Windows 10

Лабораторный практикум

Создание адаптивного пользовательского интерфейса

Октябрь 2015 года

Microsoft

Обзор

Приложения UWP могут выполняться на разных семействах устройств, которые существенно различаются в размере экранов и наборе функций. Чтобы обеспечить оптимальный пользовательский опыт на всех устройствах, дизайн должен быть адаптирован под устройство. Адаптивный UI может получать информацию об устройстве, на котором запущено приложение, и отображать макет соответственно характеристикам этого устройства.

Адаптивный (adaptive) интерфейс отличается от отзывчивого (responsive), например, он может предоставить индивидуализированный макет для каждого семейства устройств или размера экрана. Отзывчивый интерфейс отображает отдельный, гибкий дизайн, который выглядит корректно на всех устройствах. Один из недостатков такого пользовательского интерфейса - более медленная загрузка, элементы загружаются для всех устройств и разрешений, даже несмотря на то, что они, возможно, не потребуются. Адаптивный UI может быть основан на отзывчивом, однако вы также имеете возможность отображать уникальные виды для устройств, которые имеют мало общего друг с другом. Например, изображение на Xbox может полностью отличаться от вида приложения на рабочем столе компьютера и вида мобильного приложения, потому что способы взаимодействия пользователя с интерфейсом различаются.

В этом курсе мы переработаем фиксированный макет в адаптивный UI. Первоначально пользовательский интерфейс будет соответствовать более-менее стандартной практике. Затем мы создадим отдельный вид, чтобы обработать уникальные элементы дизайна для мобильного устройства.

Цели

Настоящий курс научит вас:

- Превращать фиксированный макет в адаптивный UI
- Создавать визуальные состояния приложения, чтобы поддерживать узкие, средние и широкие экраны
- Использовать класс RelativePanel, чтобы изменять состояние контента
- Использовать схему настройки, чтобы лучше приспосабливать стили для небольших экранов
- Использовать вид XAML, чтобы создавать уникальный вид для отдельных семейств устройств

Системные требования

Чтобы выполнить эту работу, необходимо обладать следующим набором программных инструментов:

- Microsoft Windows 10
- Microsoft Visual Studio 2015
- Эмулятор Windows 10 Mobile

Настройка

Вы должны осуществить следующие шаги для подготовки своего компьютера для этой работы:

1. Установите Microsoft Windows 10.
2. Установите Microsoft Visual Studio 2015. Выберите пользовательскую установку и убедитесь, что Инструменты разработки для приложений Windows выбраны из списка дополнительных функций.
3. Установите Windows 10 Mobile Emulator.

Упражнения

Эта практическая работа включает в себя следующие упражнения:

1. Переход к адаптивному пользовательскому интерфейсу
2. Класс RelativePanels с визуальными состояниями
3. Адаптивный пользовательский интерфейс с видами XAML

Расчётное время для завершения курса: **От 30 до 45 минут**.

Упражнение 1: Переход к адаптивному пользовательскому интерфейсу

При создании приложения использование фиксированного макета может показаться самым приемлемым решением из-за его простоты. Однако фиксированный дизайн не совсем подходит для приложений UWP, которые могут отображаться на различных по размеру и ориентации экранах. В этом упражнении вы создадите простой фиксированный макет, чтобы показать изображение и связанные метаданные. Затем вы преобразуете фиксированный макет в адаптивный.

Задача 1 – Создание приложения UWP

Начнём с создания проекта из шаблона пустого приложения.

1. В новой версии Visual Studio 2015 выберите **File (Файл) -> New (Новый) -> Project (Проект)**, чтобы открыть диалоговое окно New Project (Новый проект). Далее **Installed (Установленное) > Templates (Шаблоны) > Visual C# > Windows > Universal**, а затем выберите шаблон **Blank App** приложения (Universal Windows).

2. Назовите свой проект **AdaptiveUI** и выберите местоположение файловой системы, в которое будет осуществлено сохранение результатов прохождения Лабораторного практикума. Мы создали папку на диске **C** и переименовали ее в **HOL**, именно папку с этим названием вы будете видеть на скриншотах в ходе всего практического курса.

   Не изменяйте настройки, установленные для **Create new solution (Создания нового решения)** и **Create directory for solution (Создания папки для решения)**. Вы можете снять галочки как с **Add to source control (Добавить в систему контроля версий)**, так и **Show telemetry in the Windows Dev Center (Отобразить телеметрию в Windows Dev Center)**, если не хотите использовать соответствующие возможности. Нажмите **OK** для создания проекта.

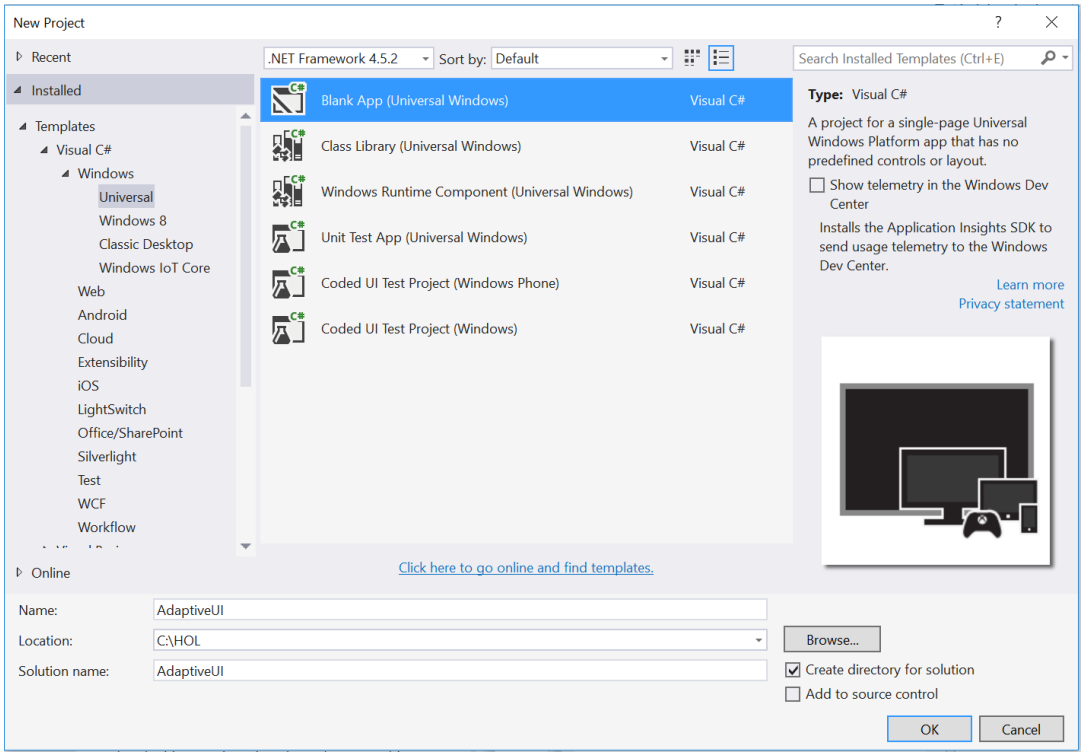

Рисунок 1

Создание нового проекта приложения в Visual Studio 2015.

3. Настройте Solution Configuration (Текущую конфигурацию решения) на **Debug (Отладка)** и Solution Platform (Платформа решений) на **x86**. Выберите **Local Machine (Локальный компьютер)** из выпадающего меню Debug Target (Цели отладки).

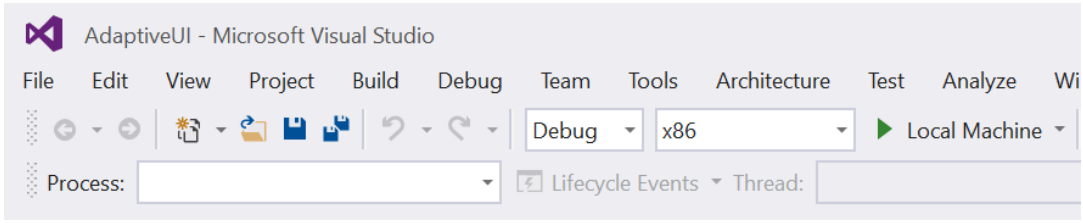

Рисунок 2

Сконфигурируйте свое приложение таким образом, чтобы оно запускалось на Local Machine (Локальном компьютере).

4. Скомпилируйте и запустите своё приложение. Вы увидите пустое окно приложения со счётчиком скорости кадров, активированном по умолчанию в режиме отладки.

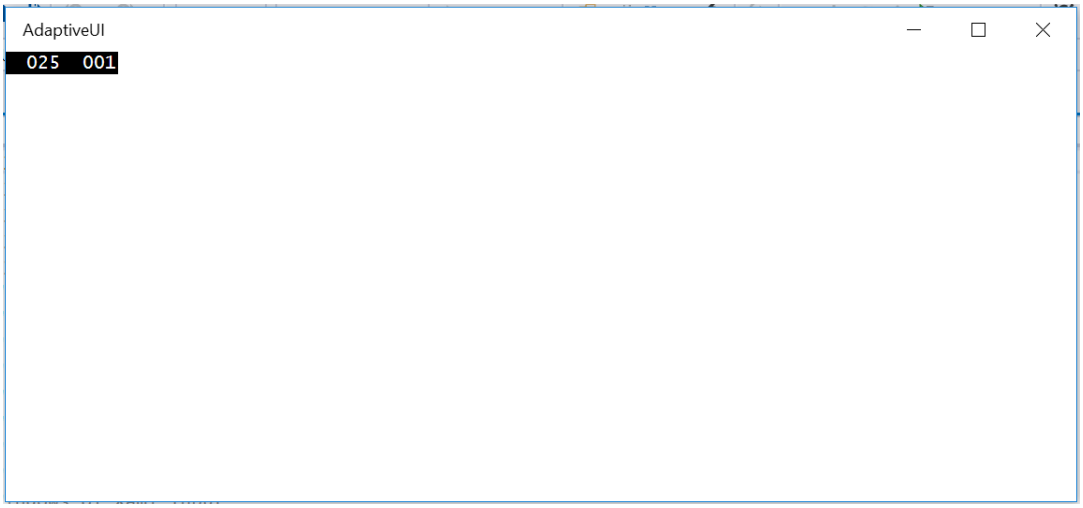

Рисунок 3

Пустое универсальное приложение, запущенное в режиме настольного компьютера.

Примечание: Счетчик частоты кадров является инструментом, используемым в процессе отладки, который помогает следить за производительностью вашего приложения. Он полезен для тех приложений, которые требуют интенсивной графической обработки, однако не подходит для простых приложений, которые будут создаваться вами в данном практическом курсе.

В шаблоне пустого приложения директива препроцессора для активации или отключения счетчика частоты кадров находится на **App.xaml.cs**. Счётчик скорости кадра может перекрываться или может скрыть ваше содержание приложения, если вы оставляете его. При выполнении данных работ вы можете отключить его, установив значение **DebugSettings.EnableFrameRateCounter** как **False (Ложное)**.

5. Вернитесь к Visual Studio и остановите отладку.

Задача 2 – Создание фиксированного макета

В качестве отправного пункта мы создадим основной фиксированный макет, содержащий простой контент. Содержание будет состоять из большого изображения и связанных метаданных: имени пользователя, аватара, имени изображения, даты и описания. Мы отформатируем содержание в две колонки.

1. Откройте **MainPage.xaml**. Добавьте два класса **ColumnDefinitions** к сетке с фиксированной шириной **500 эффективных пикселей**.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="500" />
        <ColumnDefinition Width="500" />
    </Grid.ColumnDefinitions>
</Grid>
```

2. Кликните правой кнопкой мыши на папке **Assets (Ресурсы)** и выберите **Add (Добавить) > Existing Item (Существующий элемент)**. Используйте диалоговое окно, чтобы переместиться к папке **Assets** и выберите **airtime.jpg** и **avatar.jpg**. Добавьте их в свой проект.

3. Вернитесь к **MainPage.xaml**. Добавьте изображение **airtime** в первую колонку.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="500" />
        <ColumnDefinition Width="500" />
    </Grid.ColumnDefinitions>
    <Image Grid.Column="0" Source="Assets/airtime.jpg"
      Stretch="UniformToFill" />
</Grid>
```

4. Добавьте аватар и текстовые метаданные во второй колонке.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="500" />
        <ColumnDefinition Width="500" />
    </Grid.ColumnDefinitions>
    <Image Grid.Column="0" Source="Assets/airtime.jpg"
Stretch="UniformToFill"
        />
    <StackPanel Grid.Column="1" Background="LightBlue">
        <Image Source="Assets/avatar.jpg" Width="100" Height="100"
            HorizontalAlignment="Left" />
        <TextBlock Text="phutureproof" />
        <TextBlock Foreground="White" FontSize="20" FontWeight="Light"
            Text="Airtime" />
        <TextBlock Text="9/15/15" />
        <TextBlock Text="Shot at Aiguille du Midi, Chamonix, France." />
    </StackPanel>
</Grid>
```

5. Скомпилируйте и запустите своё приложение. Для некоторых форматов экрана содержание будет четко умещаться на экране. Измените размер своего окна, чтобы увидеть поведение при меньшем и большем размерах окна. При меньших размерах экрана вторая колонка обрезается, а при больших размерах контент окружается пустым пространством.

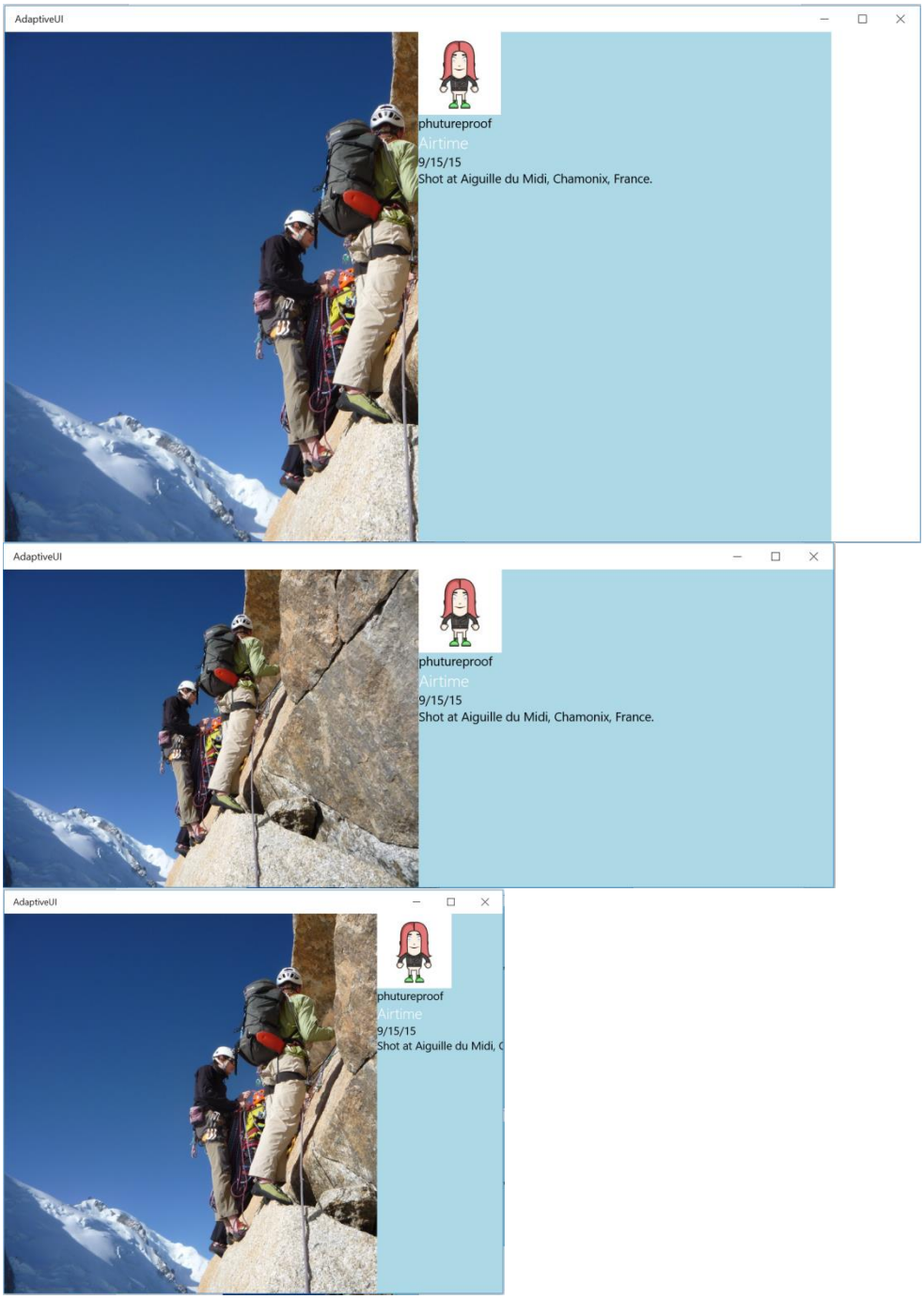

Рисунок 4

Колонки фиксированной ширины при разных размерах окна.

6. Остановите отладку и вернитесь к Visual Studio.

Задача 2 – Адаптация макета для различных размеров экрана

Сейчас, когда мы увидели недостатки фиксированного макета для приложения UWP, давайте сделаем его адаптивным. Мы будем следовать шести принципам гибкого дизайна:

- Reposition – репозиционирование, изменение положения
- Resize - изменение размера
- Reflow – изменение обтекания
- Reveal – открытие некоторых частей UI при увеличении размера
- Replace – замена
- Re-architect – перепроектирование, различные варианты дизайна

Примечание: Более подробную информацию об адаптивном дизайне в приложениях UWP см. по ссылкеhttps://msdn.microsoft.com/en-us/library/windows/apps/dn958435.aspx.

1. Добавьте определения строк к сетке. Удалите атрибуты фиксированной ширины столбца и используйте атрибуты x:Name. Мы будем использовать строки и столбцы, чтобы изменять положения контента в сетке.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition x:Name="LeftCol" />
        <ColumnDefinition x:Name="RightCol" />
    </Grid.ColumnDefinitions>
```

2. Создадим адаптивный триггер, который будет срабатывать на различных разрешениях экрана. Добавьте три визуальных состояния по условию **MinWindowWidth**, соответствующих разрешениям 320, 548 и 1024 точки по горизонтали.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="VisualStateGroup">
            <VisualState x:Name="VisualStateMin320">
                <VisualState.StateTriggers>
                    <AdaptiveTrigger MinWindowWidth="320"/>
                </VisualState.StateTriggers>
            </VisualState>
            <VisualState x:Name="VisualStateMin548">
                <VisualState.StateTriggers>
                    <AdaptiveTrigger MinWindowWidth="548"/>
                </VisualState.StateTriggers>
            </VisualState>
            <VisualState x:Name="VisualStateMin1024">
                <VisualState.StateTriggers>
                    <AdaptiveTrigger MinWindowWidth="1024"/>
                </VisualState.StateTriggers>
            </VisualState>
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
```

Примечание: Ваше приложение UWP будет запускаться на устройствах с самыми разными экранами. Полезным подходом при работе с группами устройств со сходными, но не идентичными характеристиками, является использование некоторых опорных точек. Некоторые рекомендованные точки – 320, 720, 1024 и 320, 548, 1024 в зависимости от того, как ваш UI адаптируется через визуальные состояния. То, какие именно точки вы выберете в качестве опорных, зависит от вашего приложения и конкретного дизайна, который вы создаёте. Необходимо всегда тестировать своё приложение с помощью множества различных экранов.

Адаптивные триггеры в UWP включают **MinWindowWidth** и **MinWindowHeight**. Мы будем использовать триггер MinWindowWidth в этом упражнении.

3. Используйте директивы **x:Name** для именования элементов (Hero и элемент StackPanel, содержащий контент метаданных), чтобы мы могли ссылаться на них в установщике VisualState. Расположите изображение героя в первой строке и столбце таблицы. Изображение останется в первом столбце и строке независимо от визуального состояния, таким образом, мы можем оставить его как статическую настройку. Контент метаданных будет перемещаться в зависимости от изменения ширины экрана. Удалите привязку **Grid.Column** из элемента StackPanel **MetaData**

XAML

```
<Image x:Name="Hero" Grid.Column="0" Grid.Row="0"
Source="Assets/airtime.jpg"
    Stretch="UniformToFill" />
<StackPanel x:Name="Metadata" Background="LightBlue">
```

4. Добавьте установщики визуального состояния, чтобы изменить поведение столбцов. Пока что, два визуальных состояния с меньшим размером экрана будут содержать одни и те же правила. Мы реализуем разные правила для них в следующем упражнении.

XAML

```
<VisualState x:Name="VisualStateMin320">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="320"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.Column)" Value="0" />
        <Setter Target="Metadata.(Grid.Row)" Value="1" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="1" />
    </VisualState.Setters>
</VisualState>
<VisualState x:Name="VisualStateMin548">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="548"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.Column)" Value="0" />
        <Setter Target="Metadata.(Grid.Row)" Value="1" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="1" />
    </VisualState.Setters>
</VisualState>
<VisualState x:Name="VisualStateMin1024">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="1024"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.Column)" Value="1" />
        <Setter Target="Metadata.(Grid.Row)" Value="0" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="2" />
    </VisualState.Setters>
</VisualState>
```

Примечание: Обратите внимание на синтаксис, используемый в установщиках VisualState, для установки таких свойств как (Grid.ColumnSpan) - свойство находится в скобках.

Такие свойства называются привязанными свойствами, и они обычно используются для наследования свойств дочерними элементами, т.е. позволяют дочернему элементу установить свойства родителя. Например, дочерние элементы для Grid могут использовать привязанные свойства, чтобы указать элементам Grid.Row и Grid.Column на необходимость их размещения в нужной строке и столбце сетки.

Этот метод является важной практикой для установки свойств, определяющих макет в каждом визуальном состоянии. Хотя некоторые значения могут переноситься без изменений из одного визуального состояния в другое, в то время как пользователь меняет размер окна, при запуске приложения окно может быть любого размера, так что важные свойства должны присутствовать в каждом визуальном состоянии, чтобы получить необходимый результат.

5. Отцентрируем изображение Hero с выравниванием по вертикали и горизонтали. Поскольку размер окна изменился, оно будет скрывать и показывать края изображения в зависимости от формата. Центрируя изображение, вы обеспечиваете большую вероятность того, что изображение останется на виду.

XAML

```
<Image x:Name="Hero" Grid.Column="0" Grid.Row="0"
Source="Assets/airtime.jpg"
    Stretch="UniformToFill" VerticalAlignment="Center"
    HorizontalAlignment="Center" />
```

6. Добавьте перенос текста для поля описания.

XAML

```
<TextBlock Text=" Shot at Aiguille du Midi, Chamonix, France "
    TextWrapping="WrapWholeWords" />
```

7. Добавим немного свободного пространства вокруг элемента Metadata с помощью Padding.

XAML

```
<StackPanel x:Name="Metadata" Background="LightBlue" Padding="12">
```

8. Без фиксированной ширины по умолчанию каждая колонка занимает 50% ширины окна приложения. Установите для левой колонки значение 66,6%, а для правой колонки - 33,3% для визуального состояния в 1024 пикселя, чтобы обеспечить больше места для изображения.

XAML

```
<VisualState x:Name="VisualStateMin1024">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="1024"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
```

```
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.Column)" Value="1" />
        <Setter Target="Metadata.(Grid.Row)" Value="0" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="2" />
        <Setter Target="LeftCol.Width" Value="2*" />
        <Setter Target="RightCol.Width" Value="1*" />
    </VisualState.Setters>
</VisualState>
```

Примечание: Звёздочка в присвоении ширины говорит элементу Grid, что значение ширины выражено как пропорция доступного пространства. Ваш контент охватывает оба столбца в меньших визуальных состояниях, так что не требуется устанавливать правила для ширины столбцов в этих состояниях.

9. Скомпилируйте и запустите своё приложение. Когда вы уменьшите окно, метаданные будут перемещаться в первый столбец под изображение героя. При большем размере метаданные отобразятся во втором столбце.

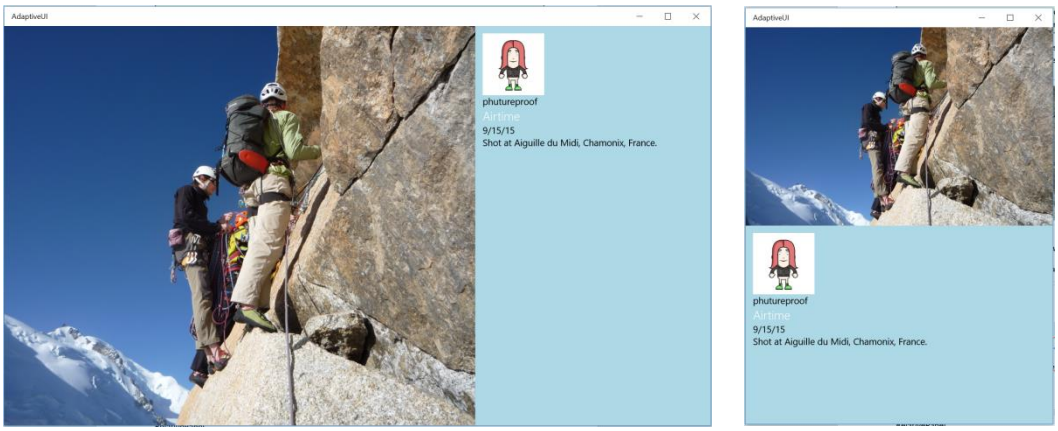

Рисунок 5

Метаданные перемещается под изображения при небольшом размере окна.

10. Остановите отладку и вернитесь к Visual Studio.

Упражнение 2: Класс RelativePanel с визуальными состояниями

В этом упражнении вы добавите класс RelativePanel, чтобы обработать семантически сгруппированный контент и ещё больше усовершенствовать отзывчивость дизайна.

Задача 1 – Добавление класса RelativePanel

Метаданные фотографии сгруппированы во всех визуальных состояниях, что делает удобным использование класса RelativePanel.

> **Примечание:** RelativePanel позволяет вам определить расположение его дочерних элементов относительно друг друга. В то время как расположение элементов относительно друг друга могут оставаться теми же в разных визуальных состояниях, вы также можете модифицировать их для разных размеров окна. Более подробную информацию о RelativePanel см. по ссылке: https://msdn.microsoft.com/library/windows/apps/windows.ui.xaml.controls.relativepanel.aspx

1. Замените элемент **StackPanel** с именем Metadata на **RelativePanel**, сохранив все прежние атрибуты без изменений.

XAML

```
<RelativePanel x:Name="Metadata" Background="LightBlue" Padding="12">
    <Image Source="Assets/avatar.jpg" Width="100" Height="100"
HorizontalAlignment="Left" />
    <TextBlock Text="phutureproof" />
    <TextBlock Foreground="White" FontSize="20" FontWeight="Light"
        Text="Airtime" />
    <TextBlock Text="9/15/15" />
    <TextBlock Text="Shot at Aiguille du Midi, Chamonix, France."
        TextWrapping="WrapWholeWords" />
</RelativePanel>
```

2. Заключите элементы TextBlock, показывающие заголовок изображения, дату и описание в в **StackPanel**. Эти блоки будут привязаны друг к другу, поэтому будет легче разместить их в StackPanel вместе, чем по отдельности.

XAML

```
<RelativePanel x:Name="Metadata" Background="LightBlue" Padding="12">
    <Image Source="Assets/avatar.jpg" Width="100" Height="100"
HorizontalAlignment="Left" />
    <TextBlock Text="phutureproof" />
    <StackPanel>
        <TextBlock Foreground="White" FontSize="20" FontWeight="Light"
            Text="Airtime" />
        <TextBlock Text="9/15/15" />
```

```
        <TextBlock Text="Shot at Aiguille du Midi, Chamonix, France."
            TextWrapping="WrapWholeWords" />
    </StackPanel>
</RelativePanel>
```

3. Присвойте атрибуты **x:Name** дочерним элементам класса **RelativePanel**. Эти элементы будут изменять свое расположение относительно друг друга.

XAML

```
<RelativePanel x:Name="Metadata" Background="LightBlue" Padding="12">
    <Image x:Name="Avatar" Source="Assets/avatar.jpg" Width="100"
Height="100"
        HorizontalAlignment="Left" />
    <TextBlock x:Name="Username" Text="phutureproof" />
    <StackPanel x:Name="Description" >
        <TextBlock Foreground="White" FontSize="20" FontWeight="Light"
            Text="Airtime" />
        <TextBlock Text="9/15/15" />
        <TextBlock Text="Shot at Aiguille du Midi, Chamonix, France."
            TextWrapping="WrapWholeWords" />
    </StackPanel>
</RelativePanel>
```

4. Используйте **RelativePanel.Below** и **RelativePanel.AlignHorizontalCenterWith** для размещения имени пользователя относительно аватара пользователя. Положение этих элементов относительно друг друга будет оставаться таким же во всех визуальных состояниях.

XAML

```
<RelativePanel x:Name="Metadata" Background="LightBlue" Padding="12">
    <Image x:Name="Avatar" Source="Assets/avatar.jpg" Width="100"
Height="100"
        HorizontalAlignment="Left" />
    <TextBlock x:Name="Username" RelativePanel.Below="Avatar"
        RelativePanel.AlignHorizontalCenterWith="Avatar"
Text="phutureproof"
        />
    <StackPanel x:Name="Description" >
        <TextBlock Foreground="White" FontSize="20" FontWeight="Light"
            Text="Airtime" />
        <TextBlock Text="9/15/15" />
        <TextBlock Text=" Shot at Aiguille du Midi, Chamonix, France."
            TextWrapping="WrapWholeWords" />
    </StackPanel>
</RelativePanel>
```

5. Добавьте установщики визуальных состояний, чтобы определить размещение описания, связанного с аватаром пользователя. Для состояний **VisualStateMin1024** и **VisualStateMin548** есть свободное пространство для отображения описания справа от аватара пользователя. Для состояния **VisualStateMin320** описание можно пометить под

изображением. Также будем изменять границы (margins) для лучшего визуального восприятия.

XAML

```
<VisualState x:Name="VisualStateMin320">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="320"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.Column)" Value="0" />
        <Setter Target="Metadata.(Grid.Row)" Value="1" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="1" />
        <Setter Target="Description.(RelativePanel.RightOf)" Value="" />
        <Setter Target="Description.(RelativePanel.Below)" Value="Username"
/>
        <Setter Target="Description.Margin " Value="0,12,0,0" />
    </VisualState.Setters>
</VisualState>
<VisualState x:Name="VisualStateMin548">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="548"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.Column)" Value="0" />
        <Setter Target="Metadata.(Grid.Row)" Value="1" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="1" />
        <Setter Target="Description.(RelativePanel.RightOf)" Value="Avatar"
/>
        <Setter Target="Description.(RelativePanel.Below)" Value="" />
        <Setter Target="Description.Margin " Value="12,0,0,0" />
    </VisualState.Setters>
</VisualState>
<VisualState x:Name="VisualStateMin1024">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="1024"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.Column)" Value="1" />
        <Setter Target="Metadata.(Grid.Row)" Value="0" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="2" />
        <Setter Target="LeftCol.Width" Value="2*" />
        <Setter Target="RightCol.Width" Value="1*" />
```

```
        <Setter Target="Description.(RelativePanel.RightOf)" Value="Avatar"
/>
        <Setter Target="Description.(RelativePanel.Below)" Value="" />
        <Setter Target="Description.Margin " Value="12,0,0,0" />
    </VisualState.Setters>
</VisualState>
```

Примечание: В некоторых визуальных состояниях свойства класса RelativePanel устанавливаются равными пустой строке. Пустая строка стирает настройки, которые могут иначе сохраниться от других визуальных состояний. Без этих установщиков некоторые нежелательные относительные положения могут наследоваться различными визуальными состояниями.

6. В дополнение к внесению изменений в расположение элементов макета, установщики в визуальном состоянии могут переопределять такие стили, как размер шрифта и отступ, для лучшего отображения на различных устройствах. Удалите атрибут шрифта из текстового блока TextBlock, отображающего имя изображения, и добавьте директиву **x:Name**, чтобы позволить вам обращаться к данному элементу.

XAML

```
<TextBlock x:Name="ImageName" Foreground="White" FontWeight="Light"
    Text="Airtime" />
```

7. Добавьте установщики для изменения размера шрифта в зависимости от VisualState.

XAML

```
<VisualState x:Name="VisualStateMin320">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="320"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.Column)" Value="0" />
        <Setter Target="Metadata.(Grid.Row)" Value="1" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="1" />
        <Setter Target="Description.(RelativePanel.RightOf)" Value="" />
        <Setter Target="Description.(RelativePanel.Below)" Value="Username"
/>
        <Setter Target="Description.Margin " Value="0,12,0,0" />
        <Setter Target="ImageName.FontSize" Value="20" />
    </VisualState.Setters>
</VisualState>
<VisualState x:Name="VisualStateMin548">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="548"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="2" />
```

```
        <Setter Target="Hero.(Grid.RowSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.Column)" Value="0" />
        <Setter Target="Metadata.(Grid.Row)" Value="1" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="1" />
        <Setter Target="Description.(RelativePanel.RightOf)" Value="Avatar"
/>
        <Setter Target="Description.(RelativePanel.Below)" Value="" />
        <Setter Target="Description.Margin " Value="12,0,0,0" />
        <Setter Target="ImageName.FontSize" Value="20" />
    </VisualState.Setters>
</VisualState>
<VisualState x:Name="VisualStateMin1024">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="1024"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.Column)" Value="1" />
        <Setter Target="Metadata.(Grid.Row)" Value="0" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="2" />
        <Setter Target="LeftCol.Width" Value="2*" />
        <Setter Target="RightCol.Width" Value="1*" />
        <Setter Target="Description.(RelativePanel.RightOf)" Value="Avatar"
/>
        <Setter Target="Description.(RelativePanel.Below)" Value="" />
        <Setter Target="Description.Margin " Value="12,0,0,0" />
        <Setter Target="ImageName.FontSize" Value="24" />
    </VisualState.Setters>
</VisualState>
```

8. Скомпилируйте и запустите своё приложение. Измените размер окна, чтобы понаблюдать, как меняется макет с классом RelativePanel.

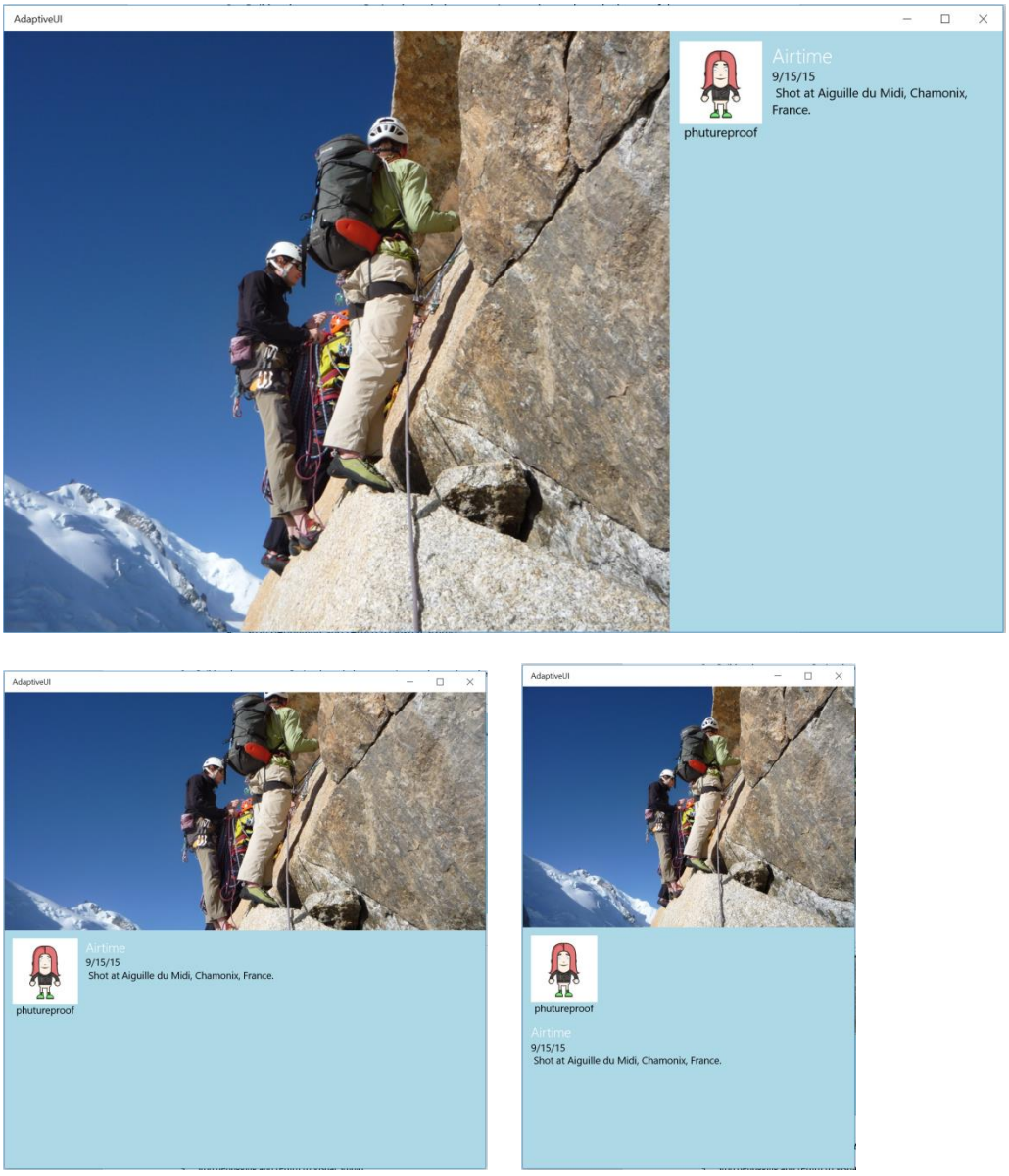

Рисунок 6

Контент с использованием класса RelativePanel перемещается внутри окна приложения при изменении размера.

9. Остановите отладку и вернитесь к Visual Studio.

Упражнение 3: Адаптивный пользовательский интерфейс на основе XAML Views

В некоторых случаях не имеет смысла использовать адаптивный дизайн для различных размеров экрана, и может быть проще начать с отдельных разметок XAML для каждого типа устройств. Виды XAML (XAML Views) позволяют вам реализовать такой подход. В этом упражнении мы добавим функцию скроллинга в приложение и создадим отдельный вид главной страницы MainPage XAML для мобильного устройства.

Задача 1 – Добавление ScrollViewer в панель метаданных

Прежде чем мы будем создавать новый вид XAML, давайте понаблюдаем, как ScrollViewer будет вести себя на существующей странице MainPage.

1. Добавьте к классу **RelativePanel** с именем Metadata **ScrollViewer,** и перенесите директиву **x:Name** для метаданных в ScrollViewer.

XAML

```
<ScrollViewer x:Name="Metadata" VerticalScrollBarVisibility="Auto">
    <RelativePanel Background="LightBlue" Padding="12">
        <Image x:Name="Avatar" Source="Assets/avatar.jpg" Width="100"
Height="100" HorizontalAlignment="Left" />
        <TextBlock x:Name="Username" Text="phutureproof"
            RelativePanel.Below="Avatar"
            RelativePanel.AlignHorizontalCenterWith="Avatar" />
        <StackPanel x:Name="Description">
            <TextBlock x:Name="ImageName" Foreground="White"
                FontWeight="Light" Text="Airtime" />
            <TextBlock Text="9/15/15" />
            <TextBlock Text=" Shot at Aiguille du Midi, Chamonix, France."
                TextWrapping="WrapWholeWords" />
        </StackPanel>
    </RelativePanel>
</ScrollViewer>
```

2. Скопируйте более длинную строку в подпись изображения.

XAML

```
<TextBlock Text="Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Nulla imperdiet pulvinar nunc. In et gravida ipsum. Morbi congue consequat
ullamcorper. Integer ornare porta convallis. Pellentesque habitant morbi
tristique senectus et netus et malesuada fames ac turpis egestas. Vivamus
sem nisi, ornare vel laoreet vel, accumsan facilisis tortor. Pellentesque
ut nunc in leo vehicula pretium et at quam. Aliquam euismod id purus nec
ultrices. Aliquam sed nisl at erat maximus finibus in sed urna. Nulla
ullamcorper vehicula ex, in porta ante ullamcorper id. Phasellus a enim
vitae odio ultricies semper. Suspendisse fermentum, erat in sodales
accumsan, lacus urna aliquam nisi, sed ultricies dolor orci quis ligula."
    TextWrapping="WrapWholeWords" />
```

3. Скомпилируйте и запустите своё приложение. Измените размер окна приложения до высоты, при которой появляется полоса прокрутки. Полоса прокрутки появляется только для метаданных, что означает, что второй столбец прокручивается при большем размере окна, а вторая строка прокручивается при меньшем размере окна.

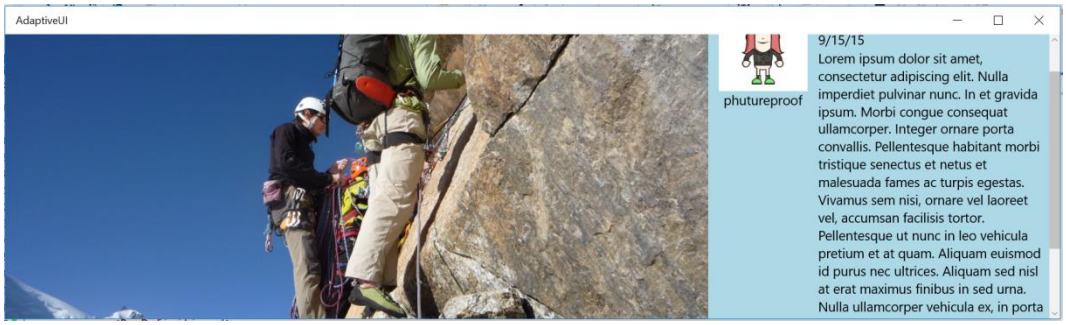

Рисунок 7

Метаданные прокручиваются, в то время как изображение остаётся на месте.

4. Остановите отладку и вернитесь в Visual Studio.

Задача 2 – Создание вида XAML

Использование прокрутки для меньших визуальных состояний - неидеальное решение. Гораздо лучше было бы, если вся страница прокручивалась на мобильных устройствах. Мы могли бы обернуть Grid в ScrollViewer, но тогда изображение героя также прокручивалось бы при большом размере окна. В этой задаче мы создадим отдельный вид для мобильного устройства, чтобы было удобнее управлять прокруткой, не меняя порядок работы приложения на настольном компьютере.

1. Щелкните правой кнопкой мыши на Solution Explorer (Обозреватель решений). **Затем Add (Добавить) > New Folder (Новая папка)**. Присвойте папке имя **DeviceFamily-Mobile**.

Примечание: Папка и вид должны называться определенным образом, чтобы XAML View правильно работал. Папки должны называться **DeviceFamily-*FamilyName***, а страницы внутри должны иметь такие же имена, как и исходные страницы, которые они будут заменять.

Кроме того, вы можете добавить файлы прямо в ту же папку, где находятся исходные страницы XAML, и присвоить им следующие имена ***OriginalPage*.DeviceFamily-*FamilyName*.xaml**.

2. Щёлкнитесь правой кнопкой мыши на папке и выберите **Add (добавить) > New Item (Новый элемент)** для добавления **вида XAML** (XAML View). Назовите его **MainPage.xaml**.

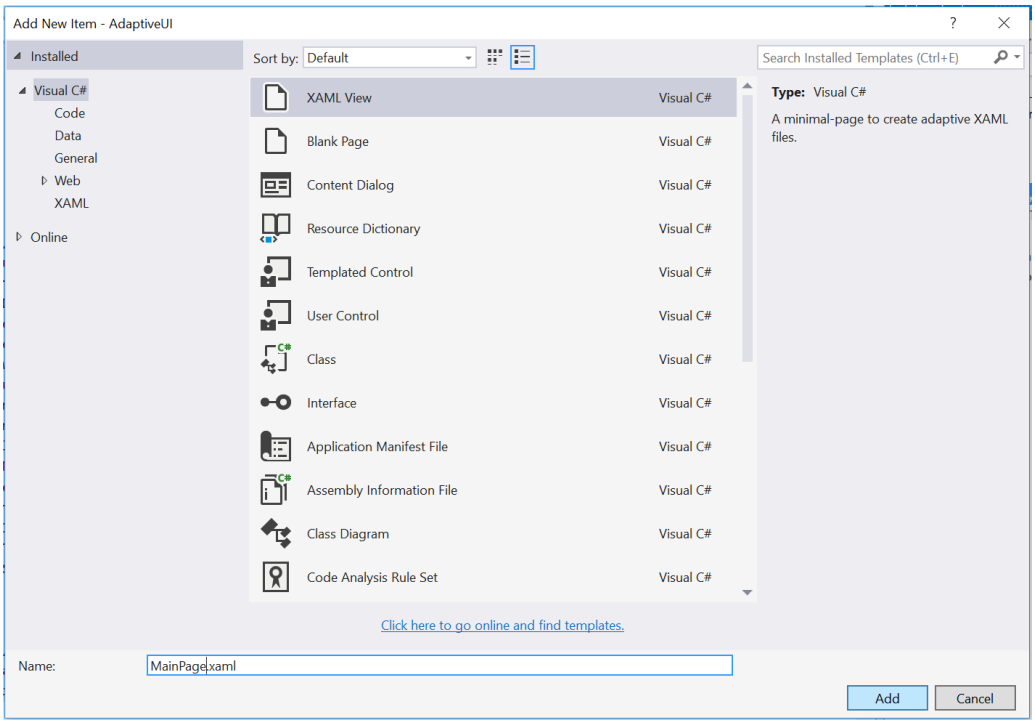

Рисунок 8

Добавдение вида XAML.

Примечание: Вид XAML отличается от пустой страницы тем, что он не генерирует файл с C#-кодом. Мы будем использовать такой же выделенный код MainPage.xaml.cs для мобильных устройств и настольного компьютера, так что мы создадим вид XAML вместо страницы.

3. Скопируйте содержание своего исходного файла **MainPage.xaml** в новый файл **MainPage.xaml**.

4. В новом файле **MainPage.xaml** удалите все визуальные состояния и VisualStateManager.

5. Удалите определения столбцов и привязку столбцов.

6. Удалите атрибуты **метаданных x:Name** и переместите класс **ScrollViewer** за пределы кода Grid.

XAML

```
<ScrollViewer VerticalScrollBarVisibility="Auto">
    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
```

```
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
        </Grid.RowDefinitions>
        <Image x:Name="Hero" Source="Assets/airtime.jpg"
            Stretch="UniformToFill" HorizontalAlignment="Center"
            VerticalAlignment="Center" />
        <RelativePanel Background="LightBlue" Padding="12">
            <Image x:Name="Avatar" Source="Assets/avatar.jpg" Width="100"
                Height="100" HorizontalAlignment="Left" />
            <TextBlock x:Name="Username" Text="phutureproof"
                RelativePanel.Below="Avatar"
                    RelativePanel.AlignHorizontalCenterWith="Avatar" />
            <StackPanel x:Name="Description">
                <TextBlock x:Name="ImageName" Foreground="White"
                    FontWeight="Light" Text="Airtime" />
                <TextBlock Text="9/15/15" />
                <TextBlock Text="Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Nulla imperdiet pulvinar nunc. In et gravida ipsum. Morbi
congue consequat ullamcorper. Integer ornare porta convallis. Pellentesque
habitant morbi tristique senectus et netus et malesuada fames ac turpis
egestas. Vivamus sem nisi, ornare vel laoreet vel, accumsan facilisis
tortor. Pellentesque ut nunc in leo vehicula pretium et at quam. Aliquam
euismod id purus nec ultrices. Aliquam sed nisl at erat maximus finibus in
sed urna. Nulla ullamcorper vehicula ex, in porta ante ullamcorper id.
Phasellus a enim vitae odio ultricies semper. Suspendisse fermentum, erat
in sodales accumsan, lacus urna aliquam nisi, sed ultricies dolor orci quis
ligula."
                    TextWrapping="WrapWholeWords" />
            </StackPanel>
        </RelativePanel>
    </Grid>
</ScrollViewer>
```

7. Добавьте привязку **Grid.Row** для изображения **Hero** и класса **RelativePanel**.

XAML

```
<Image x:Name="Hero" Grid.Row="0" Source="Assets/airtime.jpg"
    Stretch="UniformToFill" HorizontalAlignment="Center"
    VerticalAlignment="Center" />
<RelativePanel Grid.Row="1" Background="LightBlue" Padding="12">
```

8. Добавьте размер шрифта как атрибут для элемента **ImageName**.

XAML

```
<TextBlock x:Name="ImageName" FontSize="20" Foreground="White"
FontWeight="Light" Text="Airtime" />
```

9. Присвойте для **RelativePanel.Below="Username" в блок** для текста Description**.**

XAML

```
<StackPanel x:Name="Description" RelativePanel.Below="Username">
```

10. Запустите свое приложение в эмуляторе мобильной платформы. Нажмите на контент, чтобы увидеть полосу прокрутки. И изображение, и метаданные будут прокручиваться.

Рисунок 9

Адаптивный мобильный вид приложения, запущенное на эмуляторе мобильной платформы.

11. Остановите отладку и вернитесь к Visual Studio.

12. Вы также можете запустить приложение на локальном компьютере. Вы увидите, что первоначальный вид будет использоваться при запуске приложения на локальном компьютере.

Краткий обзор

Адаптивный интерфейс является важной частью дизайна приложений UWP. В этом курсе мы следовали принципам адаптивного дизайна для перемещения и обновления контента для его лучшего отображения на различных экранах. Мы также создали отдельный вид для мобильного устройства, чтобы обеспечить новые возможности, недоступные для адаптивного дизайна.