStyle" TargetType="Label">

        <Setter Property="FontSize" Value="30"/>

        <Setter Property="FontWeight" Value="Bold"/>

        <Setter Property="Foreground" Value="White"/>

    </Style>

    <Style x:Key="AskAudienceLabel" TargetType="Label">

        <Setter Property="HorizontalAlignment" Value="Center"/>

        <Setter Property="VerticalAlignment" Value="Center"/>

        <Setter Property="Foreground" Value="White"/>

        <Setter Property="FontSize" Value="30"/>

        <Setter Property="FontWeight" Value="Bold"/>

    </Style>

    <!--Button styles-->

    <Style x:Key="NormalPolygon" TargetType="{x:Type Button}">

        <Setter Property="Background" Value="Black"/>

        <Setter Property="Foreground" Value="White"/>

        <Setter Property="BorderBrush" Value="Blue"/>

        <Setter Property="FontWeight" Value="Bold"/>

        <Setter Property="Template">

            <Setter.Value>

                <ControlTemplate TargetType="{x:Type Button}">

                    <Grid>

                        <Polygon Fill="{TemplateBinding Background}"

                                 Points="0,30 60,5 360,5 420,30 360,55 60,55"

                                 Stretch="Uniform"

                                 Stroke="{TemplateBinding BorderBrush}"

                                 StrokeThickness="2"></Polygon>

                        <ContentPresenter HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}" RecognizesAccessKey="True" SnapsToDevicePixels="{TemplateBinding SnapsToDevicePixels}" VerticalAlignment="{TemplateBinding VerticalContentAlignment}"/>

                    </Grid>

                    <ControlTemplate.Triggers>

                        <Trigger Property="IsMouseOver" Value="True">

                            <Setter Property="Background" Value="Blue"/>

                            <Setter Property="Foreground" Value="White"/>

                        </Trigger>

                        <Trigger Property="IsPressed" Value="True">

                            <Setter Property="Background" Value="Black"/>

                            <Setter Property="Foreground" Value="White"/>

                        </Trigger>

                    </ControlTemplate.Triggers>

                </ControlTemplate>

            </Setter.Value>

        </Setter>

    </Style>

    <Style x:Key="EmptyButton" TargetType="Button">

        <Setter Property="Background" Value="Transparent"/>

        <Setter Property="BorderThickness" Value="0"/>

        <Setter Property="Template">

            <Setter.Value>

                <ControlTemplate TargetType="{x:Type Button}">

                    <Grid Background="{TemplateBinding Background}">

                        <ContentPresenter x:Name="MyContentPresenter"

                                          Content="{TemplateBinding Content}"

                                          HorizontalAlignment="Center"

                                          VerticalAlignment="Center" />

                    </Grid>

                </ControlTemplate>

            </Setter.Value>

        </Setter>

        <Style.Triggers>

            <Trigger Property="IsEnabled" Value="False">

                <Setter Property="Background" Value="Transparent"/>

            </Trigger>

            <Trigger Property="IsMouseOver" Value="True">

                <Setter Property="Background" Value="White"/>

                <Setter Property="Foreground" Value="Black"/>

            </Trigger>

        </Style.Triggers>

    </Style>

    <Style x:Key="StartViewButtons" BasedOn="{StaticResource NormalPolygon}" TargetType="Button">

        <Setter Property="Width" Value="500"/>

        <Setter Property="FontSize" Value="20"/>

        <Style.Triggers>

            <Trigger Property="IsEnabled" Value="False">

                <Setter Property="Foreground" Value="Gray"/>

            </Trigger>

        </Style.Triggers>

    </Style>

    <Style x:Key="OptionPolygon" BasedOn="{StaticResource NormalPolygon}" TargetType="Button">

        <Setter Property="FontSize" Value="18"/>

        <Style.Triggers>

            <Trigger Property="IsMouseOver" Value="True">

                <Setter Property="Background" Value="CadetBlue"/>

                <Setter Property="Cursor" Value="Hand"/>

            </Trigger>

        </Style.Triggers>

    </Style>

    <Style x:Key="OptionSelected" BasedOn="{StaticResource OptionPolygon}" TargetType="Button">

        <Setter Property="Background" Value="CadetBlue"/>

        <Style.Triggers>

            <Trigger Property="Polygon.IsMouseOver" Value="True">

                <Setter Property="Background" Value="CadetBlue"/>

            </Trigger>

        </Style.Triggers>

    </Style>

    <Style x:Key="RightOptionSelected" BasedOn="{StaticResource OptionPolygon}" TargetType="Button">

        <Setter Property="Background" Value="ForestGreen"/>

        <Style.Triggers>

            <Trigger Property="Polygon.IsMouseOver" Value="True">

                <Setter Property="Background" Value="ForestGreen"/>

            </Trigger>

        </Style.Triggers>

    </Style>

    <Style x:Key="WrongOptionSelected" BasedOn="{StaticResource OptionPolygon}" TargetType="Button">

        <Setter Property="Background" Value="DarkRed"/>

        <Style.Triggers>

            <Trigger Property="Polygon.IsMouseOver" Value="True">

                <Setter Property="Background" Value="DarkRed"/>

            </Trigger>

        </Style.Triggers>

    </Style>

    <Style x:Key="EllipseButton" TargetType="Button">

        <Setter Property="Margin" Value="6"/>

        <Setter Property="FontWeight" Value="Bold"/>

        <Setter Property="Foreground" Value="White"/>

        <Setter Property="FontSize" Value="20"/>

        <Setter Property="HorizontalAlignment" Value="Stretch"/>

        <Setter Property="VerticalAlignment" Value="Stretch"/>

        <Setter Property="Template">

            <Setter.Value>

                <ControlTemplate TargetType="{x:Type Button}">

                    <Grid>

                        <Ellipse Fill="Black" Stroke="DarkBlue" StrokeThickness="4"

                                 x:Name="theEllipse"/>

                        <ContentPresenter VerticalAlignment="Center" HorizontalAlignment="Center"/>

                    </Grid>

                    <ControlTemplate.Triggers>

                        <Trigger Property="IsMouseOver" Value="True">

                            <Setter Property="Fill" Value="DarkBlue" TargetName="theEllipse"/>

                        </Trigger>

                        <Trigger Property="IsPressed" Value="True">

                            <Setter Property="Fill" Value="Blue" TargetName="theEllipse"/>

                            <Setter Property="Stroke" Value="Blue" TargetName="theEllipse"/>

                        </Trigger>

                    </ControlTemplate.Triggers>

                </ControlTemplate>

            </Setter.Value>

        </Setter>

    </Style>

    <Style x:Key="btBackStyle" TargetType="Button">

        <Setter Property="Height" Value="60"/>

        <Setter Property="Width" Value="80"/>

        <Setter Property="Background" Value="Transparent"/>

        <Setter Property="BorderThickness" Value="0"/>

        <Setter Property="FontWeight" Value="Bold"/>

        <Setter Property="Foreground" Value="white"/>

        <Setter Property="FontSize" Value="50"/>

        <Setter Property="HorizontalAlignment" Value="Left"/>

        <Setter Property="VerticalAlignment" Value="Top"/>

        <Setter Property="Content" Value="&lt;"/>

        <Setter Property="FontFamily" Value="Calibri"/>

        <Setter Property="Margin" Value="10, 45, 10, 10"/>

        <Setter Property="VerticalContentAlignment" Value="Center"/>

        <Setter Property="HorizontalContentAlignment" Value="Center"/>

    </Style>

    <!--Polygon Styles-->

    <Style x:Key="QuestionPolygon" TargetType="Polygon">

        <Setter Property="Points" Value="0,40 60,10 650,10 710,40 650,70 60,70"/>

        <Setter Property="Fill" Value="Black"/>

        <Setter Property="Stretch" Value="Uniform"/>

        <Setter Property="Stroke" Value="Blue"/>

        <Setter Property="StrokeThickness" Value="2"/>

    </Style>

    <Style x:Key="QuestionStyle" TargetType="{x:Type TextBox}">

        <Setter Property="TextWrapping" Value="WrapWithOverflow"/>

        <Setter Property="FontSize" Value="15"/>

        <Setter Property="FontWeight" Value="Bold"/>

        <Setter Property="TextAlignment" Value="Center"/>

        <Setter Property="Foreground" Value="White"/>

        <Setter Property="Background" Value="Transparent"/>

        <Setter Property="BorderThickness" Value="0"/>

        <Setter Property="HorizontalAlignment" Value="Center"/>

        <Setter Property="VerticalAlignment" Value="Center"/>

        <Setter Property="Cursor" Value="Arrow"/>

        <Setter Property="Width" Value="600"/>

    </Style>

    <!--Rectangle Styles-->

    <Style x:Key="AskAudienceGraphBar" TargetType="Rectangle">

        <Setter Property="Height" Value="0"/>

        <Setter Property="VerticalAlignment" Value="Bottom"/>

        <Setter Property="Margin" Value="20 0"/>

        <Setter Property="Fill">

            <Setter.Value>

                <LinearGradientBrush>

                    <GradientStop Color="#22fefd" Offset="0.25"/>

                    <GradientStop Color="#6666f8" Offset="0.5"/>

                    <GradientStop Color="BlueViolet" Offset="1"/>

                </LinearGradientBrush>

            </Setter.Value>

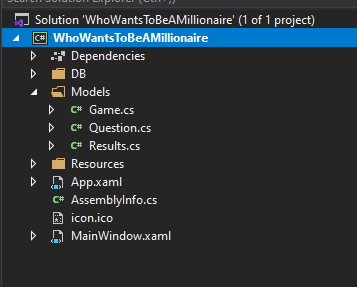
        </Setter>

    </Style>

</ResourceDictionary>

## Modelele aplicației

Urmează să adăugăm modelele (clasele) cu care vom opera. Creăm un folder cu denumirea **Models** în proiectul nostru și creăm 3 clase:



### Conținutul Game.cs

using System;

using System.Collections.Generic;

using WhoWantsToBeAMillionaireTest.Models;

namespace WhoWantsToBeAMillionaire.Models

{

    class Game

    {

        public string PlayerName { get; set; }

        // Premiul curent al jucatorului

        public string PrizeSoFar { get; set; } = "$ 0";

        // Intrebarile propriu zise

        public List<Question> Questions { get; set; }

        // Timpul total scurs

        public TimeSpan TotalEllapsedTime { get; set; } = TimeSpan.FromSeconds(0);

        public static List<string> PrizeList { get; set; } = new List<string>

            {

                "$ 100", "$ 200", "$ 300", "$ 500",

                "$ 1 000", "$ 2 000", "$ 4 000", "$ 8 000",

                "$ 16 000", "$ 32 000", "$ 64 000", "$ 125 000",

                "$ 250 000", "$ 500 000", "$ 1 000 000"

            };

        public static List<string> SafePoints = new List<String>

            {

                "$ 1 000", "$ 32 000", "$ 1 000 000"

            };

    }

}

### Conținutul Question.cs

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace WhoWantsToBeAMillionaire.Models

{

    public enum DifficultyLevel

    {

        Easy = 0, // 100 - 300                  | 4 intrebari

        Medium = 1, // 500 - 16 000             | 5 intrebari

        Hard = 2, // 32 000 - 250 000           | 4 intrebari

        Einstein = 3 // 500 000 - 1 000 000     | 2 intrebari

    }

    public class Question

    {

        public string QuestionId { get; set; }

        // Intrebarea propriu zisa (Textul intrebarii)

        [Required]

        public string QuestionText { get; set; }

        // Nivelul de dificultate al intrebarii

        public DifficultyLevel DifficultyLevel { get; set; }

        // Optiunile de raspuns

        public List<string> Options { get; set; }

        // Indicele raspunsului corect

        public int CorrectOptionIndex { get; set; }

        // Explicatii suplimentare (In cazul in care jucatorul alege o optiune gresita,

        // va fi afiat care a fost raspunsul corect, impreuna cu o explicatie)

        // Campul este optional

        public string Explanations { get; set; }

    }

}

### Conținutul Result.cs

using System;

namespace WhoWantsToBeAMillionaire.Models

{

    public class Results

    {

        // Numele jucatorului

        public string PlayerName { get; set; }

        // Premiul final

        public string FinalPrize { get; set; }

        // Numarul de raspunsuri corecte

        public int CorrectAnswers { get; set; }

        // Timpul scurs (total)

        public TimeSpan ElapsedTime { get; set; }

        // Cat timp s-a scurs in medie pentru fiecare intrebare

        public TimeSpan MediumTimeSpanPerQuestion { get; set; }

    }

}

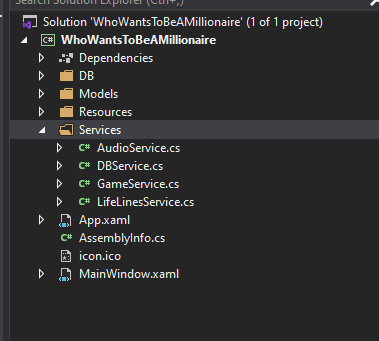
## Implementarea serviciilor

După crearea modelelor putem trece la implementarea serviciilor.

Vom crea 4 servicii:

* **AudioService** – Conține logica pe partea sonoră a aplicației.
* **DBService** – pentru lucrul cu fișierele XML
* **GameService** – Logica jocului propriu zis (Extragerea și selectarea întrebărilor, verificarea răspunsului dat de către jucător, construirea rezultatului final etc.
* **LifeLinesService** – Lifelines este termenul în engleză pentru variantele ajutătoare (50/50, întreabă audiența, sună un prieten). Aici se conține toată logica ce ține de modul de funcționare a acestor variante ajutătoare.

Creăm un nou folder cu numele Services în proiectul nostru în care adăugăm următoarele clase:



### Serviciul AudioService

Pentru a consuma mai puține resurse vom folosi clasa SoundPlayer în lucrul cu coloanele sonore. Pentru aplicațiile mai complexe sau care se axează mai mult pe sunete și efecte sonore, se recomandă utilizarea clasei MediaPlayer. Spre exemplu, aceasta din urmă ne permite să modificăm volumul sunetului, viteza de redare, să punem pauză redării și alte funcționalități pe care SoundPlayer nu le oferă.

Serviciul AudioService se va ocupa de următoarele lucruri:

* Va ține cont dacă sunetul este activat / dezactivat
* Va conține toate coloanele sonore ce pot fi utilizate
* Va reda sunetul dorit
* Va identifica și reda melodia de fundal specifică întrebării (Melodia pentru intervalul $100 - $1000 diferă de melodia din intervalul $2000 – $32000)

Pentru a avea o singură instanță a acestei clase, vom utiliza patternul **SINGLETON**. De fiecare dată când vom avea nevoie de acest serviciu vom apela funcția GetInstance() ce ne va returna instanța deja folosită, și nu constructorul metodei ce ne va returna mereu o instață noua a acestei clase.

### Conținutul AudioService.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Media;

using System.Text;

using System.Windows;

namespace WhoWantsToBeAMillionaire.Services

{

    class AudioService

    {

        public bool AudioOn { get; set; }

        public SoundPlayer MainTheme { get; set; }

        public SoundPlayer LetsPlay { get; set; }

        public SoundPlayer Music1001000 { get; set; }

        public SoundPlayer Music2k32k { get; set; }

        public SoundPlayer Music64k500k { get; set; }

        public SoundPlayer Music500k1kk { get; set; }

        public SoundPlayer FinalAnswer { get; set; }

        public SoundPlayer CorrectAnswer { get; set; }

        public SoundPlayer PhoneAFriend { get; set; }

        public SoundPlayer WrongAnswer { get; set; }

        private SoundPlayer LastSound { get; set; }

private string audioDirectory = Path.GetFullPath(Path.Combine(AppDomain.CurrentDomain.BaseDirectory, @"..\..\..\", @"Resources\Audio\"));

        // Implementarea pattern-ului Singleton

        private static AudioService instance;

        private AudioService()

        {

            AudioOn = true;

            MainTheme = new SoundPlayer(Path.Combine(audioDirectory, "maintheme.wav"));

            LetsPlay = new SoundPlayer(Path.Combine(audioDirectory, "letsplay.wav"));

            Music1001000 = new SoundPlayer(Path.Combine(audioDirectory, "1001000.wav"));

            Music2k32k = new SoundPlayer(Path.Combine(audioDirectory, "2k32k.wav"));

            Music64k500k = new SoundPlayer(Path.Combine(audioDirectory, "64k500k.wav"));

            Music500k1kk = new SoundPlayer(Path.Combine(audioDirectory, "500k1kk.wav"));

            FinalAnswer = new SoundPlayer(Path.Combine(audioDirectory, "finalanswer.wav"));

            CorrectAnswer = new SoundPlayer(Path.Combine(audioDirectory, "correctanswer.wav"));

            WrongAnswer = new SoundPlayer(Path.Combine(audioDirectory, "wronganswer.wav"));

            PhoneAFriend = Music500k1kk;

        }

        public static AudioService GetInstace()

        {

            if (instance == null)

                instance = new AudioService();

            return instance;

        }

        public void PlayAudio(SoundPlayer soundPlayer)

        {

            // Oprește sunetul precedent și pornește noua coloană sonoră

            LastSound = soundPlayer;

            if (!AudioOn)

                return;

            StopPlaying();

            // Exista sunete one-shot și melodii ce cântă incontinuu pana cand nu sunt oprite.

            if(soundPlayer == LetsPlay || soundPlayer == FinalAnswer || soundPlayer == CorrectAnswer || soundPlayer == WrongAnswer)

            {

                soundPlayer.Play();

            }

            else

            {

                soundPlayer.PlayLooping();

            }

        }

        public void PlayAudioAccordingToPriceRange(int questionId)

        {

            // In dependenta de numarul de oridine (questionId) este aleasa melodia corespunzatoare.

            if(questionId <= 4)

            {

                LastSound = Music1001000;

                PlayAudio(Music1001000);

                return;

            }

            if(questionId <= 9)

            {

                LastSound = Music2k32k;

                PlayAudio(Music2k32k);

                return;

            }

            if (questionId <= 12)

            {

                LastSound = Music64k500k;

                PlayAudio(Music64k500k);

                return;

            }

            PlayAudio(Music500k1kk);

            LastSound = Music500k1kk;

        }

        public void StopPlaying()

        {

            // Opreste ultimul sunet pornit

            if(LastSound != null && LastSound != FinalAnswer && LastSound != CorrectAnswer)

                LastSound.Stop();

        }

        public void ToggleVolume()

        {

            // Opreste / porneste sunetul.

            AudioOn = !AudioOn;

            if (!AudioOn)

                StopPlaying();

            else if(LastSound != null)

                    LastSound.PlayLooping();

        }

    }

}

### Serviciul DBSERVICE

Acest serviciu se va ocupa de comunicarea cu fișiere XML. Acesta conține adresele fișierelor XML și metodele de care avem nevoie pentru a citi, adăuga, elimina sau modifica întrebări (CRUD Operations). Serviciul mai conține și o clasă ajutătoate – XmlHelper.

### Conținutul DBService.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Windows;

using System.Xml;

using System.Xml.Serialization;

using WhoWantsToBeAMillionaire.Models;

namespace WhoWantsToBeAMillionaire.Services

{

    class DBService

    {

        private static string baseDirectory = Path.GetFullPath(Path.Combine(AppDomain.CurrentDomain.BaseDirectory, @"..\..\..\"));

        private static string xmlResultsPath = Path.Combine(baseDirectory, @"DB\Results.xml");

        private static string xmlQuestionsPath = Path.Combine(baseDirectory, @"DB\Questions.xml");

        // Metodele ce definesc operatiile CRUD: Create, Read, Update, Delete

        public static void SaveResults(Results results)

        {

            XmlHelper.AppendToXmlFile(results, xmlResultsPath);

        }

// Returneaza toate intrebarile din fisierul Questions.xml

        public static List<Question> GetQuestions()

        {

            List<Question> questionFromXaml = XmlHelper.FromXmlFile<Question>(xmlQuestionsPath);

            return questionFromXaml;

        }

// Elimină intrebarea cu ID-ul transmis ca parametru

        public static bool RemoveQuestion(string questionId)

        {

            try

            {

                List<Question> questions = GetQuestions();

                Question toBeDeleted = questions.First(q => q.QuestionId == questionId);

                questions.Remove(toBeDeleted);

                XmlHelper.ToXmlFile(questions, xmlQuestionsPath);

                return true;

            }

            catch (Exception)

            {

                return false;

            }

        }

// Înscrie lista de intrebari in fisierul xml

        public static bool WriteQuestionsToXmlFile(List<Question> QuestionsList)

        {

            try

            {

                // Validare intrebari

                // Este setata doar o conditie de validare: Campul CorrectOptionId sa contina valori din intervalul [0, 3].

                Question invalidCorrectOptionId = QuestionsList.Where(q => q.CorrectOptionIndex < 0 || q.CorrectOptionIndex > 3).FirstOrDefault();

                if(invalidCorrectOptionId != null)

                {

                    MessageBox.Show(string.Format($"Intrebarea cu ID-ul {invalidCorrectOptionId.QuestionId} contine campuri invalide: {nameof(invalidCorrectOptionId.CorrectOptionIndex)}"));

                    return false;

                }

                XmlHelper.ToXmlFile(QuestionsList, xmlQuestionsPath);

                return true;

            }

            catch (Exception)

            {

                return false;

            }

        }

    }

    public class XmlHelper

    {

// O funcție generică ce primeste ca parametru un cod xml si returneaza obiectul reprezentat de acel cod

        public static T FromXml<T>(string xml)

        {

            XmlSerializer xs = new XmlSerializer(typeof(T));

            using (StringReader sr = new StringReader(xml))

            {

                return (T)xs.Deserialize(sr);

            }

        }

// Înscrie obiectul obj in format xml in fisierul specificat prin filepath

        public static void ToXmlFile(Object obj, string filePath)

        {

            try

            {

                var xs = new XmlSerializer(obj.GetType());

                using (XmlWriter writer = XmlWriter.Create(filePath))

                {

                    xs.Serialize(writer, obj);

                }

            }

            catch (Exception e)

            {

                MessageBox.Show(e.Message);

            }

        }

// Adauga o intregistrare noua in fisierul xml

        public static bool AppendToXmlFile<T>(T obj, string filePath)

        {

            try

            {

                // Daca fisierul nu a fost inca folosit, il aducem in starea in care vom putea sa lucram cu el.

                if (File.ReadAllText(filePath).Length < 80)

                {

                    ToXmlFile(new List<T> { }, filePath);

                }

                List<T> objects = FromXmlFile<T>(filePath);

                objects.Add(obj);

                ToXmlFile(objects, filePath);

                return true;

            }

            catch (Exception e)

            {

                MessageBox.Show(e.Message);

                return false;

            }

        }

// Returnează o lista cu datele din fisierul xml specificat

        public static List<T> FromXmlFile<T>(string filePath)

        {

            StreamReader sr = new StreamReader(filePath);

            try

            {

                var result = FromXml<List<T>>(sr.ReadToEnd());

                return result;

            }

            catch (Exception e)

            {

                MessageBox.Show("Datele nu au putu fi citite din fisierul XML, " + e.Message);

                return new List<T>();

            }

            finally

            {

                sr.Close();

            }

        }

    }

}

### Serviciul gameservice

Serviciul GameService conține logica din spatele jocului. La fel ca și în cazul AudioService, acesta implementeaza pattern-ul singleton.

GameService se află în strânsă legătură cu DBService și LifeLinesService.

Pentru mai multe detalii consultați comentariile din cod.

### Conținutul GameService.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using WhoWantsToBeAMillionaire.Models;

namespace WhoWantsToBeAMillionaire.Services

{

    class GameService

    {

        // Singleton pattern: Vom utiliza acest design pattern pentru a avea o singura instanta a clasei

        private static GameService instance;

        private GameService() { }

        public static GameService GetInstace()

        {

            if (instance == null)

                instance = new GameService();

            return instance;

        }

        public Game Game { get; set; }

        public Results Results { get; set; }

        public LifelineService LifelineService { get; set; }

        public int CurrentQuestionId { get; set; }

        public bool Init(string playerName)

        {

            try

            {

                // Initializarea jocului: Este creat un obiect Game. Sunt extrase intrebarile jocului din fisierul XML

                Game = new Game();

                LifelineService = new LifelineService();

                Results = new Results();

                Game.Questions = RetrieveAndSelect15Questions();

                // Este nevoie de minim 15 intrebari pentru a incepe jocul

                if (Game.Questions.Count < 15)

                {

                    MessageBox.Show("Este nevoie de minim 15 intrebari pentru a initia un joc.");

                    return false;

                }

                Game.PlayerName = Results.PlayerName = playerName;

                CurrentQuestionId = 0;

                return true;

            }

            catch (Exception)

            {

                MessageBox.Show("S-a produs o eroare la initializarea jocului.");

                return false;

            }

        }

        public Question PickNext()

        {

            if (CurrentQuestionId == 14) // Au fost raspunse deja 15 intrebari

                return null;

            return Game.Questions[++CurrentQuestionId];

        }

        public List<Question> RetrieveAndSelect15Questions()

        {

            // Easy, // 100 - 300                  | 4 intrebari

            // Medium, // 500 - 16 000             | 5 intrebari

            // Hard, // 32 000 - 250 000           | 4 intrebari

            // Einstein // 500 000 - 1 000 000     | 2 intrebari

            List<Question> allQuestions = DBService.GetQuestions();

            if (allQuestions.Count < 15)

                return allQuestions;

            List<Question> gameQuestions = new List<Question>();

            // Popularea listei cu intrebari din diferite categorii de dificultate.

            PopulateList(allQuestions, gameQuestions, DifficultyLevel.Easy, 4);

            PopulateList(allQuestions, gameQuestions, DifficultyLevel.Medium, 5);

            PopulateList(allQuestions, gameQuestions, DifficultyLevel.Hard, 4);

            PopulateList(allQuestions, gameQuestions, DifficultyLevel.Einstein, 2);

            // Populam cu intrebari aleatoare daca lista contine mai putin de 15 intrebari,

            Random rnd = new Random();

            while(gameQuestions.Count < 15)

            {

                Question selectedQ = allQuestions[rnd.Next(0, allQuestions.Count)];

                gameQuestions.Add(selectedQ);

                allQuestions.Remove(selectedQ);

            }

            // Sortarea intrebarilor astfel incat primele sa fie mereu cele mai usoare.

            gameQuestions = gameQuestions.OrderBy(q => q.DifficultyLevel).ToList();

            return gameQuestions;

        }

        public void PopulateList(List<Question> AllQuestions, List<Question> FinalQuestionsList, DifficultyLevel difficultyLevel, int maximumQuestions)

        {

            // Populeaza lista 'FinalQuestionsList' cu maxim 'maximumQuestions' intrebari de dificultatea

            // 'difficultyLevel' din lista tuturor intrebarilor 'AllQuestions'

            List<Question> questionsByDiffLevel = AllQuestions.Where(q => q.DifficultyLevel == difficultyLevel).ToList();

            int questionsCount = questionsByDiffLevel.Count;

            if (questionsCount > maximumQuestions)

                questionsCount = maximumQuestions;

            Random rnd = new Random();

            for (int i = 0; i < questionsCount; i++)

            {

                Question selectedQ = questionsByDiffLevel[rnd.Next(0, questionsByDiffLevel.Count)];

                FinalQuestionsList.Add(selectedQ);

                questionsByDiffLevel.Remove(selectedQ);

            }

        }

        public void AddToEllapsedTime(TimeSpan timeSpan)

        {

            Results.ElapsedTime += timeSpan;

        }

        public bool CheckOption(int userOptionId, TimeSpan ellapsedTimeForQuestion)

        {

            AddToEllapsedTime(ellapsedTimeForQuestion);

            var isCorrect = Game.Questions[CurrentQuestionId].CorrectOptionIndex == userOptionId;

            // Indicii castigurilor safe: 4 9 14 ($1000, $32000, $1000000).

            if (isCorrect)

            {

                switch (CurrentQuestionId)

                {

                    case 4: Game.PrizeSoFar = Game.PrizeList[4]; break;

                    case 9: Game.PrizeSoFar = Game.PrizeList[9]; break;

                    case 14: Game.PrizeSoFar = Game.PrizeList[14]; break;

                    default: break;

                }

            }

            return isCorrect;

        }

        public List<int> AskAudience()

        {

            // Returneaza o lista cu 4 valori - fiecare numar reprezentand procentajul

            // din aduienta care a indicat anume acea optiune;

            return LifelineService.AskAudience(Game.Questions[CurrentQuestionId]);

        }

        public void SaveResults()

        {

            Results.FinalPrize = Game.PrizeSoFar;

            Results.CorrectAnswers = CurrentQuestionId;

            if(Game.PrizeSoFar == Game.PrizeList[14]) Results.CorrectAnswers = CurrentQuestionId + 1;

            DBService.SaveResults(Results);

        }

        public void Retreat()

        {

            // Daca jucatorul decide sa se retraga din joc, atunci premiul lui final va fi cel al utimei intrebari raspunse corect.

            if (CurrentQuestionId == 0)

                Game.PrizeSoFar = "$ 0";

            else

                Game.PrizeSoFar = Game.PrizeList[CurrentQuestionId - 1];

        }

        public List<string> CallAFriend(string friendName)

        {

            // Returneaza o lista cu replicile prietenului si participantului

            return LifelineService.CallAFriend(friendName, Results.PlayerName, Game.Questions[CurrentQuestionId]);

        }

        public void FiftyFifty()

        {

            // Exclude 2 raspunsuri incorecte (textul raspunsurilor este marcat ca fiind "", deja

            // este treaba ViewModel-ului sa dezactiveze butoanele ce contin aceste raspunsuri).

            LifelineService.FiftyFifty(Game.Questions[CurrentQuestionId]);

        }

    }

}

### Serviciul Lifelines service

Aici se conține logica din spatele variantelor ajutătoare.

* **Întreabă publicul:** Această metodă returnează o listă cu numere întregi reprezentând procentajul din public ce a votat pentru fiecare opțiune de răspuns. Această metodă ajutătoare devinde din ce în ce mai puțin utilă odată cu creșterea în dificultate a întrebărilor.
  + Pentru întrebările ușoare (Easy), răspunsul corect va acumula peste 80% din voturi.
  + Pentru întrebările de dificultate medie (Medium), acesta va lua o valoare între 60% și 100%
  + Pentru întrebările grele (Hard), răspunsul corect va avea peste 35% din votui. Aici apare riscul ca o altă variantă de răspuns să ia majoritoatea voturilor
  + Pentru întrebările extrem de grele (Einstein), acestă metodă te poate duce în eroare. Răspunsul corect va acumula peste 20% din voturi, ceea ce oferă teren pentru celelalte opțiuni de a acumula mai multe voturi.
  + Este tratat și cazul în care această metodă de ajutor este chemată după ce au fost exclude 2 întrebări cu varianta 50/50
* **50/50:** Metoda primește ca parametru de intrare intrebarea la care a fost chemată varianta de ajutor. Ea va alege în mod aleator 2 opțiuni de răspuns incorecte și le va invalida.
* **Sună un prieten:** Aici încercă să simulăm discuția cu un prieten. Există 3 tipuri de prieteni:
  + Primul tip: Cunoaște răspunsul corect
  + Al doilea tip: Nu este sigur de răspunsul său. Acesta va oferi 2 variante de răspuns din care una la sigur este cea corectă.
  + Al treilea tip: Nu cunoaște nici pe aproape răspunsul la întrebare, dar oricum va da 2 variante de răspuns absolut aleatoare. Nu știm dacă printre aceastea două se numără și răspunsul corect. Acesta va menționa la sfârșit că nu este sigur de răspunsul oferit.
  + Acum, în dependență de dificultatea întrebării va fi ales unul dintre aceste 3 tipuri.
    - Pentru întrebările ușoare, probabilitatea că prietenul sunat
      * va ști răspunsul este de 70%,
      * va ști parțial răspunsul 20%
      * nu va ști răspunsul 10%
    - Pentru întrebările medium, probabilitatea că prietenul sunat
      * va ști răspunsul este de 60%,
      * va ști parțial răspunsul 25%
      * nu va ști răspunsul 15%
    - Pentru întrebările grele și extrem de grele, probabilitatea că prietenul sunat
      * va ști răspunsul este de 40%,
      * va ști parțial răspunsul 40%
      * nu va ști răspunsul 20%
  + Replicile sunt extrase în mod aleator dintr-o listă cu mai multe replici posibile pentru un anumit context.

### Conținutul LifelineService.cs

using System;

using System.Collections.Generic;

using System.Linq;

using WhoWantsToBeAMillionaire.Models;

namespace WhoWantsToBeAMillionaire.Services

{

    class LifelineService

    {

        public static List<int> AskAudience(Question question)

        {

            // Pentru intrebarile Easy: Raspunsul corect va avea peste 80% din voturi

            // Pentru intrebarile Medium: Raspunsul corect va avea peste 60% din voturi

            // Pentru intrebarile Hard: Raspunsul corect va avea peste 35% din voturi

            // Pentru intrebarile Einstein: Raspunsul corect va avea peste 20% din voturi

            List<int> results = new List<int>() { 0, 0, 0, 0 };

            Random random = new Random();

            int op1 = 0, op2 = 0, op3 = 0, op4 = 0; // op1 este optiunea castigatoare

            switch (question.DifficultyLevel)

            {

                case DifficultyLevel.Easy:

                    op1 = random.Next(80, 100); // Optiunea corecta

                    op2 = random.Next(0, 100 - op1);

                    op3 = random.Next(0, 100 - (op1 + op2));

                    op4 = 100 - (op1 + op2 + op3);

                    break;

                case DifficultyLevel.Medium:

                    op1 = random.Next(60, 100);  // Optiunea corecta

                    op2 = random.Next(0, 100 - op1);

                    op3 = random.Next(0, 100 - (op1 + op2));

                    op4 = 100 - (op1 + op2 + op3);

                    break;

                case DifficultyLevel.Hard:

                    op1 = random.Next(35, 100);  // Optiunea corecta

                    op2 = random.Next(0, 100 - op1);

                    op3 = random.Next(0, 100 - (op1 + op2));

                    op4 = 100 - (op1 + op2 + op3);

                    break;

                case DifficultyLevel.Einstein:

                    op1 = random.Next(20, 100);  // Optiunea corecta

                    op2 = random.Next(0, 100 - op1);

                    op3 = random.Next(0, 100 - (op1 + op2));

                    op4 = 100 - (op1 + op2 + op3);

                    break;

            }

            // Acum avem rezultatele sondajului. A ramas sa verificam un caz aparte - cazul in care

            // a fost interogat publicul dupa ce a fost folosita metoda 50/50.

            // In acest caz, 2 optiuni vide (sa presupunem ca sunt op3 si op4) vor fi egalate cu 0,

            // iar diferenta este impartita in mod egal intre op1 si op2.

            if(question.Options.Count(q => q.Length == 0) == 2)

            {

                // 50/50 A fost utilizat

                int diferenta = op3 + op4;

                op3 = op4 = 0;

                op1 += diferenta / 2;

                op2 += diferenta / 2;

                int correctOptionId = question.CorrectOptionIndex;

                // Avand indicele raspunsului corect, vom itera toti indicii de la 0 la 3 inclusiv:

                // Daca indicele i curent este indicele raspunsului corect, atunci result[i] = op1,

                // Daca indicele este al unui raspuns eliminat in urma 50/50, atunci valoarea result[i] va fi 0.

                // Daca indicele este al unui raspuns valid, result[i] va fi op2.

                // Intr-un final vom avea urmatoarea imagine:

                // Raspunsul corect: op1%

                // Celalalt raspuns: op2%

                // Raspunsurile eliminate: 0% fiecare.

                for(int i = 0; i < 4; i++)

                {

                    if(i == correctOptionId)

                        results[i] = op1;

                    else

                        if (question.Options[i].Length != 0)

                            results[i] = op2;

                }

            }

            else

            {

                // 50/50 nu a fost utilizat.

                // Stocam rezultatele sondajului intr-o stiva.

                Stack<int> OtherOptions = new Stack<int>();

                OtherOptions.Push(op2);

                OtherOptions.Push(op3);

                OtherOptions.Push(op4);

                int correctOptionId = question.CorrectOptionIndex;

                // Populam lista results in 3 etape:

                // Prima etapa: intervalul results[0] - results[correctOptionId-1] este completat cu valori din stiva.

                for (int i = 0; i < correctOptionId; i++)

                    results[i] = OtherOptions.Pop();

                // A Doua etapa: in results[correctOptionId] este inserata valoarea ce reprezinta

                // procentajul din public ce a votat pentru optiunea intr-adevar corecta.

                results[correctOptionId] = op1;

                // A 3-a etapa: intervalul results[correctOptionId+1] - results[3] este completat cu valorile ramase in stiva.

                for (int i = correctOptionId + 1; i < 4; i++)

                    results[i] = OtherOptions.Pop();

            }

            // Intr-un final am obtinut o lista in care pentru Optiunea cu indicele i avem results[i] ce reprezinta

            // procentajul din public ce a votat pentru optiunea respectia.

            return results;

        }

        public static void FiftyFifty(Question question)

        {

            // Marcheaza 2 intrebari gresite prin "".

            List<int> idOptiuni = new List<int>() {0, 1, 2, 3};

            idOptiuni.Remove(question.CorrectOptionIndex);

            Random rnd = new Random();

            // Prima optiune care va fi eliminata:

            int op1 = idOptiuni[rnd.Next(0, 3)];

            question.Options[op1] = "";

            idOptiuni.Remove(op1);

            int op2 = idOptiuni[rnd.Next(0, 2)];

            question.Options[op2] = "";

            idOptiuni.Remove(op1);

            return;

        }

        public static List<string> CallAFriend(string friendName, string playerName, Question question)

        {

            // Vor exista 3 tipuri de prieteni

            // Primul tip: Nu cunoaste raspunsul si va da 2 raspunsuri absolut aleatorii

            // Al doilea tip: Nu este sigur pe sine, dar cunoaste cate ceva. Acesta va da 2 raspunsuri dintre care unul

            // sigur este cel corect

            // Al treilea tip: Este un bun cunoscator, acesta va da raspunsul corect.

            // Pentru intrebarile Easy:

            //  10% sansa ca prietenul va fin din primul tip

            //  20% ca va fi din al 2-lea tip

            //  70% ca va fi din al 3-lea tip

            // Pentru intrebarile Medium

            //  15% - primul tip

            //  25% - al doilea tip

            //  60% - al treilea tip

            // Pentru intrebarile Hard si Einstein

            //  20% - primul tip

            //  40% - al doilea tip

            //  40% - al treilea tip

            // Functia returneaza o lista ce va contine replicile dupa cum urmeaza:

            // Persoana telefonata: Alo

            // Participant: Salut

            // si tot asa, astfel incat replicile cu indice par sunt ale persoanei telefonate, respectiv cele cu indice impar -

            // al participantului.

            List<string> r1 = new List<string>() { "Alo?", "Pronto", "Da", "Alo alo", "Hello", "Cine acolo?" };

            List<string> r2 = new List<string>() {

                $"Salut {friendName}, sunt {playerName}, am nevoie de ajutorul tau",

                $"Buna! {playerName} te deranjeaza. Daca ma ajuti impartim banutii",

                $"Daca nu ma ajuti ma supar pe tine, cu {playerName} vorbesti."

            };

            List<string> r3 = new List<string>() { "Zi", "N-am bani", "Ce s-a intamplat?", $"Ce ai ma {playerName}, esti Ok?", "Cu mare placere!" };

            List<string> r4 = new List<string>() {

                $"Sunt la 'Vrei sa fii Milionar' si sunt blocat la o intrebare",

                $"Vreau sa ma ajuti la o intrebare, apropo sunt la 'Vrei sa fii milionar'!",

                $"Sunt la 'Vrei sa fii milionar', am ajuns la o intrebare mai grea la care doar tu ma poti ajuta."

            };

            List<string> r5 = new List<string> { "Zi", "Ascult cu mare atentie", "Incercam!", "Uoof, pe altcineva n-ai gasit sa suni?", "Nici o problema!" };

            List<string> r6 = new List<string>

            {

                $"Asculta atent aci: {question.QuestionText},  variantele de raspuns: {question.Options[0]}, {question.Options[1]}, {question.Options[2]}, {question.Options[3]}",

                $"Intrebarea e: {question.QuestionText}, astea sunt raspunsurile: {question.Options[0]}, {question.Options[1]}, {question.Options[2]}, {question.Options[3]}",

                $"{question.QuestionText}, sunt sigur ca stii raspunsul dar iti zic si variantele pe care le am aici: {question.Options[0]}, {question.Options[1]}, {question.Options[2]}, {question.Options[3]}",

            };

            List<string> r7\_type1 = new List<string>

            {

                $"E usoara intrebarea... E una din optiunile alea. As merge pe primele 2, imi plac cum suna.",

                $"Mi-a dat cu eroare... Nu cunosc, imi pare rau. Eu as alege {question.Options[1]} sau {question.Options[3]}",

                $"Vezi si tu..  {question.Options[1]} sau {question.Options[3]}, sau  {question.Options[0]}, sau {question.Options[2]}, la sigur e una din ele",

                $"Imi pare rau, nu cunosc nici pe aproape raspunsul, ar putea fi optiunea 3 sau 4, dar nu sunt sigur deloc.",

            };

            List<string> r8\_type1 = new List<string>

            {

                $"Ok, multumesc!",

                $"Bine :( ",

                $"Eh, aia e. Mergem la risc. Mersi!",

            };

            // Pentru al 2-lea tip stocam 2 raspunsuri intr-o lista, dintre care unul cu siguranta este castigator, dupa care le extragem in mod aleator.

            List<string> options = new List<string>();

            options.Add(question.Options[question.CorrectOptionIndex]);

            List<string> wrongOptions = new List<string>();

            for(int i = 0; i < 4; i++)

                if (i != question.CorrectOptionIndex)

                    wrongOptions.Add(question.Options[i]);

            Random rnd = new Random();

            string optiuneGresita = wrongOptions[rnd.Next(0, 3)];

            options.Add(optiuneGresita);

            int randomIndex = rnd.Next(0, 2);

            string option1 = options[randomIndex];

            options.RemoveAt(randomIndex);

            string option2 = options[0];

            //Acum avem option1 si option2, suntem siguri ca una este varianta corecta, dar nu stim exact care.

            List<string> r7\_type2 = new List<string>

            {

                $"La sigur e {option1} sau {option2}",

                $"{option1} sau {option2}, cu siguranta una dintre ele",

                $"Ma inseala memoria, raspunsul e {option1} sau {option2}, cu siguranta.",

            };

            List<string> r8\_type2 = new List<string>

            {

                $"Ok, multumesc!",

                $"Bine",

                $"Eh, aia e. Mergem la risc. Mersi!",

                $"Multumesc! Urmeaza sa iau o decizie. O Zi buna!",

            };

            string correctOption = question.Options[question.CorrectOptionIndex];

            List<string> r7\_type3 = new List<string>

            {

                $"Nici nu stam la discutii, raspunsul e {correctOption}",

                $"{correctOption}, 100%",

                $"Asa intrebari usoare se dau la Vrei sa fii milionar? Raspunsul e {correctOption}",

                $"Ai nimerit bine! Raspunsul e {correctOption}",

                $"Prea usoara intrebarea, {correctOption}",

                $"{correctOption}, te achiti si cu mine ;)",

                $"{correctOption}, altceva?"

            };

            List<string> r8\_type3 = new List<string>

            {

                $"Te respect! Multumesc",

                $"Prietenul la nevoie se cunoaste, multumesc!",

                $"Raman dator!",

                $"Stiam eu ca sun pe cine trebuie ;)",

                $"Eh, intr-adevar era usoara intrebarea, am vrut doar sa-ti aud vocea",

                $"Am incredere! Merg pe aceasta optiune.",

                $"Nice!",

            };

            // Construim converatia cu cate o replica de fiecare tip

            List<string> finalConversation = new List<string>();

            finalConversation.Add(r1[rnd.Next(0, r1.Count)]);

            finalConversation.Add(r2[rnd.Next(0, r2.Count)]);

            finalConversation.Add(r3[rnd.Next(0, r3.Count)]);

            finalConversation.Add(r4[rnd.Next(0, r4.Count)]);

            finalConversation.Add(r5[rnd.Next(0, r5.Count)]);

            finalConversation.Add(r6[rnd.Next(0, r6.Count)]);

            // Alegem tipul de prieten in dependenta de dificultatea intrebarii.

            int friendType = 0;

            int randomNumber = rnd.Next(1, 101);

            switch (question.DifficultyLevel)

            {

                case DifficultyLevel.Easy: // 10% tip1,  20% tip 2,  70% tip 3

                    if (randomNumber <= 10) { friendType = 1;  break; }

                    if (randomNumber <= 30) { friendType = 2;  break; }

                    friendType = 3;

                    break;

                case DifficultyLevel.Medium: // 15% tip1,  25% tip 2,  60% tip 3

                    if (randomNumber <= 15) { friendType = 1; break; }

                    if (randomNumber <= 40) { friendType = 2; break; }

                    friendType = 3;

                    break;

                case DifficultyLevel.Hard: // 20% tip1,  40% tip 2,  40% tip 3

                case DifficultyLevel.Einstein: // 20% tip1,  40% tip 2,  40% tip 3

                    if (randomNumber <= 20) { friendType = 1; break; }

                    if (randomNumber <= 60) { friendType = 2; break; }

                    friendType = 3;

                    break;

            }

            List<string> replica7 = new List<string>();

            List<string> replica8 = new List<string>();

            switch (friendType)

            {

                case 1: replica7 = r7\_type1; replica8 = r8\_type1; break;

                case 2: replica7 = r7\_type2; replica8 = r8\_type2; break;

                case 3: replica7 = r7\_type3; replica8 = r8\_type3; break;

            }

            finalConversation.Add(replica7[rnd.Next(0, replica7.Count)]);

            finalConversation.Add(replica8[rnd.Next(0, replica8.Count)]);

            return finalConversation;

        }

    }

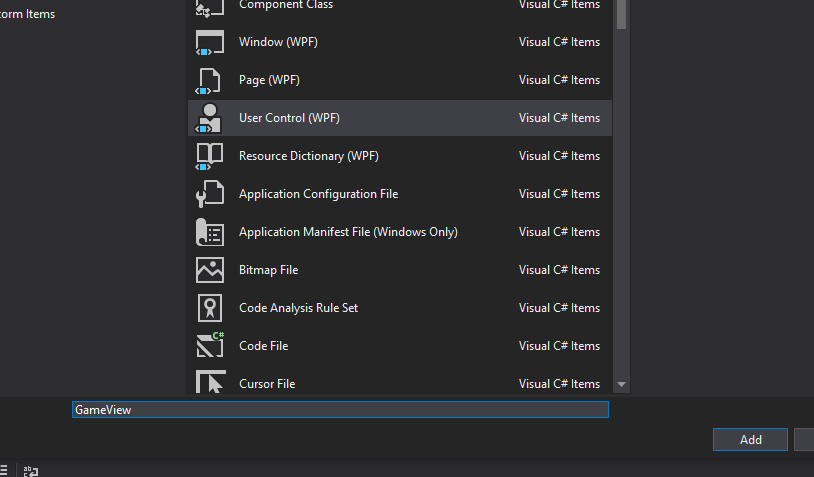
}

## Ferestrele aplicației

### Adăugarea ferestrelor

Pentru a avea o structură mai curată a proiectului, vom crea un folder nou cu denumirea **Views** în soluția noastră și vom adăuga următoarele ferestre:

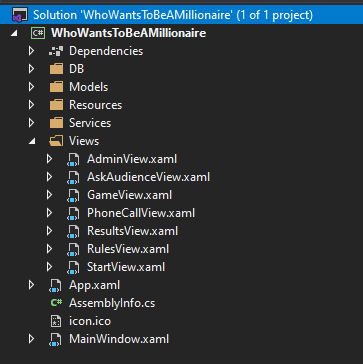
* StartView, RulesView, GameView, ResultsView **de tip UserControl**



* AdminView, AskAudienceView, PhoneCallView **de tip Window**

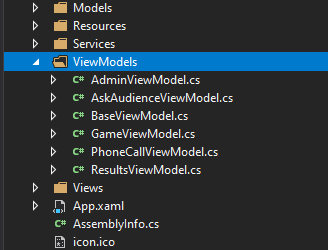


Aceasta trebuie să fie structura proiectului până în momentul de față:



După cum am spus la început, vom implementa pattern-ul **MVVM** pentru ferestrele mai mari (care conțin mai multe date), de aceea, următorul pas după adăugarea ferestrelor este adăugarea ViewModel-urilor pentru ferestrele ce vor urma pattern-ul **MVVM.**

Creăm un încă un folder nou cu denumirea **ViewModels,** în care adăugam următoarele clase:



Vom începe prin a defini clasa BaseViewModel.

**Puțină teorie:** Pentru ca interfața aplicației să fie notificată despre schimbarea valorilor unor proprietăți, aceasta trebuie să fie notificată.

Spre exemplu, să zicem că avem pe suprafața ferestrei noastre un textblock a cărui conținut este legat de valoarea unei proprietăți a unei clase. Atunci când valoarea proprietății respective este modificată, conținutul acelui textblock nu se va modifica deoarece nu a fost notificat.

Această problemă poate fi rezolvată în 2 metode:

1. Prin a utiliza DependecyProperties
2. Prin notificarea interfeței atunci când valoarea unei proprietăți a fost modificată.

În cadrul aceste aplicații vom aborda a 2-a metodă, și anume – Clasa BaseViewModel va implementa interfața InotifyPropertyChanged, iar toate celelalte ViewModel-uri vor deriva din BaseViewModel, iar atunci când vom modifica valorile proprietăților, vom apela totodată metoda OnPropertyChanged ce va notifica interfața privind schimbarea valorii.

### Conținutul baseviewmodel

using System.ComponentModel;

namespace WhoWantsToBeAMillionaire.ViewModels

{

    public class BaseViewModel : INotifyPropertyChanged

    {

        public event PropertyChangedEventHandler PropertyChanged;

        protected void OnPropertyChanged(string propertyName)

        {

            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

        }

    }

}

### Fereastra mainwindow

Acesta este rezultatul spre care tindem:



După cum vedeți, în colțul din stânga-sus se află 2 butoane. Ele se vor afla mereu în acel loc, indiferent de fereastra care este afișată. Pentru a realiza acest lucru, creăm un ContentControl în care vor fi încărcate celelate ferestre, și un grid în care includem butoanele mai sus menționate.

### Conținut MainWindow.xaml

<Window x:Class="WhoWantsToBeAMillionaire.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:WhoWantsToBeAMillionaire"

        mc:Ignorable="d"

        FontFamily="Source Sans Pro"

        Title="Who wants to be a Millionaire" Height="630" Width="1140"

        MinWidth="1092" MinHeight="615"

        WindowStartupLocation="CenterScreen"

        Icon="icon.ico">

    <Window.Background>

        <ImageBrush ImageSource="pack://application:,,,/WhoWantsToBeAMillionaire;component/Resources/Media/bg.jpg" Stretch="UniformToFill"/>

    </Window.Background>

    <Grid>

        <Grid.Background>

            <SolidColorBrush Opacity="0.5" Color="Black"/>

        </Grid.Background>

        <!--Spatiul in care vor fi randate celelalte ferestre-->

        <ContentControl Name="childWindow" Grid.Row="1" Padding="0, 30, 0, 0">

        </ContentControl>

        <!--Meniul de sus-->

        <Grid Height="30" VerticalAlignment="Top" Margin="0 5 0 0">

            <Grid.ColumnDefinitions>

                <ColumnDefinition Width="1\*"/>

                <ColumnDefinition Width="1\*"/>

                <ColumnDefinition Width="15\*"/>

            </Grid.ColumnDefinitions>

            <!--Button-ul pentru activarea / dezactivarea sunetelor-->

            <Button Grid.Column="0"

                    HorizontalAlignment="Center"

                    Name="btToggleVolume" Style="{StaticResource EmptyButton}"

                    Click="btToggleVolume\_Click">

                <Image Source="pack://application:,,,/Resources/Media/volume.png"/>

            </Button>

            <!--In cazul in care sunetul este dezativat, deasupra iconitei va aparea un X rosu-->

            <Image Name="VolumeOff"

                   HorizontalAlignment="Center"

                   Grid.Column="0"

                   Source="pack://application:,,,/Resources/Media/cross.png"

                   Height="{Binding ElementName=btToggleVolume, Path=Height}"

                   Width="{Binding ElementName=btToggleVolume, Path=Width}"

                   MouseDown="btToggleVolume\_Click"

                   Opacity="0.7" Visibility="Hidden"/>

            <!--Butonul pentru accesarea panoului de administrare-->

            <Button Grid.Column="1" Name="btAdminPanel"

                    HorizontalAlignment="Center"

                    Style="{StaticResource EmptyButton}"

                    Click="btAdminPanel\_Click">

                <Image Source="pack://application:,,,/Resources/Media/admin.png"/>

            </Button>

        </Grid>

    </Grid>

</Window>

### Code behind (Mainwindow.xaml.cs)

using System.Windows;

using WhoWantsToBeAMillionaire.Services;

using WhoWantsToBeAMillionaire.Views;

namespace WhoWantsToBeAMillionaire

{

    public partial class MainWindow : Window

    {

        private AudioService \_audioService;

        public MainWindow()

        {

            InitializeComponent();

            UpdateView("Start");

            \_audioService = AudioService.GetInstace();

        }

        private void btToggleVolume\_Click(object sender, RoutedEventArgs e)

        {

            if (VolumeOff.Visibility == Visibility.Hidden)

                VolumeOff.Visibility = Visibility.Visible;

            else

                VolumeOff.Visibility = Visibility.Hidden;

            \_audioService.ToggleVolume();

        }

        private void btAdminPanel\_Click(object sender, RoutedEventArgs e)

        {

            AdminView adminView = new AdminView();

            adminView.ShowDialog();

        }

        public void UpdateView(string parameter)

        {

            switch (parameter)

            {

                case "Start":

                    childWindow.Content = new StartView();

                    break;

                case "Game":

                    childWindow.Content = new GameView();

                    break;

                case "Results":

                    childWindow.Content = new ResultsView();

                    break;

                case "Rules":

                    childWindow.Content = new RulesView();

                    break;

                default:

                    childWindow.Content = new StartView();

                    break;

            }

        }

        public void DeactivateAdminPanel()

        {

            btAdminPanel.IsEnabled = false;

        }

        public void ActivateAdminPanel()

        {

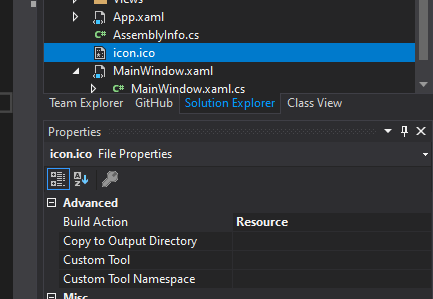
            btAdminPanel.IsEnabled = true;

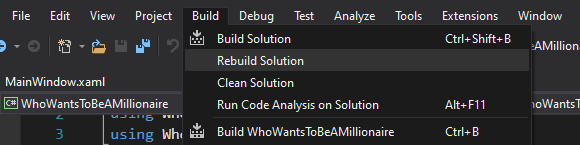
        }

    }

}

Dacă este aruncată o excepție de tip Cannot locate resource icon.ico, asigurați-vă că build-action pentru icon.ico este setat ca Resource, după care faceți un rebuild.

****

****

Dacă problema nu a fost rezolvată, alegeți mai întâi opțiunea Clean Solution, după care Rebuild Solution.

### fereastra startview

Acesta este rezultatul spre care tindem



Ea va fi randată în interiorul ferestrei MainView.

### Conținutul startview.xaml

<UserControl x:Class="WhoWantsToBeAMillionaire.Views.StartView"

             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

             xmlns:local="clr-namespace:WhoWantsToBeAMillionaire.Views"

             mc:Ignorable="d"

             d:DesignHeight="450" d:DesignWidth="800">

    <Grid>

        <Grid.Background>

            <SolidColorBrush Opacity="0.5" Color="Black"/>

        </Grid.Background>

        <Grid.RowDefinitions>

            <RowDefinition/>

            <RowDefinition/>

            <RowDefinition/>

            <RowDefinition/>

            <RowDefinition/>

        </Grid.RowDefinitions>

        <Label x:Name="Title" Grid.Row="1"

               FontSize="40"

               Foreground="White"

               Content="Bine ai venit!"

               HorizontalAlignment="Center"

               VerticalAlignment="Center"/>

        <Grid Grid.Row="2">

            <Grid.RowDefinitions>

                <RowDefinition Height="2\*"/>

                <RowDefinition Height="4\*"/>

            </Grid.RowDefinitions>

            <Label Grid.Row="0" Content="Introdu numele"

                   Foreground="White" FontSize="16"

                   HorizontalContentAlignment="Center" VerticalContentAlignment="Center"/>

            <TextBox Name="tbPlayerName" Grid.Row="1"

                     Style="{StaticResource SimpleTextBox}"

                     TextChanged="tbPlayerName\_TextChanged"

                     />

        </Grid>

        <Button x:Name="btStartGame" Grid.Row="3"

                IsEnabled="False"

                Style="{StaticResource StartViewButtons}"

                Content="Incepe jocul"

                Click="btStartGame\_Click"/>

        <Button x:Name="btShowRules" Grid.Row="4"

                Content="Regulamentul"

                Style="{StaticResource StartViewButtons}"

                Click="btShowRules\_Click"/>

    </Grid>

</UserControl>

### Conținutul startview.xaml.cs

using System.Windows;

using System.Windows.Controls;

using WhoWantsToBeAMillionaire.Services;

namespace WhoWantsToBeAMillionaire.Views

{

    public partial class StartView : UserControl

    {

        private MainWindow \_mainWindow;

        private AudioService \_audioService;

        public StartView()

        {

            InitializeComponent();

            \_mainWindow = (MainWindow)System.Windows.Application.Current.MainWindow;

            \_mainWindow.ActivateAdminPanel();

            \_audioService = AudioService.GetInstace();

            \_audioService.PlayAudio(\_audioService.MainTheme);

        }

        private void btStartGame\_Click(object sender, RoutedEventArgs e)

        {

            if (GameService.GetInstace().Init(tbPlayerName.Text))

            {

                \_mainWindow.DeactivateAdminPanel();

                \_mainWindow.UpdateView("Game");

            }

        }

        private void btShowRules\_Click(object sender, RoutedEventArgs e)

        {

            \_mainWindow.UpdateView("Rules");

        }

        private void tbPlayerName\_TextChanged(object sender, TextChangedEventArgs e)

        {

            btStartGame.IsEnabled = (tbPlayerName.Text.Trim() != "");

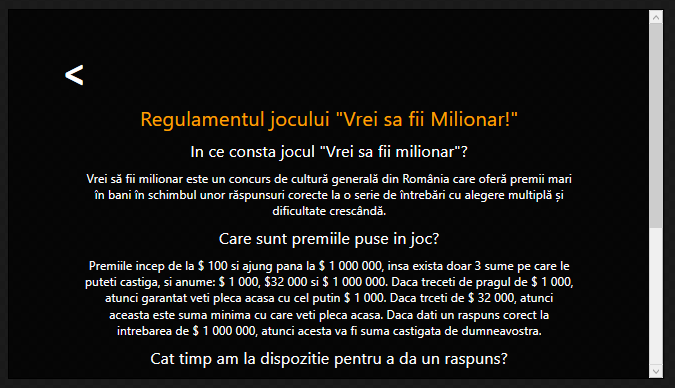
        }

    }

}

### Fereastra rulesview

Rezultatul spre care tindem



Întreg conținutul este plasat într-un ScrollViewer.

### Conținutul rulesview.xaml

<UserControl x:Class="WhoWantsToBeAMillionaire.Views.RulesView"

             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

             xmlns:local="clr-namespace:WhoWantsToBeAMillionaire.Views"

             mc:Ignorable="d"

             d:DesignHeight="450" d:DesignWidth="800">

    <ScrollViewer>

        <ScrollViewer.Background>

            <SolidColorBrush Opacity="0.8" Color="Black"/>

        </ScrollViewer.Background>

        <StackPanel Orientation="Vertical" Margin="30 0 30 10">

            <Button Style="{StaticResource btBackStyle}"

                    Click="btInapoi\_Click"/>

            <TextBlock Style="{StaticResource Heading}" Foreground="Orange">

            Regulamentul jocului "Vrei sa fii Milionar!"

            </TextBlock>

            <TextBlock Style="{StaticResource LargeText}" HorizontalAlignment="Center" Margin="0 10">

            In ce consta jocul "Vrei sa fii milionar"?

            </TextBlock>

            <TextBlock Style="{StaticResource NormalText}">

            Vrei să fii milionar este un concurs de cultură generală din România care oferă premii mari în bani în schimbul unor răspunsuri corecte la o serie de întrebări cu alegere multiplă și dificultate crescândă.

            </TextBlock>

            <TextBlock Style="{StaticResource LargeText}" HorizontalAlignment="Center" Margin="0 10">

            Care sunt premiile puse in joc?

            </TextBlock>

            <TextBlock Style="{StaticResource NormalText}">

            Premiile incep de la $ 100 si ajung pana la $ 1 000 000, insa exista doar 3 sume pe care le puteti castiga, si anume: $ 1 000, $32 000 si $ 1 000 000. Daca treceti de pragul de $ 1 000, atunci garantat veti pleca acasa cu cel putin $ 1 000. Daca trceti de  $ 32 000, atunci aceasta este suma minima cu care veti pleca acasa. Daca dati un raspuns corect la intrebarea de $ 1 000 000, atunci acesta va fi suma castigata de dumneavostra.

            </TextBlock>

            <TextBlock Style="{StaticResource LargeText}" HorizontalAlignment="Center" Margin="0 10">

            Cat timp am la dispozitie pentru a da un raspuns?

            </TextBlock>

            <TextBlock Style="{StaticResource NormalText}">

            Dispuneti de 60 de secunde pentru a da un raspuns la intrebarea acordata. Daca decideti sa utilizati una din variantele ajutatoare, atunci timpul este resetat, ceea ce inseamna ca veti avea inca un minut pentru a oferi raspunsul final.

            </TextBlock>

            <TextBlock Style="{StaticResource LargeText}" HorizontalAlignment="Center" Margin="0 10">

            Ce se intampla daca expira cele 60 de secunde iar eu nu am ales un raspuns?

            </TextBlock>

            <TextBlock Style="{StaticResource NormalText}">

            In momentul in care expira cele 60 de secunde, este afisat raspunsul corect pentru intrebarea respectiva, iar dumneavoastra parasiti jocul.

            </TextBlock>

            <TextBlock Style="{StaticResource LargeText}" HorizontalAlignment="Center" Margin="0 10">

            Care sunt variantele ajutatoare?

            </TextBlock>

            <TextBlock Style="{StaticResource NormalText}">

            Exista 3 variante ajutatoare (Lifelines) de care va puteti folosi de-a lungul jocului, si anume: <LineBreak/>

                50/50 - Elimina 2 raspunsuri incorecte, <LineBreak/>

                Intreaba publicul - Publicul este intergoat privind raspunsul corect la intrebarea curenta, rezultatele sunt afisate sub forma de grafic <LineBreak/>

                Suna un prieteni - Ofera posibilitatea de a telefona un prieten.

            </TextBlock>

        </StackPanel>

    </ScrollViewer>

</UserControl>

### Conținutul rulesview.xaml.cs

using System.Windows;

using System.Windows.Controls;

namespace WhoWantsToBeAMillionaire.Views

{

    public partial class RulesView : UserControl

    {

        public RulesView()

        {

            InitializeComponent();

        }

        private void btInapoi\_Click(object sender, RoutedEventArgs e)

        {

            ((MainWindow)System.Windows.Application.Current.MainWindow).UpdateView("Start");

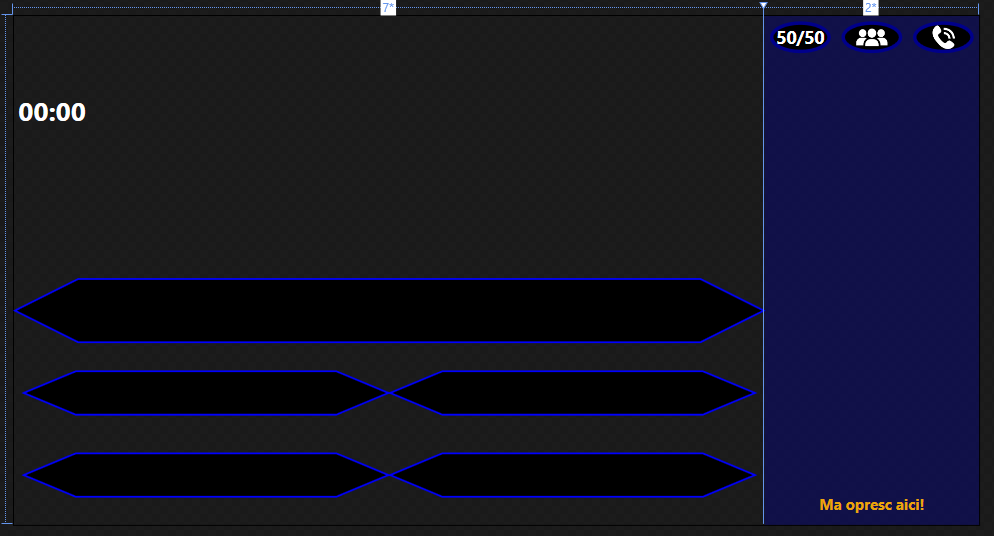
        }

    }

}

### Fereastra Gameview

Rezultatul spre care tindem:



Avem poligonul din centru, în interiorul căruia va apărea întrebarea. Mai jos se găsesc 4 butoane, câte unul pentru fiecare opțiune. Conținutul acestor elemente este dinamic, fiind legat de valoarea unor proprietăți ale întrebării curente (Întrebarea curentă se află în GameViewModel). Atunci când vom selecta altă întrebare, conținutul de pe pagină se va modifica corespunzător.

În partea dreaptă avem panoul jocului, acolo vor fi adăugate în mod dinamic premiile jocului în momentul lansării paginii.

**IMPORTANT:** Pentru a vedea mai bine conținutul paginii, setați Lățimea UserControl-ului 580, iar lățimea 1100, iar când lansați aplicație, setațile valorile **auto** pentru Height și Width. Astfel, butoanele se vor redimensiona odată cu fereastra.

### Conținutul gameview.xaml

<UserControl x:Class="WhoWantsToBeAMillionaire.Views.GameView"

             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

             xmlns:local="clr-namespace:WhoWantsToBeAMillionaire.Views"

             mc:Ignorable="d" Height="auto" Width="auto">

    <!--Development: Height="580" Width="1100"-->

    <!--Production: Height="auto" Width="auto"-->

    <Grid>

        <Grid.ColumnDefinitions>

            <ColumnDefinition Width="7\*"/>

            <ColumnDefinition Width="2\*"/>

        </Grid.ColumnDefinitions>

        <Grid Grid.Column="0">

            <Grid.RowDefinitions>

                <RowDefinition Height="2\*"/>

                <RowDefinition Height="4\*"/>

                <RowDefinition Height="3\*"/>

                <RowDefinition Height="5\*"/>

                <RowDefinition Name="RowForAdditionalInformation" Height="0\*"/>

            </Grid.RowDefinitions>

            <!--Timer-ul-->

            <StackPanel Grid.Row="1">

                <Label Name="timer" Content="00:00" Style="{StaticResource timerStyle}"/>

                <Label Style="{StaticResource timerStyle}" FontSize="13" FontWeight="Normal"

                       Content="{Binding Path=SecondsPerQuestion}"

                       ContentStringFormat="secunde per intrebare: {0}"/>

            </StackPanel>

            <!--Containerul in care va fi afisata intrebarea-->

            <Grid Grid.Row="2" HorizontalAlignment="Center" VerticalAlignment="Bottom" >

                <Polygon Style="{StaticResource QuestionPolygon}"/>

                <TextBox

                    Style="{StaticResource QuestionStyle}"

                    Width="620"

                    Text="{Binding CurrentQuestion.QuestionText}"/>

            </Grid>

            <Grid Name="optionsGrid" Grid.Row="3" Margin="10">

                <Grid.RowDefinitions>

                    <RowDefinition/>

                    <RowDefinition/>

                </Grid.RowDefinitions>

                <Grid.ColumnDefinitions>

                    <ColumnDefinition/>

                    <ColumnDefinition/>

                </Grid.ColumnDefinitions>

                <!--Prima optiune-->

                <Grid Grid.Row="0" Grid.Column="0" HorizontalAlignment="Center" VerticalAlignment="Center" >

                    <Button x:Name="btOption0" Style="{StaticResource OptionPolygon}"

                            Tag="0"

                            Click="btOption\_Click"

                            IsEnabled="{Binding Op0IsEnabled}">

                        <TextBlock x:Name="option0" Style="{StaticResource optionText}"

                                   Text="{Binding CurrentQuestion.Options[0], StringFormat={}A: {0}}"/>

                    </Button>

                </Grid>

                <!--A doua optiune-->

                <Grid Grid.Row="0" Grid.Column="1" HorizontalAlignment="Center" VerticalAlignment="Center" >

                    <Button x:Name="btOption1" Style="{StaticResource OptionPolygon}"

                            Tag="1"

                            Click="btOption\_Click"

                            IsEnabled="{Binding Op1IsEnabled}">

                        <TextBlock x:Name="option1" Style="{StaticResource optionText}"

                                   Text="{Binding CurrentQuestion.Options[1], StringFormat={}B: {0}}"/>

                    </Button>

                </Grid>

                <!--A treia optiune-->

                <Grid Grid.Row="1" Grid.Column="0" HorizontalAlignment="Center" VerticalAlignment="Center" >

                    <Button x:Name="btOption2" Style="{StaticResource OptionPolygon}"

                            Tag="2"

                            Click="btOption\_Click"

                            IsEnabled="{Binding Op2IsEnabled}">

                        <TextBlock x:Name="option2" Style="{StaticResource optionText}"

                                   Text="{Binding CurrentQuestion.Options[2], StringFormat={}C: {0}}"/>

                    </Button>

                </Grid>

                <!--A patra optiune-->

                <Grid Grid.Row="1" Grid.Column="1" HorizontalAlignment="Center" VerticalAlignment="Center" >

                    <Button x:Name="btOption3" Style="{StaticResource OptionPolygon}"

                            Tag="3"

                            Click="btOption\_Click"

                            IsEnabled="{Binding Op3IsEnabled}">

                        <TextBlock x:Name="option3" Style="{StaticResource optionText}"

                                   Text="{Binding CurrentQuestion.Options[3], StringFormat={}D: {0}}"/>

                    </Button>

                </Grid>

            </Grid>

            <!--Panoul pentru afisarea explicatiilor si a butonului de continuare-->

            <StackPanel Grid.Row="4" Orientation="Vertical"

                        Background="White" Opacity="0.7" Margin="4"

                        Visibility="Collapsed"

                        Name="panelExplications">

                <TextBox Name="tbExplications"

                   Text="{Binding Explanations}"

                   HorizontalContentAlignment="Center" VerticalContentAlignment="Center"

                   FontSize="15"

                   BorderThickness="0"

                   IsReadOnly="True"

                   TextWrapping="WrapWithOverflow"

                   Background="Azure"/>

                <Button x:Name="btGenerateResults"

                        Click="btGenerateResults\_Click"

                        Content="Mai Departe"

                        FontSize="18"

                        FontWeight="Bold"/>

            </StackPanel>

        </Grid>

        <Grid Grid.Column="1" Style="{StaticResource RightPanelGrid}">

            <Grid.RowDefinitions>

                <RowDefinition Height="1\*"/>

                <RowDefinition Height="10\*"/>

                <RowDefinition Height="1\*"/>

            </Grid.RowDefinitions>

            <!--Grid pentru afisarea butoanelor asociate variantelor ajutatoare-->

            <Grid x:Name="lifeLinesGrid" Grid.Row="0">

                <Grid.ColumnDefinitions>

                    <ColumnDefinition/>

                    <ColumnDefinition/>

                    <ColumnDefinition/>

                </Grid.ColumnDefinitions>

                <Button Grid.Column="0" x:Name="btFiftyFifty"

                        Click="btFiftyFifty\_Click"  Content="50/50"

                        Style="{StaticResource EllipseButton}"/>

                <Button Grid.Column="1" x:Name="btAskAudience"

                        Click="btAskAudience\_Click"

                        Style="{StaticResource EllipseButton}">

                    <Image Source="pack://application:,,,/Resources/Media/askaudience.png"/>

                </Button>

                <Button Grid.Column="2" x:Name="btPhoneCall"

                        Click="btPhoneCall\_Click"

                        Style="{StaticResource EllipseButton}">

                    <Image Source="pack://application:,,,/Resources/Media/phonecall.png"/>

                </Button>

                <Image Name="FiftyFiftyUsed"

                       Grid.Column="0" Source="pack://application:,,,/Resources/Media/cross.png"

                       Opacity="0.7" Visibility="Collapsed"/>

                <Image Name="AskAudienceUsed"

                       Grid.Column="1" Source="pack://application:,,,/Resources/Media/cross.png"

                       Opacity="0.7" Visibility="Collapsed"/>

                <Image Name="PhoneCallUsed"

                       Grid.Column="2" Source="pack://application:,,,/Resources/Media/cross.png"

                       Opacity="0.7" Visibility="Collapsed"/>

            </Grid>

            <!--Grid in care vor fi adaugate in mod dinamic premiile-->

            <Grid Grid.Row="1"

                  Name="prizeStack"

                  Margin="10 20 10 10"

                  Height="auto">

            </Grid>

            <Button Grid.Row="2" x:Name="btRetreat"

                    IsEnabled="{Binding BtRetreatIsEnabled}"

                    Style="{StaticResource EmptyButton}"

                    Margin="10"

                    Foreground="#eca609"

                    FontWeight="Bold"

                    FontSize="17"

                    Content="Ma opresc aici!"

                    Click="btRetreat\_Click"/>

        </Grid>

    </Grid>

</UserControl>

### Conținutul gameview.xaml.cs

using System.Windows;

using System.Windows.Controls;

using WhoWantsToBeAMillionaire.ViewModels;

namespace WhoWantsToBeAMillionaire.Views

{

    public partial class GameView : UserControl

    {

        private GameViewModel \_viewModel;

        public GameView()

        {

            // Este creata o instanta a clasei GameViewModel si este setata ca DataContext

            InitializeComponent();

            DataContext = new GameViewModel(this);

            \_viewModel = (DataContext as GameViewModel);

        }

        private void btAskAudience\_Click(object sender, RoutedEventArgs e)

        {

            \_viewModel.AskAudience();

        }

        private void btPhoneCall\_Click(object sender, RoutedEventArgs e)

        {

            \_viewModel.CallFriend();

        }

        private void btFiftyFifty\_Click(object sender, RoutedEventArgs e)

        {

            \_viewModel.FiftyFifty();

        }

        private void btOption\_Click(object sender, RoutedEventArgs e)

        {

            \_viewModel.OptionSubmitted(sender as Button);

        }

        private void btGenerateResults\_Click(object sender, RoutedEventArgs e)

        {

            \_viewModel.SaveAndDisplayResults();

        }

        private void btRetreat\_Click(object sender, RoutedEventArgs e)

        {

            \_viewModel.Retreat();

        }

    }

}

### Conținutul GAMEVIEWMODEL.CS

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Media;

using System.Windows.Threading;

using WhoWantsToBeAMillionaire.Models;

using WhoWantsToBeAMillionaire.Services;

using WhoWantsToBeAMillionaire.Views;

using WhoWantsToBeAMillionaire.ViewModels;

namespace WhoWantsToBeAMillionaire.ViewModels

{

    class GameViewModel : BaseViewModel

    {

        private Question \_currentQuestion;

        public Question CurrentQuestion

        {

            get

            {

                return \_currentQuestion;

            }

            set

            {

                \_currentQuestion = value;

                OnPropertyChanged(nameof(CurrentQuestion));

            }

        }

        private int \_secondsPerQuestion;

        public int SecondsPerQuestion

        {

            get { return \_secondsPerQuestion; }

            set

            {

                \_secondsPerQuestion = value;

                OnPropertyChanged(nameof(SecondsPerQuestion));

            }

        }

        private bool \_op0IsEnabled;

        public bool Op0IsEnabled

        {

            get { return \_op0IsEnabled; }

            set

            {

                \_op0IsEnabled = value;

                OnPropertyChanged(nameof(Op0IsEnabled));

            }

        }

        private bool \_op1IsEnabled;

        public bool Op1IsEnabled

        {

            get { return \_op1IsEnabled; }

            set

            {

                \_op1IsEnabled = value;

                OnPropertyChanged(nameof(Op1IsEnabled));

            }

        }

        private bool \_op2IsEnabled;

        public bool Op2IsEnabled

        {

            get { return \_op2IsEnabled; }

            set

            {

                \_op2IsEnabled = value;

                OnPropertyChanged(nameof(Op2IsEnabled));

            }

        }

        private bool \_op3IsEnabled;

        public bool Op3IsEnabled

        {

            get { return \_op3IsEnabled; }

            set

            {

                \_op3IsEnabled = value;

                OnPropertyChanged(nameof(Op3IsEnabled));

            }

        }

        private bool \_btRetreatIsEnabled;

        public bool BtRetreatIsEnabled

        {

            get { return \_op3IsEnabled; }

            set

            {

                \_btRetreatIsEnabled = value;

                OnPropertyChanged(nameof(BtRetreatIsEnabled));

            }

        }

        public GameService GameService { get; set; }

        private AudioService \_audioService { get; set; }

        public GameView \_gameView;

        public DispatcherTimer ellapsedTime;

        public DateTime start;

        private Style lastItemStyle = null;

        public List<Label> PrizeLabels { get; set; }

        public GameViewModel(GameView gameView)

        {

            Op0IsEnabled = Op1IsEnabled = Op2IsEnabled = Op3IsEnabled = true;

            \_gameView = gameView;

            \_audioService = AudioService.GetInstace();

            // Cream o instanta a clasei GameService, Acolo se contine cea mai mare parte a logicii

            // din spatele jocului.

            GameService = GameService.GetInstace();

            SecondsPerQuestion = 60;

            // Afisam stiva premiilor

            PrizeLabels = new List<Label>();

            var Prizes = Game.PrizeList;

            var SafePoints = Game.SafePoints;

            // Pentru fiecare premiu este creat un label si un row in grid-ul

            // dedicat afisarii premiilor.

            for (int i = Prizes.Count-1; i>=0; i--)

            {

                Label lbPrize = new Label { Content = Prizes[i] };

                // Premiile de $1000, $32000 si $1000000 vor avea alta culoare

                string styleIdentifier = "PrizeItemStyle";

                if (SafePoints.Contains(lbPrize.Content.ToString()))

                    styleIdentifier = "SafePrizeItemStyle";

                lbPrize.Style = Application.Current.TryFindResource(styleIdentifier) as Style;

                // Inserarea premiului in grid

                gameView.prizeStack.RowDefinitions.Add(new RowDefinition());

                Grid.SetRow(lbPrize, gameView.prizeStack.RowDefinitions.Count - 1);

                lbPrize.HorizontalContentAlignment = HorizontalAlignment.Center;

                lbPrize.VerticalAlignment = VerticalAlignment.Stretch;

                gameView.prizeStack.Children.Add(lbPrize);

                PrizeLabels.Add(lbPrize);

            }

            lastItemStyle = PrizeLabels[PrizeLabels.Count - 1].Style;

            // Marcam premiul curent in stiva

            PrizeLabels[PrizeLabels.Count - 1].Style = Application.Current.TryFindResource("CurrentPrize") as Style;

            // Selectam prima intrebare

            CurrentQuestion = GameService.Game.Questions[0];

            // Initializam un timer pentru contorizarea timpului la fiecare intrebare.

            ellapsedTime = new DispatcherTimer(TimeSpan.FromSeconds(1), DispatcherPriority.Background,

                   t\_Tick, Dispatcher.CurrentDispatcher); ellapsedTime.IsEnabled = true;

            start = DateTime.Now;

            // Melodia de inceput

            \_audioService.PlayAudio(\_audioService.LetsPlay);

        }

        private void t\_Tick(object sender, EventArgs e)

        {

            // Extragem diferenta de timp dintre momentul actual si cel in care a fost afisata intrebarea

            var difference = DateTime.Now.Subtract(start);

            \_gameView.timer.Content = difference.ToString(@"mm\:ss");

            // Daca mai raman 5 secunde - timer-ul devine rosu

            // 30 secunde - timer-ul galben

            // > 30 secunde - timer-ul alb

            if (difference.Seconds > SecondsPerQuestion - 5)

                \_gameView.timer.Foreground = Brushes.Red;

            else if (difference.Seconds > SecondsPerQuestion - 30)

                \_gameView.timer.Foreground = Brushes.Yellow;

            else

                \_gameView.timer.Foreground = Brushes.White;

            // Cazul in care expira timpul:

            if (difference > TimeSpan.FromSeconds(SecondsPerQuestion))

            {

                ellapsedTime.Stop();

                GameService.AddToEllapsedTime(DateTime.Now.Subtract(start));

                GameOver();

                return;

            }

        }

        public async void OptionSubmitted(Button btOption)

        {

            // Oprim contorizarea timpului si dezactivam butoanele

            ellapsedTime.Stop();

            ButtonsSetIsEnabledTo(false);

            \_audioService.PlayAudio(\_audioService.FinalAnswer);

            btOption.Style = Application.Current.TryFindResource("OptionSelected") as Style;

            await Task.Delay(3000);

            string feedBack = CheckSubmittedOption(int.Parse(btOption.Tag.ToString()), DateTime.Now.Subtract(start));

            switch (feedBack)

            {

                case "Success!":

                    // Raspuns corect

                    btOption.Style = Application.Current.TryFindResource("RightOptionSelected") as Style;

                    \_audioService.PlayAudio(\_audioService.CorrectAnswer);

                    break;

                default:

                    // Raspuns gresit

                    btOption.Style = Application.Current.TryFindResource("WrongOptionSelected") as Style;

                    GameOver();

                    return;

            }

            // Daca am ajuns pana aici, inseamna ca raspunsul a fost corect.

            await Task.Delay(3000);

            // Daca nu mai exista intrebari - afisam rezultatele.

            // Daca mai exista - pregatim terenul pentru urmatoarea intrebare.

            if (!PickNextQuestion())

                SaveAndDisplayResults();

            else

            {

                ButtonsSetIsEnabledTo(true);

                btOption.Style = Application.Current.TryFindResource("OptionPolygon") as Style;

                start = DateTime.Now;

                \_gameView.timer.Content = "00:00";

                ellapsedTime.Start();

            }

        }

        public string CheckSubmittedOption(int optionId, TimeSpan ellapsedTimeForQuestion)

        {

            if (GameService.CheckOption(optionId, ellapsedTimeForQuestion))

                return "Success!";

            else

                return "NotSuccess";

        }

        public bool PickNextQuestion()

        {

            CurrentQuestion = GameService.PickNext();

            if (CurrentQuestion == null)

                return false; // Nu mai sunt intrebari

            MarkCurrentPrizeWithinPrizeStack();

            \_audioService.PlayAudioAccordingToPriceRange(GameService.CurrentQuestionId);

            return true;

        }

        public void MarkCurrentPrizeWithinPrizeStack()

        {

            // Evidentiaza premiul curent in stiva premiilor, iar premiul care a fost evidentiat revine la stilul sau normal.

            var previousPrize = PrizeLabels[PrizeLabels.Count - GameService.CurrentQuestionId];

            var currentPrize = PrizeLabels[PrizeLabels.Count - 1 - GameService.CurrentQuestionId];

            previousPrize.Style = lastItemStyle;

            lastItemStyle = currentPrize.Style;

            currentPrize.Style = Application.Current.TryFindResource("CurrentPrize") as Style;

        }

        public void SaveAndDisplayResults()

        {

            GameService.SaveResults();

            ((MainWindow)System.Windows.Application.Current.MainWindow).UpdateView("Results");

        }

        private void ButtonsSetIsEnabledTo(bool flag)

        {

            // Dezactiveaza / Activeaza (in dependenta de valoarea flag-ului) optiunile de raspuns si cele ajutatoare.

            Op0IsEnabled = Op1IsEnabled = Op2IsEnabled = Op3IsEnabled =  flag;

            for (int i = 0; i < 3; i++)

                (\_gameView.lifeLinesGrid.Children[i] as Button).IsEnabled = flag;

        }

        public void Retreat()

        {

            GameService.Retreat();

            GameOver();

        }

        public void GameOver()

        {

            \_audioService.PlayAudio(\_audioService.WrongAnswer);

            // Oprim timer-ul, dezactivam butoanele si afisam eplicatia privind raspunsul corect.

            ellapsedTime.Stop();

            ButtonsSetIsEnabledTo(false);

            \_gameView.RowForAdditionalInformation.Height = GridLength.Auto;

            \_gameView.panelExplications.Visibility = Visibility.Visible;

            string explanations = CurrentQuestion.Explanations;

            if (explanations != null && explanations.Length > 0)

                \_gameView.tbExplications.Text = explanations;

            else

                \_gameView.tbExplications.Height = 0;

            MarkCorrectOption();

        }

        public void MarkCorrectOption()

        {

            // Identifica care este butonul ce contine raspunsul corect si ii atribuie stilul RightOptionSelected

            var btCorrectOption = (\_gameView.optionsGrid.Children[CurrentQuestion.CorrectOptionIndex] as Grid).Children[0] as Button;

            btCorrectOption.Style = Application.Current.TryFindResource("RightOptionSelected") as Style;

        }

        public void FiftyFifty()

        {

            start = DateTime.Now;

            GameService.FiftyFifty();

            OnPropertyChanged(nameof(CurrentQuestion));

            \_gameView.FiftyFiftyUsed.Visibility = Visibility.Visible;

            \_gameView.btFiftyFifty.IsEnabled = false;

            Op0IsEnabled = \_gameView.option0.IsEnabled;

            Op1IsEnabled = \_gameView.option1.IsEnabled;

            Op2IsEnabled = \_gameView.option2.IsEnabled;

            Op3IsEnabled = \_gameView.option3.IsEnabled;

        }

        public void AskAudience()

        {

            start = DateTime.Now;

            List<int> results = GameService.AskAudience();

            var window = new AskAudienceView(results);

            window.ShowDialog();

            \_gameView.AskAudienceUsed.Visibility = Visibility.Visible;

            \_gameView.btAskAudience.IsEnabled = false;

        }

        public void CallFriend()

        {

            start = DateTime.Now;

            var window = new PhoneCallView(CurrentQuestion, GameService.Results.PlayerName);

            window.ShowDialog();

            \_gameView.PhoneCallUsed.Visibility = Visibility.Visible;

            \_gameView.btPhoneCall.IsEnabled = false;

        }

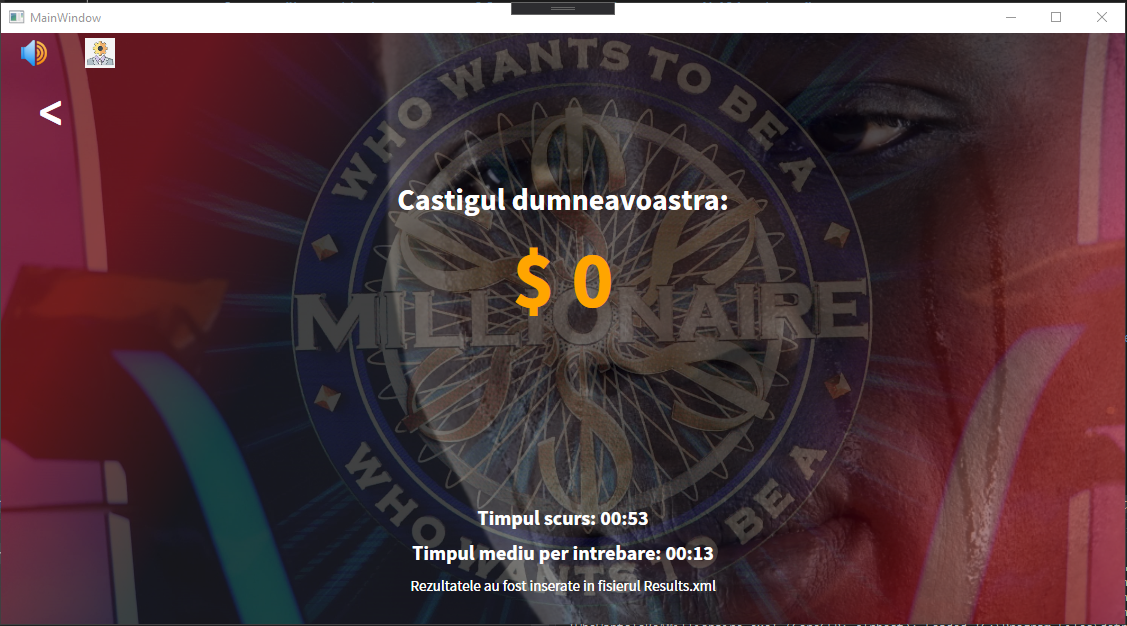
    }

}

După cum ați observat, în metoda **set** a proprietăților ce comunică cu interfața este apelată funcția OnPropertyChanged ce notifică despre modificările ce au avut loc asupra proprietății respective.

### ResultsView

Rezultatul spre care tindem: Dacă jocul a fost finalizat cu câștigul de $0



Dacă jocul a fost finalizat cu un premiu între $100 și $500 000



Dacă a fost câștigat premiul cel mare



### Conținutul resultsview.xaml

<UserControl x:Class="WhoWantsToBeAMillionaire.Views.ResultsView"

             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

             xmlns:local="clr-namespace:WhoWantsToBeAMillionaire.Views"

             mc:Ignorable="d"

             d:DesignHeight="580" d:DesignWidth="1100"

             Name="mainUserControl">

    <!--set height and width to auto when done!-->

    <!--Height="580" Width="1100"-->

    <Grid x:Name="mainGrid">

        <Grid.RowDefinitions>

            <RowDefinition Height="3\*"/>

            <RowDefinition/>

        </Grid.RowDefinitions>

        <Button Name="btHome" Style="{StaticResource btBackStyle}" Click="btHome\_Click"/>

        <StackPanel Grid.Row="0" HorizontalAlignment="Center" VerticalAlignment="Center">

            <Label VerticalAlignment="Center" FontSize="30" FontWeight="Bold" Foreground="White" Content="Castigul dumneavoastra:" HorizontalAlignment="Center"/>

            <Label FontSize="80" FontWeight="Bold" Foreground="Orange"

                   Content="{Binding PrizeWon}" HorizontalAlignment="Center"/>

        </StackPanel>

        <StackPanel Orientation="Vertical" Grid.Row="1" HorizontalAlignment="Center" VerticalAlignment="Center">

            <Label FontWeight="Bold" FontSize="20" Foreground="White"

                   HorizontalAlignment="Center"

                   ContentStringFormat="Timpul scurs: {0}"

                   Content="{Binding TotalEllapsedTime}"/>

            <Label FontWeight="Bold" FontSize="20" Foreground="White"

                   HorizontalAlignment="Center"

                   ContentStringFormat="Timpul mediu per intrebare: {0}"

                   Content="{Binding MediumTimeEllapsedPerQuestion}"/>

            <Label Content="Rezultatele au fost inserate in fisierul Results.xml" FontSize="15" Foreground="White" HorizontalAlignment="Center"/>

        </StackPanel>

    </Grid>

</UserControl>

### conținutul Resultsview.xaml.cs

using System.Windows;

using System.Windows.Controls;

using WhoWantsToBeAMillionaire.ViewModels;

namespace WhoWantsToBeAMillionaire.Views

{

    public partial class ResultsView : UserControl

    {

        public ResultsView()

        {

            InitializeComponent();

            DataContext = new ResultsViewModel(this);

        }

        private void btHome\_Click(object sender, RoutedEventArgs e)

        {

            ((MainWindow)System.Windows.Application.Current.MainWindow).UpdateView("Start");

        }

    }

}

### Conținutul resultsviewmodel.cs

using System;

using System.Windows;

using WhoWantsToBeAMillionaire.Models;

using WhoWantsToBeAMillionaire.Services;

using WhoWantsToBeAMillionaire.Views;

using WhoWantsToBeAMillionaire.ViewModels;

namespace WhoWantsToBeAMillionaire.ViewModels

{

    class ResultsViewModel : BaseViewModel

    {

        private GameService \_gameService;

        private ResultsView \_view;

        public string PrizeWon { get; set; }

        public string TotalEllapsedTime { get; set; }

        public string MediumTimeEllapsedPerQuestion { get; set; }

        public ResultsViewModel(ResultsView view)

        {

            \_view = view;

            \_gameService = GameService.GetInstace();

            PrizeWon = \_gameService.Results.FinalPrize;

            TotalEllapsedTime = \_gameService.Results.ElapsedTime.ToString(@"mm\:ss");

            if (\_gameService.Results.CorrectAnswers == 0)

                MediumTimeEllapsedPerQuestion = TotalEllapsedTime;

            else

                MediumTimeEllapsedPerQuestion = TimeSpan.FromSeconds((\_gameService.Results.ElapsedTime.Seconds / \_gameService.Results.CorrectAnswers)).ToString(@"mm\:ss");

            int prizeId = Game.PrizeList.IndexOf(PrizeWon);

            switch (prizeId)

            {

                case -1:

                    \_view.mainGrid.Style = Application.Current.TryFindResource("DangerGrid") as Style;

                    \_view.mainUserControl.Style = Application.Current.TryFindResource("NoWin") as Style;

                    break;

                case int n when(n >= 0 && n < 14):

                    \_view.mainGrid.Style = Application.Current.TryFindResource("SuccessGrid") as Style;

                    \_view.mainUserControl.Style = Application.Current.TryFindResource("Win") as Style;

                    break;

                case 14:

                    \_view.mainGrid.Style = Application.Current.TryFindResource("SuccessGrid") as Style;

                    \_view.mainUserControl.Style = Application.Current.TryFindResource("BigWin") as Style;

                    break;

            }

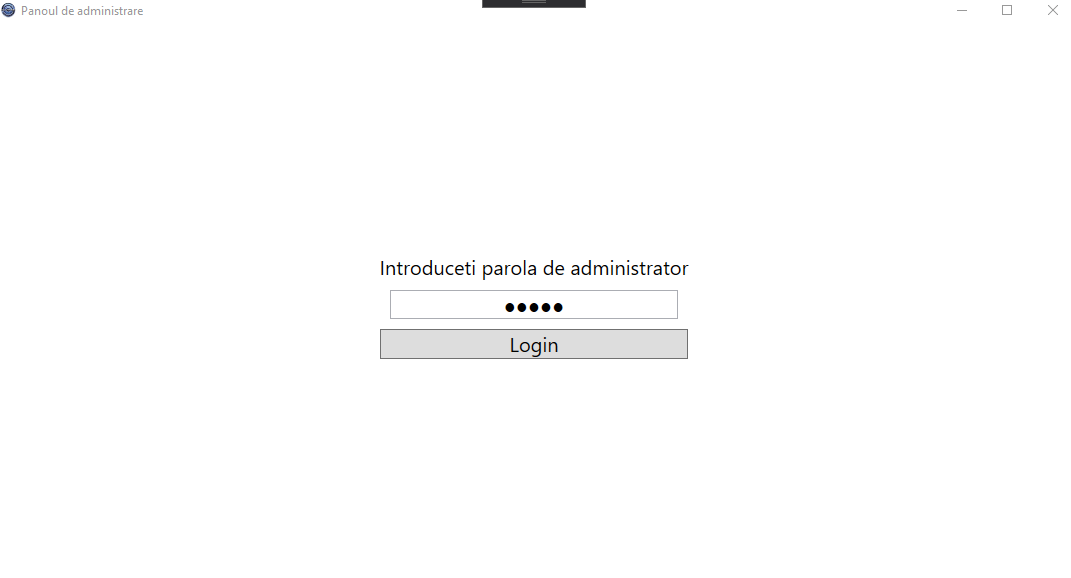
        }

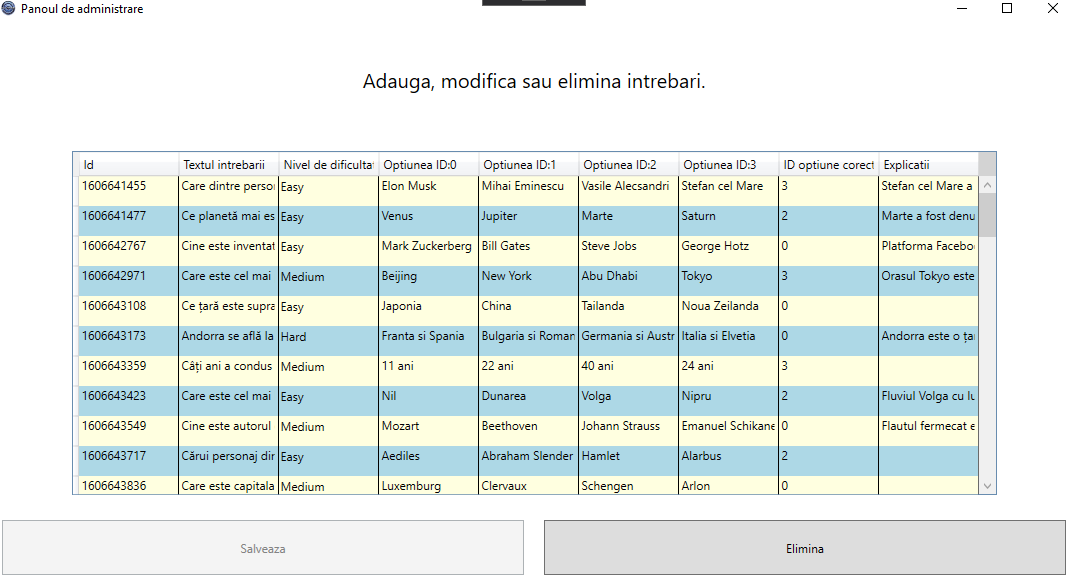
    }

}

### fereastra Adminview

Rezultatul spre care tindem



 Dacă parola introdusă este validă, vom afișa această fereastră

Din acest panou vom putea adăuga, modifica sau elimina întrebări.

### Conținutul adminview.xaml

<Window x:Class="WhoWantsToBeAMillionaire.Views.AdminView"

        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:WhoWantsToBeAMillionaire.Views"

        mc:Ignorable="d"

        Title="Panoul de administrare" Height="600" Width="1100"

        WindowStartupLocation="CenterScreen"

        Closing="Window\_Closing" Icon="pack://application:,,,/icon.ico">

    <Grid>

        <!--Grid-ul ce reprezinta pagina de logare-->

        <Grid Name="LoginPage" Visibility="{Binding LoginPageVisibility}">

            <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">

                <TextBlock Style="{StaticResource LargeText}" HorizontalAlignment="Center" Foreground="Black">

                    Introduceti parola de administrator

                </TextBlock>

                <PasswordBox Name="passwordBox" FontSize="20" Margin="10" HorizontalContentAlignment="Center"/>

                <Button x:Name="btLogin" Click="btLogin\_Click" Content="Login" FontSize="20" IsDefault="True"/>

            </StackPanel>

        </Grid>

        <!--Grid-ul ce reprezinta panoul de administrare-->

        <Grid Name="AdminPanel" Visibility="{Binding AdminPanelVisibility}">

            <Grid.RowDefinitions>

                <RowDefinition Height="3\*"/>

                <RowDefinition Height="10\*"/>

                <RowDefinition Height="2\*"/>

            </Grid.RowDefinitions>

            <Label Grid.Row="0" Content="Adauga, modifica sau elimina intrebari." HorizontalContentAlignment="Center" VerticalContentAlignment="Center"

               FontSize="20"/>

            <!--Aici vor fi afisate intrebarile-->

            <DataGrid Grid.Row="1" AutoGenerateColumns="False" HorizontalAlignment="Center" Margin="15" InitializingNewItem="QuestionsDataGrid\_InitializingNewItem"

        Name="QuestionsDataGrid" RowHeight="30" ColumnWidth="100" ItemsSource="{Binding QuestionsList}" CellEditEnding="QuestionsDataGrid\_CellEditEnding"  GridLinesVisibility="Vertical" HeadersVisibility="All"   VerticalAlignment="Center" Background="LightGray" RowBackground="LightYellow" AlternatingRowBackground="LightBlue">

                <DataGrid.Columns>

                    <DataGridTextColumn Header="Id" Binding="{Binding QuestionId, ValidatesOnExceptions=True}" IsReadOnly="True"/>

                    <DataGridTextColumn Header="Textul intrebarii" Binding="{Binding QuestionText, ValidatesOnExceptions=True}"/>

                    <DataGridComboBoxColumn Header="Nivel de dificultate" x:Name="DifficultyDropDown"

                                        SelectedItemBinding="{Binding DifficultyLevel}"/>

                    <DataGridTextColumn Header="Optiunea ID:0" Binding="{Binding Options[0], ValidatesOnExceptions=True}"/>

                    <DataGridTextColumn Header="Optiunea ID:1" Binding="{Binding Options[1], ValidatesOnExceptions=True}"/>

                    <DataGridTextColumn Header="Optiunea ID:2" Binding="{Binding Options[2], ValidatesOnExceptions=True}"/>

                    <DataGridTextColumn Header="Optiunea ID:3" Binding="{Binding Options[3], ValidatesOnExceptions=True}"/>

                    <DataGridTextColumn Header="ID optiune corecta" Binding="{Binding CorrectOptionIndex, ValidatesOnExceptions=True}"/>

                    <DataGridTextColumn Header="Explicatii" Binding="{Binding Explanations, ValidatesOnExceptions=True}"/>

                </DataGrid.Columns>

            </DataGrid>

            <Grid Grid.Row="2">

                <Grid.ColumnDefinitions>

                    <ColumnDefinition/>

                    <ColumnDefinition/>

                </Grid.ColumnDefinitions>

                <Button Name="btSave" Grid.Column="0" Content="Salveaza" Margin="10" Click="btSave\_Click" IsEnabled="False"/>

                <Button Name="btDelete" Grid.Column="1" Content="Elimina" Margin="10" Click="btDelete\_Click"/>

            </Grid>

        </Grid>

    </Grid>

</Window>

### Conținutul adminview.xaml.cs

using System.Collections.Generic;

using System.ComponentModel;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using WhoWantsToBeAMillionaire.Models;

using WhoWantsToBeAMillionaire.ViewModels;

namespace WhoWantsToBeAMillionaire.Views

{

    public partial class AdminView : Window

    {

        private AdminViewModel \_viewModel;

        public AdminView()

        {

            // Initializarea ferestrei. Setam o instanta a clasei AdminViewModel ca DataContext pentru aceasta fereastra.

            DataContext = new AdminViewModel();

            \_viewModel = DataContext as AdminViewModel;

            InitializeComponent();

            // Populam dropdown-ul cu nivelurile de dificultate

            DifficultyDropDown.ItemsSource = new List<DifficultyLevel> { DifficultyLevel.Easy, DifficultyLevel.Medium, DifficultyLevel.Hard, DifficultyLevel.Einstein };

            \_viewModel.QuestionsDataGrid = QuestionsDataGrid;

        }

        private void QuestionsDataGrid\_CellEditEnding(object sender, DataGridCellEditEndingEventArgs e)

        {

            // Daca au fost efectuate unele modificari, butonul pentru salvare a modificarilor devine activ.

            btSave.IsEnabled = true;

        }

        private void btSave\_Click(object sender, RoutedEventArgs e)

        {

            \_viewModel.Save();

            btSave.IsEnabled = false;

        }

        private void btDelete\_Click(object sender, RoutedEventArgs e)

        {

            // Eliminam inregistrarea selectata si reimprospatam datagrid-ul pentru a vizualiza imediat schimbarile.

            \_viewModel.DeleteSelectedRecord();

            CollectionViewSource.GetDefaultView(QuestionsDataGrid.ItemsSource).Refresh();

        }

        private void QuestionsDataGrid\_InitializingNewItem(object sender, InitializingNewItemEventArgs e)

        {

            // Cand este adaugat un nou rand in datagrid apelam metoda NewRow care va seta o valoare unica pentru campul QuestionId

            \_viewModel.NewRow();

        }

        private void Window\_Closing(object sender, CancelEventArgs e)

        {

            // Inainte de a inchide fereastra ne asiguram ca nu au ramas modificari nesalvate.

            if (btSave.IsEnabled)

            {

                if (!\_viewModel.CloseWindow())

                    e.Cancel = true;

            }

        }

        private void btLogin\_Click(object sender, RoutedEventArgs e)

        {

            \_viewModel.Login(passwordBox.Password);

        }

    }

}

### conținutul adminviewmodel.cs

using System;

using System.Collections.Generic;

using System.Windows;

using System.Windows.Controls;

using WhoWantsToBeAMillionaire.Models;

using WhoWantsToBeAMillionaire.Services;

namespace WhoWantsToBeAMillionaire.ViewModels

{

    class AdminViewModel : BaseViewModel

    {

        private Visibility \_loginPageVisibility;

        public Visibility LoginPageVisibility

        {

            get { return \_loginPageVisibility; }

            set

            {

                \_loginPageVisibility = value;

                OnPropertyChanged(nameof(LoginPageVisibility));

            }

        }

        private Visibility \_adminPanelVisibility;

        public Visibility AdminPanelVisibility

        {

            get { return \_adminPanelVisibility; }

            set

            {

                \_adminPanelVisibility = value;

                OnPropertyChanged(nameof(AdminPanelVisibility));

            }

        }

        private List<Question> \_questionsList;

        public List<Question> QuestionsList

        {

            get { return \_questionsList; }

            set

            {

                \_questionsList = value;

                OnPropertyChanged(nameof(QuestionsList));

            }

        }

        public List<DifficultyLevel> DifficultyLevelItems { get; set; }

        public DataGrid QuestionsDataGrid;

        public AdminViewModel()

        {

            // Initial este visibila doar pagina de logare.

            LoginPageVisibility = Visibility.Visible;

            AdminPanelVisibility = Visibility.Collapsed;

        }

        public void DeleteSelectedRecord()

        {

            // Verificam daca randul selectat reprezinta o inregistrare reala. In caz afirmativ - o eliminam, dar nu inainte

            // de a afisa un messageBox pentru confirmare.

            int selectedIndex = QuestionsDataGrid.SelectedIndex;

            if (selectedIndex > -1 && selectedIndex <= QuestionsList.Count)

            {

                MessageBoxResult messageBoxResult = MessageBox.Show(

                    "Sunteti siguri ca doriti sa excludeti intrebarea [ID: " + QuestionsList[selectedIndex].QuestionId + "]: "

                    + QuestionsList[selectedIndex].QuestionText + "?", "",

                    MessageBoxButton.YesNoCancel);

                if (messageBoxResult == MessageBoxResult.Yes)

                {

                    if (DBService.RemoveQuestion(QuestionsList[selectedIndex].QuestionId))

                        MessageBox.Show("Intrebarea a fost eliminata cu succes.");

                    QuestionsList.RemoveAt(selectedIndex);

                }

            }

        }

        public void Login(string password)

        {

            // Verificam daca parola indicata este valida. In caz afirmativ, pagina de logare este ascunsa, intrebarile sunt extrase

            // din fisierul xml si panoul de administrare devine visibil.

            if(password == "admin")

            {

                QuestionsList = DBService.GetQuestions();

                AdminPanelVisibility = Visibility.Visible;

                LoginPageVisibility = Visibility.Collapsed;

            }

            else

            {

                MessageBox.Show("Parola introdusa este invalida.");

            }

        }

        public bool CloseWindow()

        {

            // Aceasta metoda este apelata din code behind atunci cand utilizatorul doreste sa inchida fereastra in timp ce exista

            // unele modificari nesalvate. In acest caz, utilizatorul va fi instiintat printr-un messageBox. El va avea posiblitatea

            // de a salva modificarile, de a nu le salva si de a anula actiunea.

            // Daca functia returneaza true, atunci fereastra se va inchide.

            MessageBoxResult messageBoxResult = MessageBox.Show(

                "Doriti sa salvati modificarile?", "", MessageBoxButton.YesNoCancel);

            switch (messageBoxResult)

            {

                case MessageBoxResult.Yes:

                    DBService.WriteQuestionsToXmlFile(QuestionsList);

                    return true;

                case MessageBoxResult.No:

                    return true;

                default:

                    return false;

            }

        }

        public void NewRow()

        {

            // Utilizatorul a creat un nou rand in datagrid,campul pentru Id-ul intrebarii este setat automat

            // Valoarea acestuia este numarul de secunde scurse incepand cu nasterea epocii Unix - January 1st, 1970

            QuestionsList[QuestionsList.Count - 1].QuestionId = DateTimeOffset.UtcNow.ToUnixTimeSeconds().ToString();

            QuestionsList[QuestionsList.Count - 1].Options = new List<string>() { "", "", "", "" };

        }

        public void Save()

        {

            if (DBService.WriteQuestionsToXmlFile(QuestionsList))

                MessageBox.Show("Modificarile au fost efectuate cu succes!");

            else

                MessageBox.Show("S-au produs erori la salvarea datelor.");

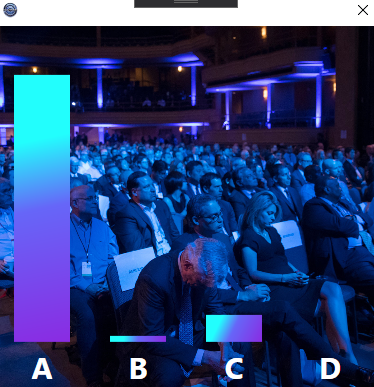
        }

    }

}

### Fereastra askaudienceview

Rezultatul spre care tindem:



Sarcina acestei ferestre este de a transforma lista de rezultate ale sondajului pe care o primește de la LifelinesService astfel încăt să poată fi vizualizată sub forma unei diagrame cu bare.

### Conținutul Askaudienceview.xaml

<Window x:Class="WhoWantsToBeAMillionaire.Views.AskAudienceView"

        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:WhoWantsToBeAMillionaire.Views"

        mc:Ignorable="d"

        Height="400" Width="400"

        WindowStartupLocation="CenterScreen"

        ResizeMode="NoResize"

        Icon="pack://application:,,,/icon.ico">

    <Grid>

        <Grid.Background>

            <ImageBrush ImageSource="pack://application:,,,/WhoWantsToBeAMillionaire;component/Resources/Media/audiencebg.jpg" Stretch="UniformToFill"/>

        </Grid.Background>

        <Grid.ColumnDefinitions>

            <ColumnDefinition/>

            <ColumnDefinition/>

            <ColumnDefinition/>

            <ColumnDefinition/>

        </Grid.ColumnDefinitions>

        <Grid.RowDefinitions>

            <RowDefinition Height="7\*"/>

            <RowDefinition/>

        </Grid.RowDefinitions>

        <!--

        Pentru fiecare optiune este creat un obiect de tip Rectangle. Fiecarui ii atribuim 2 animatii:

        Prima animatie: Inaltimea obiectului creste de la valoarea 0 la MaximumHeight, indicata in viewModel.

        Prima animatie pentru fiecare dreptunghi incepe cu un mic delay (0.5 secunde) fata de dreptunghul precedent

        simuland astfel efectul de val (wave), dupa care se revine la inaltimea initiala - 0.

        Presupunem ca in timpul in care este efectuata aceasta animatie, publicul face o alegere.

        A doua animatie: Aceasta incepe odata ce se termina prima. Inaltimea dreptunghiului creste de la 0 la

        RectangleHeight, indicata in viewModel pentru fiecare dreptunghi aparte.

        -->

        <Rectangle Grid.Row="0" Grid.Column="0" Style="{StaticResource AskAudienceGraphBar}" Name="rectangle1">

            <Rectangle.Triggers>

                <EventTrigger RoutedEvent="Rectangle.Loaded">

                    <BeginStoryboard>

                        <Storyboard>

                            <DoubleAnimation

                                Storyboard.TargetName="rectangle1"

                                Storyboard.TargetProperty="Height"

                                From="0" To="{Binding MaximumHeight}" Duration="0:0:2" AutoReverse="True"/>

                            <DoubleAnimation

                                BeginTime="0:0:7"

                                Storyboard.TargetName="rectangle1"

                                Storyboard.TargetProperty="Height"

                                From="0" To="{Binding RectangleAHeight}" Duration="0:0:2" AutoReverse="False"/>

                        </Storyboard>

                    </BeginStoryboard>

                </EventTrigger>

            </Rectangle.Triggers>

        </Rectangle>

        <Rectangle Grid.Row="0" Grid.Column="1" Style="{StaticResource AskAudienceGraphBar}" Name="rectangle2">

            <Rectangle.Triggers>

                <EventTrigger RoutedEvent="Rectangle.Loaded">

                    <BeginStoryboard>

                        <Storyboard>

                            <DoubleAnimation

                                BeginTime="0:0:0.5"

                                Storyboard.TargetName="rectangle2"

                                Storyboard.TargetProperty="Height"

                                From="0" To="{Binding MaximumHeight}" Duration="0:0:2" AutoReverse="True"/>

                            <DoubleAnimation

                                BeginTime="0:0:6.5"

                                Storyboard.TargetName="rectangle2"

                                Storyboard.TargetProperty="Height"

                                From="0" To="{Binding RectangleBHeight}" Duration="0:0:2" AutoReverse="False"/>

                        </Storyboard>

                    </BeginStoryboard>

                </EventTrigger>

            </Rectangle.Triggers>

        </Rectangle>

        <Rectangle Grid.Row="0" Grid.Column="2" Style="{StaticResource AskAudienceGraphBar}" Name="rectangle3">

            <Rectangle.Triggers>

                <EventTrigger RoutedEvent="Rectangle.Loaded">

                    <BeginStoryboard>

                        <Storyboard>

                            <DoubleAnimation

                                BeginTime="0:0:1"

                                Storyboard.TargetName="rectangle3"

                                Storyboard.TargetProperty="Height"

                                From="0" To="{Binding MaximumHeight}" Duration="0:0:2" AutoReverse="True"/>

                            <DoubleAnimation

                                BeginTime="0:0:6"

                                Storyboard.TargetName="rectangle3"

                                Storyboard.TargetProperty="Height"

                                From="0" To="{Binding RectangleCHeight}" Duration="0:0:2" AutoReverse="False"/>

                        </Storyboard>

                    </BeginStoryboard>

                </EventTrigger>

            </Rectangle.Triggers>

        </Rectangle>

        <Rectangle Grid.Row="0" Grid.Column="3" Style="{StaticResource AskAudienceGraphBar}" Height="0" Name="rectangle4">

            <Rectangle.Triggers>

                <EventTrigger RoutedEvent="Rectangle.Loaded">

                    <BeginStoryboard>

                        <Storyboard>

                            <DoubleAnimation

                                BeginTime="0:0:1.5"

                                Storyboard.TargetName="rectangle4"

                                Storyboard.TargetProperty="Height"

                                From="0" To="{Binding MaximumHeight}" Duration="0:0:2" AutoReverse="True"/>

                            <DoubleAnimation

                                BeginTime="0:0:5.5"

                                Storyboard.TargetName="rectangle4"

                                Storyboard.TargetProperty="Height"

                                From="0" To="{Binding RectangleDHeight}" Duration="0:0:2" AutoReverse="False"/>

                        </Storyboard>

                    </BeginStoryboard>

                </EventTrigger>

            </Rectangle.Triggers>

        </Rectangle>

        <Label Grid.Row="1" Grid.Column="0" Style="{StaticResource AskAudienceLabel}" Content="A"/>

        <Label Grid.Row="1" Grid.Column="1" Style="{StaticResource AskAudienceLabel}" Content="B"/>

        <Label Grid.Row="1" Grid.Column="2" Style="{StaticResource AskAudienceLabel}" Content="C"/>

        <Label Grid.Row="1" Grid.Column="3" Style="{StaticResource AskAudienceLabel}" Content="D"/>

    </Grid>

</Window>

### Conținutul askaudienceview.xaml.cs

using System.Collections.Generic;

using System.Windows;

using WhoWantsToBeAMillionaire.ViewModels;

namespace WhoWantsToBeAMillionaire.Views

{

    public partial class AskAudienceView : Window

    {

        public AskAudienceView(List<int> results)

        {

            // Este creata o noua instanta a clasei AskAudienceViewModel ce este atrbuita ca DataContext pentru fereastra.

            DataContext = new AskAudienceViewModel(results);

            InitializeComponent();

        }

    }

}

### Conținutul askaudienceviewmodel.cs

using System.Collections.Generic;

namespace WhoWantsToBeAMillionaire.ViewModels

{

    public class AskAudienceViewModel : BaseViewModel

    {

        private int \_maximumHeight;

        public int MaximumHeight

        {

            get

            {

                return \_maximumHeight;

            }

            set

            {

                \_maximumHeight = value;

                OnPropertyChanged(nameof(MaximumHeight));

            }

        }

        private int \_rectangleAHeight;

        public int RectangleAHeight

        {

            get

            {

                return \_rectangleAHeight;

            }

            set

            {

                \_rectangleAHeight = value;

                OnPropertyChanged(nameof(RectangleAHeight));

            }

        }

        private int \_rectangleBHeight;

        public int RectangleBHeight

        {

            get

            {

                return \_rectangleBHeight;

            }

            set

            {

                \_rectangleBHeight = value;

                OnPropertyChanged(nameof(RectangleBHeight));

            }

        }

        private int \_rectangleCHeight;

        public int RectangleCHeight

        {

            get

            {

                return \_rectangleCHeight;

            }

            set

            {

                \_rectangleCHeight = value;

                OnPropertyChanged(nameof(RectangleCHeight));

            }

        }

        private int \_rectangleDHeight;

        public int RectangleDHeight

        {

            get

            {

                return \_rectangleDHeight;

            }

            set

            {

                \_rectangleDHeight = value;

                OnPropertyChanged(nameof(RectangleDHeight));

            }

        }

        public AskAudienceViewModel(List<int> results)

        {

            // Inaltimea maxima la care poate ajunge un dreptunghi

            MaximumHeight = 300;

            // Constructorul a primit ca parametru de intrare lista cu rezultate in urma sondajului.

            // Lista consta din 4 valori ce reprezinta procentajele din public ce a votat pentru fiecare optiune.

            // Exemplu: results[0] = 30     pentru optiunea A au votat 30% din public.

            // Exemplu: results[1] = 10     pentru optiunea B au votat 10% din public.

            // Avand lungimea maxima a dreptunghiului, putem calcula cum poate fi reprezentat procentajul (de la 1 la 100)

            // intr-o alta valoare de la 1 la MaximumHeight.

            RectangleAHeight = MaximumHeight \* results[0] / 100;

            RectangleBHeight = MaximumHeight \* results[1] / 100;

            RectangleCHeight = MaximumHeight \* results[2] / 100;

            RectangleDHeight = MaximumHeight \* results[3] / 100;

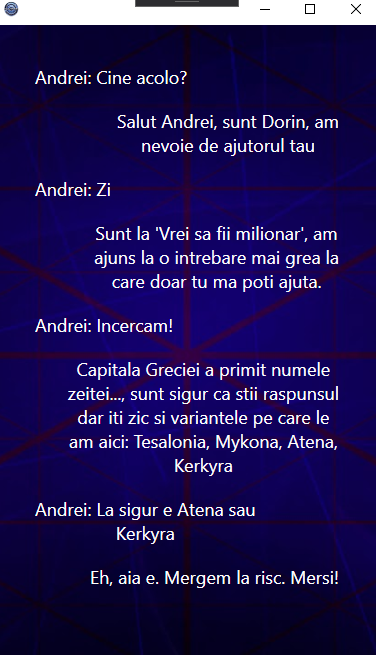
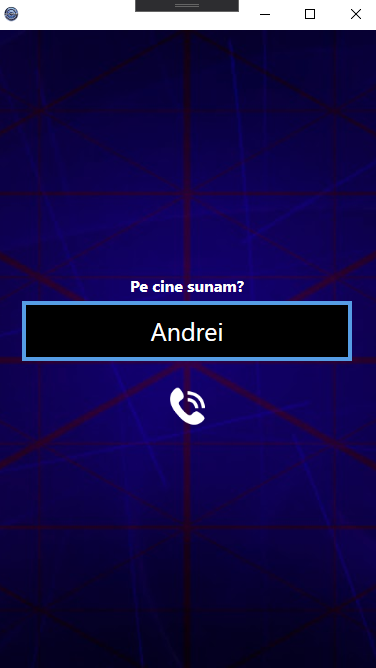
        }

    }

}

### Fereastra phonecallview

Rezultatul spre care tindem



După ce jucătorul a indicat numele persoanei pe care dorește să o telefoneze, butonul de telefonare se activează. Un click pe acest buton va trece la etapa următoare – cea de generare a conversației de către LifelineService. Conversația primită este mai apoi afișată la ecran cu un delay de 2 secunde între replici.

### Conținutul phonecallview.xaml

<Window x:Class="WhoWantsToBeAMillionaire.Views.PhoneCallView"

        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:WhoWantsToBeAMillionaire.Views"

        mc:Ignorable="d"

        Height="700" Width="400"

        WindowStartupLocation="CenterScreen"

        Icon="pack://application:,,,/icon.ico">

    <Grid>

        <Grid.Background>

            <ImageBrush ImageSource="pack://application:,,,/WhoWantsToBeAMillionaire;component/Resources/Media/phonefriendbg.jpg" Stretch="UniformToFill"/>

        </Grid.Background>

        <!--Grid-ul pentru indicarea numelui persoanei ce urmeaza a fi "telefonata"-->

        <Grid Name="GridInput" Visibility="{Binding GridInputVisibility}">

            <StackPanel Orientation="Vertical" Margin="20" HorizontalAlignment="Center" VerticalAlignment="Center">

                <Label Content="Pe cine sunam?" FontWeight="Bold"

                   Foreground="White" FontSize="16"

                   HorizontalContentAlignment="Center" VerticalContentAlignment="Center"/>

                <TextBox Name="tbFriendName"

                     Style="{StaticResource SimpleTextBox}"

                     Width="330"

                     TextChanged="tbPlayerName\_TextChanged"

                     />

                <Button Name="ButtonCall" Style="{StaticResource EmptyButton}"

                        Height="50" Margin="20" IsEnabled="{Binding CallButtonIsEnabled}"

                        Click="ButtonCall\_Click">

                    <Image Source="pack://application:,,,/WhoWantsToBeAMillionaire;component/Resources/Media/phonecall.png"/>

                </Button>

            </StackPanel>

        </Grid>

        <!--Grid-ul pentru afisarea dialogului propriu zis-->

        <Grid Name="GridDialog" Visibility="{Binding GridDialogVisibility}">

            <ScrollViewer VerticalScrollBarVisibility="Auto">

                <!--In acest dockpanel vor fi adaugate replicile in mod programat-->

                <DockPanel Name="DialogDockPanel" Margin="30">

                </DockPanel>

            </ScrollViewer>

        </Grid>

    </Grid>

</Window>

### Conținutul phonecallview.xaml.cs

using System.Windows;

using System.Windows.Controls;

using WhoWantsToBeAMillionaire.Models;

using WhoWantsToBeAMillionaire.ViewModels;

namespace WhoWantsToBeAMillionaire.Views

{

    public partial class PhoneCallView : Window

    {

        private PhoneCallViewModel \_viewModel;

        public PhoneCallView(Question question, string playerName)

        {

            // Cream o noua instanta a clasei PhoneCallViewModel pe care o atribuim ca Datacontext pentru fereastra

            InitializeComponent();

            DataContext = new PhoneCallViewModel(question, playerName);

            \_viewModel = DataContext as PhoneCallViewModel;

        }

        private void tbPlayerName\_TextChanged(object sender, TextChangedEventArgs e)

        {

            // Activam butonul de telefonare daca a fost indicata vreo valoare in campul pentru indicarea numelui

            if ((sender as TextBox).Text.Length > 0)

                \_viewModel.CallButtonIsEnabled = true;

            else

                \_viewModel.CallButtonIsEnabled = false;

        }

        private void ButtonCall\_Click(object sender, RoutedEventArgs e)

        {

            // Trecem nemijlocit la afisarea dialogului.

            \_viewModel.FriendName = tbFriendName.Text;

            \_viewModel.DisplayConversation(DialogDockPanel);

        }

    }

}

### Conținutul phonecallviewmodel.cs

using System.Collections.Generic;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using WhoWantsToBeAMillionaire.Models;

using WhoWantsToBeAMillionaire.Services;

namespace WhoWantsToBeAMillionaire.ViewModels

{

    class PhoneCallViewModel : BaseViewModel

    {

        private bool \_callButtonIsEnabled;

        public bool CallButtonIsEnabled

        {

            get

            {

                return \_callButtonIsEnabled;

            }

            set

            {

                \_callButtonIsEnabled = value;

                OnPropertyChanged(nameof(CallButtonIsEnabled));

            }

        }

        private Visibility \_gridInputVisibility;

        public Visibility GridInputVisibility

        {

            get

            {

                return \_gridInputVisibility;

            }

            set

            {

                \_gridInputVisibility = value;

                OnPropertyChanged(nameof(GridInputVisibility));

            }

        }

        private Visibility \_gridDialogVisibility;

        public Visibility GridDialogVisibility

        {

            get

            {

                return \_gridDialogVisibility;

            }

            set

            {

                \_gridDialogVisibility = value;

                OnPropertyChanged(nameof(GridDialogVisibility));

            }

        }

        public string FriendName { get; set; }

        public string PlayerName { get; set; }

        public List<string> Conversation { get; set; }

        public Question CurrentQuestion{ get; set; }

        private AudioService \_audioService;

        public PhoneCallViewModel(Question question, string playerName)

        {

            PlayerName = playerName;

            CurrentQuestion = question;

            \_audioService = AudioService.GetInstace();

            GridInputVisibility = Visibility.Visible;

            GridDialogVisibility = Visibility.Collapsed;

        }

        public async void DisplayConversation(DockPanel parent)

        {

            GridInputVisibility = Visibility.Collapsed;

            GridDialogVisibility = Visibility.Visible;

            \_audioService.PlayAudio(\_audioService.PhoneAFriend);

            // Facem rost de lista de replici

            Conversation = LifelineService.CallAFriend(FriendName, PlayerName, CurrentQuestion);

            // Pentru fiecare replica este creat un textblock.

            // care este adaugat in mod programat in lista dockpanel-ul pentru replici.

            // Va fi un delay de 2 secunde inainte de a aparea urmatoarea replica.

            for (int i = 0; i < Conversation.Count; i++)

            {

                TextBlock tbReply = new TextBlock();

                if (i % 2 == 1)

                {

                    tbReply.Style = Application.Current.TryFindResource("playerReply") as Style;

                    tbReply.Text = Conversation[i];

                }

                else

                {

                    tbReply.Style = Application.Current.TryFindResource("friendReply") as Style;

                    tbReply.Text = $"{FriendName}: {Conversation[i]}";

                }

                parent.Children.Add(tbReply);

                await Task.Delay(2000);

            }

        }

    }

}

# notă de final

Printre metodele de îmbunătățire a aplicației se numără:

* Crearea unui fișier XML de configurare, acesta va conține parola de administrator și numărul de secunde per întrebare. Aceste date vor putea fi modificate din panoul de administrare.
* Criptarea fișierelor XML pentru ca utilizatorii aplicației să nu poată vedea întrebările și răspunsurile.
* Adăugarea mai multor opțiuni ajutătoare.

Pentru a fi la curent cu următoarele versiuni / modificări / îmbunătățiri ale aplicației,

puteți consulta repozitoriul

[**https://github.com/BabaDorin/who-wants-to-be-a-millionaire**](https://github.com/BabaDorin/who-wants-to-be-a-millionaire)