**Министерство образования и науки Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,**

**МЕХАНИКИ И ОПТИКИ»**

**Факультет «ИКТ»**

**Дисциплина**

«Создание программного обеспечения инфокоммуникационных систем»

**Лабораторная работа 3**

**«**Реализация иерархии классов на языке С#**»**

**Выполнил:**

студент группы K3415

Дружинин К. А.

**Проверил**:

Осипов Н. А.

Санкт-Петербург

2021

**Практическое задание 8. Использование стилей и анимации Упражнение 1. Использование стилей для создания высококонтрастных элементов**

1. Создадим новый проект WPF Application с именем «Style01».
2. Определим стили для элементов Button, Label и TextBox в секции Window.Resources.
3. Изменим XAML разметку формы добавив элементы Button, Label и TextBox.
4. Построим и запустим приложение, чтобы протестировать его работу и проверить, что созданные стили применились к добавленным элементам.

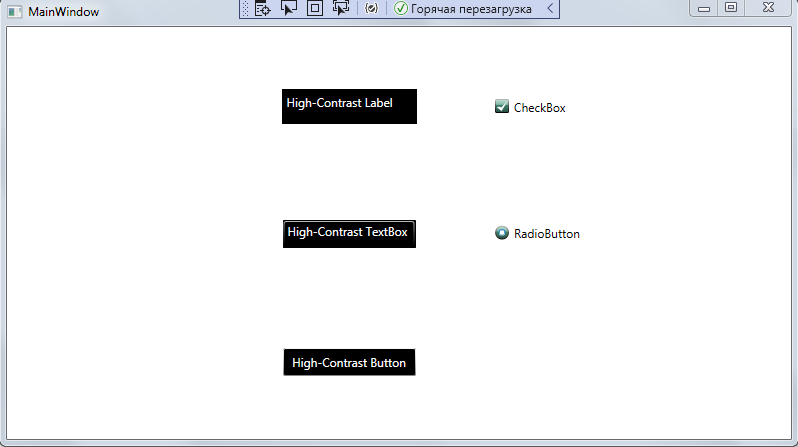


Рисунок 1 - тестирование приложения "Style01"

**Практическое задание 8. Использование стилей и анимации Упражнение 2. Использование триггеров для улучшения внешнего вида интерфейса**

1. Откроем проект предыдущего упражнения.
2. Добавим для стилей элементов **TextBox, Button** и **Label** секции Style.Triggers и триггеры для свойства **IsMouseOver**.

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

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

1. Построим и запустим приложение, чтобы протестировать его работу.

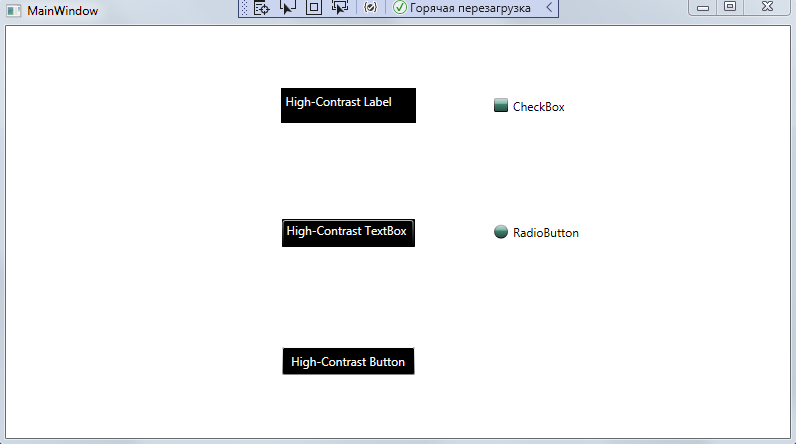


Рисунок 2 - тестирование приложения, проверка интерактивного отклика при наведении.

**Практическое задание 8. Использование стилей и анимации Упражнение 3. Анимация высококонтрастных стилей**

1. Откроем проект предыдущего упражнения.
2. Заменим секции Setter секциями **Trigger.EnterActions** и **Trigger.ExitActions**. И определим параметры анимации в соответствии с заданием.
3. Построим и запустим приложение, чтобы протестировать его работу

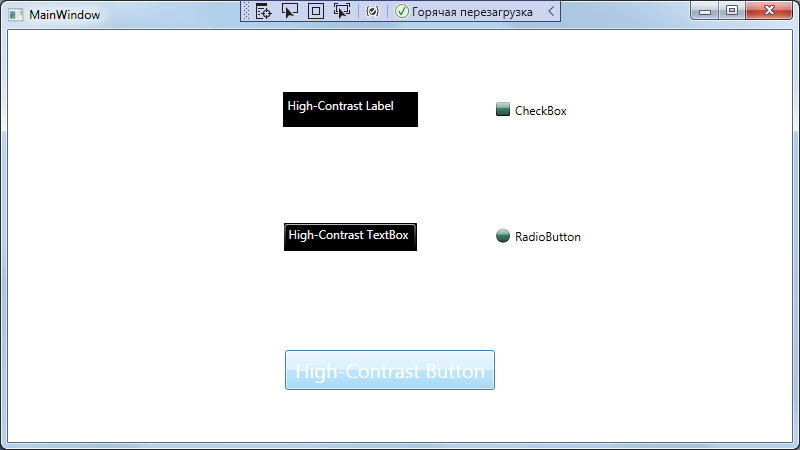


Рисунок 3 - тестирование приложения, проверка интерактивного отклика при наведении. Происходит плавное увеличение размеров элемента.

**Практическое задание 8. Использование стилей и анимации Упражнение 4. Добавление готовых тем в приложение**

1. Откроем проект предыдущего упражнения.
2. Скачаем и добавим в проект XAML файл темы, меняющий внешний вид базовых элементов.
3. В файле App.xaml, добавим ссылку на желаемую тему в секции Application.Resources.
4. Добавим несколько новых элементов (CheckBox, RadioButton), чтобы продемонстрировать применение добавленных стилей.
5. Построим и запустим приложение, чтобы протестировать его работу

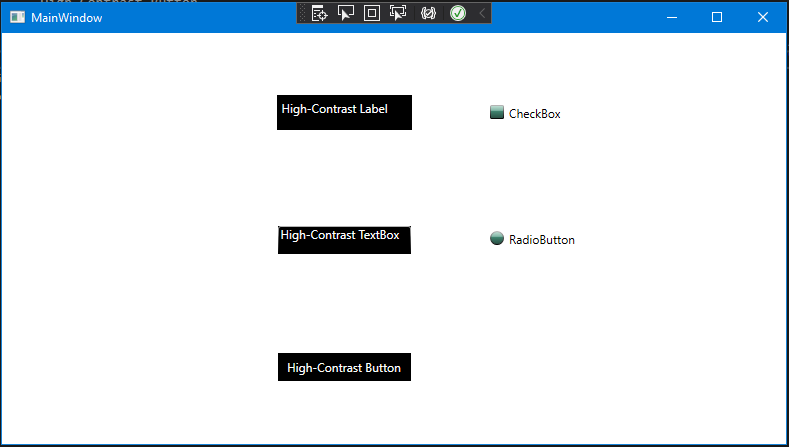


Рисунок 4 - тестирование приложения

<Window x:Class="Style01.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:Style01"

mc:Ignorable="d"

Title="MainWindow" Height="450" Width="800">

<Window.Resources>

<Style TargetType="TextBox">

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

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

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

<Setter Property="Width" Value="135" />

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

<Style.Triggers>

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

<Trigger.EnterActions>

<BeginStoryboard Name="Storyboard1">

<Storyboard Duration="0:0:1">

<DoubleAnimation Storyboard.TargetProperty="FontSize" To="20" />

<ThicknessAnimation Storyboard.TargetProperty="Margin" To="26,118,45,104" />

<DoubleAnimation Storyboard.TargetProperty="Width" To="210"/>

<DoubleAnimation Storyboard.TargetProperty="Height" To="40" />

</Storyboard>

</BeginStoryboard>

</Trigger.EnterActions>

<Trigger.ExitActions>

<StopStoryboard BeginStoryboardName="Storyboard1" />

</Trigger.ExitActions>

</Trigger>

</Style.Triggers>

</Style>

<Style TargetType="Label">

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

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

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

<Setter Property="Width" Value="135" />

<Setter Property="Height" Value="35" />

<Style.Triggers>

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

<Trigger.EnterActions>

<BeginStoryboard Name="Storyboard1">

<Storyboard Duration="0:0:1" >

<DoubleAnimation Storyboard.TargetProperty="FontSize" To="20" />

<ThicknessAnimation Storyboard.TargetProperty="Margin" To="26,62,46,-10" />

<DoubleAnimation Storyboard.TargetProperty="Width" To="210"/>

<DoubleAnimation Storyboard.TargetProperty="Height" To="40"/>

</Storyboard>

</BeginStoryboard>

</Trigger.EnterActions>

<Trigger.ExitActions>

<StopStoryboard BeginStoryboardName="Storyboard1" />

</Trigger.ExitActions>

</Trigger>

</Style.Triggers>

</Style>

<Style TargetType="Button">

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

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

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

<Setter Property="Width" Value="135" />

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

<Style.Triggers>

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

<Trigger.EnterActions>

<BeginStoryboard Name="Storyboard1">

<Storyboard Duration="0:0:1" >

<DoubleAnimation Storyboard.TargetProperty="FontSize" To="20" />

<ThicknessAnimation Storyboard.TargetProperty="Margin" To="26,0,46,52" />

<DoubleAnimation Storyboard.TargetProperty="Width" To="210"/>

<DoubleAnimation Storyboard.TargetProperty="Height" To="40"/>

</Storyboard>

</BeginStoryboard>

</Trigger.EnterActions>

<Trigger.ExitActions>

<StopStoryboard BeginStoryboardName="Storyboard1" />

</Trigger.ExitActions>

</Trigger>

</Style.Triggers>

</Style>

</Window.Resources>

<Grid>

<Label Margin="26,62,126,0" VerticalAlignment="Top">

High-Contrast Label

</Label>

<TextBox Margin="26,117,126,115">

High-Contrast TextBox

</TextBox>

<Button Margin="26,0,126,62" VerticalAlignment="Bottom">

High-Contrast Button

</Button>

<CheckBox Content="CheckBox" HorizontalAlignment="Left" Margin="487,71,0,0" VerticalAlignment="Top"/>

<RadioButton Content="RadioButton" HorizontalAlignment="Left" Margin="487,0,0,0" VerticalAlignment="Center"/>

</Grid>

</Window>

Текст 1 – MainWindow.xaml

**Практическое задание 9. Использование стилей и анимации Упражнение 1. Создание привязки данных с помощью конструктора WPF**

1. Создадим новый проект WPF Application с именем «DataBindingDemo».
2. Добавим в проект новый класс с именем **Student**.

namespace DataBindingDemo

{

public class Student

{

public string StudentName { get; set; }

public bool IsEnrolled { get; set; }

}

}

Текст 2 – Вносимые изменения в код

1. В классе Student определим два автоматических свойства **StudentName** и **IsEnrolled**.
2. В пространстве имен **DataBindingDemo** определим новый класс **StudentList** производный от класса ObservableCollection<Student>. Объект данного класса будет служить источником данных для элемента интерфейса **ListBox**.

namespace DataBindingDemo

{

public class StudentList : ObservableCollection<Student>

{

}

public class Student

{

public string StudentName { get; set; }

public bool IsEnrolled { get; set; }

}

}

Текст 3 – Вносимые изменения в код

1. В пространстве имен **DataBindingDemo** определим новый класс **BoolToBrushConverter** и отредактируем в соответствии с заданием лабораторной работы, добавив методы Convert, ConvertBack и атрибут ValueConversion.

namespace DataBindingDemo

{

[ValueConversion(typeof(bool), typeof(Brush))]

public class BoolToBrushConverter : IValueConverter

{

public object Convert(

object value,

Type targetType,

object parameter,

CultureInfo culture

)

{

Brush b = null;

if (value != null && value.GetType() == typeof(bool))

{

b = (bool)value ? Brushes.Green : Brushes.Red;

}

return b;

}

public object ConvertBack(

object value,

Type targetType,

object parameter,

CultureInfo culture

)

{

return null;

}

}

Текст 4 – Вносимые изменения в код

1. Отредактируем файл MainWindow.xaml, определив набор объектов класса **Student** в коллекции **StudentList**, и шаблон для отображения элементов списка **ListBox.**

<Window x:Class="DataBindingDemo.MainWindow"

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

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

xmlns:local="clr-namespace:DataBindingDemo"

Title="Databinding Demo" Height="300" Width="300">

<Window.Resources>

<local:StudentList x:Key="studentCollection" >

<local:Student StudentName="Syed Abbas" IsEnrolled="false" />

<local:Student StudentName="Lori Kane" IsEnrolled="true" />

<local:Student StudentName="Steve Masters" IsEnrolled="false" />

<local:Student StudentName="Tai Yee" IsEnrolled="true" />

<local:Student StudentName="Brenda Diaz" IsEnrolled="true" />

</local:StudentList>

<local:BoolToBrushConverter x:Key="boolToBrushConverter" />

<DataTemplate x:Key="listBoxTemplate">

<StackPanel Orientation="Horizontal" >

<Rectangle Fill="{Binding Path=IsEnrolled, Converter={StaticResource boolToBrushConverter}}" Height="10" Width="10" Margin="0,0,5,0" />

<TextBlock Text="{Binding Path=StudentName}" />

</StackPanel>

</DataTemplate>

</Window.Resources>

<Grid>

<ListBox ItemsSource="{Binding Mode=OneWay, Source={StaticResource studentCollection}}" IsSynchronizedWithCurrentItem="True" ItemTemplate="{Binding Mode=OneWay, Source={StaticResource listBoxTemplate}}"/>

</Grid>

</Window>

Текст 5 – Вносимые изменения в код

1. С помощью конструктора WPF настроим привязку данных для элемента **ListBox**.
2. Выберем шаблон DataTemplate для преобразования данных и их более наглядного отображения.
3. Построим и запустим приложение, чтобы протестировать его работу

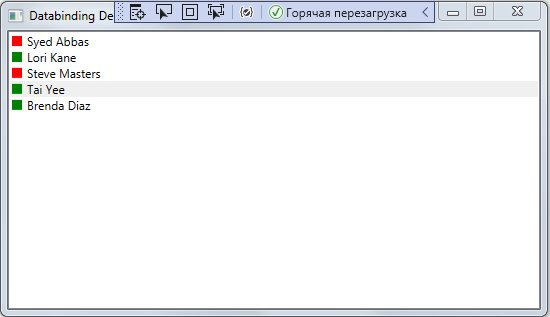


Рисунок 5 - тестирование приложения

**Практическое задание 10. Использование параметров приложения**

**Упражнение 1. Работа с параметрами**

1. Создадим новый проект WPF Application с именем «Param».
2. Добавим два параметра в приложение **ApplicationName** и **BackgroundColor.**

namespace Param

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

this.Title = Properties.Settings.Default.ApplicationName;

this.Background = new System.Windows.Media.SolidColorBrush(Properties.Settings.Default.BackgroundColor);

}

System.Windows.Media.SolidColorBrush(Properties.Settings.Default.BackgroundColor);

Properties.Settings.Default.Save();

}

}

}

Текст 6 – Вносимые изменения в код

1. Отредактируем XAML представление формы, добавив список **ListBox** с четырьмя элементами, описывающими доступные цвета, и кнопку смены цвета фона.
2. Добавим обработчик события Click для созданной кнопки, производящий смену цвета фона окна приложения.

private void button1\_Click(object sender, RoutedEventArgs e)

{

if (!(listBox1.SelectedItem == null))

{

String astring = ((ListBoxItem)listBox1.SelectedItem).Content.ToString();

switch (astring)

{

case "Red":

Properties.Settings.Default.BackgroundColor = Colors.Red;

break;

case "Blue":

Properties.Settings.Default.BackgroundColor = Colors.Blue;

break;

case "Green":

Properties.Settings.Default.BackgroundColor = Colors.Green;

break;

case "Tomato":

Properties.Settings.Default.BackgroundColor = Colors.Tomato;

break;

}

this.Background = new System.Windows.Media.SolidColorBrush(Properties.Settings.Default.BackgroundColor);

Текст 7 – Вносимые изменения в код

1. Добавим присвоение свойствам **Title** и **Background** параметров, определенных ранее, в конструкторе класса MainWindow.

public MainWindow()

{

InitializeComponent();

this.Title = Properties.Settings.Default.ApplicationName;

this.Background = new System.Windows.Media.SolidColorBrush(Properties.Settings.Default.BackgroundColor);

Текст 8 – Вносимые изменения в код

1. Построим и запустим приложение, чтобы протестировать его работу

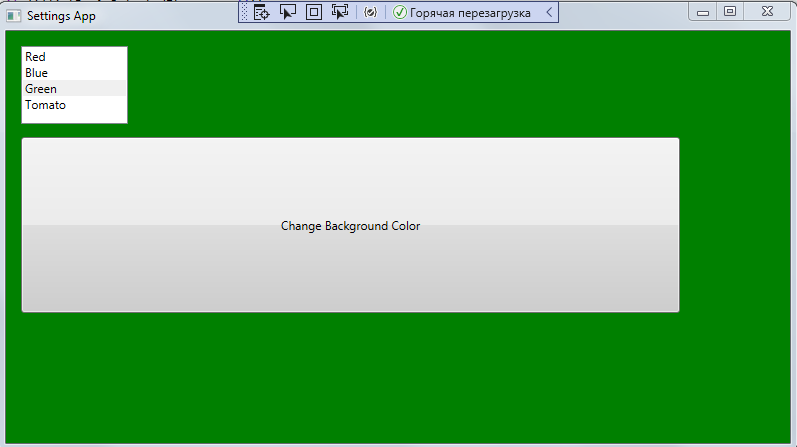


Рисунок 6 - тестирование приложения.

**Лабораторная работа 1 (Задание). Разработка приложения WPF**

**Упражнение 1. Работа с параметрами**

1. Создадим новый проект WPF Application с именем «ExpenseIt».
2. Заменим элемент Window на NavigationWindow.
3. Отредактируем и определим свойства элемента NavigationWindow в соответствии с заданием.
4. Добавим в проект новые страницы с именами **ExpenseItHome** и **ExpenseReportPage**, настроим их параметры.
5. Добавим на страницу тестовое изображение и построим проект.

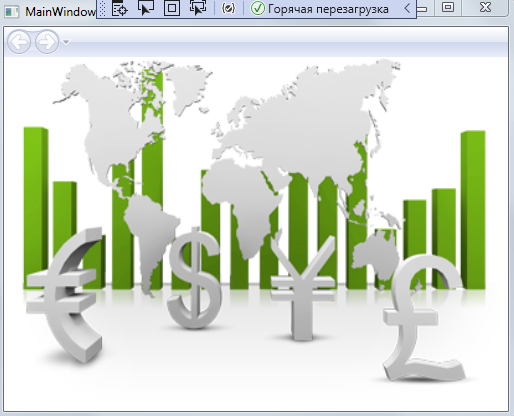


Рисунок 7 - промежуточное тестирование приложения.

**Упражнение 2. Реализация пользовательского интерфейса приложения**

1. Отредактируем файл ExpenseItHome.xaml в соответствии с заданием, определив сетку компонентов и элементы пользовательского интерфейса.
2. Зададим добавленное ранее изображение в качестве фона для класса Grid.
3. Добавим обработчик события Click для кнопки View, открывающий новое окно просмотра информации о выбранном человеке.
4. Построим и проверим наше приложение.

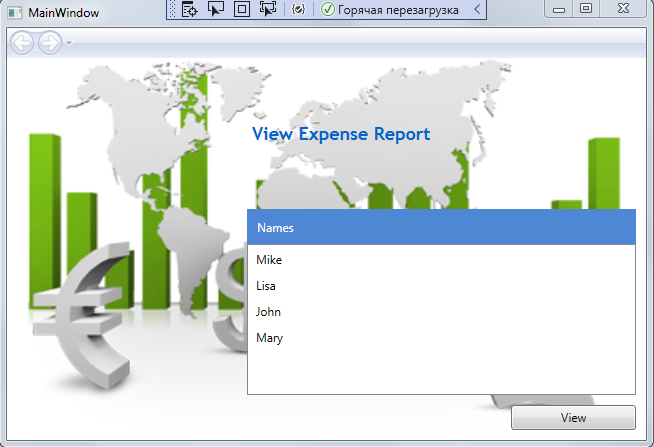


Рисунок 8 - промежуточное тестирование приложения

**Упражнение 3. Создание пользовательского интерфейса отдельной страницы**

1. Отредактируем файл ExpenseReportPage.xaml в соответствии с заданием, определив сетку компонентов и элементы пользовательского интерфейса. Данная страница отвечает за отображения подробной информации о человеке, выбранном на главной странице.
2. Построим и проверим наше приложение.

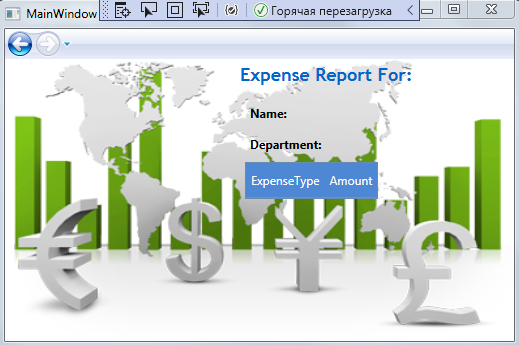


Рисунок 9 - промежуточное тестирование приложения

Как и ожидалось, после нажатия кнопки View происходит переход на пока не наполненную страницу информации о выбранном человеке.

**Упражнение 4. Использование стилей**

1. Определим стили для нашего приложения в секции Application.Resources файла App.xaml.

<Application.Resources>

<!-- Header text style -->

<Style x:Key="headerTextStyle">

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

<Setter Property="Label.FontFamily" Value="Trebuchet MS"></Setter>

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

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

<Setter Property="Label.Foreground" Value="#0066cc"></Setter>

</Style>

<!-- Label style -->

<Style x:Key="labelStyle" TargetType="{x:Type Label}">

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

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

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

<Setter Property="Margin" Value="0,0,0,5" />

</Style>

<!-- DataGrid header style -->

<Style x:Key="columnHeaderStyle" TargetType="{x:Type DataGridColumnHeader}">

<Setter Property="Height" Value="35" />

<Setter Property="Padding" Value="5" />

<Setter Property="Background" Value="#4E87D4" />

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

</Style>

<!-- List header style -->

<Style x:Key="listHeaderStyle" TargetType="{x:Type Border}">

<Setter Property="Height" Value="35" />

<Setter Property="Padding" Value="5" />

<Setter Property="Background" Value="#4E87D4" />

</Style>

<!-- List header text style -->

<Style x:Key="listHeaderTextStyle" TargetType="{x:Type Label}">

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

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

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

</Style>

<!-- Button style -->

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

<Setter Property="Width" Value="125" />

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

<Setter Property="Margin" Value="0,10,0,0" />

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

</Style>

</Application.Resources>

Текст 9 – Вносимые изменения в код

1. Применим описанные выше стили к элементам страниц ExpenseReportPage и ExpenseItHome.

<Page x:Class="ExpenseIt.ExpenseReportPage"

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:ExpenseIt"

mc:Ignorable="d"

d:DesignHeight="350" d:DesignWidth="525"

Title="ExpenseIt - View Expense">

<Grid>

<!--Templates to display expense report data-->

<Grid.Resources>

<!-- Reason item template -->

<DataTemplate x:Key="typeItemTemplate">

<Label Content="{Binding XPath=@ExpenseType}"/>

</DataTemplate>

<!-- Amount item template -->

<DataTemplate x:Key="amountItemTemplate">

<Label Content="{Binding XPath=@ExpenseAmount}"/>

</DataTemplate>

</Grid.Resources>

<Grid.Background>

<ImageBrush ImageSource="pngkey.com-photoshop-logo-png-177601.png" />

</Grid.Background>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="230" />

<ColumnDefinition />

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition />

</Grid.RowDefinitions>

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

Expense Report For:

</Label>

<Grid Margin="10" Grid.Column="1" Grid.Row="1">

<Grid.ColumnDefinitions>

<ColumnDefinition />

<ColumnDefinition />

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition />

</Grid.RowDefinitions>

<!-- Name -->

<StackPanel Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="0" Orientation="Horizontal">

<Label Style="{StaticResource labelStyle}">Name:</Label>

<Label Style="{StaticResource labelStyle}" Content="{Binding XPath=@Name}"></Label>

</StackPanel>

<!-- Department -->

<StackPanel Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="1" Orientation="Horizontal">

<Label Style="{StaticResource labelStyle}">Department:</Label>

<Label Style="{StaticResource labelStyle}" Content="{Binding XPath=@Department}"></Label>

</StackPanel>

<Grid Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="2" VerticalAlignment="Top" HorizontalAlignment="Left">

<!-- Expense type and Amount table -->

<DataGrid ItemsSource="{Binding XPath=Expense}" ColumnHeaderStyle="{StaticResource columnHeaderStyle}" AutoGenerateColumns="False" RowHeaderWidth="0" >

<DataGrid.Columns>

<DataGridTextColumn Header="ExpenseType" Binding="{Binding XPath=@ExpenseType}" />

<DataGridTextColumn Header="Amount" Binding="{Binding XPath=@ExpenseAmount}" />

</DataGrid.Columns>

</DataGrid>

</Grid>

</Grid>

</Grid>

</Page>

Текст 10 – Вносимые изменения в код

1. Построим и проверим наше приложение.

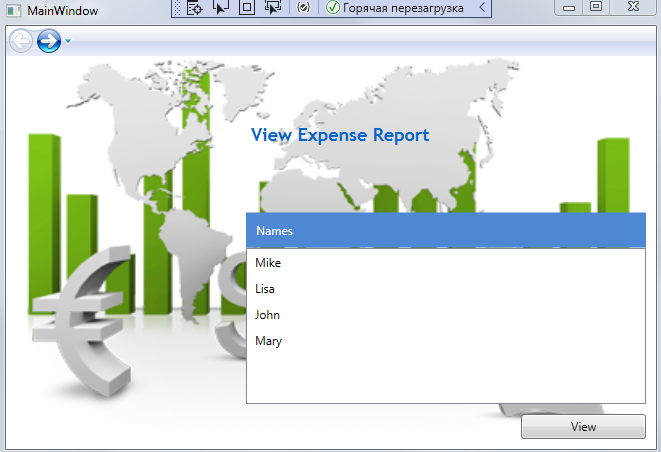


Рисунок 10 - промежуточное тестирование приложения

**Упражнение 5. Привязка данных к элементу управления**

1. Добавим в проект папку с именем Data и файлом Data.xml, который будет выступать в качестве источника данных для нашего приложения.
2. Отредактируем файл ExpenseItHome.xaml, определив раннее созданный файл в качестве ресурса и задав шаблон для отображения элементов компонента ListBox.
3. Отредактируем файл ExpenseReportPage.xaml.cs, добавив в конструктор параметр data – информации о выбранном человеке.

public ExpenseReportPage(object data) : this()

Текст 11 – Вносимые изменения в код

1. Переопределим создание новой страницы в файле ExpenseItHome.xaml.cs.
2. Отредактируем код ExpenseReportPage.xaml для отображения информации о выбранном человеке.

public ExpenseReportPage(object data) : this()

{

// Bind to expense report data.

this.DataContext = data;

Текст 12 – Вносимые изменения в код

1. Построим и проверим наше приложение.

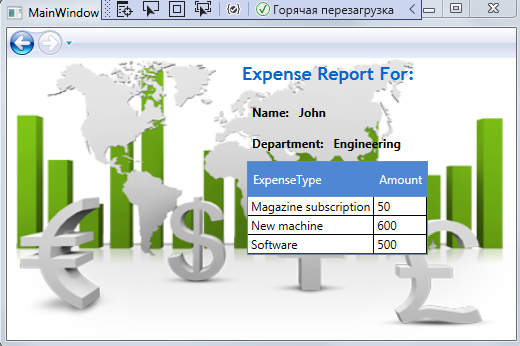


Рисунок 1 - тестирование приложения. Страница 2.

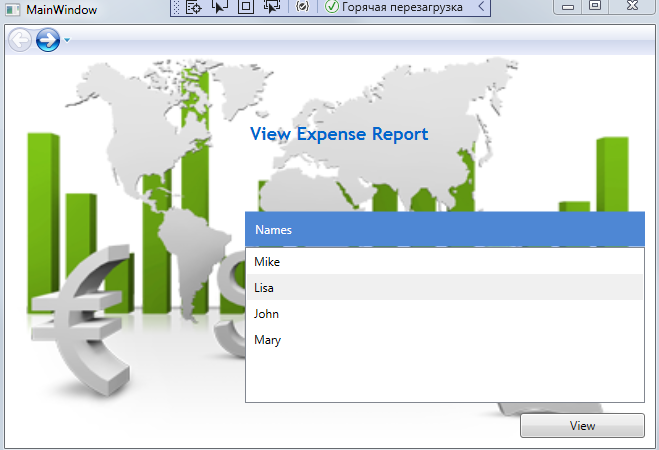


Рисунок 12 - тестирование приложения. Страница 1.

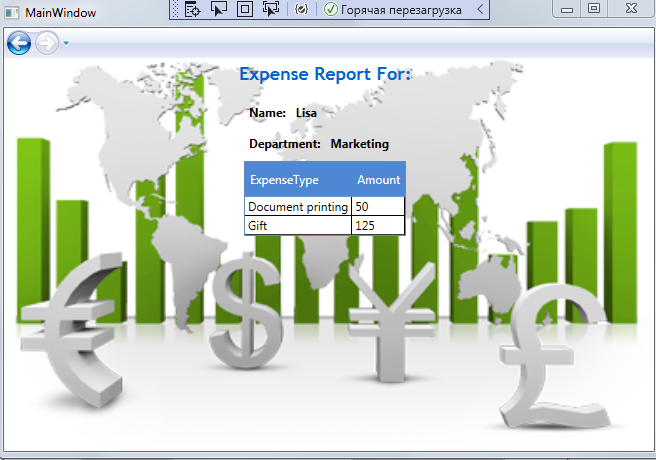


Рисунок 13 - тестирование приложения. Страница 2.

**Вывод**

В данной лабораторной работе были рассмотрены:

• Использование стилей для изменения внешнего вида элементов управления.

• Использование триггеров для реакции на поведение пользователя (наведение курсора мыши на элементы управления)

• Анимация стилей

• Использование сторонних тем для изменения внешнего вида элементов управления.

• Использование привязки данных к компонентам для их отображения.

• Использование преобразователей значений для редактирования отображения данных.

• Рассмотрена работа с параметрами приложения для сохранения состояния между сессиями.

• Рассмотрены механизмы создания и переходов между страницами в приложении.

• Рассмотрены механизмы передачи данных между страницами через конструктора страницы.

На практике были освоены следующие принципы окон WPF:

• Окно WPF может содержать только один элемент.

• Чтобы разместить более одного элемента и создать практичный пользовательский интерфейс, нужно поместить в окно контейнер и добавлять элементы в этот контейнер.

• В WPF компоновка формируется с использованием разнообразных контейнеров.

• Каждый контейнер обладает собственной логикой компоновки

• WPF использует потоковую модель расположения (flow-based layout) для позиционирования элементов управления

• Потоковая модель расположения позволяет контролам WPF быть адаптированным к размеру окна, когда они начинают визуализацию

В лабораторной были изучены принципы добавления стилей и анимаций к элементам. По итогу выполнения, можно отметить, что WPF имеет богатые возможности, а использование сторонних библиотек и стилевых компонентов только расширяют их.

Многие компоненты WPF поддерживают привязку данных, что значительно упрощает наполнение и отображение информации.

Было реализовано сложное приложение с древовидным интерфейсом. Благодаря WPF реализация была значительно ускорена с помощью переиспользуемых компонентов с привязкой данных, что позволило не создавать отдельную страничку под каждый случай, а динамически обрабатывать имеющиеся данные