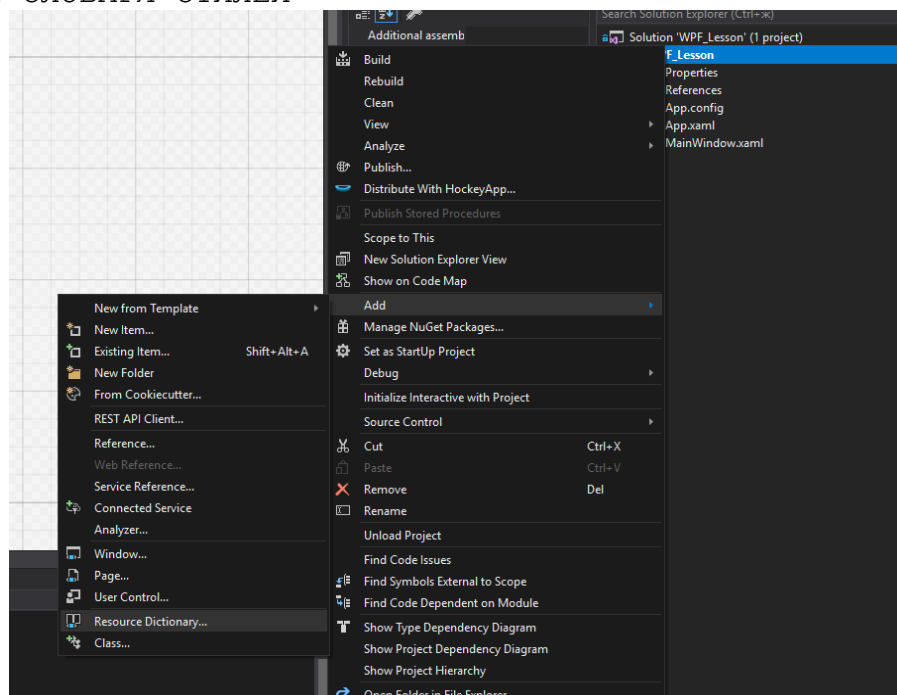
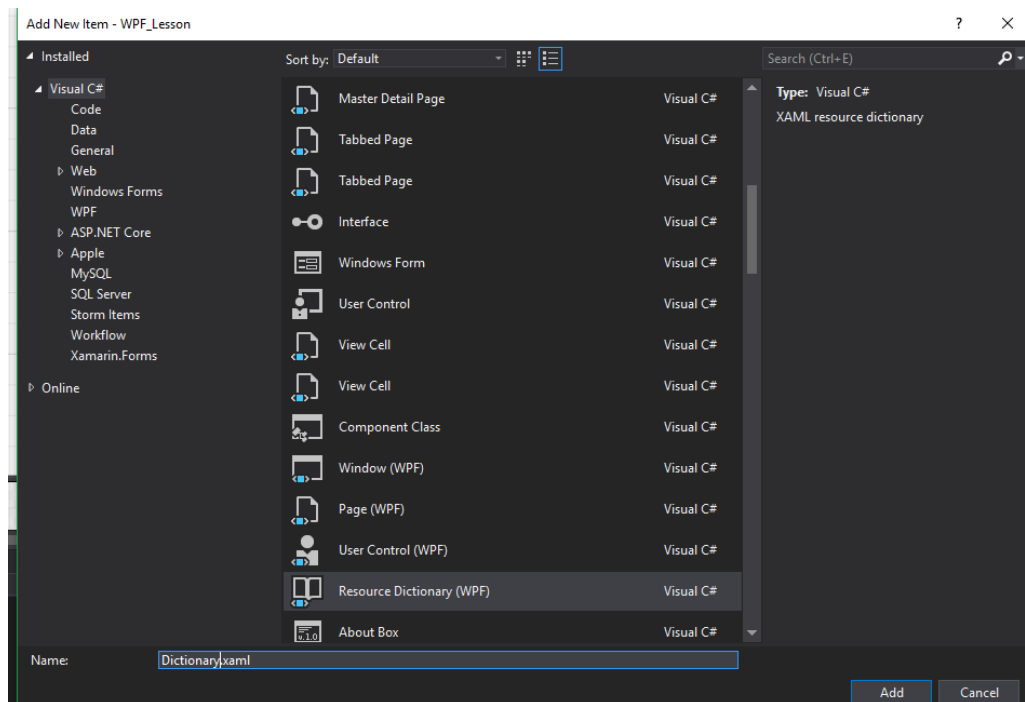


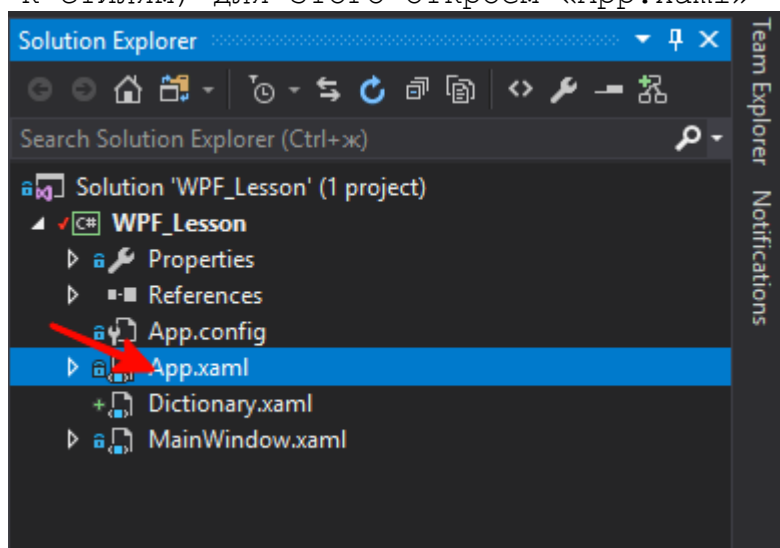
## СОЗДАНИЕ СЛОВАРЯ СТИЛЕЙ



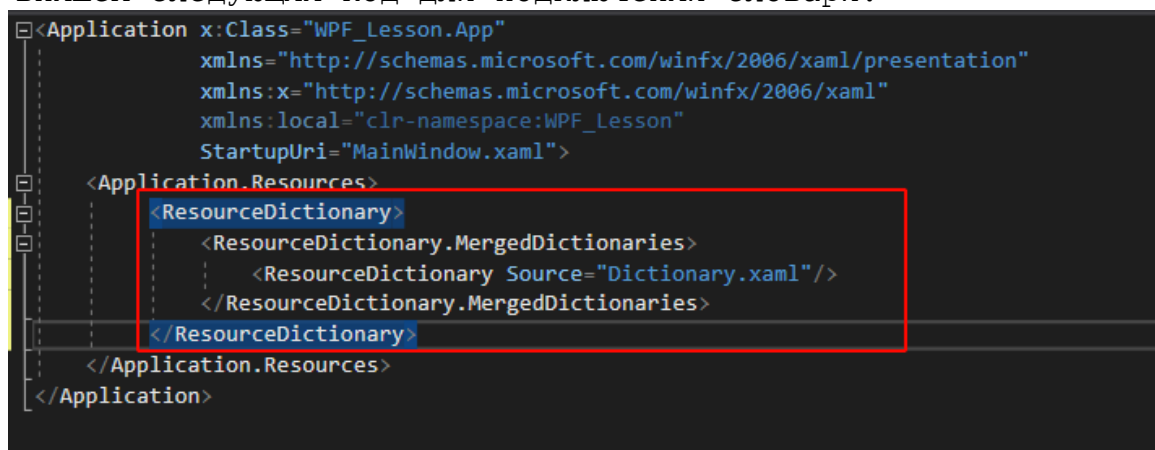
Укажем имя «Dictionary.xaml» и создадим его.



Теперь словарь необходимо подключить к проекту (чтобы компоненты могли обращаться к стилям) для этого откроем «App.xaml»

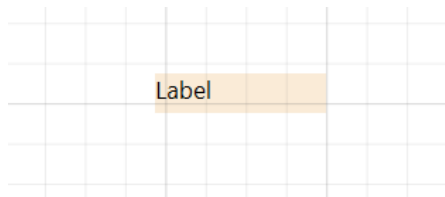


И впишем следующий код для подключения словаря.



После данных операций можно начинать писать стили и константы. Напишем первый стиль, который будет менять задний фон и внутренний отступ, и он применяться ко всем компонентам типа «Label»

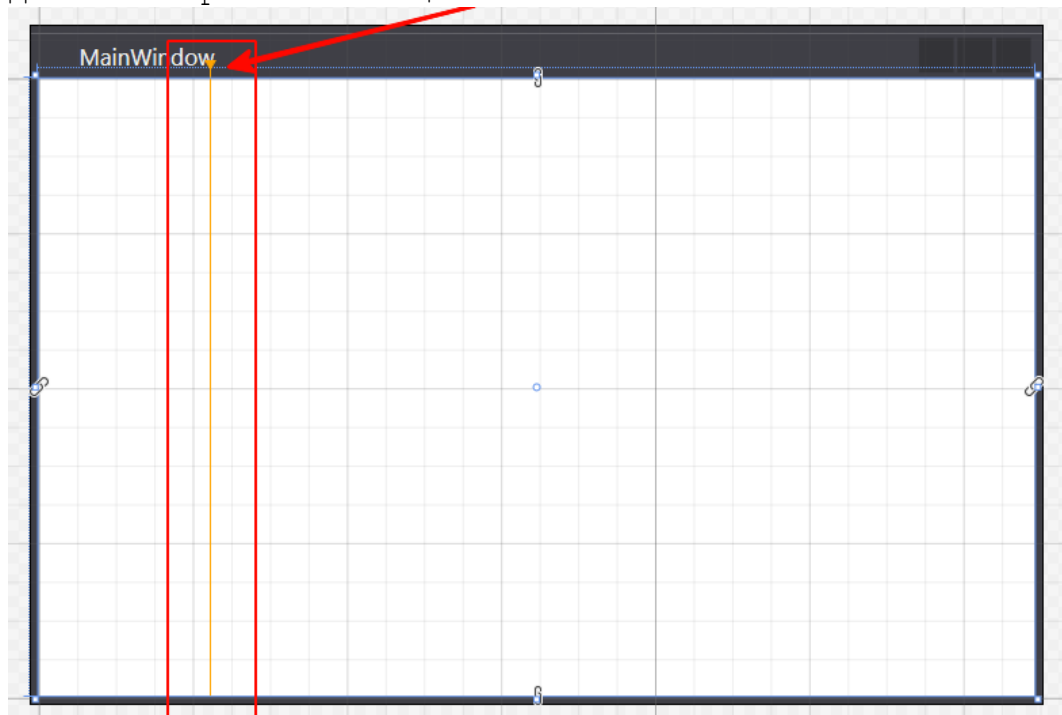
```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:local="clr-namespace:WPF_Lesson">
    <Style TargetType="{x:Type Label}">
        <Setter Property="Background" Value="AntiqueWhite"/>
        <Setter Property="Padding" Value="0"/>
    </Style>
</ResourceDictionary>
```



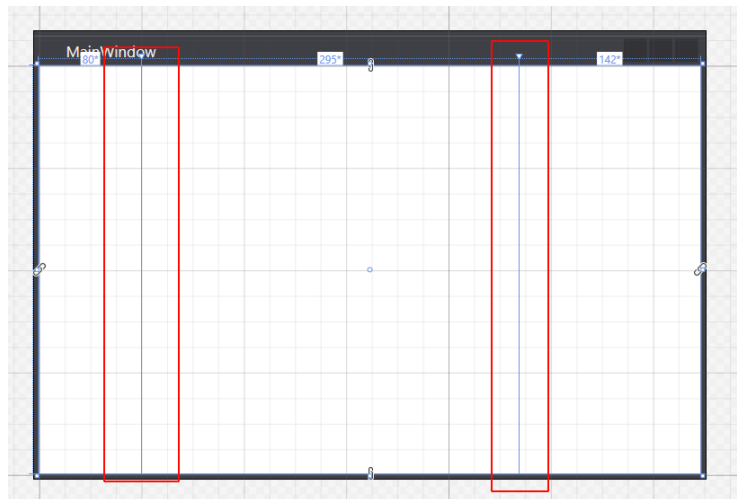
#### РАЗМЕЩЕНИЕ КОНТЕНТА ПО ЦЕНТРУ ФОРМЫ

Откройте форму или страницу. Выберите компонент Grid, после чего по краям (сверху и слева) появится возможность добавлять строки и столбцы.

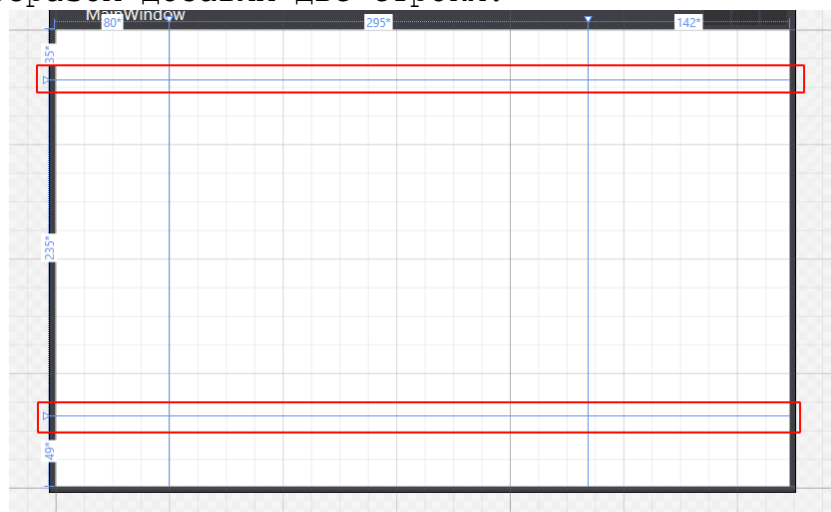
Добавим первый столбец.



Добавим еще один столбец.



Таким же образом добавим две строки.



После всех добавление мышкой, мы можем увидеть следующий код XAML.

```

<Window x:Class="MainWindow"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="35*" />
            <RowDefinition Height="235*" />
            <RowDefinition Height="49*" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="80*" />
            <ColumnDefinition Width="295*" />
            <ColumnDefinition Width="142*" />
        </Grid.ColumnDefinitions>
    </Grid>
</Window>

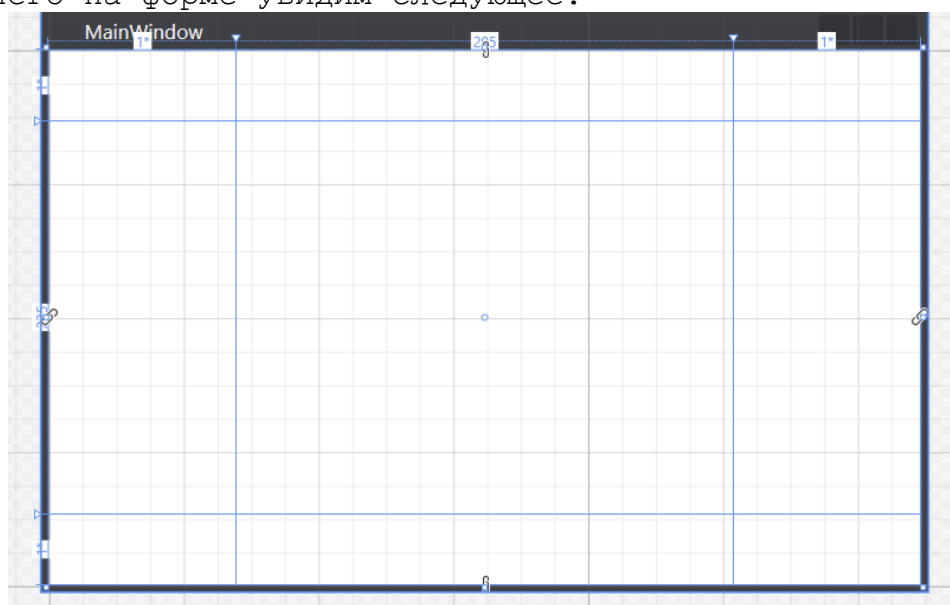
```

Так же столбцы и строки можно добавлять, редактируя код XAML. Здесь же можно редактировать размер строк и столбцов. Чтобы разместить

элементы по центру (которые будут во второй строки и во втором столбце) необходимо задать размеры строкам и столбцам. Зададим столбцам и строкам следующие размеры.

```
<Grid.RowDefinitions>
  <RowDefinition Height="*" />
  <RowDefinition Height="235" />
  <RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
  <ColumnDefinition Width="*" />
  <ColumnDefinition Width="295" />
  <ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
```

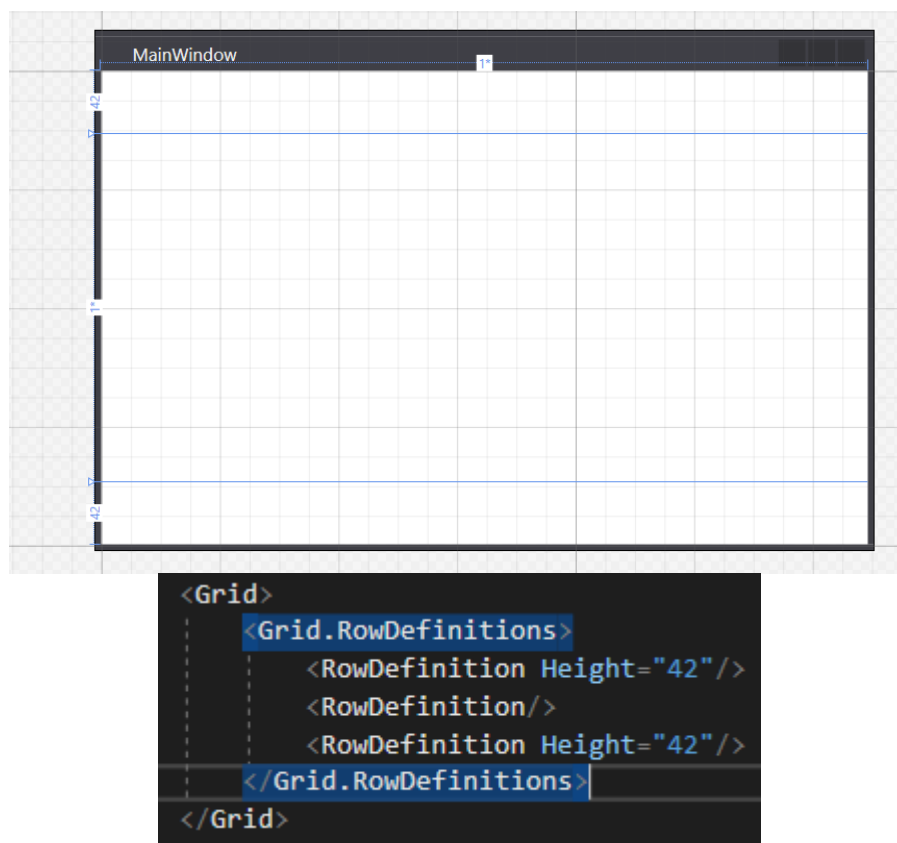
После чего на форме увидим следующее.



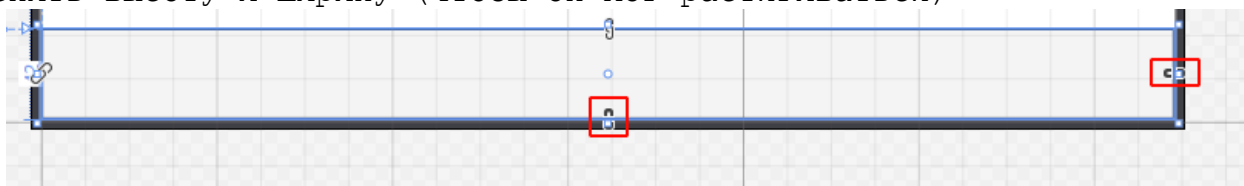
В результате средний столбец и средняя строка имеют фиксированный размер, в то время как последние и первые имеют динамический размер (меняется в зависимости от размера контейнера, окна).

#### СОЗДАНИЕ БАЗОВОЙ ФОРМЫ

Создадим несколько строк с фиксированными размерами.



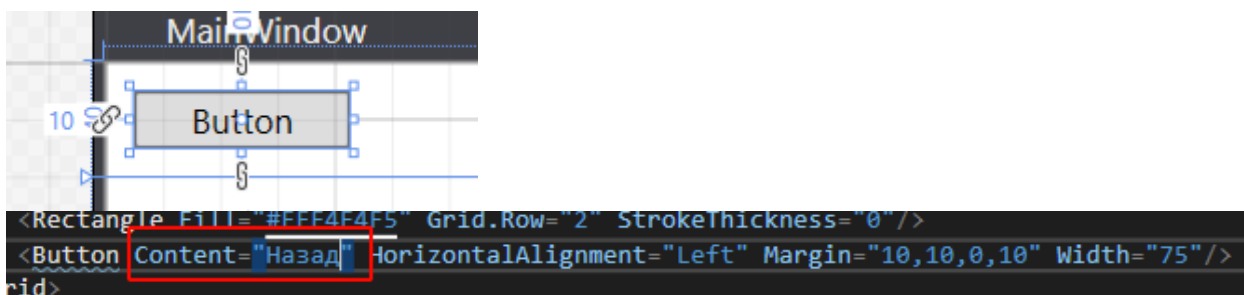
Добавим компонент «Rectangle» на форму и разместим его в 3 строке. Компонент необходимо растянуть на всю ширину и высоту строки, а затем закрепить высоту и ширину (чтобы он мог растягиваться)



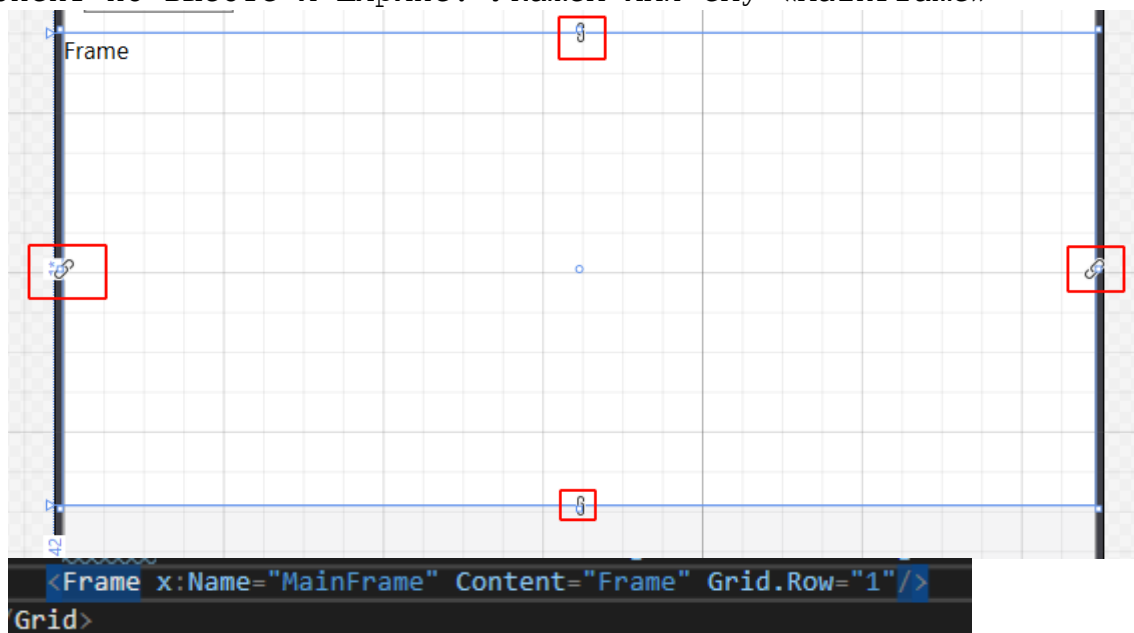
Укажем у компонента «Rectangle» свойство «StrokeThickness» равное 0, это задаст размер границы прямоугольника.

```
</Grid.RowDefinitions>
<Rectangle Fill="#FFF4F4F5" Grid.Row="2" StrokeThickness="0"/>
id>
```

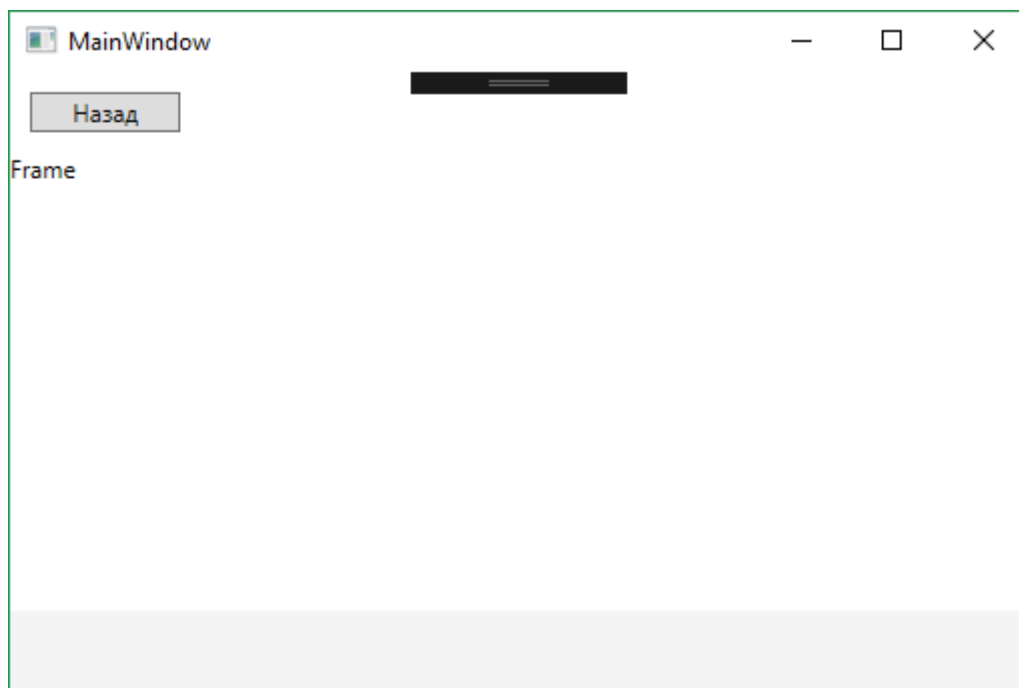
Добавим на форму кнопку и разместим в левом углу первой строки. И укажем имя кнопки «Назад», которая в последствии будет выполнять соответствующую функцию.



Добавим на форму компонент «Frame» во вторую строку, в него будут загружаться страницы (Авторизация, Регистрация и другие). Закрепим компонент по высоте и ширине. Укажем имя ему «MainFrame»

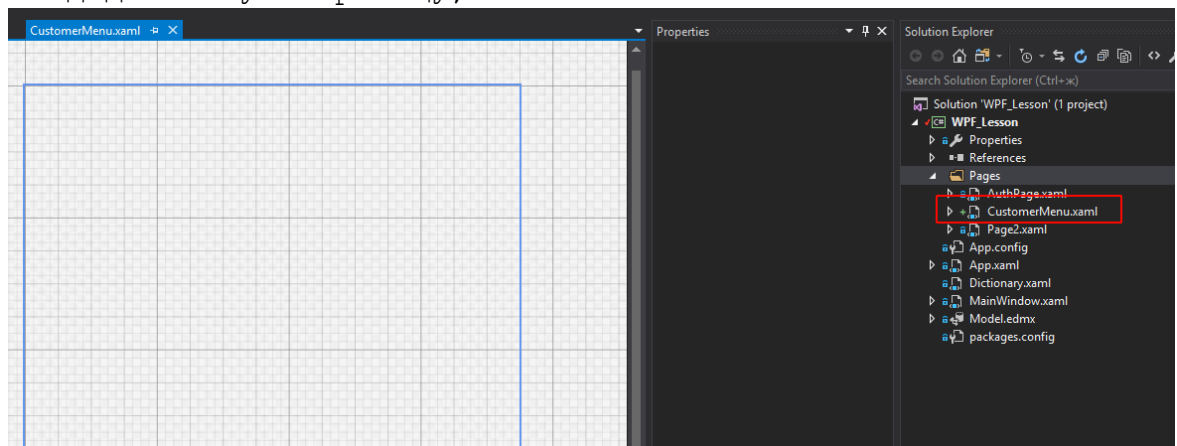


Запустим проект и увидим примерно следующее.

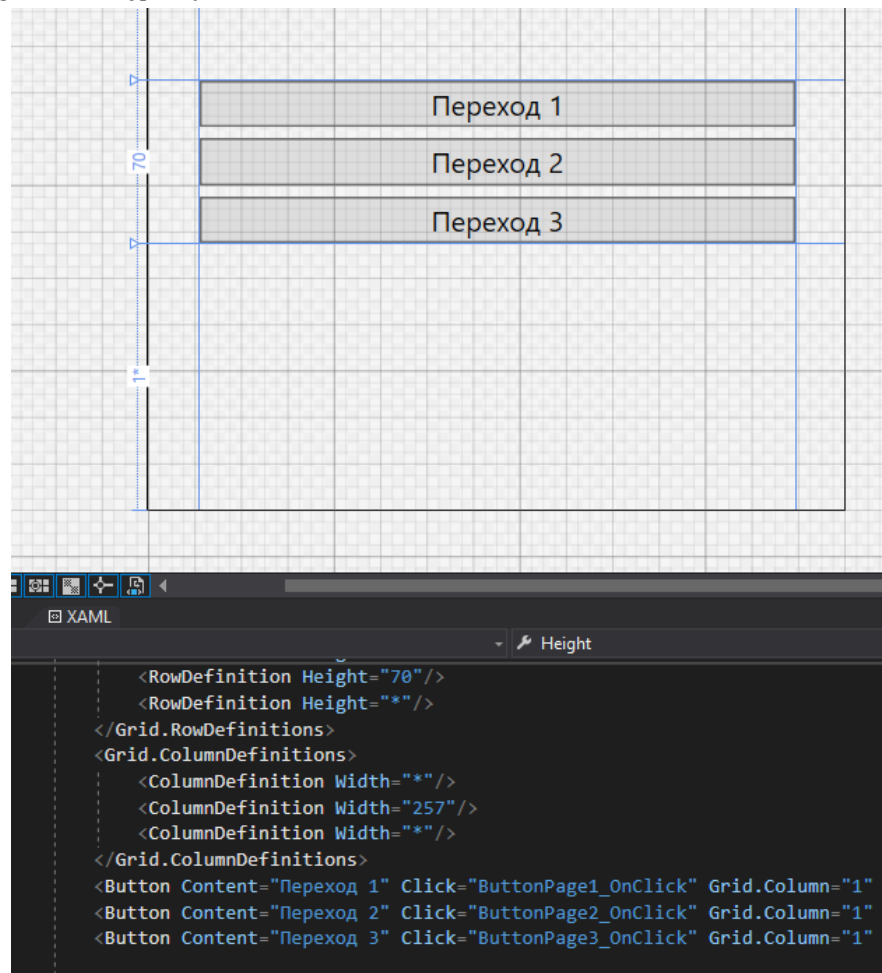


## СОЗДАНИЕ МЕНЮ ПОЛЬЗОВАТЕЛЯ

Создадим новую страницу, назовем ее CustomerMenu.



Добавим несколько кнопок, разместим их по центру и добавим им обработчик события клик.



И в коде каждому обработчику укажем свой переход



```

- references | 0 changes | 0 authors, 0 changes
private void ButtonPage1_OnClick(object sender, RoutedEventArgs e)
{
    NavigationService?.Navigate(new Page2());
}

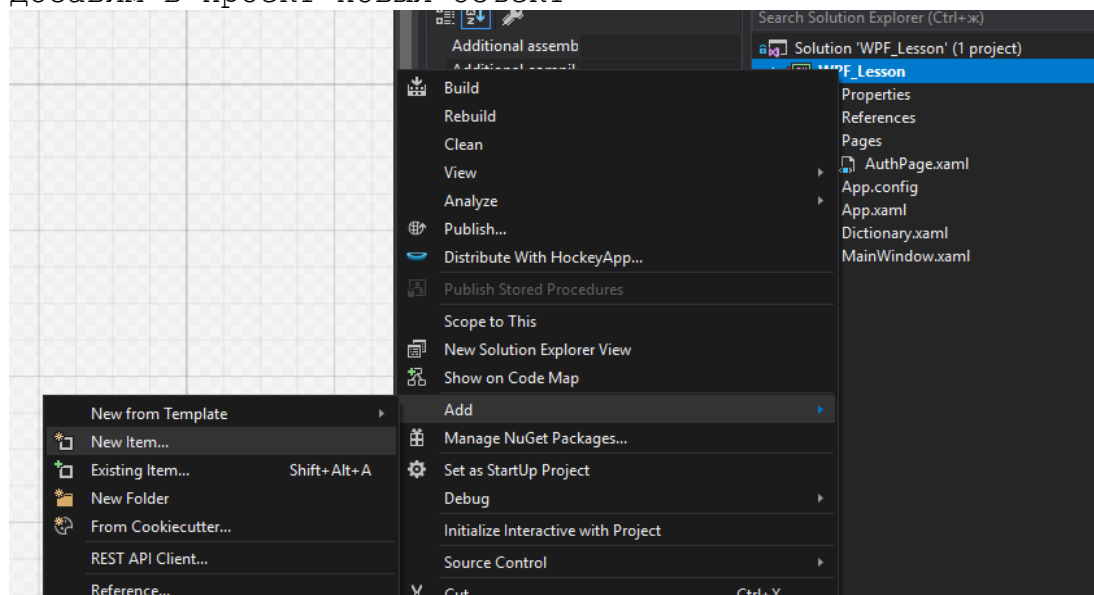
- references | 0 changes | 0 authors, 0 changes
private void ButtonPage2_OnClick(object sender, RoutedEventArgs e)
{
    NavigationService?.Navigate(new AuthPage());
}

- references | 0 changes | 0 authors, 0 changes
private void ButtonPage3_OnClick(object sender, RoutedEventArgs e)
{
    NavigationService?.Navigate(new Menu());
}

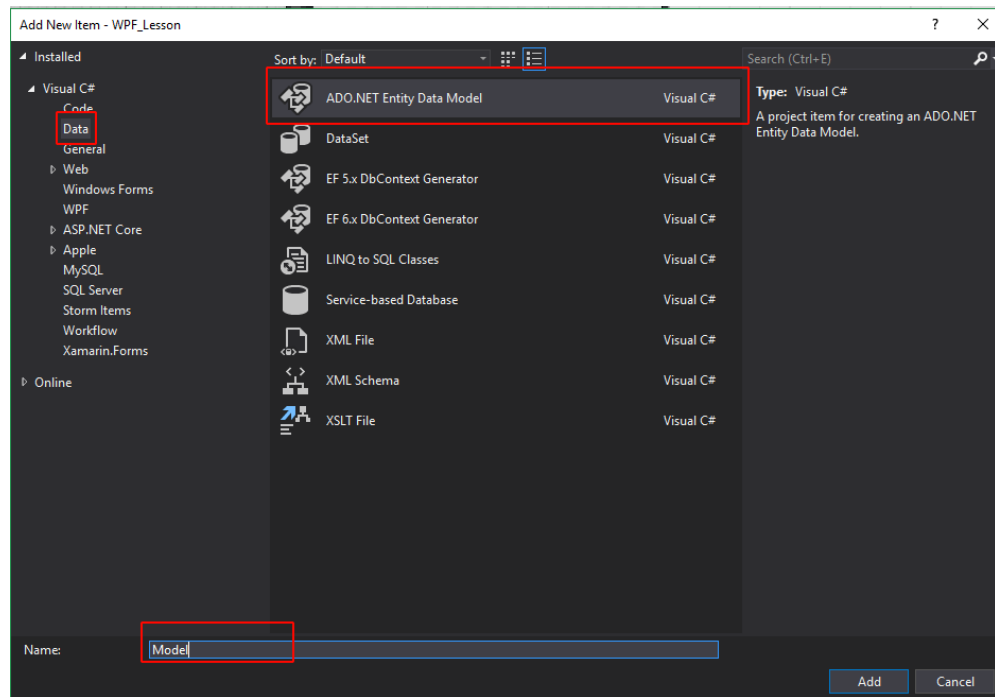
```

## СОЗДАНИЕ ПОДКЛЮЧЕНИЯ К БАЗЕ ДАННЫХ

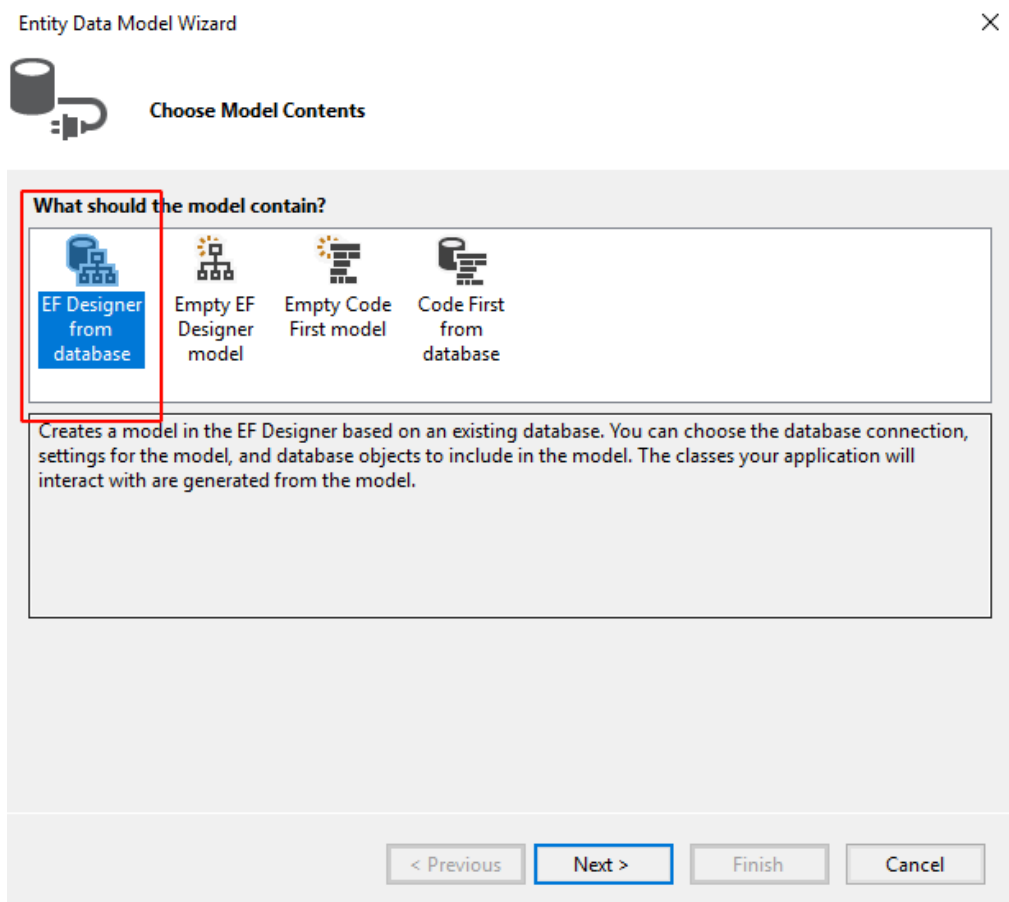
Добавим в проект новый объект



Выберем Data – ADO.NET Entity Data Model, назовем Model и добавим.




Появится диалоговое окно и выберем пункт «EF Designer from database»



После появится следующее окно. Нажмем «New Connection»

Entity Data Model Wizard ×

 **Choose Your Data Connection**

**Which data connection should your application use to connect to the database?**

Database.mdf1 New Connection...

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

☐ No, exclude sensitive data from the connection string. I will set it in my application code.

☐ Yes, include the sensitive data in the connection string.

**Connection string:**

```
metadata=res://*/Model.csdl|res://*/Model.ssdl|
res://*/Model.msl;provider=System.Data.SqlClient;provider connection string="data source=
(LocalDB)\MSSQLLocalDB;attachdbfilename=C:\Users\Maxim\Desktop\IntegrationEmulator
\IntegrationEmulator\bin\Debug\Database.mdf;integrated security=True;connect
timeout=30;MultipleActiveResultSets=True;App=EntityFramework"
```

☒ Save connection settings in App.Config as:

DatabaseEntities

< Previous **Next >** Finish Cancel

В данном окне выберем «Change».

Connection Properties ? ×

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:

MySQL Database (MySQL Data Provider) Change...

Server name:

User name:

Password:

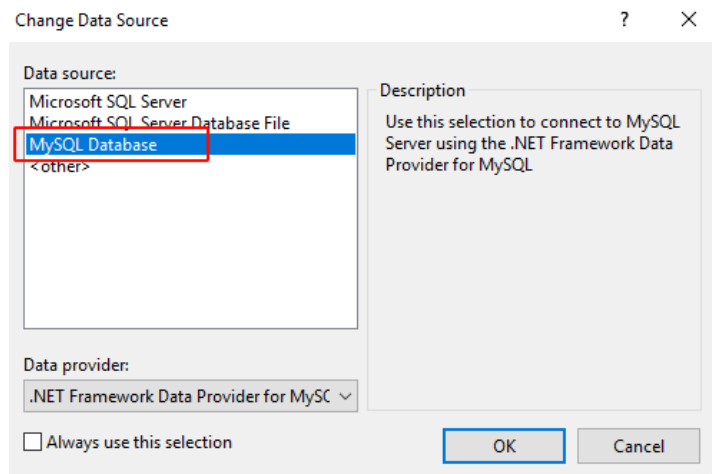
☐ Save my password

Database name:

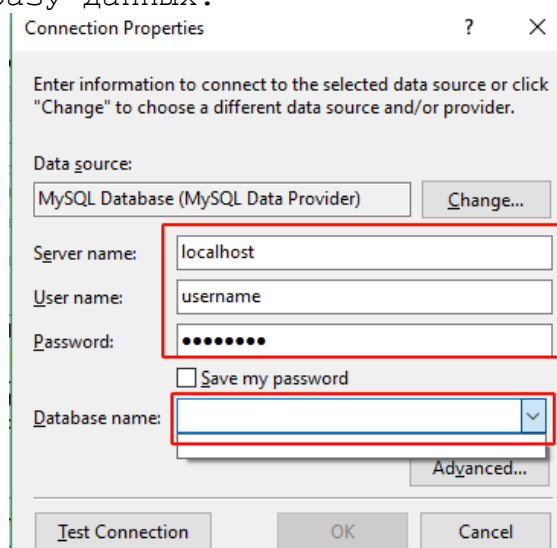
Advanced...

Test Connection OK Cancel

В следующем окне выберем «MySQL Database» и нажмем «OK».



После выбора введем данные сервера: IP адрес, логин и пароль. После выберем нужную базу данных.



На следующем окне, выберем созданное подключение и поставим «Yes, include the sensitive data in the connection string», а также укажем имя нашего объекта базы данных «Entitites».



## Choose Your Data Connection

Which data connection should your application use to connect to the database?

192.168.100.200(WSR\_EXAM) ▾

New Connection...

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

- ☐ No, exclude sensitive data from the connection string. I will set it in my application code.
- ☒ Yes, include the sensitive data in the connection string.

Connection string:

```
metadata=res://*/Model.csdl|res://*/Model.ssdl|
res://*/Model.msl;provider=MySQL.Data.MySqlClient;provider connection
string="server=192.168.100.200;user id=theworst;database=WSR_EXAM"
```

☒ Save connection settings in App.Config as:

WSR\_EXAMEntities

&lt; Previous

Next &gt;

Finish

Cancel

В следующем окне оставим по умолчанию:



## Choose Your Version

Which version of Entity Framework do you want to use?

- ☐ Entity Framework 6.x
- ☒ Entity Framework 5.0

**i** An Entity Framework database provider compatible with the latest version of Entity Framework could not be found for your data connection. If you have already installed a compatible provider, ensure you have rebuilt your project before performing this action. Otherwise, exit this wizard, install a compatible provider, and rebuild your project before performing this action.

[Learn more about this](#)

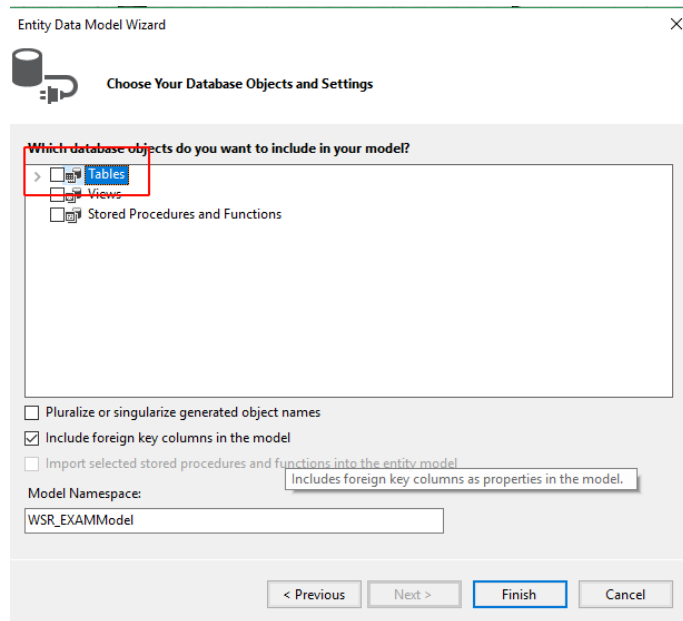
&lt; Previous

Next &gt;

Finish

Cancel

В этом окне выберем таблицы, которые хотим использовать в проекте:



## ПОЛУЧЕНИЕ ДАННЫХ ИЗ БАЗЫ ДАННЫХ

Для подключения к базе данных необходимо создать контекст.

```
using (var db = new Entities())  
{  
  
}
```

Теперь загрузим всю таблицу пользователей.

```
using (var db = new Entities())  
{  
    var users = db.User.AsNoTracking().ToList();  
}
```

Получим пользователей по определенному критерию

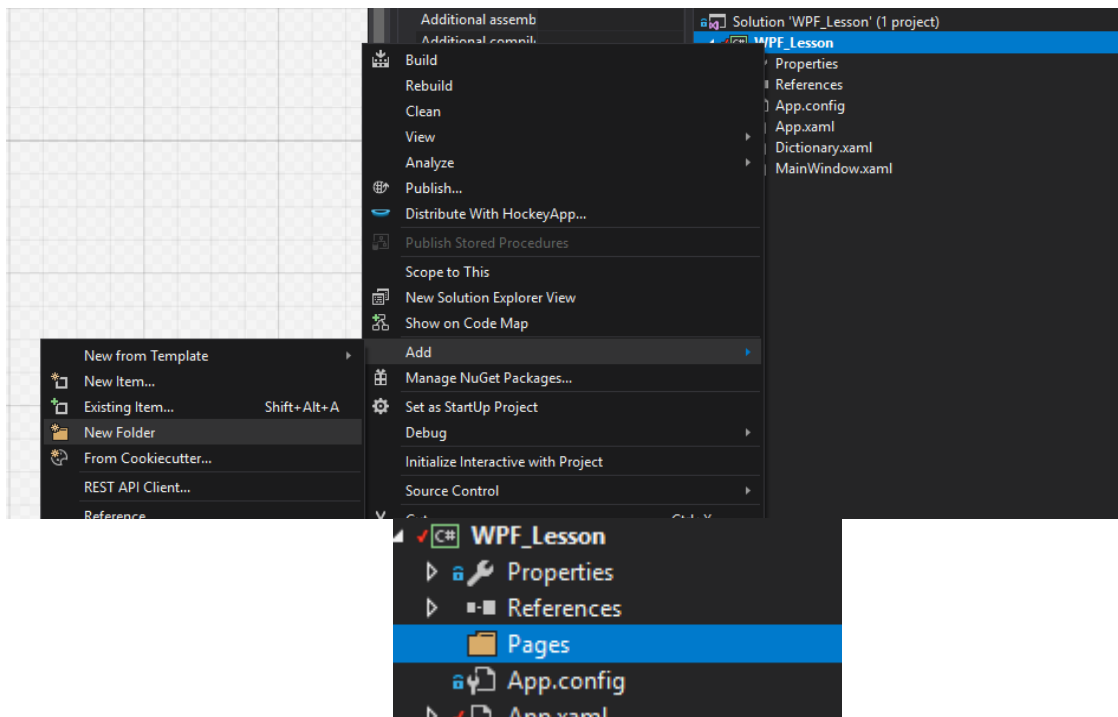
```
using (var db = new Entities())  
{  
    var users = db.User.AsNoTracking().Where(u => u.Login.StartsWith("max")).ToList();  
}
```

Получим пользователя по определенным критериям

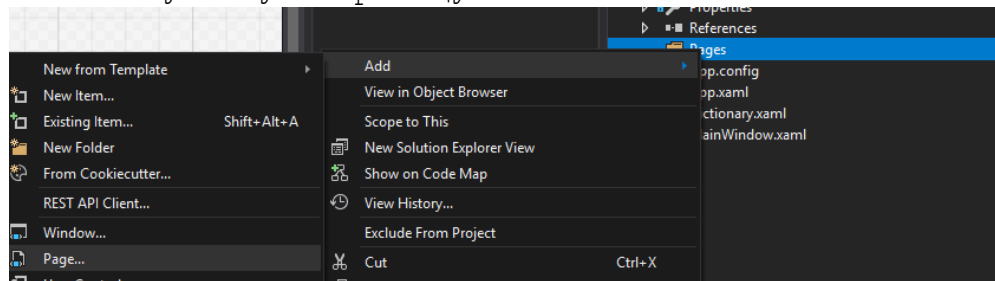
```
using (var db = new Entities())  
{  
    var user = db.User.AsNoTracking().FirstOrDefault(u => u.Login == "max" && u.Password == "test");  
}
```

## Создание формы авторизации

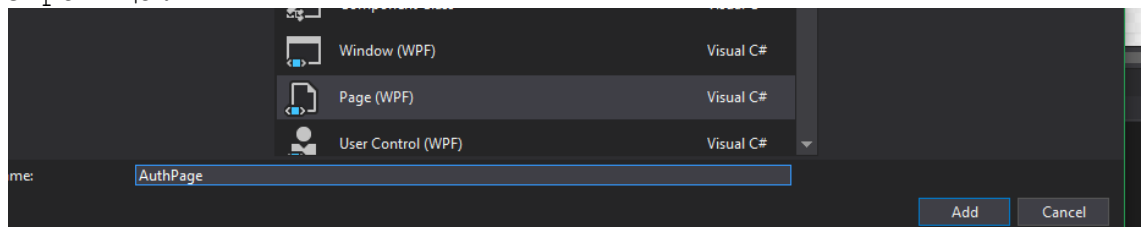
Добавим в проект новую папку и назовем ее «Pages», в этой папке будут находиться страницы (авторизация, регистрация и другие)

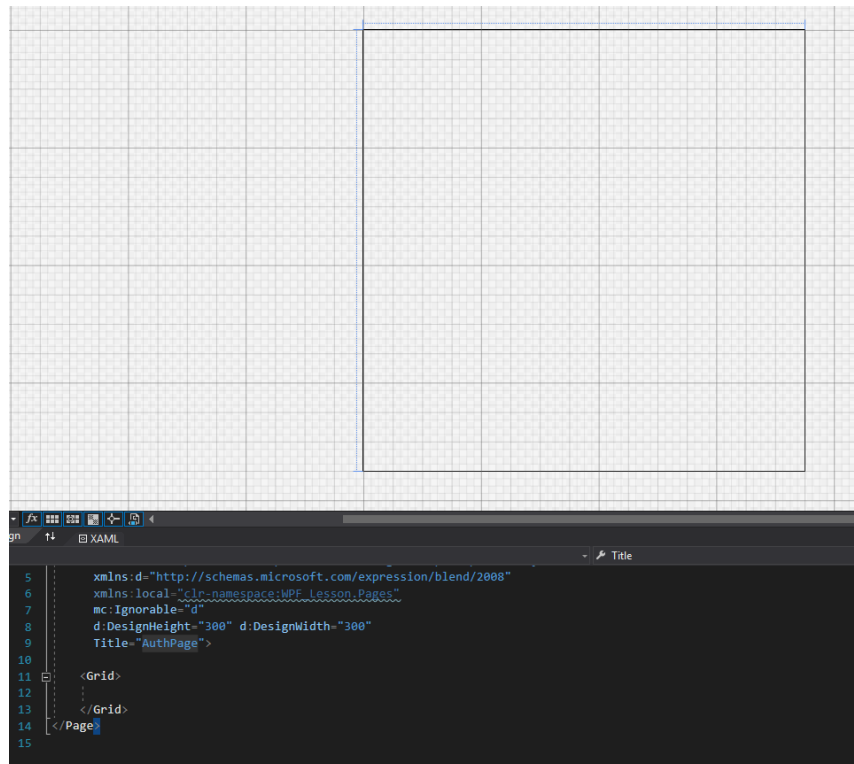


Добавим в папку новую страницу.

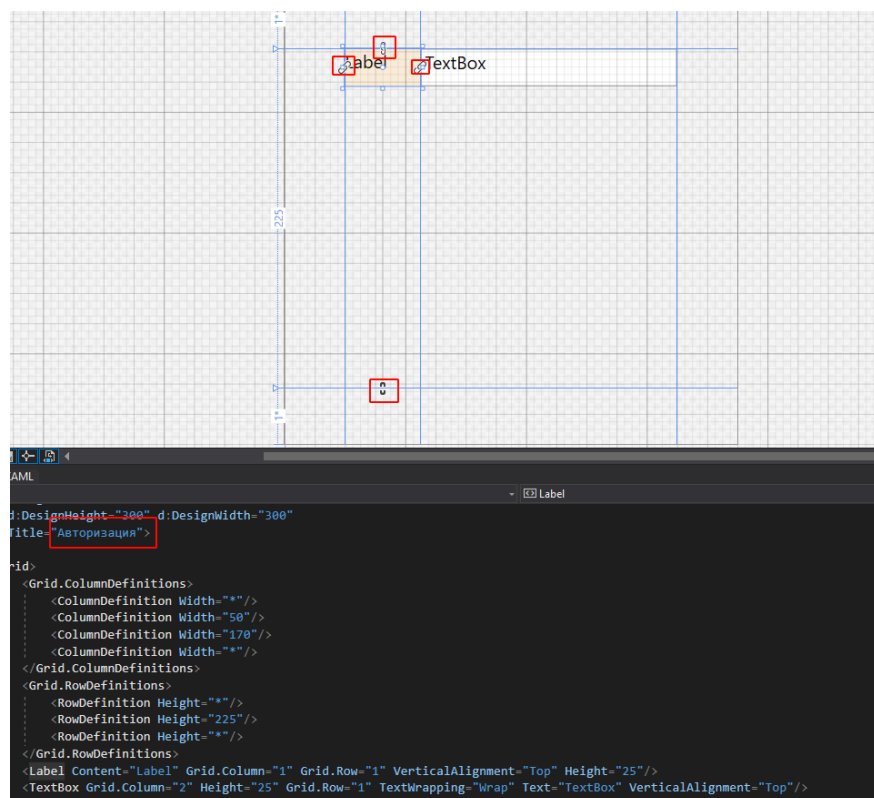


Назовем страницу AuthPage и создадим ее. После чего появится пустая страница.



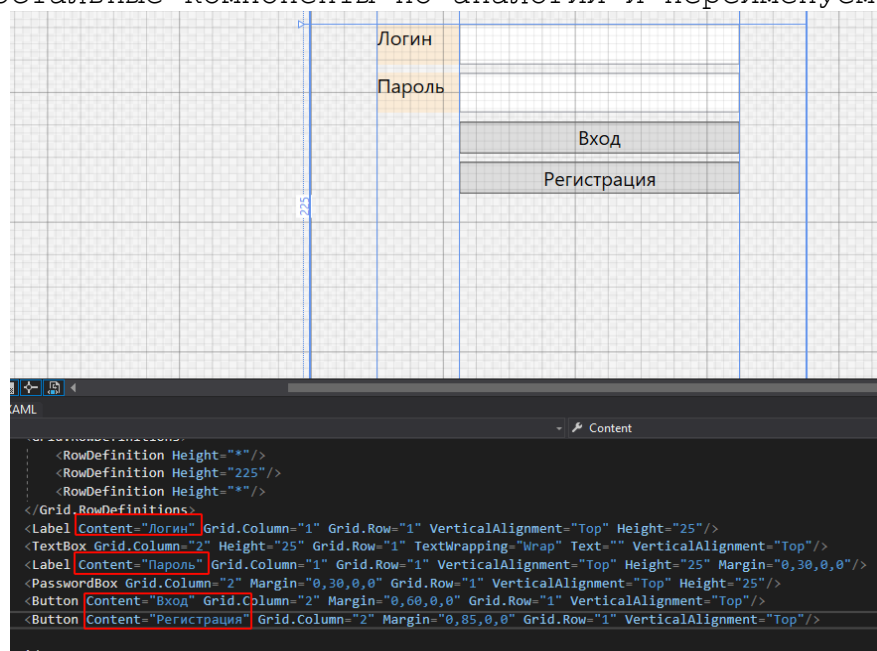


Добавим на форму компоненты «Label» и «TextBox», а затем отцентрируем их и добавим еще один столбец (для размещения лейблов). Также не забудем переименовать страницу, а затем закрепить лейбл и текстовый блок, чтобы они растягивались по ширине.

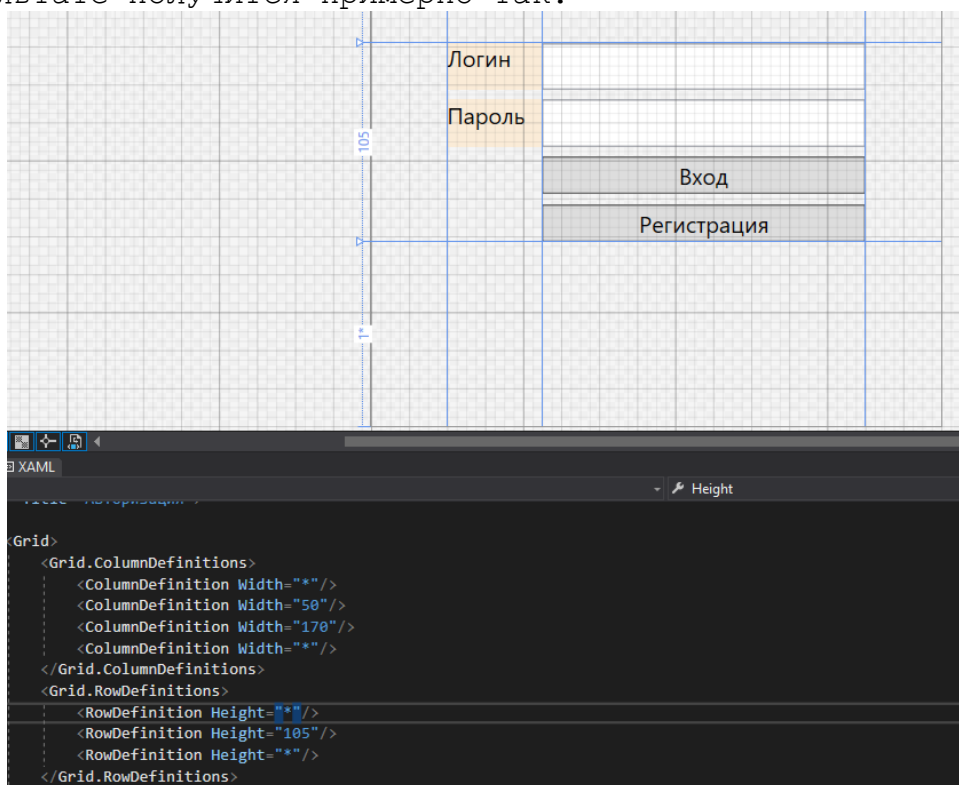




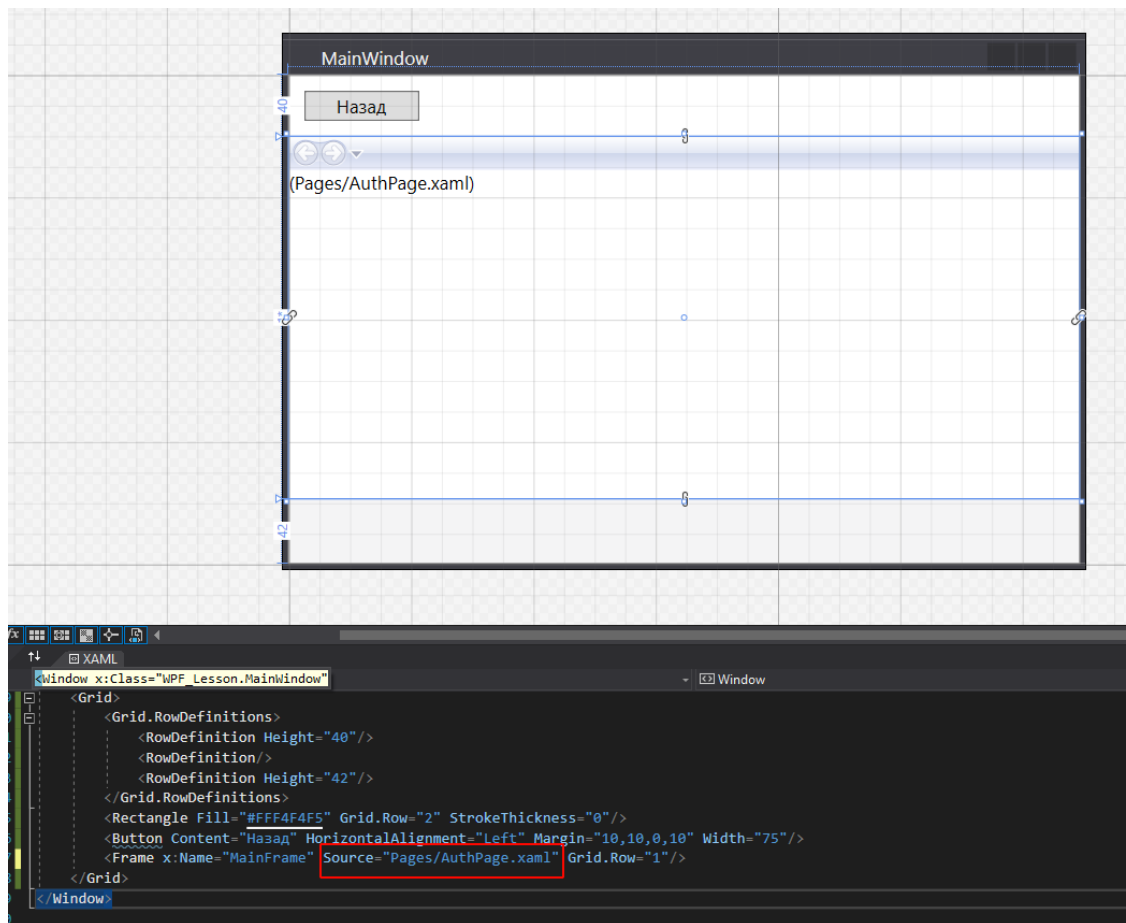
Добавим остальные компоненты по аналогии и переименуем их.



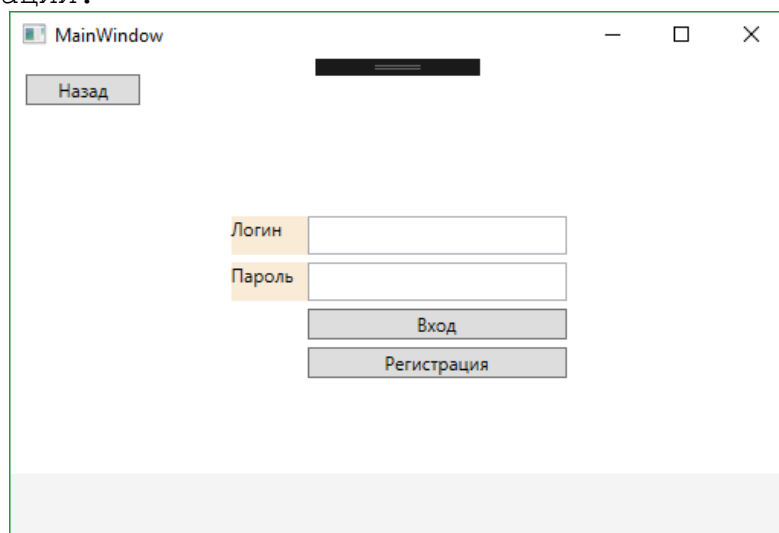
В результате получится примерно так.



Перейдем на главную форму и в компоненте «Frame» укажем в свойстве «Source» нашу страницу «AuthPage.xaml»



Запустим проект и увидим, что при запуске теперь отображается страница авторизации.



Теперь добавим функционал. Добавим обработчик события на кнопку вход.

```
<PasswordBox Content="Пароль" Grid.Column="1" Grid.Row="1" VerticalAlignment="Top" Height="20"/>
<Button Click="ButtonEnter_OnClick" Content="Вход" Grid.Column="2" Margin="0,60,0,0" Grid.Row="1" VerticalAlignment="Top" Height="20"/>
<Button Content="Регистрация" Grid.Column="2" Margin="0,85,0,0" Grid.Row="1" VerticalAlignment="Top" Height="20"/>

0 references | 0 changes | 0 authors, 0 changes
private void ButtonEnter_OnClick(object sender, RoutedEventArgs e)
{
    // ...
}
```

Добавим полям имена:

```
</Grid.RowDefinitions>
<Label Content="Логин" Grid.Column="1" Grid.Row="1" VerticalAlignment="Top" Margin="5" />
<TextBox x:Name="TextBoxLogin" Grid.Column="2" Height="30" Margin="5" />
<Label Content="Пароль" Grid.Column="1" Grid.Row="2" VerticalAlignment="Top" Margin="5" />
<PasswordBox x:Name="PasswordBox" Grid.Column="2" Margin="5" />
<Button Click="ButtonEnter_OnClick" Content="Вход" Grid.Column="2" Margin="5" />
<Button Content="Регистрация" Grid.Column="2" Margin="5" />
</Grid>
</Page>
```

Добавим в код базовую проверку

```
private void ButtonEnter_OnClick(object sender, RoutedEventArgs e)
{
    if (string.IsNullOrEmpty(TextBoxLogin.Text) || string.IsNullOrEmpty>PasswordBox.Password))
    {
        MessageBox.Show("Введите логин и пароль!");
        return;
    }
}
```

Добавим запрос к базе данных:

```
0 references | 0 changes | 0 authors, 0 changes
private void ButtonEnter_OnClick(object sender, RoutedEventArgs e)
{
    if (string.IsNullOrEmpty(TextBoxLogin.Text) || string.IsNullOrEmpty>PasswordBox.Password))
    {
        MessageBox.Show("Введите логин и пароль!");
        return;
    }

    using (var db = new Entities())
    {
        var user = db.User
            .AsNoTracking()
            .FirstOrDefault(u => u.Login == TextBoxLogin.Text && u.Password == PasswordBox.Password);

        if (user == null)
        {
            MessageBox.Show("Пользователь с такими данными не найден!");
            return;
        }
    }
}
```

И теперь добавим переходы в зависимости от роли на меню пользователя (для этого необходимо создать страницы меню для каждого типа пользователя, CustomerMenu или DirectorMenu и тд)

```

0 references to 0 changes to 0 authors, 0 changes
private void ButtonEnter_OnClick(object sender, RoutedEventArgs e)
{
    if (string.IsNullOrEmpty(TextBoxLogin.Text) || string.IsNullOrEmpty>PasswordBox.Password))
    {
        MessageBox.Show("Введите логин и пароль!");
        return;
    }

    using (var db = new Entities())
    {
        var user = db.User
            .AsNoTracking()
            .FirstOrDefault(u => u.Login == TextBoxLogin.Text && u.Password == PasswordBox.Password);

        if (user == null)
        {
            MessageBox.Show("Пользователь с такими данными не найден!");
            return;
        }

        MessageBox.Show("Пользователь успешно найден!");
        // Переход на меню пользователя в зависимости от роли

        switch (user.Role)
        {
            case "Заказчик":
                NavigationService?.Navigate(new Menu());
                break;
            case "Директор":
                NavigationService?.Navigate(new Menu());
                break;
        }
    }
}

```

## ПЕРЕХОД МЕЖДУ СТРАНИЦАМИ

Перейдем на базовую форму и создадим обработчик события Navigated у Frame.

```

<Rectangle Fill="#FFF4F4F5" Grid.Row="2" StrokeThickness="0"/>
<Button Content="Назад" HorizontalAlignment="Left" Margin="10,10,0,10" Width="75"/>
<Frame x:Name="MainFrame" Source="Pages/AuthPage.xaml" Grid.Row="1" Navigated="MainFrame_OnNavigated"/>
Grid>

```

В обработчик события напишем следующий код. Сначала мы проверяем что получили ли мы страницу на вход, затем устанавливаем заголовок формы в соответствии с шаблоном, после в зависимости от страницы отображаем или скрываем кнопку «Назад».

```

- references | 0 changes | 0 authors, 0 changes
private void MainFrame_OnNavigated(object sender, NavigationEventArgs e)
{
    if (!(e.Content is Page page)) return;
    this.Title = $"LESSON - {page.Title}";

    if (page is AuthPage)
    {
        ButtonBack.Visibility = Visibility.Hidden;
    }
    else
    {
        ButtonBack.Visibility = Visibility.Visible;
    }
}

```

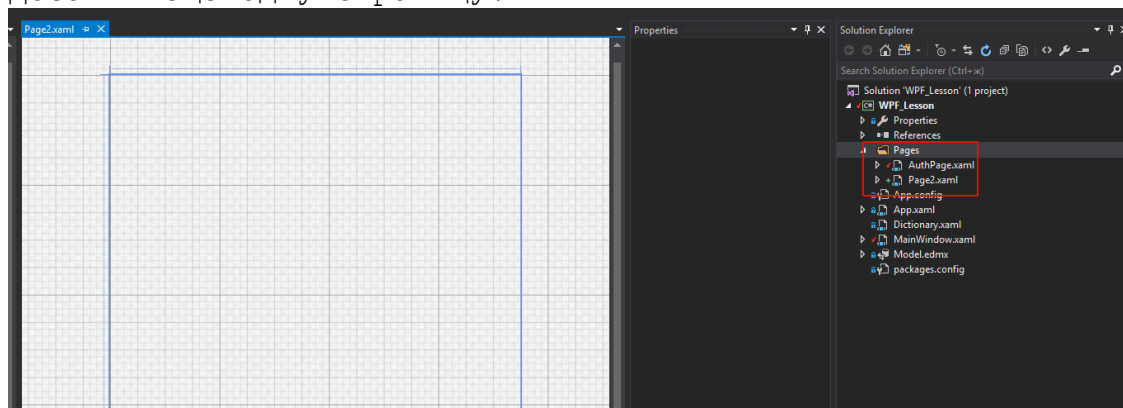
А в обработчик события нажатия кнопки «Назад». При нажатии на кнопку будет выполнен переход назад, если такой возможен.

```

- references | 0 changes | 0 authors, 0 changes
private void ButtonBack_OnClick(object sender, RoutedEventArgs e)
{
    if(MainFrame.CanGoBack) MainFrame.GoBack();
}

```

Добавим еще одну страницу.



Теперь выполним переход со страницы AuthPage на Page2. Перейдем на страницу AuthPage и добавим обработчик события на кнопку «Регистрация»:

```

<Button Content="Регистрация" Grid.Column="2" Margin="0,85,0,0" Grid.Row="1" VerticalAlignment="Top" Height="20" Click="ButtonRegistration_OnClick"/>

```

Напишем следующий код в обработчике:

```

- references | 0 changes | 0 authors, 0 changes
private void ButtonRegistration_OnClick(object sender, RoutedEventArgs e)
{
    NavigationService?.Navigate(new Page2());
}

```

После этого переход по кнопке будет осуществляться на страницу Page2, а по кнопке «Назад» обратно на AuthPage.

