**Задание. XAML интерфейс мобильного приложения**

Данное методическое указание содержит инструкции для изучения особенностей программирования интерфейсов с использованием языка XAML.

Разработка кроссплатформенных приложений Xamarin.Forms выполняется в Visual Studio 2019 с использованием шаблона проекта, который называется Mobile App (Xamarin.Forms).

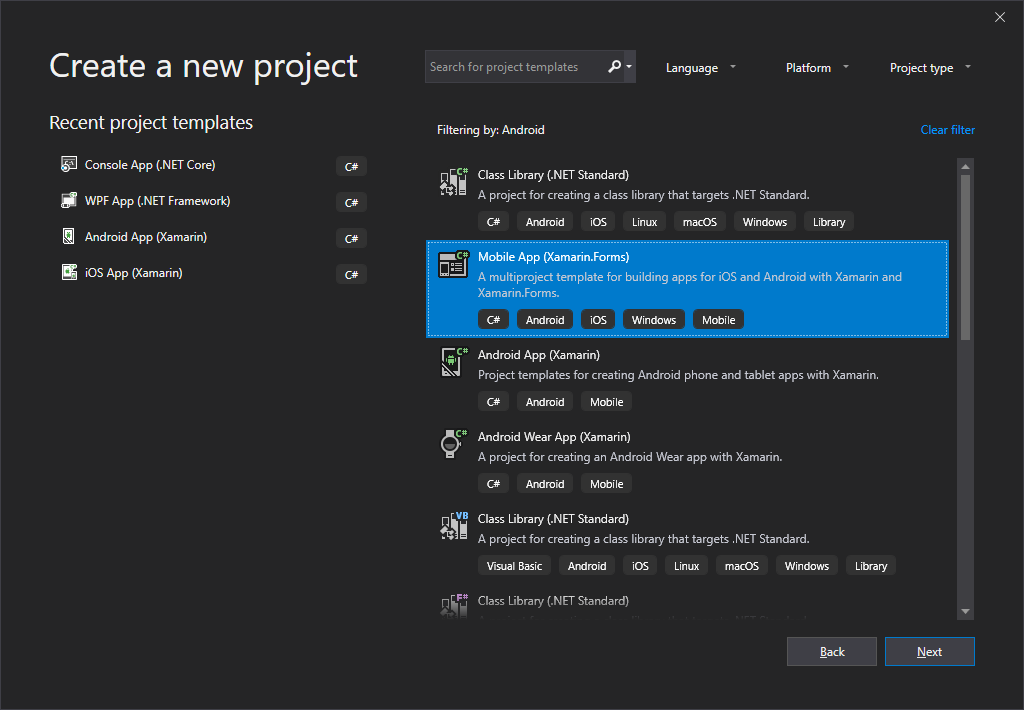


Рисунок 1 – создание нового проекта

Можно выбрать проект Mobile App (Xamarin.Forms), работающий на всех платформах, либо отдельно Android App или iOS App.

Затем нужно выбрать шаблон. В этом руководстве рассматривается создание мобильного приложения на примере пустого (Blank) проекта. Обратите внимание, что можно выбрать платформы Android, iOS и UWP. Для последнего нужно скачать дополнительный компонент в Visual Studio Installer — Universal Windows Platform development.

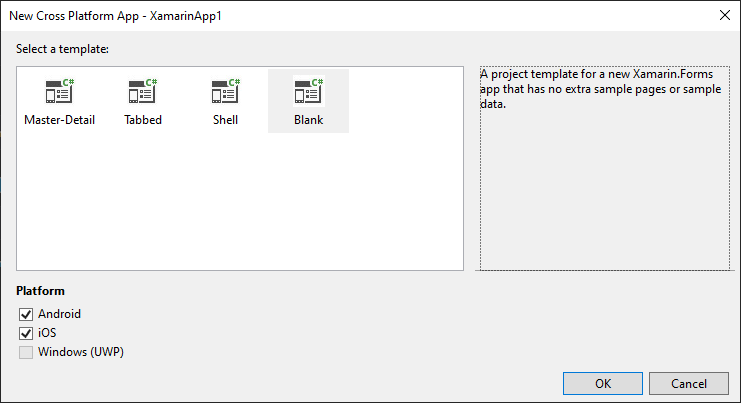


Рисунок 2 – выбор шаблона

Visual Studio понадобится какое-то время, чтобы подготовить все файлы проекта, а потом стоит сразу установить средства для отладки приложений. Для этого понадобится либо устройство на iOS или Android, либо эмулятор.

Чтобы создать эмулятор, выберите пункт Create Android Emulator…, как показано на скриншоте:

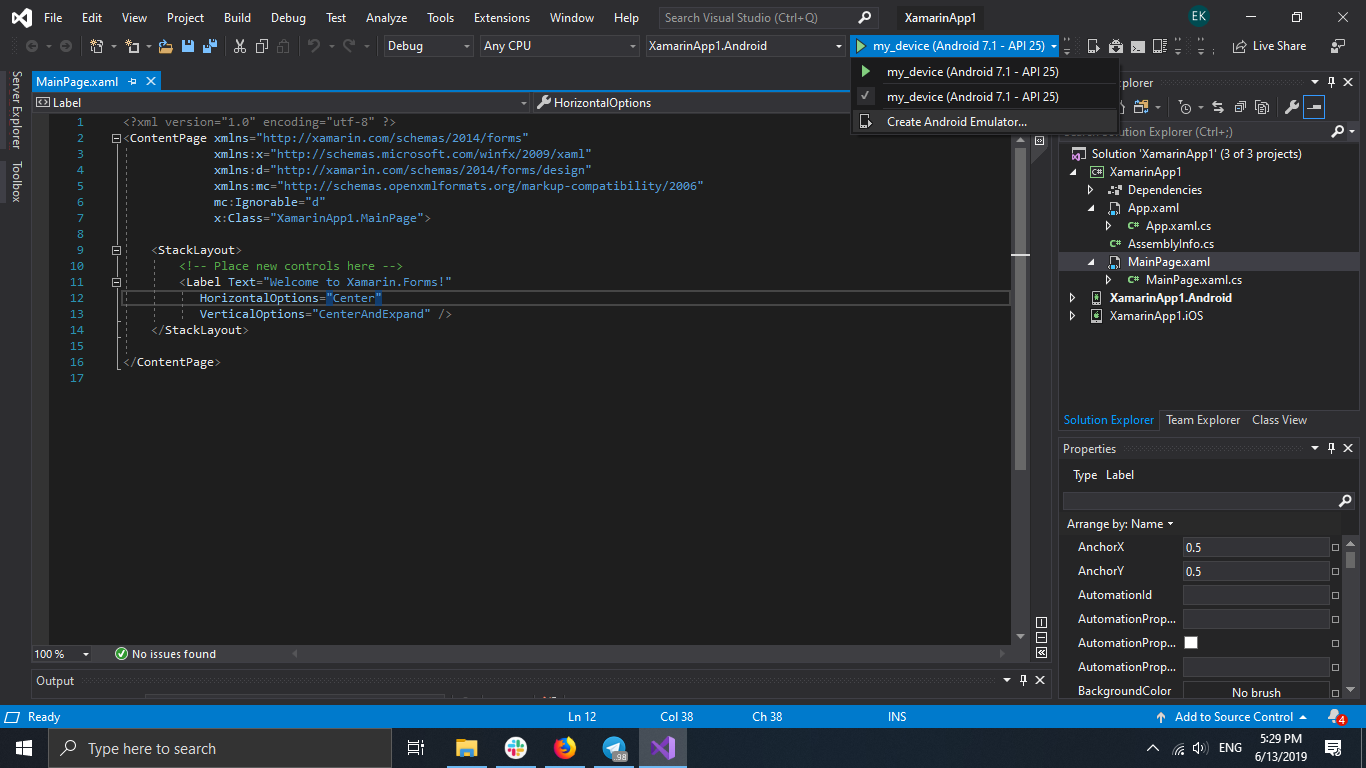


Рисунок 3 – выбор эмулятора

Откроется меню добавления устройства. Можно выбрать готовый шаблон из списка, разрядность, версию ОС и многие другие параметры. Не указывайте высокие характеристики, если используется слабый компьютер.

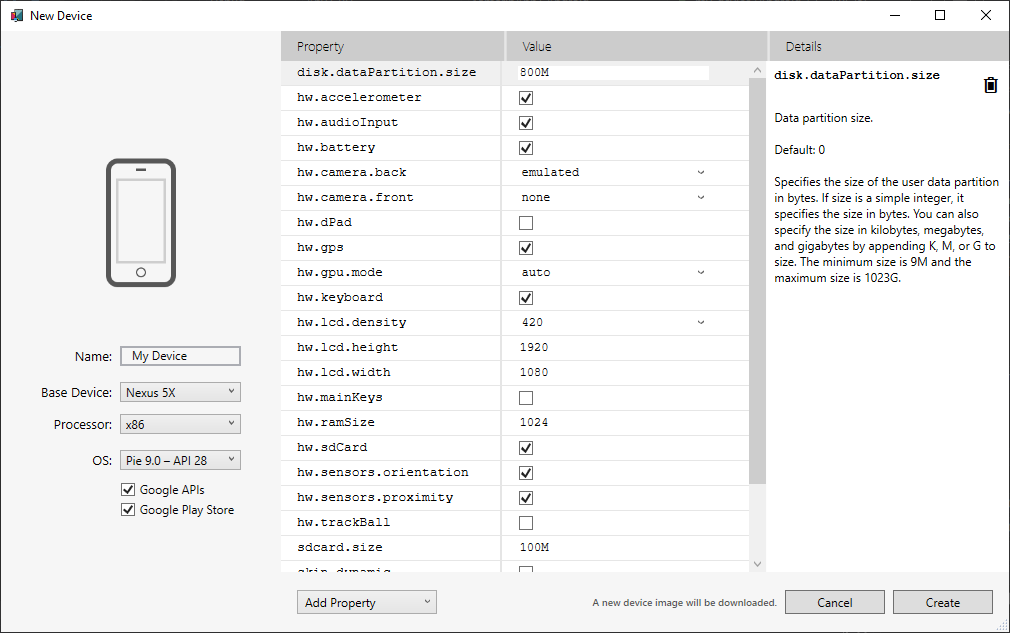


Рисунок 4 – меню добавления устройства

После создания устройства выберите его в списке и нажмите Start:

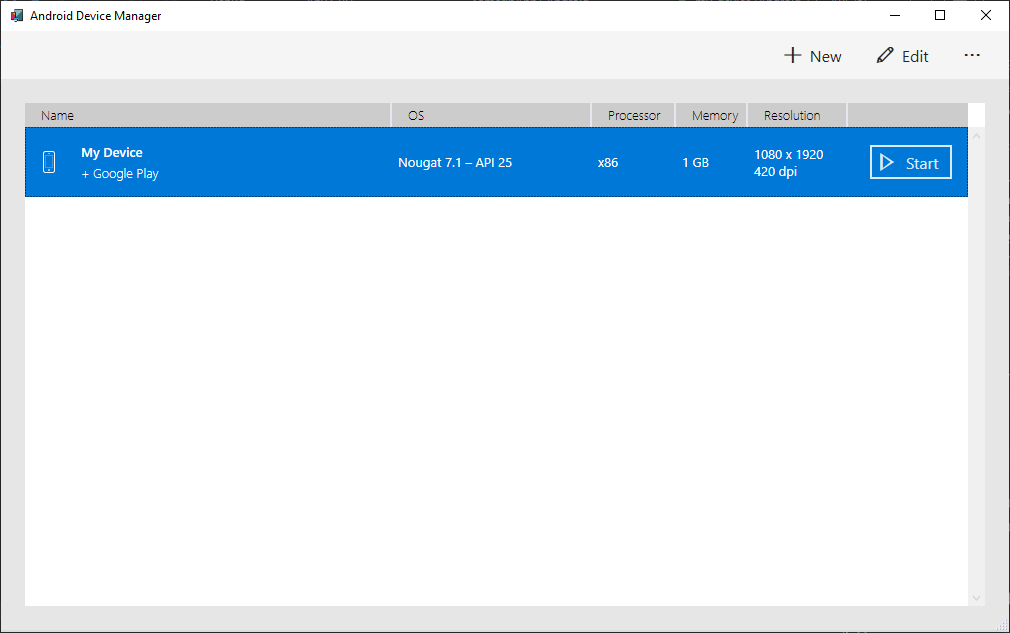


Рисунок 5 – запуск эмулятора

Эмулятор начнет скачиваться. После установки его можно будет запустить, чтобы отлаживать приложения.

На устройстве откроется приложение, которое было создано по шаблону:



Рисунок 6 - запущенное приложение

## **Введение в язык XAML**

XAML (eXtensible Application Markup Language) - язык разметки, используемый для инициализации объектов в технологиях на платформе .NET. Применительно к WPF (а также к Silverlight) данный язык используется прежде всего для создания пользовательского интерфейса декларативным путем.

Мы можем создавать весь интерфейс и в коде C#, но более предпочтительным способом является его описание в XAML. Имеются некоторые преимущества:

* Возможность отделить графический интерфейс от логики приложения, благодаря чему над разными частями приложения могут относительно автономно работать разные специалисты: над интерфейсом - дизайнеры, над кодом логики - программисты.
* Компактность, понятность, код на XAML относительно легко поддерживать.

XAML представляет язык разметки на основе xml для создания объектов декларативным образом. Собственно поэтому при создании проекта уже по умолчанию в него добавляются два файла MainPage.xaml и MainPage.xaml.cs.

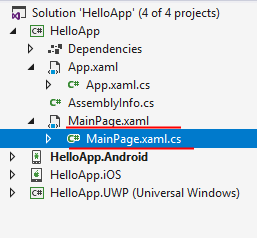


Рисунок 7 – структура проекта HelloApp

После создания проекта мы видим следующие файлы: Библиотека XamlSamples.NET Standard, XamlSamples.Android, XamlSamples.iOS  и универсальная платформа Windows решение XAML. UWP.

## **Элементы в XAML и их атрибуты**

XAML предлагает очень простую и ясную схему определения различных элементов и их свойств. Каждый элемент, как и любой элемент XML, должен иметь открытый и закрытый тег, как в случае с элементом ContentPage:

|  |  |
| --- | --- |
|  | <Label></Label> |

Либо элемент может иметь сокращенную форму с закрывающим слешем в конце, наподобие:

|  |  |
| --- | --- |
|  | <Label /> |
|  |  |

Каждый элемент в XAML представляет объект определенного класса C#, а атрибуты элементов соотносятся со свойствами этих классом. Например, элемент ContentPage фактически будет представлять объект одноименного класса ContentPage, а элемент Label - объект класса Label.

Каждый элемент в xaml можно сопоставить определенному классу в C#, а каждый атрибут элемента - свойству этого класса. Однако если свойства классов могут принимать значения различных типов: string, double, int и т.д. То в XAML атрибуты элементов имеют текстовые значения. Например, возьмем один из определенных на странице элементов

<Label Text="Welcome to Xamarin.Forms!" HorizontalTextAlignment="Center" TextColor="White" FontSize="36" />

Свойство Text класса Label в качестве значение принимает строку, поэтому здесь никаких преобразований не потребуется. Но, к примеру, свойство HorizontalTextAlignment, которое устанавливает выравнивание текста по горизонтали, в качестве значения принимает одну из констант перечисления TextAlignment, например, TextAlignment.Center. Однако в коде XAML мы присваиваем просто значение Center. Дело в том, что в Xamarin Forms действуют конвертеры типов, которые позволяют преобразовать от одного типа к другому.

## **Сложные свойства**

Кроме простых свойств, которые могут устанавливаться в с помощью простой строки, например, <Label Text="Welcome to Xamarin.Forms!" />, в XAML могут применяться сложные или комплексные свойства. В таких случаях свойство класса может принимать в качестве значения какой-нибудь сложный объект, который в свою очередь имеет некоторый набор свойств. Сложные свойства определяются в формате:

<Элемент.Свойство>

    <Сложный\_объект />

</Элемент.Свойство>

Например, возьмем другой, определенный на странице элемент Label:

<Label FontSize="16" Padding="30,24,30,0">

    <Label.FormattedText>

        <FormattedString>

            <FormattedString.Spans>

                <Span Text="Learn more at "/>

                <Span Text="<https://aka.ms/xamarin-quickstart>" FontAttributes="Bold"/>

            </FormattedString.Spans>

        </FormattedString>

    </Label.FormattedText>

</Label>

Свойство FormattedText представляет форматированный текст (текст со сложным оформлением) и является сложным свойством. В качестве значения оно принимает объект FormattedString, который передается в качестве вложенного объекта:

<FormattedString>

    <FormattedString.Spans>

        <Span Text="Learn more at "/>

        <Span Text="<https://aka.ms/xamarin-quickstart>" FontAttributes="Bold"/>

    </FormattedString.Spans>

</FormattedString>

У объекта FormattedString, в свою очередь, имеет свойство Spans, которое также является сложным свойством и которое представляет набор элементов Span - отдельных кусочков текста. А у каждого элемента Span устанавливается атрибут Text, который задает выводимый текст.

## **Взаимодействие с кодом c#**

Определим на станице MainPage.xaml какое-либо примитивное содержание, например, кнопку:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?><ContentPage xmlns="<http://xamarin.com/schemas/2014/forms>"               xmlns:x="<http://schemas.microsoft.com/winfx/2009/xaml>"               x:Class="HelloApp.MainPage">      <Button x:Name="button1" Text="Нажми на кнопку" Clicked="Button\_Click" />    </ContentPage> |

В данном случае определяется кнопка, которая занимает все пространство страницы MainPage. Свойства элементов определяются в виде атрибутов, например, Text="Оп. Кнопка нажата!!!". События также определяются как атрибуты. Например, с помощью атрибута Clicked устанавливается обработчик для события нажатия: Clicked="Button\_Click". И чтобы определить обработчик, перейдем в файл MainPage.xaml.cs и определим в классе MainPage следующий метод:

|  |  |
| --- | --- |
|  | public partial class MainPage : ContentPage  {      public MainPage()      {          InitializeComponent();      }        private void Button\_Click(object sender, EventArgs e)      {          button1.Text = "Оп. Кнопка нажата!!";      }  } |

С помощью атрибута x:Name элементу назначается имя. При компиляции приложения будет создаваться приватная переменная с этим именем. Через это имя в файле отделенного кода C# мы сможем ссылать на этот элемент, точнее на объект, который представляет в файле кода данный элемент. В частности, здесь кнопке назначено имя "button1". Соответственно в файле кода мы можем обратиться к свойствам и методам кнопки через это имя. В итоге по нажатию на кнопку у нее изменится текст.

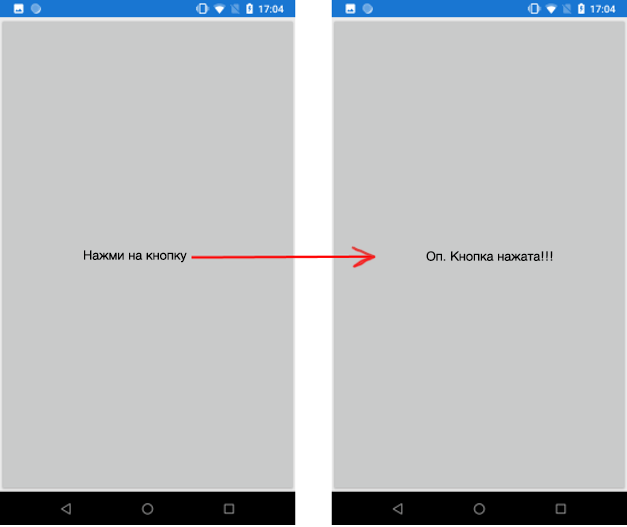


Рисунок 8 – выполненное приложение

## **Компоненты**

## **Кнопки**

Для произведения какого-либо действия нам понадобятся кнопки, представленные классом Button. Создадим кнопку в коде:

using Xamarin.Forms;

namespace HelloApp

{

    class MainPage : ContentPage

    {

        public MainPage()

        {

            StackLayout stackLayout = new StackLayout();

            Button button = new Button

            {

                Text = "Нажми!",

                FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Button)),

                BorderWidth = 1,

                HorizontalOptions = LayoutOptions.Center,

                VerticalOptions = LayoutOptions.CenterAndExpand

            };

            button.Clicked += OnButtonClicked;

            stackLayout.Children.Add(button);

            this.Content = stackLayout;

        }

        private void OnButtonClicked(object sender, System.EventArgs e)

        {

            Button button = (Button)sender;

            button.Text = "Нажато!";

            button.BackgroundColor = Color.Red;

        }

    }

}

Для кнопки можно задать обработчик нажатия для события Clicked. В данном случае по нажатию просто изменяем текст и цвет фона кнопки.

Обрботчик во многим аналогичен стандартным обработчикам в Windows Forms. Он принимает два параметра: объект типа object (источника события) и System.EventArgs (аргумент события, хранящий некоторую дополнительную информацию).

Анлогичная кнопка в xaml:

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="<http://xamarin.com/schemas/2014/forms>"

             xmlns:x="<http://schemas.microsoft.com/winfx/2009/xaml>"

             x:Class="HelloApp.MainPage">

  <StackLayout>

    <Button Text = "Нажми!" FontSize="Large" BorderWidth="1"

        HorizontalOptions="Center" VerticalOptions="CenterAndExpand"

        Clicked="OnButtonClicked" />

  </StackLayout>

</ContentPage>

И тогда в файле отделенного кода MainPage.xaml.cs надо прописать обработчик OnButtonClicked:

using Xamarin.Forms;

namespace HelloApp

{

    public partial class MainPage : ContentPage

    {

        public MainPage()

        {

            InitializeComponent();

        }

        private void OnButtonClicked(object sender, System.EventArgs e)

        {

            Button button = (Button)sender;

            button.Text = "Нажато!";

            button.BackgroundColor = Color.Red;

        }

    }

}

using Xamarin.Forms;

namespace HelloApp

{

    public partial class MainPage : ContentPage

    {

        public MainPage()

        {

            InitializeComponent();

        }

        private void OnButtonClicked(object sender, System.EventArgs e)

        {

            Button button = (Button)sender;

            button.Text = "Стать студентом!";

            button.BackgroundColor = Color.Red;

        }

    }

}



Рисунок 9 – создание кнопки

Кнопка имеет много различных свойств, из которых следует выделить следующие:

* FontFamily: шрифт, который используется для текста на кнопке
* FontSize: размер текста на кнопке
* FontAttributes: выделение жирным или курсивом текста на кнопке
* TextColor: цвет шрифта
* BorderColor: цвет границы
* BorderWidth: ширина границы
* BorderRadius: радиус границы
* Image: позволяет задать изображение на кнопке

В Xamarin Forms текстовые поля представлены несколькими классами:

* Label: текстовая метка для вывода текста
* Entry: однострочное текстовое поле
* Editor: многострочное текстовое поле

## **Label**

Label представляет обычную текстовую метку, которая выводит информацию с помощью свойства Text. Label удобен для создания заголоков и меток к элементам ввода.

Label довольно простой элемент, однако даже с ним могут возникать некоторые сложности. Например, кнопки определяют событие нажатия, которое мы можем обработать. У Label подобного события нет. Но что делать, если мы вдруг хотим обрабатывать нажатие на заголовок? В этом случае мы можем воспользоваться специальным классом TapGestureRecognizer, который позволяет обрабатывать нажатия:

public class MainPage : ContentPage

{

    public MainPage()

    {

        Label label = new Label

        {

            VerticalTextAlignment = TextAlignment.Center,

            HorizontalTextAlignment = TextAlignment.Center,

            Text = "Welcome to Xamarin Forms!",

            FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label))

        };

        TapGestureRecognizer tapGesture = new TapGestureRecognizer

        {

            NumberOfTapsRequired = 2

        };

        int count = 0;  // счетчик нажатий

        tapGesture.Tapped += (s, e) =>

        {

            count++;

            if (count % 2 == 0)

            {

                label.BackgroundColor = Color.Black;

                label.TextColor = Color.White;

            }

            else

            {

                label.TextColor = Color.Black;

                label.BackgroundColor = Color.White;

            }

            label.Text = $"Вы нажали {count} раз";

        };

        label.GestureRecognizers.Add(tapGesture);

        Content = label;

    }

}

TapGestureRecognizer представляет специальный класс, который позволяет распознать нажатия. С помощью свойства NumberOfTapsRequired мы можем установить, сколько нажатий необходимо. В данном случае мы устанавливаем двойное нажатие. И именно по двойному нажатию будет генерироваться событие Tapped, в обработчике которого мы будем переключать цвет текста и фона метки.

И чтобы связать объект TapGestureRecognizer с меткой, необходимо добавить этот объект в коллекцию GestureRecognizers:

|  |  |
| --- | --- |
|  | label.GestureRecognizers.Add(tapGesture); |
|  |

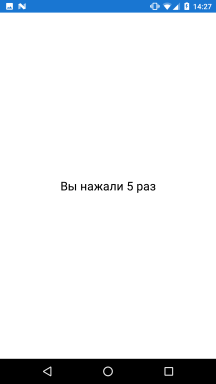


Рисунок 10 – создание label

## **Текстовое поле Entry**

Entry представляет текстовое поле для ввода однострочной информации:

class MainPage : ContentPage

{

    Label textLabel;

    Entry loginEntry, passwordEntry;

    public MainPage()

    {

        StackLayout stackLayout = new StackLayout();

        loginEntry = new Entry { Placeholder = "Login" };

        loginEntry.TextChanged+=loginEntry\_TextChanged;

        passwordEntry = new Entry

        {

            Placeholder = "Password",

            IsPassword = true

        };

        textLabel = new Label { FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)) };

        stackLayout.Children.Add(loginEntry);

        stackLayout.Children.Add(passwordEntry);

        stackLayout.Children.Add(textLabel);

        this.Content = stackLayout;

    }

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

    {

        textLabel.Text = loginEntry.Text;

    }

}

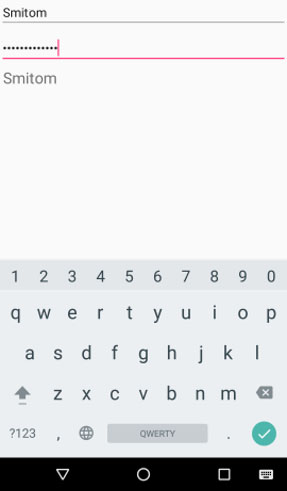


Рисунок 11 – создание entry

Обработать ввод пользователя мы можем с помощью обработки события TextChanged

Аналогичный пример в xaml:

<StackLayout>

    <Entry x:Name="loginEntry" TextChanged="loginEntry\_TextChanged" />

    <Entry x:Name="passwordEntry"  Placeholder = "Password" IsPassword ="True" />

    <Label x:Name="textLabel" FontSize="Large" />

</StackLayout>

Основные свойства элемента Entry, которые мы можем использовать:

* Text: введенный в поле текст
* TextColor: цвет вводимого текста
* Placeholder: текст-заменитель, который отображается в поле до ввода и исчезает при начале печати
* IsPassword: при значении true указывает, то поле будет предназначено для ввода пароля, и поэтому все вводимые символы будут заменяться звездочкой

Также класс Entry определяет два события:

* TextChanged: возникает при вводе символов в поле
* Completed: возникает при завершении ввода

### **Создание калькулятора**

Теперь создадим первое приложение. Перейдем к файлу MainPage.xaml и напишем следующий код.

<Grid Padding="20">

    <Grid.RowDefinitions>

        <RowDefinition Height="50" />

        <RowDefinition Height="50" />

        <RowDefinition Height="50" />

        <RowDefinition Height="50" />

        <RowDefinition Height="50" />

        <RowDefinition Height="50" />

    </Grid.RowDefinitions>

    <Grid.ColumnDefinitions>

        <ColumnDefinition/>

        <ColumnDefinition/>

    </Grid.ColumnDefinitions>

<Label x:Name="Labelnum1" Text="Num1:" Grid.Row="0" Grid.Column="0" />

<Entry x:Name="num1" Placeholder="Enter num1:" Grid.Row="0" Grid.Column="1" />

<Label x:Name="Labelnum2" Text="num2:" Grid.Row="1" Grid.Column="0" />

<Entry x:Name="num2" Placeholder="Enter num:" Grid.Row="1" Grid.Column="1" />

<Label x:Name="Labelres" Text="Result is:" Grid.Row="2" Grid.Column="0" />

<Label x:Name="res" Grid.Row="2" Grid.Column="1" />

<Button x:Name="BtnAdd" Text="Addition" Grid.Row="3" Grid.Column="0" Clicked="BtnAdd\_Clicked" />

<Button x:Name="BtnSub" Text="Substraction" Grid.Row="3" Grid.Column="1" Clicked="BtnSub\_Clicked" />

<Button x:Name="BtnMul" Text="Multiplication" Grid.Row="4" Grid.Column="0" Clicked="BtnMul\_Clicked" />

<Button x:Name="BtnDiv" Text="Division" Grid.Row="4" Grid.Column="1" Clicked="BtnDiv\_Clicked" />

<Button x:Name="BtnRem" Text="Remainder" Grid.Row="5" Grid.Column="0" Clicked="BtnRem\_Clicked" />

<Button x:Name="BtnClr" Text="Clear" Grid.Row="5" Grid.Column="1" Clicked="BtnClr\_Clicked" />

</Grid>

В приведенном выше коде XAML мы создаем макет сетки, записывая сетку в теги. Затем мы применяем padding = 20 к макету; . т.е., когда мы создаем какой-либо элемент в макете сетки, текст этих элементов перемещается на значение 20 к его элементу. Далее мы создаем в макете строки и столбцы.

Grid.RowDefinitions используется для объявления, что мы хотим создать строку, и аналогично Grid.ColumnDefinitions используется для создания столбцов в макете. Теперь мы используем RowDefinition для создания строки, т.е. если мы хотим создать 4 строки, мы объявляем RowDefinition 4 раза. Точно так же, если мы хотим создать 10 столбцов, мы объявляем ColumnDefinition 10 раз. Мы можем регулировать высоту строк и ширину столбцов.

После объявления необходимых строк и столбцов мы объявляем необходимые элементы для приложения. Теперь мы хотим разместить эти элементы в требуемых строках и столбцах, затем мы определяем необходимую строку и столбец с помощью Grid.Row используется для отображения Row и Grid.Column используется для отображения столбца.

Теперь откройте ваш MainPage.xaml.cs для написания кода C# вашего приложения.

private void BtnAdd\_Clicked(object sender, EventArgs e) {

    double a = Double.Parse(num1.Text);

    double b = Double.Parse(num2.Text);

    res.Text = (a + b).ToString();

}

private void BtnSub\_Clicked(object sender, EventArgs e) {

    double a = Double.Parse(num1.Text);

    double b = Double.Parse(num2.Text);

    res.Text = (a - b).ToString();

}

private void BtnMul\_Clicked(object sender, EventArgs e) {

    double a = Double.Parse(num1.Text);

    double b = Double.Parse(num2.Text);

    res.Text = (a \* b).ToString();

}

private void BtnDiv\_Clicked(object sender, EventArgs e) {

    double a = Double.Parse(num1.Text);

    double b = Double.Parse(num2.Text);

    res.Text = (a / b).ToString();

}

private void BtnRem\_Clicked(object sender, EventArgs e) {

    double a = Double.Parse(num1.Text);

    double b = Double.Parse(num2.Text);

    res.Text = (a % b).ToString();

}

private void BtnClr\_Clicked(object sender, EventArgs e) {

    num1.Text = String.Empty;

    num2.Text = String.Empty;

    res.Text = String.Empty;

}

Затем в переносимом файле щелкните параметр сборки, чтобы создать приложение.

Затем выберите тип платформы, на которой вы хотите запустить свое приложение.

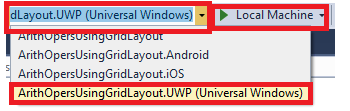


Рисунок 12 – выбор платформы для запуска

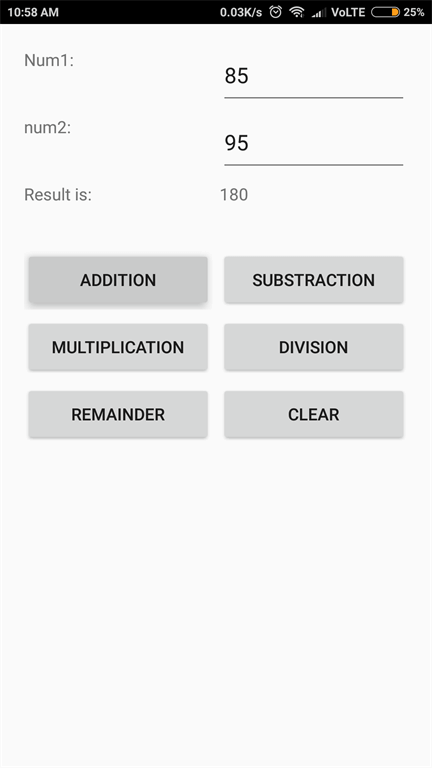


Рисунок 13 – интерфейс приложения «Калькулятор»

**Варианты заданий**

На основе написанного выше кода создайте приложение с использованием компонент XAML (указаны по ссылке ниже):

1. Конвертер физических величин. Предусмотреть ручной выбор величины, в которую нужно переводить данные.

2. Конвертер валют. Предусмотреть ручной выбор величины, в которую нужно переводить данные.

3. Конвертер температуры (F,C). Предусмотреть ручной выбор величины, в которую нужно переводить данные.

4. Добавьте в калькулятор возможность вычисления дополнительных математических функций (на ваше усмотрение), которые активируются при переключении Toggle компоненты (см. ссылку на компоненты)

5. Создать список из элементов (картинка+текст), которые можно удалять

6. Программа для расчёта корней квадратного уравнения

7. Теорема Пифагора. Подсчёт сторон треугольника

8. Расчёт значений нескольких функций. Пользователь вводит значение Х. И выбирает уравнение, для которого хочет посчитать данное значение.

9. Создать список элементов (картинка+текст), который можно редактировать.

10. Добавить функцию расчёта факториала числа в калькулятор.

11. Конвертер масс. Вводится число, нужно выбрать изначальную величину и конечную.

12. Расчёт периметра и площади треугольника. Пользователь с помощью переключателя сам выбирает, что рассчитывать

13. Добавить в калькулятор функции расчёта логарифма

14. Добавить в калькулятор возможно рассчитывать sin, cos, tg угла

15. Приложение для определения максимального и минимального числа среди списка чисел. Пользователь с помощью переключателя выбирает- найти максимальный или минимальный элемент

16. Дан список чисел. Пользователь в зависимости от нажатой кнопки увидит либо только четные, либо только нечетные числа.

17. Дан список чисел. Пользователь в зависимости от нажатой кнопки видит либо только отрицательные либо только положительные числа.

18. Конвертер длины. Пользователь выбирает конвертируемую и конечную величину.

19. Вводятся числа. С помощью переключателя выводить первый четный или последний нечетный элемент.

20. Вводятся числа. При нажатии на кнопку рассчитывается его среднее арифметическое

Ссылка на компоненты XAML:

**https://docs.microsoft.com/ru-ru/xamarin/xamarin-forms/xaml/xaml-controls**