Упражнение №2 по ПС

Въведение в WPF. Разлики между WPF и WinForms. Създаване на приложение с WPF.

Целта на това упражнение е студентите да се запознаят технологията WPF за създаване на Windows приложения, да установят приликите и разликите с по-старата технология Windows Forms, да видят предимствата пред нея, разработвайки първото си приложение с разглежданата технология.

Какво е WPF?

Windows Presentation Foundation (WPF) е система за разработката на клиентски Windows приложения, позволяваща удивително потребителско изживяване. С WPF могат да се създават широк набор от самостоятелни или базирани на браузъра приложения.

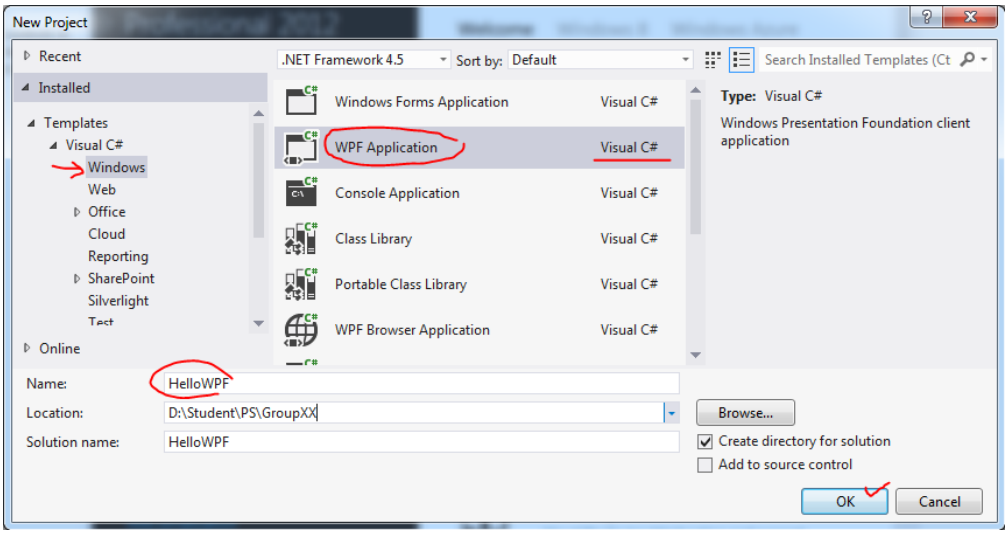
Ядрото на WPF е независимо от резолюцията и векторно базирано, така че да се възползва от модерния графичен хардуер. WPF дава достъп до обширен набор от функции за разработка на приложения, включващи Extensible Application Markup Language (XAML), контроли (controls), връзка с данните (data binding), 2D и 3D графика, анимация, стилове, шаблони, документи, медия, текст и текстооформление. WPF е включен в .NET Framework и е възможно в него да се вграждат и други елементи от библиотеките с класове на .NET.

Програмирането с WPF включва разгледаните в предното упражнение дейности като: инстанциране на класове, настройка на свойства, извикване на методи и прихващане на събития, използвайки програмен език C# или Visual Basic.

WPF включва допълнителни програмни конструкции, които допълват свойствата и събитията: ***dependency properties*** и ***routed events***. Тях ще разгледаме по-подробно нататък в упражнението.

Hello, WPF! Създаване на нов проект.

За да създадете нов проект отворете Visual Studio и от менюто “File” изберете “New” → “Project…”. Отваря се следния диалогов прозорец:



От различните видове Windows приложения изберете **WPF Application**. Кръстете го "HelloWPF" и след това натиснете **Ok** бутона.

Вашият току що създаден проект се състои от няколко файла, като първо ще се фокусираме върху един от тях: *MainWindox.xaml*. Това е началния прозорец на приложението, този, който се показва при стартиране му, освен ако изрично не зададете нещо друго. XAML кодът, който ще намерите в него, трябва да изглежда така:


```
<Window x:Class="HelloWPF.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
    </Grid>
</Window>
```

Това е основния XAML, който Visual Studio създава за вашия прозорец. Повече информация за отделните му части можете да прочетете тук XAML (http://www.wpf-tutorial.com/xaml/basic-xaml/) и тук "The Window" (http://www.wpf-tutorial.com/wpf-application/the-window/). Можете да стартирате приложението (натиснете **F5**) и да видите празния прозорец на вашето приложение.

За да добавим класическото съобщение „**Hello, WPF!**“ към нашия прозорец, ни е необходима контрола от типа **TextBlock.** Добавяме контролата към Grid панела и кодът на приложението ви трябва да има следния вид:

```
<Window x:Class="HelloWPF.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
  <Grid>
     <TextBlock HorizontalAlignment="Center" VerticalAlignment="Center" FontSize="72">
           Hello, WPF!
     </TextBlock>
  </Grid>
</Window>
```

Стартирайте приложението (от менюто изберете Debug -> Start debugging или натиснете **F5**). Ако няма грешки, то честито първо приложение с WPF:

Може да забележите, че в елемента TextBlock използвахме различни атрибути (свойства), за да центрираме текста в прозореца (HorizontalAlignment и VerticalAlignment) и FontSize, за да уголемим текста. Пълния списък със свойства за всеки елемент можем да видим в Properties прозореца.

WPF vs. WinForms

В предното упражнение си говорихме за WinForms и тук ще се опитаме да сравним двете технологии, защото те имат еднаква цел, но постигана с различни средства.

Основната най-важна разлика между WinForms и WPFе факта, че докато WinForms надгражда стандартните Windows контроли (например TextBox), WPF е изграден от нулата и не разчита на Windows контролите в повечето ситуации. Това е много осезаемо ако ви се наложи да работите с framework основаващ се на Win32/WinAPI.

Добър пример за това е бутон, представляващ изображение и текст върху него. Това не е стандартен Windows контрол и WinForms не поддържа такава възможност. За да го реализирате, трябва да изчертаете изображението, да имплементирате свой собствен бутон, който да поддържа изображение или да използвате чужди контроли. В WPF бутонът може да съдържа каквото и да е, защото по принцип представлява някакво съдържание с граница и различни състояния (например untouched, hovered, pressed). За WPF контролите и в частност бутона, не е описано как да изглеждат, което означава, че те могат да съдържат в себе си определен тип други контроли. Искаме бутона да има изображение и текст, просто поставяме Image и TextBlock в бутона и сме готови. При WinForms просто няма такава свобода и гъвкавост.

Това е само една разлика, но докато работите с WPF, ще осъзнаете че всъщност тя е основополагаща за много други разлики – WPF прави нещата по собствен начин, за добро или лошо. Вече липсват ограниченията от специфицираните в Windows елементи. За да се получи тази гъвкавост обаче се налага писането на малко повече код, отколкото когато просто се използват стандартните за Windows елементи.

Предлагаме ви един доста субективен списък от предимства на двете технологии, който да ви даде по-добра идея с какво се захващаме.

Предимства на WPF

- По-нова технология и следователно повече в крак с текущите стандарти и тенденции;
- Microsoft я използва за много нови приложения, включително и Visual Studio;
- По-гъвкава е и можете да правите повече неща, без да се налага да пишете или купувате нови контроли;
- XAML прави лесно създаването и модификацията на вашия GUI, като позволява работата да бъде разделена между дизайнер (XAML) и програмист (C#, VB.NET);
- Използва се т.нар. Databinding, който позволява да има по-чисто разделяне между данните и изгледа;
- Използва хардуерно ускорение при изчертаване на GUI за по-добра производителност;
- Позволява да направите потребителски интерфейс както за Windows приложение, така и за уеб приложение (Silverlight/XBAP).

Предимства на WinForms

- И понеже ни трябва някакво предимство, да кажем че е по-стара технология и следователно по-тествана и изчистена от грешки ;-)
- Стандартните контроли и елементи за Windows се реализират без почти никакво писане на код.

Някои неща, които трябва да се знаят…

Преди да се захванем с правенето на програми с WPF има някои неща, които трябва да се разберат...

Какво се има в предвид с Dispatcher & Thread Affinity?

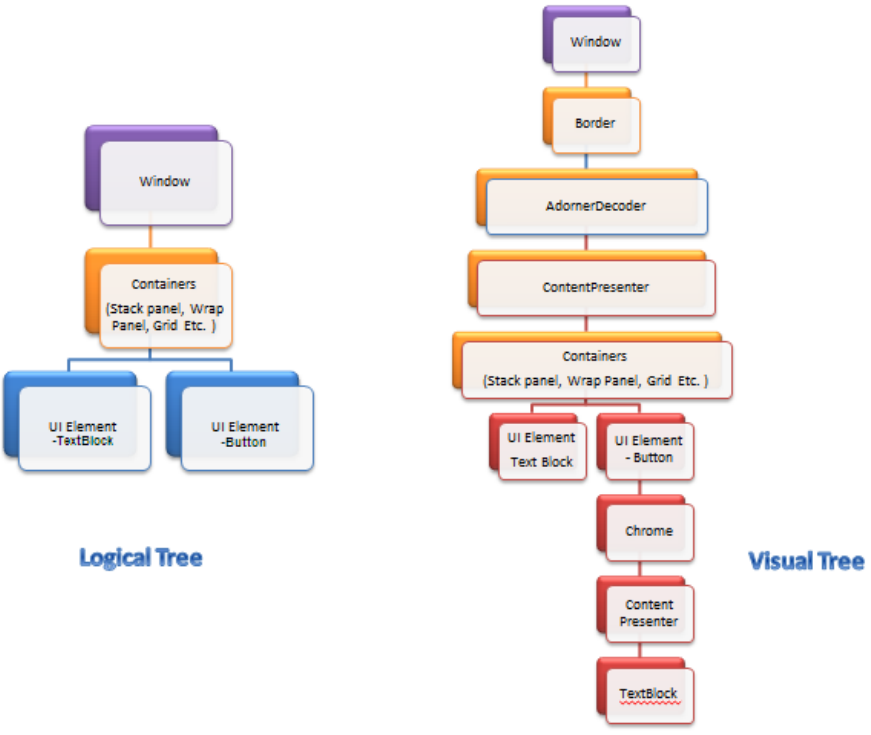
Когато WPF приложението стартира, то обикновено стартира две нишки. Едната се нарича зареждаща нишка (***Rendering Thread***) и програмиста няма достъп до нея. Втората е диспечерната нишка (***Dispatcher Thread***), която всъщност държи всички UI елементи. Така с други думи може да се каже, че диспечера е всъщност Ui нишката, която свързва всички елементи създадени в WPF приложението. Обратно, WPF изисква всички UI елементи да бъдат свързани с нишката на диспечера, а това се нарича ***Thread Affinity***. Следва че никой елемент създаден в диспечер нишката не може да бъде променян от друга нишка, както е попринцип в стандартните APIs базирани на Win32.

Dispatcher е клас, който обработва ***Thread Affinity***. Той е всъщност приоритизиран цикъл на съобщения, през които всички елементи се канализират. Всеки **UIElement** произлиза от **DispatcherObject**, който от своя страна дефинира свойство, наречено Dispatcher и сочещо към UI нишката. Обектът DispatcherObject има две основни задачи – да проверява и верифицира дали нишката има достъп до обекта.

Какво е визуално (Visual Tree) и логическо (Logical Tree) дърво?

Всеки програмен стил съдържа някакъв тип логическо дърво (LogicalTree), което обхваща елементите на цялата програма. Логическото дърво обхваща елементите, така както са описани в XAML. То съдържа само контролите, които са декларирани във вашия XAML.

Визуалното дърво (VisualTree) от своя страна включва частите, които съставят индивидуалните контроли. Обикновено няма нужда директно да се занимавате с визуалното дърво, но трябва да знаете какво включва всяка контрола, така че по-лесно да съставяте специфични шаблони.



Защо ни трябва RoutedEvent?

RoutedEvent е сравнително ново нещо в C# езика, но за тези които са се занимавали с JavaScript или други уеб технологии е нещо познато от браузъра. Всъщност има два типа RoutedEvent – Bubbles и Tunnels, в зависимост от начина на обхождане на елементите от визуалното дърво. Съществува и трети тип RoutedEvent, който е директен (Direct).

Когато се регистрира или предизвика Routed събитие, елементите на визуалното дърво се обхождат по начина за Bubbles или Tunnels, и се извикват всички асоциирани с тях RoutedEventHandlers един по един.

За да се различават двата типа, WPF отбелязва с Preview*** събития които тунелират, а само с *** - Bubbles събитията. Например *IsPreviewMouseDown* е събитие, което тунелира измежду елементите наследници на визуалното дърво, докато *MouseDown* ще направи Bubbles. Така събитието „Mouse Down“ за най-външния елемент ще бъде извикано в случай на IsPreviewMouseDown, а „Mouse Down“ за най-вътрешния елемент ще се прихване първо при MouseDown събитието.

Защо се използва DependencyObject?

Всяка WPF контрола произлиза от ***DependencyObject***. Това е клас, който поддържа новата за WPF система от свойства ***DependencyProperty***. След като всеки обект наследява DependencyObject, то той може да асоциира себе си множество вградени функционалности на WPF като EventTriggers, PropertyBindings, Animations и др.

Всеки DependencyObject има „наблюдател“ ***Observer*** или списък ***List*** и декларира 3 метода с имена **ClearValue**, **SetValue** и **GetValue**, които се използват за добавяне, промяна и изтриване на тези свойства. Едно DependencyProperty свойство ще се създаде само ако използвате SetValue, за да съхраните нещо.

Повече за хардуерното ускорение и Graphics Rendering Tiers в WPF?

Друго важно нещо, което трябва да знаете, е как се зарежда WPF графиката. Всъщност WPF рендирането автоматично засича до колко е поддържано хардуерното ускорение от конкретната система и се самонастройва от това.

За хардуерно рендиране, нещата от най-голямо значение са:

1. **Video RAM**: определя размера и броя на буферите, които приложението може да използва за визуализирането.
2. **Pixel Shader**: функционалност, която изчислява ефекта на базата на пикселите.
3. **Vertex Shader**: Това е програма за графична обработка, която изпълнява математически изчисления за Vertex на изхода. Те се използват за добавяне на специални ефекти към обекти в 3D среда.
4. **MultiTexture Blending**: Това е специална функция, която позволява да се прилагат две или повече текстури върху един и същ обект в 3D.

Рендериращата машина на WPF определея кой слой е най-подходящ за текущото приложение и прилага съответно това рендериращите слоеве.

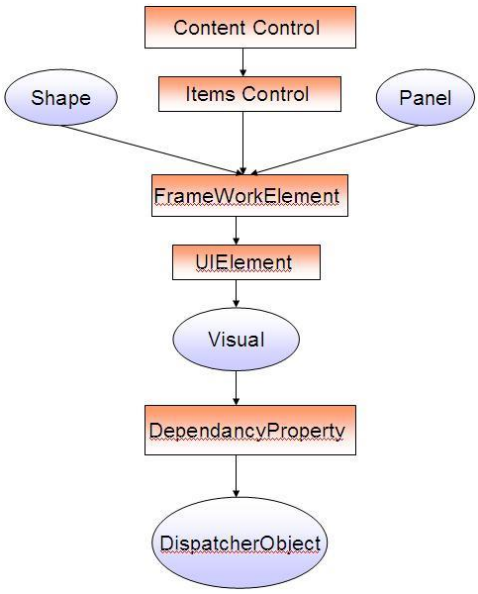
1. **TIER 0**: Не се извършва никакво хардуерно ускорение, вместо това всички се зарежда само със софтуерни изчисления. Всяка версия на DirectX 9 или по стара е способна да

рендерира този изход.

2. **TIER 1**: Използва се частично хардуерно и софтуерно зареждане. Необходим е Directx9 или по-нов.
3. **TIER 2**: Изцяло хардуерно ускорение. Необходим е Directx9 или по-нов.

Йерархия на обектите

В коя и да е WPF контрола има доста обекти. Нека ги разгледаме един по един както са на фигурата. Абстрактните класове са отбелязани с елипси, а конкретните класове с правоъгълници.



- `DispatcherObject`: Родител на всички WPF контроли, грижи се за UI нишката;
- `DependencyObject`: Построява Observer за Dependency свойствата;
- `Visual`: връзка между managed libraries и milcore
- `UIElement`: добавя поддръжка за WPF функционалности катоподредба, вход, събития и др.
- `FrameworkElement`: имплементация на `UIElement`
- `Shape`: Базов клас за всички основни форми
- `Control`: UI елементите, които взаимодействат с потребителя. Може да им се променя изгледа с Template.
- `ContentControl`: базов клас за всички контроли, които имат единично съдържание
- `ItemsControl`: базов клас за всички контроли, които показват колекции
- `Panel`: базов клас за всички панели, които показват една или повече контроли в себе си

Обратно към нашето тестово приложение

Нека добавим кода за един бутон в нашия XAML. Добавете следния код в Grid-а, след вече поставения TextBlock:

```
<Button HorizontalAlignment="Stretch" VerticalAlignment="Bottom" Height="100">
    Click me!
</Button>
```

Нека разгледаме по-подробно XAML-а. Той дефинира прозорец и бутон използвайки съответно **Window** и **Button** елементите. Всеки елемент се конфигурира с атрибути, като например **Title** атрибута на **Window**, който задава текста в заглавната лента на прозореца. При стартиране WPF преобразува елементите и техните атрибути от маркъп кода в инстанции на WPF класове. Например елемента **Window** се преобразува до инстанция на класа Window, чието Title свойство е стойността на **Title** атрибута.

Тъй като XAML е XML базиран, графичния интерфейс (UI), който съставяте с него, се асемблира в йерархична структура от вложени елементи, познато като дърво на елементите. Това дърво на елементите предоставя логически и интуитивен начин за създаване и управление на UI.

Code-behind или нашия C# код

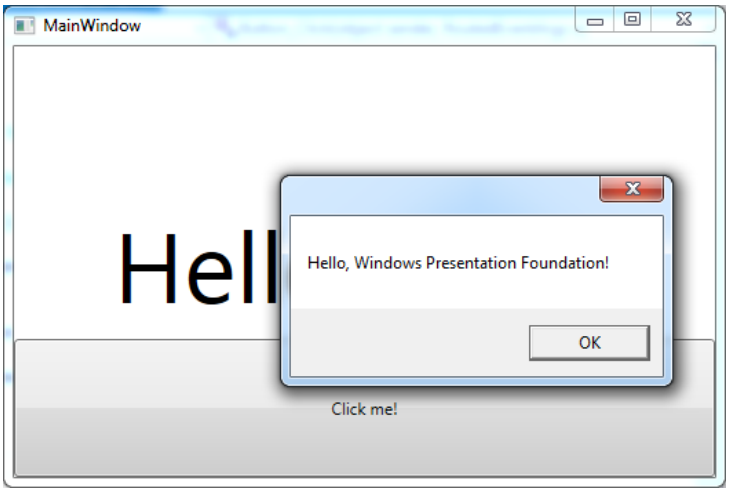
Поведението на приложението се имплементира посредством функционалността, която

отговаря на потребителското взаимодействие с програмата. Това включва обработката на събития (например натискане на меню, лента с инструменти или бутон) или обръщение към слоя на бизнес логиката или данните в отговор. В WPF поведението е основно имплементирано с код, който се асоциира с маркъпа. Този тип код се нарича code-behind.

В нашия пример, за да направим бутона да прави нещо, трябва да напишем неговия code-behind и да го свържем с маркъпа. Най-лесно това може да стане като изберете бутона и в Properties прозореца на Visual Studio изберете да ви се покажат неговите събития. Изберете Click и щракнете два пъти върху него. Ще ви се генерира метода Button_Click, а в XAML маркъпа ще се появи следния атрибут `Click="Button_Click"`. Добавете вече познатия MessageBox за извеждане на съобщение в Windows.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Hello, Windows Presentation Foundation!");
}
```

Тествайте програмата с F5.



В този пример code-behind имплементира клас, който наследява Window класа. Атрибута **x:Class** се използва, за да свърже маркъпа с класа от code-behind файла. Методът **InitializeComponent** се извиква от конструктора на code-behind класа, за да свърже дефинирания с маркъп UI и code-behind класа. **InitializeComponent** се генерира автоматично при компилирането на програмата и не е нужно да имплементиране нищо ръчно. Комбинацията от **x:Class** и **InitializeComponent** осигурява на вашата програма правилна инициализация при създаването ѝ. Code-behind класа също имплементира event handler за click събитието на бутона. Така, когато се натисне бутона event handler-а показва съобщение със стандарния за Windows MessageBox.Show() метод.

WPF контроли

В WPF “контрола” е обединяващ термин, който се прилага за категория от WPF класове, които се ползват в прозорец или страница, имат графичен потребителски интерфейс и имплементират някакво поведение.

Вградените WPF контроли, групирани по предназначение, са изредени по-долу:

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

Малко по-сериозно WPF приложение

Със следващите стъпки ще изградим едно по-сложно приложение на WPF, което ползва навигаця и страници. Ще разположим елементите в него с помощта на Grid. Ще направим малко по-съвременен дизайн и ще предаваме данни от една страница в друга.

Създаване и конфигуриране на приложение ExpenseIt

1. Към вече направения Solution, добавете нов проект (натиснете с десен бутон върху името на Solution-а и изберете Add → New Project…) и го кръстете **ExpenseIt**.
2. Отворете App.xaml. Този XAML файл дефинира WPF приложението и бъдещи негови ресурси. Ще използваме този файл по-късно, за да уточним кой UI ще бъде автоматично зареждан при стартиране на програмата; в нашия случай това е MainWindow.xaml.
3. Отворете MainWindow.xaml. Този XAML файл е главния прозорец на вашето приложение и ще изобразява съдържанието в страници. Класът Window дефинира свойствата на прозорец, а именно – заглавие, големина, икона, обработва събития като затваряне или скриване.
4. Изтрийте Grid елемента в MainWindow.xaml.
5. Променете типа на Window на NavigationWindow.

Правим тази смяна защото приложението ще навигира между различно съдържание в зависимост от действията на потребителя. NavigationWindow наследява всички свойства на Window, но в същото време предоставя възможност за навигация подобно на тази в уеб браузър.

6. Променете следните свойства на NavigationWindow елемента:
   - Променете свойството Title на "ExpenseIt".
   - Променете свойството Width на 500 pixels.
   - Променете свойството Height на 350 pixels.

Кодът ви трябва да изглежда така:

```
<NavigationWindow x:Class="ExpenseIt.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="ExpenseIt" Height="350" Width="500">

</NavigationWindow>
```

7. Отворете MainWindow.xaml.cs. Това е файла съдържащ code-behind, обработващ събитията декларирани в MainWindow.xaml. Този файл съдържа частичен клас за прозореца дефиниран в XAML.
8. Променете MainWindow класа да наследява **NavigationWindow**, а не **Window**.

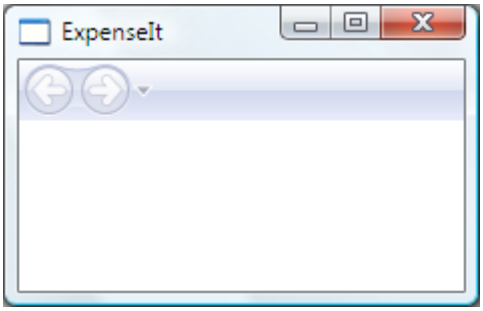
Добавяне на файлове към приложението

В тази част от упражнението ще добавим две страници и изображение към приложението.

1. Добавете нова страница Page към проекта (с десен бутон натиснете върху името на проекта и от контекстното меню изберете Add → New Item → Page). Дайте ѝ име **ExpenseItHome.xaml** .

Това ще е началната страница, която ще се показва при стартиране на приложението. Тя ще показва списък от хора, от които потребителя ще може да избере един, за да се покаже справката за разходите му.

2. Във отворилия се **ExpenseItHome.xaml** настройте заглавието **Title** на "**ExpenseIt - Home**".
3. В **MainWindow.xaml** сменете стойността на **Source** свойството на NavigationWindow на "**ExpenseItHome.xaml**". Това ще зададе ExpenseItHome.xaml да бъде първата страница отворена, когато приложението стартира.
4. Добавете още една страница както в точка 1, но я кръстете **ExpenseReportPage.xaml** този път. В ExpenseReportPage.xaml сменете името **Title** на "**ExpenseIt - View Expense**". Тази страница ще покзва справката за разходите на избрания от първата страница човек.
5. Свалете изображението от адрес https://i-msdn.sec.s-msft.com/dynimg/IC816740.jpeg и го запаметете на компютъра под името watermark.png. Добавете изображението към проекта, като натиснете десен бутон върху проекта → Add → Existing Item ... и избирате файла на изображението.
6. Направете приложението основно за Solution-а. С десен бутон върху него, от контекстното меню изберете **Set as StartUp project**.
7. Компилирайте и стартирайте приложението като натиснете F5 или изберете Start Debugging от менюто Debug.



Приложението ви трябва да изглежда по подобен начин. Забележете характерните за NavigationWindow бутони горе в ляво.

8. Затворете приложението и нека продължим с разработката му във Visual Studio.

Подредба на елементите

Подредбата (Layout) позволява различни начини на позициониране на UI елементите, като се грижи за тяхната колемина и положение при преоразмеряване на потребителския интерфейс. В тази част ще създадем таблица с една колона и 3 реда, като добавим дефиниции за редове и колони в Grid-а на ExpenseItHome.xaml.

1. В ExpenseItHome.xaml, настройте **Margin** свойството на Grid елемента на "**10,0,10,10**". Тези стойности определят разстоянията от ляво, горе, дясно и долу.
2. Добавете следния XAML код между отварящия и затварящ таг Grid, за да създадете дефинициите на редове и колони.

```
<Grid.ColumnDefinitions>
    <ColumnDefinition />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition />
    <RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
```

Височината на два от редовете се задава като Auto, което означава, че реда ще се оразмери в зависимост от неговото съдържание. Височината по подразбиране е със стойност звездичка (*), което означава, че реда ще заеме претеглена пропорция от свободното място. Например ако имаме два реда, всеки от които е с височина "*", и двата ще имат височина равна на половината от заделеното пространство.

Добавяне на контроли

В тази подточка ще променим UI на началната страница, така че да показва списък от хора, от които потребителя да може да избира. Контролите, които добавяме, са UI обекти, които позволяват на потребителя да взаимодейства с приложението.

За да създадем този UI, ще използваме следните елементи и ще ги добавим към ExpenseItHome.xaml:

- ListBox (за списъка от хора).
- Label (за хедър на списъка).
- Button (за да може да се натиска и да сочи към справката за разходите на избрания човек).

Всяка контрола се поставя в ред на Grid-а като се настройва свойството Grid.Row (attached property).

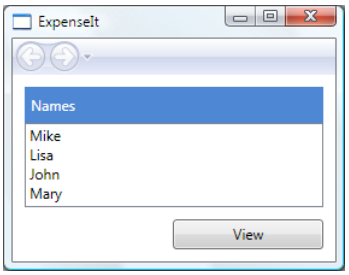
1. В ExpenseItHome.xaml добавете следния XAML код между </Grid.RowDefinitions> и </Grid> таговете.

```
  <!-- People list -->
  <Border Grid.Column="0" Grid.Row="0" Height="35" Padding="5"
Background="#4E87D4">
      <Label VerticalAlignment="Center" Foreground="White">Names</Label>
  </Border>
  <ListBox Name="peopleListBox" Grid.Column="0" Grid.Row="1">
```

```
        <ListBoxItem>Mike</ListBoxItem>
        <ListBoxItem>Lisa</ListBoxItem>
        <ListBoxItem>John</ListBoxItem>
        <ListBoxItem>Mary</ListBoxItem>
    </ListBox>

    <!-- View report button -->
    <Button Grid.Column="0" Grid.Row="2" Margin="0,10,0,0" Width="125"
Height="25" HorizontalAlignment="Right">View</Button>
```

2. Компилирайте и стартирайте приложението.



Приложението би трябвало да изглежда така.

Добавяне на изображение и заглавие

1. В ExpenseItHome.xaml добавете още една колона към ColumnDefinitions с фиксирана ширина Width на 230 пиксела.

```
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="230" />
    <ColumnDefinition />
</Grid.ColumnDefinitions>
```

2. Добавете и още един ред в RowDefinitions.

```
<Grid.RowDefinitions>
    <RowDefinition />
    <RowDefinition Height="Auto"/>
    <RowDefinition />
    <RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
```

3. „Преместете“ контролите във втората колона като настроите **Grid.Column** на **1**. Преместете и всяка от контролите с 1 ред надолу, като увеличите стойността на **Grid.Row** с **1**.

```
        <Border Grid.Column="1" Grid.Row="1" Height="35" Padding="5"
Background="#4E87D4">
            <Label VerticalAlignment="Center" Foreground="White">Names</Label>
        </Border>
        <ListBox Name="peopleListBox" Grid.Column="1" Grid.Row="2">
            <ListBoxItem>Mike</ListBoxItem>
            <ListBoxItem>Lisa</ListBoxItem>
            <ListBoxItem>John</ListBoxItem>
            <ListBoxItem>Mary</ListBoxItem>
        </ListBox>

        <!-- View report button -->
        <Button Grid.Column="1" Grid.Row="3" Margin="0,10,0,0" Width="125"
Height="25" HorizontalAlignment="Right">View</Button>
```

4. Настройте фона на Grid елемента, така че да приема картинката **watermark.png**, която

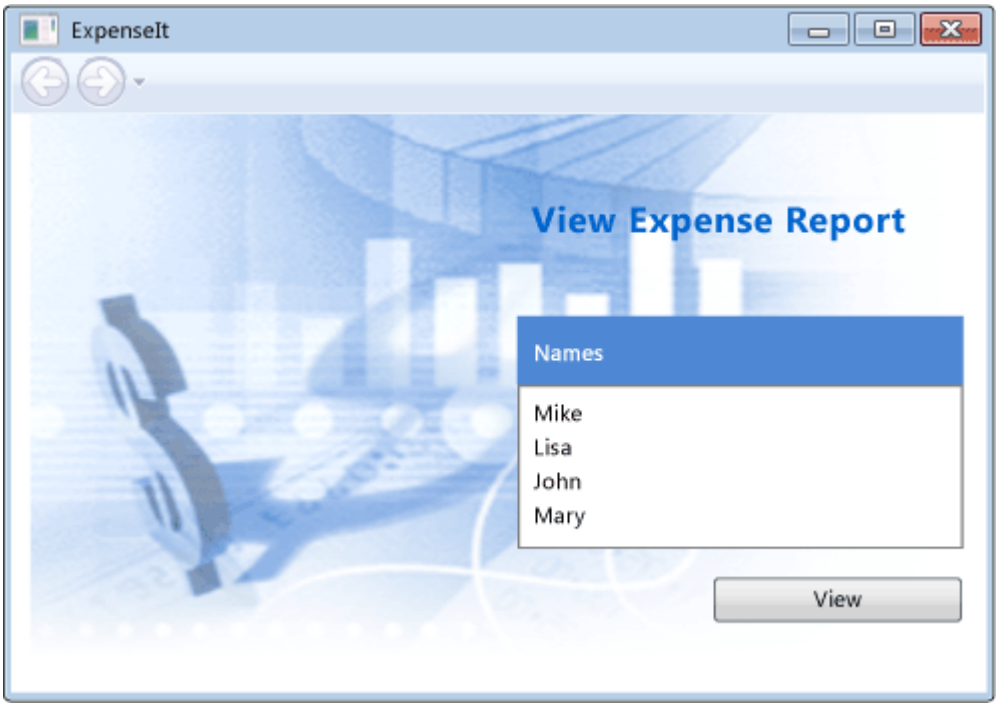
добавихме в предните стъпки. Добавете следния код **след** затварянето на </Grid.RowDefinitions> тага.

```
<Grid.Background>
    <ImageBrush ImageSource="watermark.png"/>
</Grid.Background>
```

5. Преди Border, добавете Label елемент със съдържание "View Expense Report", представляващ заглавието на страницата.

```
<Label Grid.Column="1">
    View Expense Report
</Label>
```

6. Компилирайте и стартирайте приложението.



Добавяне на код за прихващане на събития

1. В **ExpenseItHome.xaml** изберете **бутона** и в Properties прозореца превключете на **събития**. Изберете събитието **Click**, натиснете два пъти върху него, за да се генерира **event handler** за събитието и метода, в който ще пишем.
2. Добавете следния код в метода **Button_Click**

```
ExpenseReportPage expenseReportPage = new ExpenseReportPage();
this.NavigationService.Navigate(expenseReportPage);
```

Създаване на UI за ExpenseReportPage

ExpenseReportPage.xaml е предназначен да показва справката с разходите на конкретен човек, избран от страницата ExpenseItHome.xaml. В тази подточка ще добавим контроли и ще създадем графичния интерфейс на ExpenseReportPage.xaml. Ще добавим фонове и цветове за някои UI елементи.

1. Отворете **ExpenseReportPage.xaml** и добавете в него следния XAML код между Grid

таговете.

```
        <Grid.Background>
            <ImageBrush ImageSource="watermark.png" />
        </Grid.Background>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="230" />
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition />
        </Grid.RowDefinitions>

        <Label Grid.Column="1" VerticalAlignment="Center" FontFamily="Trebuchet MS"
FontWeight="Bold" FontSize="18" Foreground="#0066cc">
            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 Margin="0,0,0,5" FontWeight="Bold">Name:</Label>
                <Label Margin="0,0,0,5" FontWeight="Bold"></Label>
            </StackPanel>

            <!-- Department -->
            <StackPanel Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="1"
Orientation="Horizontal">
                <Label Margin="0,0,0,5" FontWeight="Bold">Department:</Label>
                <Label Margin="0,0,0,5" FontWeight="Bold"></Label>
            </StackPanel>

            <Grid Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="2"
VerticalAlignment="Top"
          HorizontalAlignment="Left">
                <!-- Expense type and Amount table -->
                <DataGrid  AutoGenerateColumns="False" RowHeaderWidth="0" >
                    <DataGrid.ColumnHeaderStyle>
                        <Style 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>
                    </DataGrid.ColumnHeaderStyle>
                    <DataGrid.Columns>
                        <DataGridTextColumn Header="ExpenseType" />
                        <DataGridTextColumn Header="Amount"  />
                    </DataGrid.Columns>
                </DataGrid>
            </Grid>
```

```
</Grid>
```

Този UI е подобен на графичния интерфейс в ExpenseItHome.xaml с тази разлика, че данните за разходите се представят в DataGrid.

2. Компилирайте и стартирайте приложението.
3. Натиснете бутона **View,** за да се покаже страницата за справката с разходи.

Страницата ExpenseReportPage.xaml би трябвало да изглежда така. Забележете, че бутона „назад“ от навигацията е станал автоматично активен.

Стилизиране на контроли

Често външния вид на елементите може да е еднакъв за всички елементи от даден тип в един UI. Потребителския интерфейс използва стилове, за да направи преизползваем този изглед измежду множество контроли. Това преизползване позволява да се опрости XAML кода. Повече за стиловете ще разгледаме в упражнение 7.

Тази подточка заменя със стилове отделните атрибути досега дефинирани в елементите

1. Отворете **App.xaml** и добавете следния XAML код между отварящия и затварящия таг на **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>
```

Горният XAML собавя следните стилове:

- **headerTextStyle**: за форматирането на заглавния Label на страницата.
- **labelStyle**: за форматиране на останалите Label контроли.
- **columnHeaderStyle**: за форматиране на DataGridColumnHeader.
- **listHeaderStyle**: за форматиране на контролите Border за заглавния ред на списъка.
- **listHeaderTextStyle**: за форматиране на Label контролата в заглавния ред на списъка.
- **buttonStyle**: за форматиране на бутона в ExpenseItHome.xaml.

Забележете, че стиловете са ресурси и наследници на елемента Application.Resources. Написани на това място, стиловете се прилагат към всички елементи в приложението.

2. В **ExpenseItHome.xaml** заменете всичко между **Grid** елементите със следния XAML.

```
<Grid.Background>
    <ImageBrush ImageSource="watermark.png"  />
</Grid.Background>

<Grid.ColumnDefinitions>
    <ColumnDefinition Width="230" />
    <ColumnDefinition />
</Grid.ColumnDefinitions>

<Grid.RowDefinitions>
    <RowDefinition/>
    <RowDefinition Height="Auto"/>
    <RowDefinition />
    <RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
```

```
<!-- People list -->

<Label Grid.Column="1" Style="{StaticResource headerTextStyle}" >
    View Expense Report
</Label>

<Border Grid.Column="1" Grid.Row="1" Style="{StaticResource listHeaderStyle}">
    <Label Style="{StaticResource listHeaderTextStyle}">Names</Label>
</Border>
<ListBox Name="peopleListBox" Grid.Column="1" Grid.Row="2">
    <ListBoxItem>Mike</ListBoxItem>
    <ListBoxItem>Lisa</ListBoxItem>
    <ListBoxItem>John</ListBoxItem>
    <ListBoxItem>Mary</ListBoxItem>
</ListBox>

<!-- View report button -->
<Button Grid.Column="1" Grid.Row="3" Click="Button_Click"
        Style="{StaticResource buttonStyle}">View</Button>
```

Свойства като VerticalAlignment и FontFamily, които дефинират изгледа на всяка контрола са премахнати и са заменени от прилагане на стилове. Например, headerTextStyle се прилага в елемента Label "View Expense Report".

3. В ExpenseReportPage.xaml заменете всичко между Grid елементите със следния XAML

```
        <Grid.Background>
            <ImageBrush ImageSource="watermark.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}"></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}"></Label>
            </StackPanel>

            <Grid Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="2"
VerticalAlignment="Top"
          HorizontalAlignment="Left">
                <!-- Expense type and Amount table -->
                <DataGrid ColumnHeaderStyle="{StaticResource columnHeaderStyle}"
                  AutoGenerateColumns="False" RowHeaderWidth="0" >
                    <DataGrid.Columns>
                        <DataGridTextColumn Header="ExpenseType" />
                        <DataGridTextColumn Header="Amount"  />
                    </DataGrid.Columns>
                </DataGrid>
            </Grid>
        </Grid>
```

Този код добавя стилове за Label и Border елементите.

4. Компилирайте и стартирайте приложението.

След като направите промените в XAML кода от тази точка, приложението трябва да изглежда по същия начин както преди да се реализират стиловете.

Свързване на данни (Binding Data) към контрола

1. Отворете **ExpenseItHome.xaml**, и след отварящия **Grid** елемент, добавете следния XAML код за създаването на **XmlDataProvider**, който ще съдържа данните за всеки един човек.

В случая ще зададем данните като ресурс на Grid-а, но обикновено те ще трябва да се четат от файл или база данни.

```
<Grid.Resources>
    <!-- Expense Report Data -->
    <XmlDataProvider x:Key="ExpenseDataSource" XPath="Expenses">
        <x:XData>
            <Expenses xmlns="">
                <Person Name="Mike" Department="Legal">
                    <Expense ExpenseType="Lunch" ExpenseAmount="50" />
                    <Expense ExpenseType="Transportation" ExpenseAmount="50" />
                </Person>
                <Person Name="Lisa" Department="Marketing">
                    <Expense ExpenseType="Document printing" ExpenseAmount="50"/>
                    <Expense ExpenseType="Gift" ExpenseAmount="125" />
                </Person>
                <Person Name="John" Department="Engineering">
                    <Expense ExpenseType="Magazine subscription" ExpenseAmount="50"/>
                    <Expense ExpenseType="New machine" ExpenseAmount="600" />
                    <Expense ExpenseType="Software" ExpenseAmount="500" />
                </Person>
                <Person Name="Mary" Department="Finance">
                    <Expense ExpenseType="Dinner" ExpenseAmount="100" />
                </Person>
            </Expenses>
        </x:XData>
    </XmlDataProvider>
</Grid.Resources>
```

2. В ресурсите на Grid-а, добавете следния **DataTemplate**, който дефинира как да се

визуализират данните в ListBox-а.

```
            ...
            <!-- Name item template -->
            <DataTemplate x:Key="nameItemTemplate">
                <Label Content="{Binding XPath=@Name}"/>
            </DataTemplate>
        </Grid.Resources>
```

3. Заменете съществуващия ListBox със следния XAML код.

```
<ListBox Name="peopleListBox" Grid.Column="1" Grid.Row="2"
   ItemsSource="{Binding Source={StaticResource ExpenseDataSource}, XPath=Person}"
   ItemTemplate="{StaticResource nameItemTemplate}">
</ListBox>
```

Този XAML свързва (binds) ItemsSource свойството на ListBox с източника на данни (data source) и прилага шаблона на данните (data template) като ItemTemplate.

Остана да напишем кода за извличането на информацията за човека, който е избран от списъка в страницата ExpenseItHome.xaml, и предаването ѝ към конструктора на ExpenseReportPage. ExpenseReportPage настройва своя контекст на данните (data context) с получения обект. А с него от своя страна ще се свържат (bind) контролите дефинирани в ExpenseReportPage.xaml.

4. Отворете ExpenseReportPage.xaml.cs. и добавете конструктор, който приема параметър обект, с който да подадете информация за разходните данни на избрания човек.

```
        // Custom constructor to pass expense report data
        public ExpenseReportPage(object data):this()
        {
            // Bind to expense report data.
            this.DataContext = data;
        }
```

5. В ExpenseItHome.xaml.cs променете метода на клик събитието, така че да извиква новия конструктор, предавайки данните за избрания човек.

```
        private void Button_Click(object sender, RoutedEventArgs e)
        {
            // View Expense Report
            //ExpenseReportPage expenseReportPage = new ExpenseReportPage();
            //this.NavigationService.Navigate(expenseReportPage);

            ExpenseReportPage expenseReportPage = new
ExpenseReportPage(this.peopleListBox.SelectedItem);
            this.NavigationService.Navigate(expenseReportPage);
        }
```

Стилизиране на данните с Data Templates

1. Отворете ExpenseReportPage.xaml, свържете (bind) съдържанието (content) на Label елементите за "Name" и "Department" със съответните data source свойства.

```
<!-- 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>
```

2. След отварящия таг на Grid елемента, добавете следните data templates, които дефинират как да бъдат изобразени данните за справката за разходи.

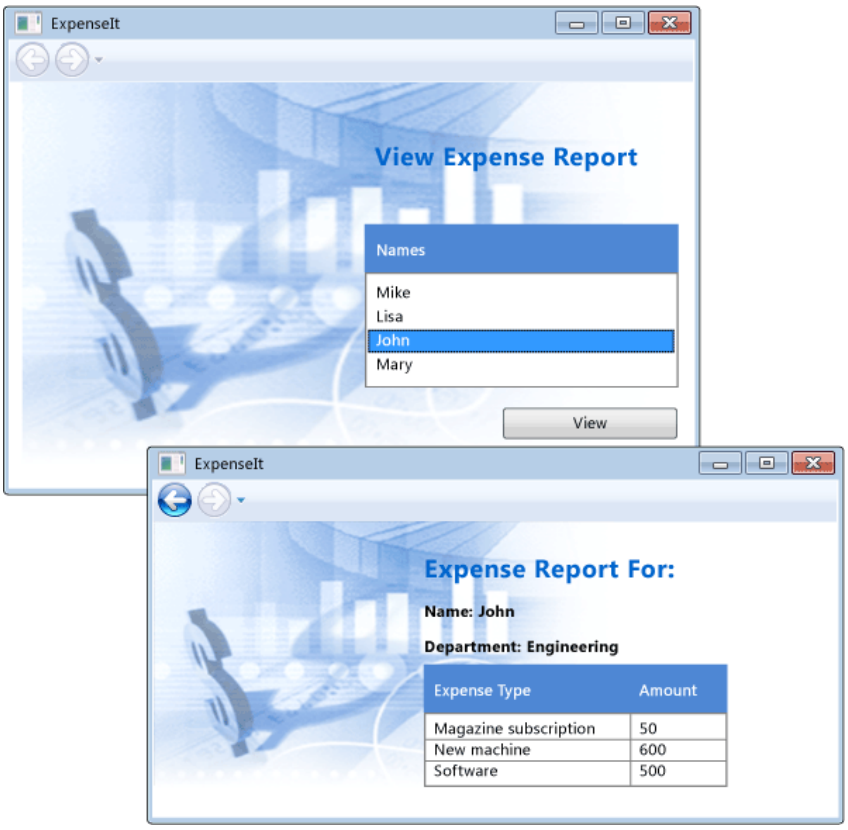
```
<!--Templates to display expense report data-->
<Grid.Resources>
    <!-- Reason item template -->
    <DataTemplate x:Key="typeItemTemplate">
        <Label Content="{Binding XPath=@ExpenseType}"/>
    </DataTemplate>
    <!-- Amount item template -->
    <DataTemplate x:Key="amountItemTemplate">
        <Label Content="{Binding XPath=@ExpenseAmount}"/>
    </DataTemplate>
</Grid.Resources>
```

3. За да се приложат горните темплейти към колоните на DataGrid, трябва да се добавят следните връзки, маркирани в жълто.

```
<!-- 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>
```

4. Стартирайте приложението, изберете човек и натиснете **View** бутона.

Програмата заедно с двата и прозореца трябва да изглежда по следния начин:



Добри практики

Този пример показва специфични особености на WPF и, следователно, не следва най-добрите практики за разработка на приложения. Повече за тези практики ще се опитаме да говорим в следващите упражнения, а дотогава самостоятелно можете да прочетете следните теми:

- Accessibility - Accessibility Best Practices
- Security - Security (WPF)
- Localization - WPF Globalization and Localization Overview
- Performance - Optimizing WPF Application Performance

Източници:

[1] Microsoft MSDN, Walkthrough: Getting Started with WPF, https://msdn.microsoft.com/library/ms752299%28v=vs.100%29.aspx
[2] The complete WPF tutorial, http://www.wpf-tutorial.com/
[3] Microsoft MSDN, Introduction to WPF, https://msdn.microsoft.com/library/aa970268%28v=vs.100%29.aspx
[4] Abhishek Sur, WPF Tutorial: Beginning, http://www.codeproject.com/Articles/140611/WPF-Tutorial-Beginning