# Лабораторная работа 8

Разработка мобильных приложений на языке C# при помощи платформы Xamarin.Forms

## Установка и настройка библиотеки Xamarin.Forms

Xamarin — это платформа с открытым исходным кодом, предназначенная для построения современных производительных приложений для iOS, Android и Windows с .NET. Платформа Xamarin представляет собой уровень абстракции, который обеспечивает управление взаимодействием между общим кодом и кодом базовой платформы. Xamarin выполняется в управляемой среде, которая реализует такие возможности, как выделение памяти и сборка мусора.

Благодаря Xamarin в среднем 90 % кода приложения может использоваться без изменений на разных платформах. С помощью этого шаблона разработчик может написать всю бизнес-логику на одном языке (или использовать существующий код приложения), но при этом получить характеристики производительности, оформление и поведение, характерные для каждой соответствующей платформы.

Как правило, при установке Visual Studio по умолчанию не производится установка всех необходимых библиотек, которые необходимы для разработки мобильных приложений. Поэтому их необходимо установить дополнительно.

Для этого используется установщик среды разработки - Visual Studio Installer. Его необходимо запустить из меню приложений (Пуск\Программы\...). В открывшемся окне выбрать пункт “Дополнительно\Изменить”

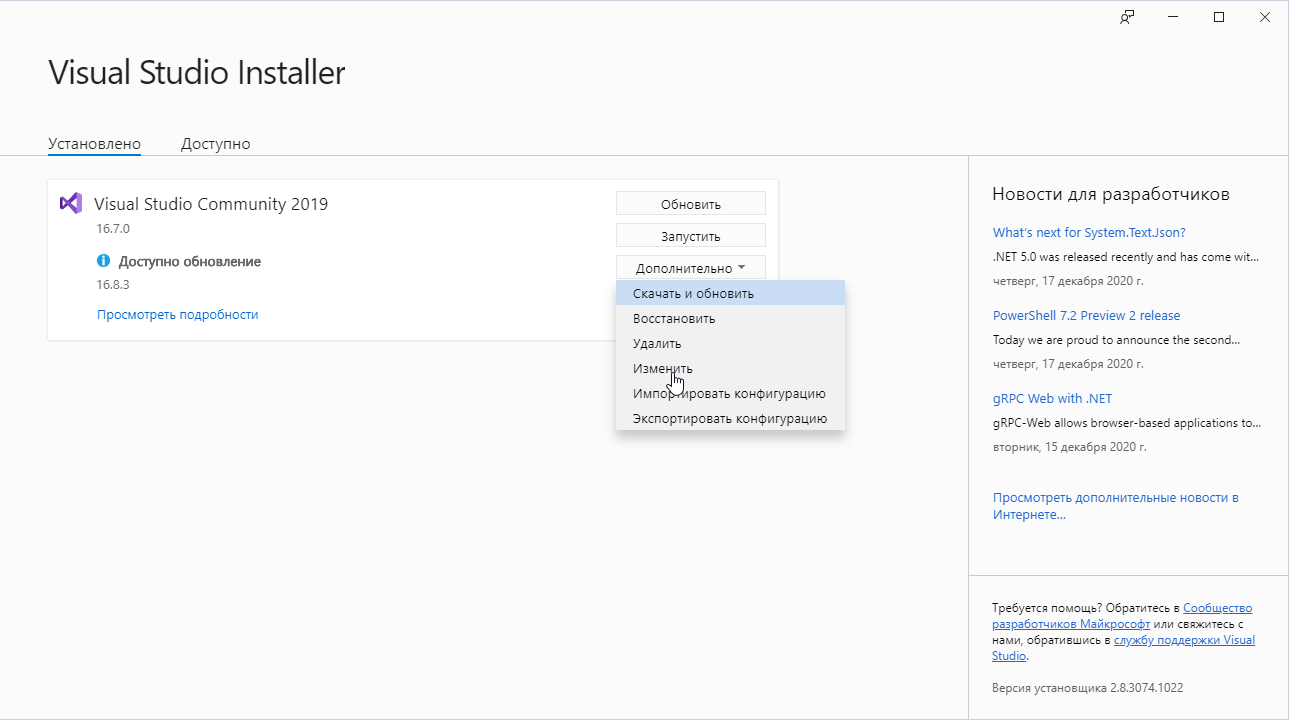


Рисунок 1 Установка библиотеки Xamarin

Затем отметить галочкой блок “Разработка мобильных приложений на .Net” и нажать кнопку “Установить при скачивании”

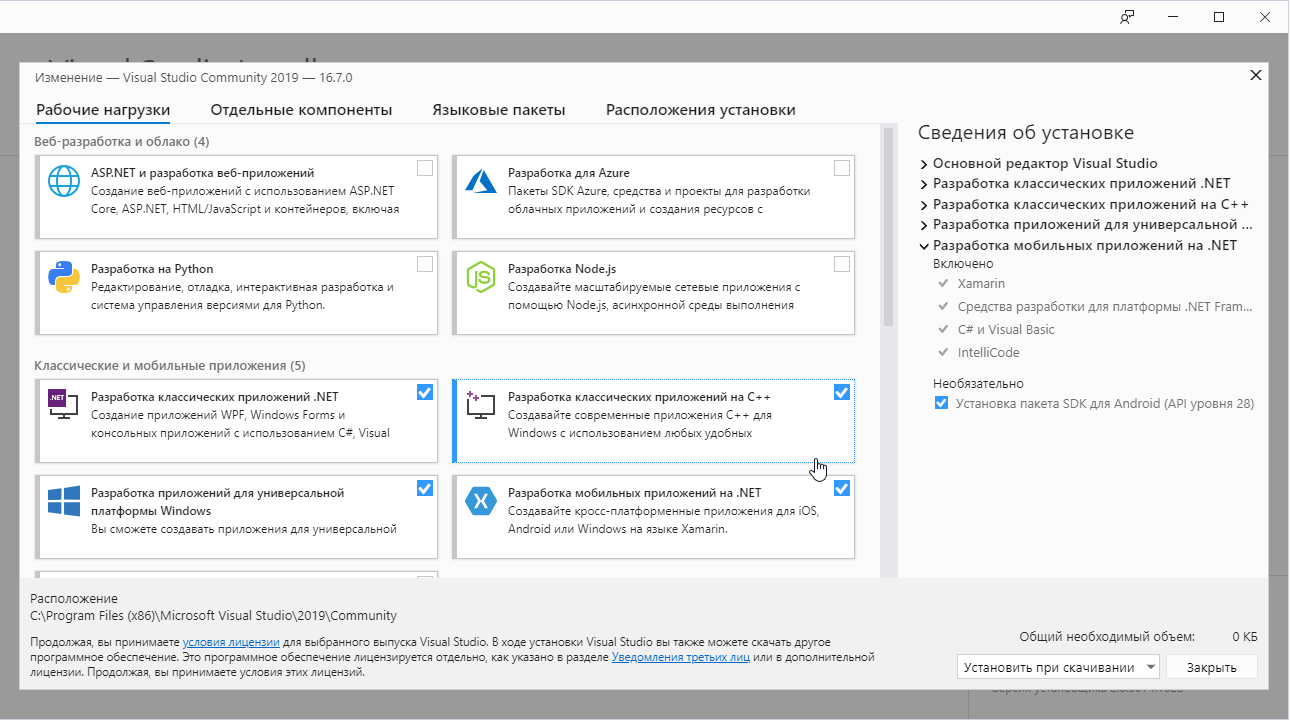


Рисунок 2 Окно выбора компонентов для установки

После этого нужно дождаться окончания установки и можно приступать к разработке.

## Создание проекта

Платформа Xamarin предлагает несколько видов проектов для разработки:

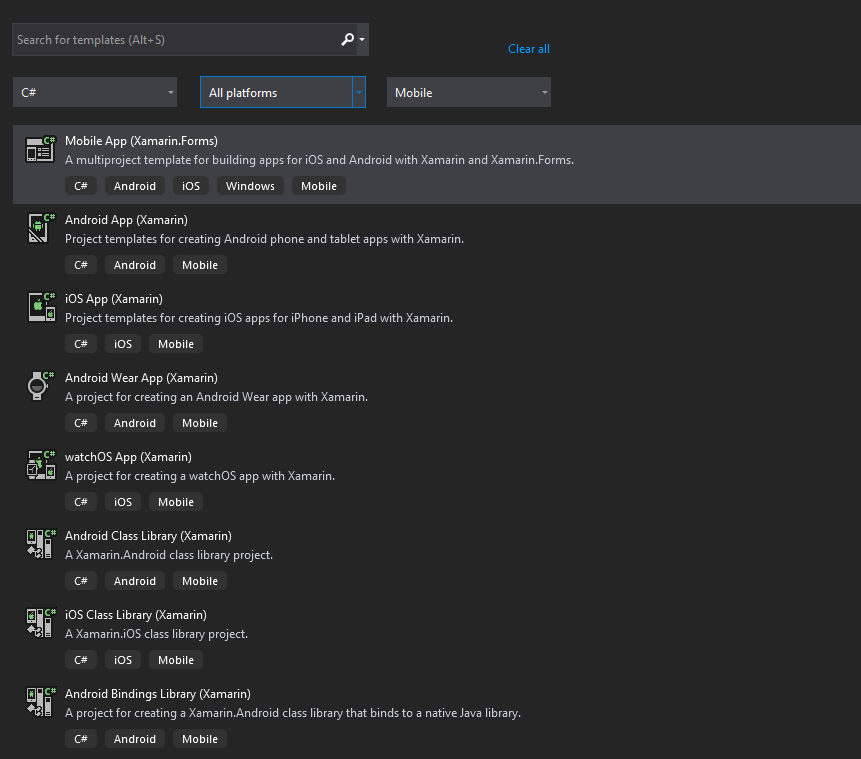


Рисунок 3 Шаблоны проектов Xamarin

Их можно разбить на две основных группы:

* Приложения, предназначенные для одной специфической платформы: Android, iOS, watchOS, tvOS и др.
* Универсальное приложение Xamarin. Forms

В этой работе мы будем использовать именно Xamarin.Forms, так как этот тип проекта позволяет разработать приложение, которое сможет полноценно работать под всеми основными видами мобильных платформ.

Поэтому выбираем в окне создания проекта “Mobile App (Xamarin.Forms)” и нажимаем кнопку Next, а на следующем экране Create.

Затем среда разработки предложит выбрать стартовый шаблон приложения и отметить галочками целевые платформы:

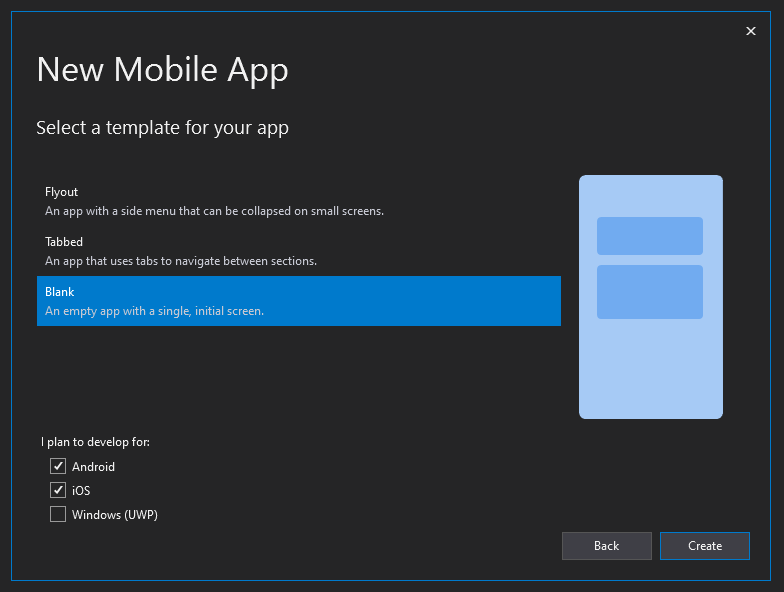


Рисунок 4 Окно выбора типа стартового шаблона Xamarin.Forms

Типы шаблонов Flyout и Tabbed могут оказаться сложными в самом начале работы, однако, они представляют правильно организованную структуру проекта, которая позволит создать поддерживаемое и расширяемое приложение.

Шаблон Blank в свою очередь представляет собой пустой проект с одним стартовым окном приложения и ничего лишнего. Поэтому в этой лабораторной работе выберем именно его и нажимаем Create.

После этого у вас откроется окно с исходным кодом проекта.

## Запуск и отладка проекта

Далее предполагаем, что программа собирается и отлаживается для Android

При разработке приложений под Android доступно два варианта запуска приложений во время отладки:

* Запуск на реальном смартфоне
* Запуск в эмуляторе

Так же возможен запуск приложения UWP, что позволяет быстро и удобно отладить логику, но не позволяет убедиться в том, что приложение будет вести себя на смартфоне именно так, как задумано.

Запуск на смартфоне является предпочтительным вариантом, поскольку запуск в эмуляторе требует значительных ресурсов ПК и места на жестком диске.

### Запуск и отладка приложения на реальном смартфоне

Для запуска приложений на смартфоне необходимо перевести его в режим разработчика и включить “Отладку по USB”. Так как этот процесс различается на разных версиях ОС, воспользуйтесь инструкцией по ссылке <https://androidp1.ru/otladka-po-usb/> или поиском в интернет.

Затем необходимо подключить смартфон к ПК и разрешить удаленную отладку.

После этого, ваш смартфон должен обнаружиться в Visual Studio в выпадающем списке целей отладки:

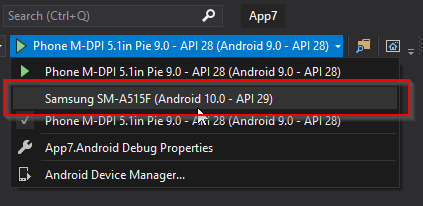


Рисунок 5 Выбор устройства для запуска отладки

И после выбора нужного пункта меню, вы сможете запустить приложение на нем:

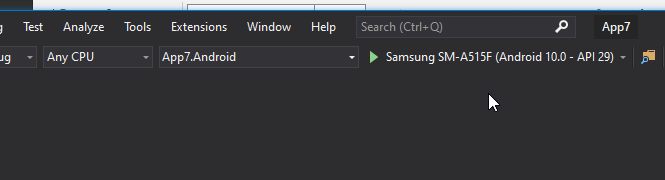


Рисунок 6 Смартфон Samsung A51 выбран в качестве цели отладки приложения

Выберите этот пункт меню и запустите приложение. Процесс сборки в зависимости от производительности ПК может занять от 10 секунд до пары минут, после чего приложение будет закачано на смартфон и запущено. По окончанию процесса, на вашем телефоне должно быть запущено приложение, которое выглядит следующим образом.

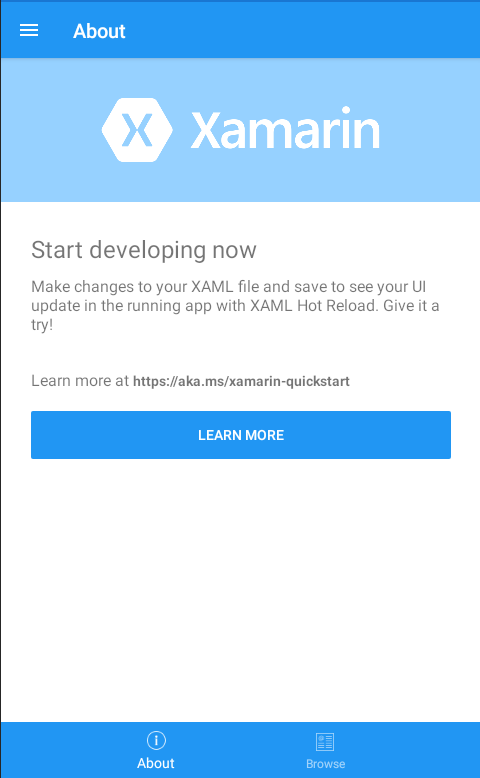


Рисунок 7 Внешний вид стартового приложения Xamarin.Forms

### Установка и настройка эмулятора Android

Если у вас нет возможности запустить отладку на смартфоне, вы можете использовать эмулятор Android для запуска и отладки приложений.

Для этого после открытия проекта необходимо запустить менеджер устройств Android. Для этого выберите пункт меню Tools\Android\Android Device Manager:

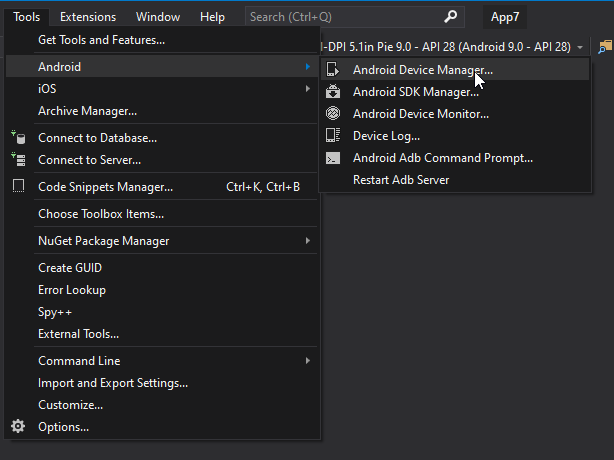


Рисунок 8 Открытие менеджера устройств Android

Далее в открывшемся окне Диспетчера устройств Android нажмите кнопку “Создать” и следуйте указаниям мастера создания виртуального устройства:

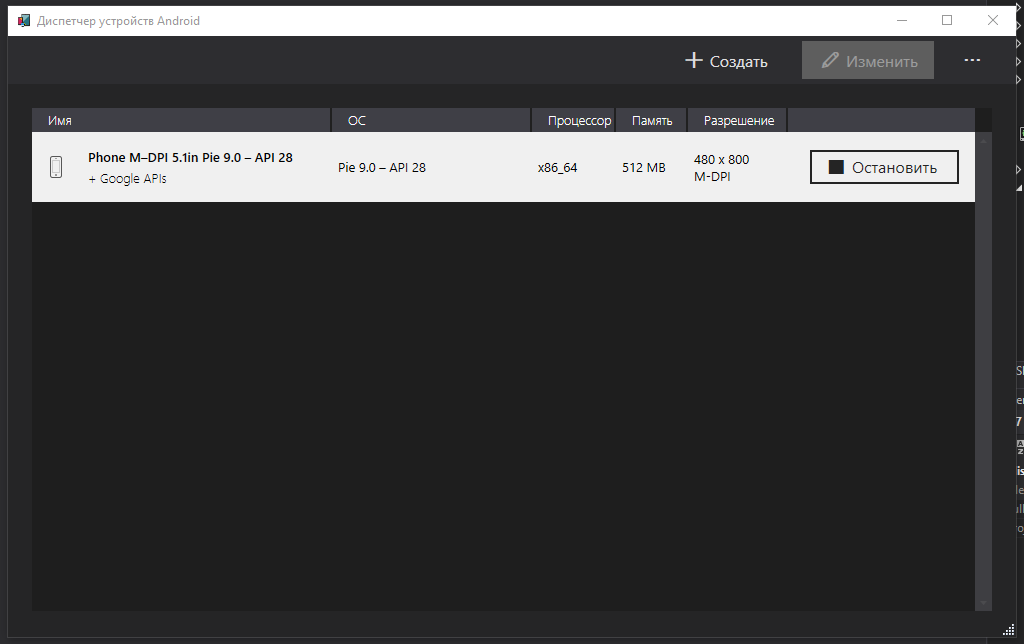


Рисунок 9 Запущенный эмулятор в окне менеджера устройств Android

Если в процессе создания или запуска устройства вы увидите сообщение о несоответствии версии, невозможности или ошибке запуска гипервизора Hyper-V, это будет означать то, что эмулятор все еще будет работать, но не будет использовать аппаратную виртуализацию, а значит будет работать очень медленно. Данная проблема как правило носит индивидуальный характер и не имеет общего решения, обратитесь ко мне, либо к интернету.

После запуска эмулятора и окончания установки сервисов андроид, у вас появится возможность отладки приложения в эмуляторе. Для этого необходимо нажать кнопку  на панели Visual Studio.

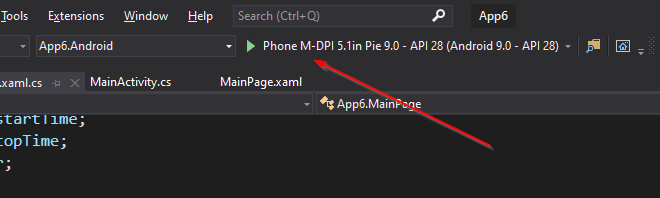


Рисунок 10 Эмулятор выбран в качестве цели отладки

После этого, в окне эмулятора будет запущено (возможно спустя несколько минут, в зависимости от производительности вашего ПК) приложение:

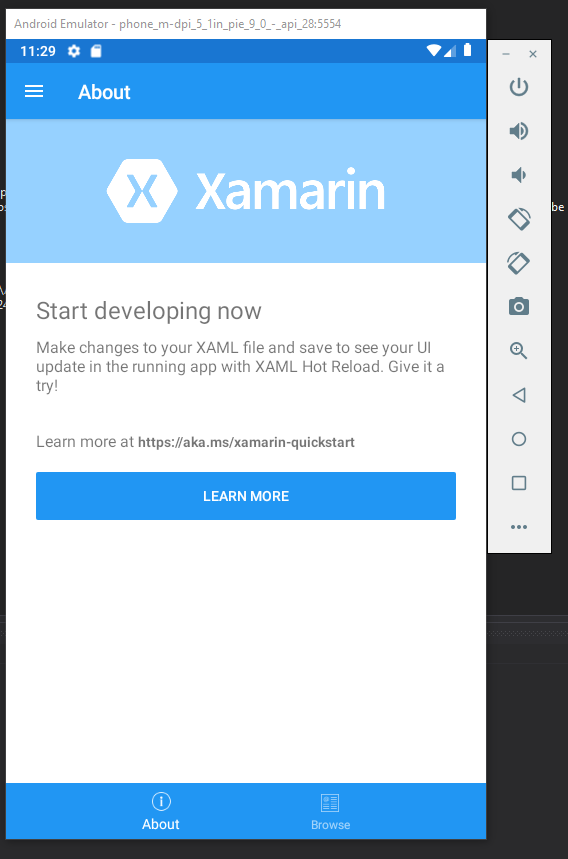


Рисунок 11 Приложение запущенное в эмуляторе

## Основы разработки мобильных приложений в Xamarin.Forms

Приложения в Xamarin.Forms являются кроссплатформенными и могут быть скомпилированы для нескольких мобильных платформ. Это приводит к некоторому усложнению структуры проекта таких приложений.

**В этой работе мы создадим приложение-секундомер для замера времени.**

После создания приложения с настройками по умолчанию в окне Solution Explorer будут отображены следующие проекты:

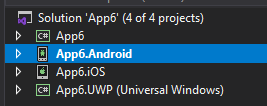


Рисунок 12 Состав решения приложения Xamarin.Forms

На этом скриншоте видно, что решение состоит из четырех проектов, при этом 3 из них имеют тип одной конкретной платформы:

* Android – для мобильных телефонов и планшетов на базе ОС Android
* iOS – для мобильных телефонов iPhone и планшетов iPad
* UWP – для ПК и планшетов на базе ОС Windows

Эти три проекта нужны для настройки приложения под соответствующую платформу.

Четвертый проект, для которого не указана платформа является общим и базовым для остальных. Именно в нем размещается весь основной код и логика работы приложения и именно в нем мы будем писать код.

### Структура проекта

Базовая структура общего проекта во многом аналогична структуре приложения на WPF и использует много общих элементов.

Если раскрыть базовый проект, то мы увидим следующие элементы:

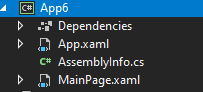


Рисунок 13 Список компонентов базового проекта Xamarin

* Группа Dependencies позволяет подключить к проекту дополнительные библиотеки
* App.xaml / App.xaml.cs – Главный класс приложения, с него начинается запуск программы
* AssemblyInfo.cs – задает настройки сборки приложения
* MainPage.xaml / MainPage.xaml.cs – Начальное окно приложения, в нем описывается что будет отображаться после запуска программы

Так как при создании проекта был выбран шаблон Blank, то среда создала лишь одно главное окно приложение с базовым заполнением. Если же при создании приложения выбрать шаблон Flyout, то будет создано большое количество файлов и классов для реализации шаблона проектирования MVVM о котором было рассказано на лекции. Но для упрощения работы, мы не будем его использовать.

За стартовый экран приложения отвечает класс MainPage, перейдем к нему.

### Определения оформления и внешнего вида программы

Библиотека Xamarin.Forms базируется на WPF и повторяет его структуру. Поэтому все знания, полученные при разработке графических приложений на WPF, будут актуальны и при разработке мобильных приложений. Однако, есть и несколько отличий от разработки приложений для ПК:

* В Xamarin.Forms отсутствует графический редактор форм. К сожалению, нет возможности графического размещения элементов и задания их свойств. Все редактирование внешнего вида должно производиться путем редактирования исходного кода XAML
* Некоторые компоненты имеют другие названия или свойства. Например, в WPF текст элемента Label записывался в свойство Content, а в Xamarin – в свойство текст. Однако, все основные элементы совпадают с WPF

В этой работе будем использовать следующие компоненты:

* Button – кнопка, позволяет выполнять действия при нажатии
* Label – текстовая подпись, позволяет выводить текст заданного размера, цвета и шрифта
* Grid – контейнер который позволяет размещать другие элементы в соответствии с описанными правилами в виде таблицы. Нужен для того чтобы разместить дочерние элементы в окне в заданном порядке
* StackLayout – контейнер, который позволяет размещать другие элементы в одну строку или столбец
* ScrollView – контейнер, который позволяет прокручивать содержимое пальцем, как ListBox

Начнем разработку программы с редактирования внешнего вида.

Хотя Xamarin.Forms и не имеет графического редактора как WPF, в Visual Studio 2019 **начиная с версии 16.8** появилась возможность редактирования свойств элементов через панель Properties и добавление графических элементов через панель Toolbox, а также предварительный просмотр внешнего вида.

Откроем файл MainPage.xaml и нажмем кнопку  в правом нижнем углу:

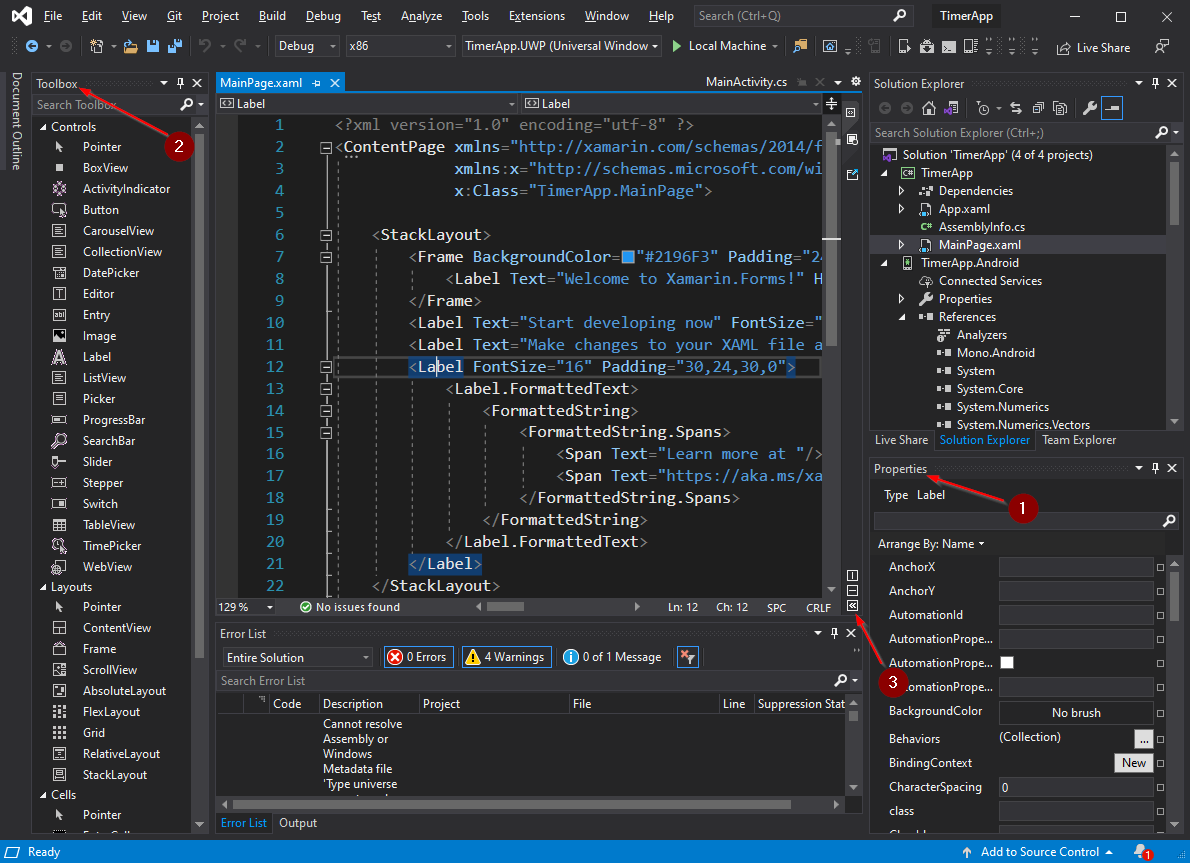
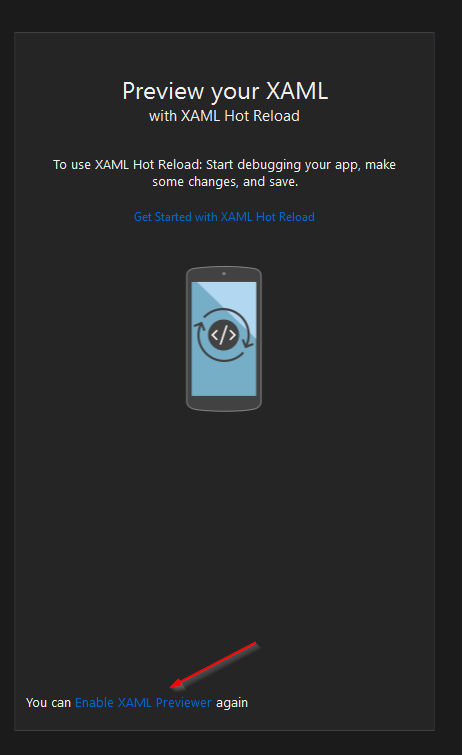


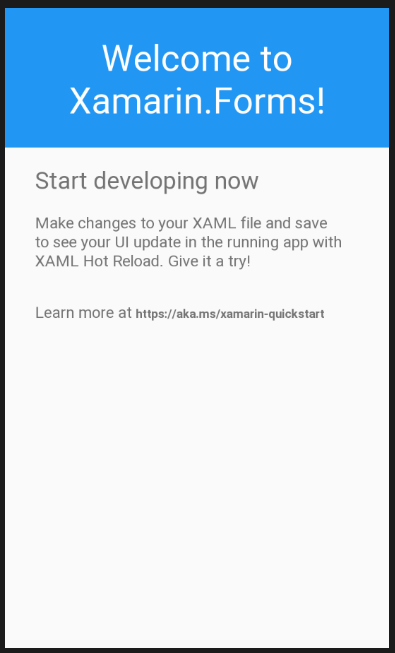
Рисунок 14 Элементы окна среды разработки XAML

1. Панель свойств для редактирования свойств того элемента, на котором стоит текстовый курсор в редакторе кода
2. Панель элементов для добавления новых элементов в окно
3. Кнопка для открытия окна предварительного просмотра внешнего вида

В открывшемся окне предварительного просмотра нужно нажать на текст “Enable XAML Previewer” для запуска просмотра:



После этого откроется следующее окно, в котором отображается то, как будет выглядеть приложение при запуске его на смартфоне:



Заменим разметку XAML внутри элемента <ContentPage> на следующий код:

<Grid x:Name="grid">

<Grid.ColumnDefinitions>

<!--Окно состоит из трех столбцов - Column

Это столбцы - время запуска таймера, значение таймера, и время окончания

при этом для заголовка приложения и кнопки эти столбцы объединены

ширина столбцов 25\* - 50\* - 25\* означает, что центральный столбец будет занимать

50% экрана, а левый и правый - по 25%

-->

<ColumnDefinition Width="25\*"/>

<ColumnDefinition Width="50\*"/>

<ColumnDefinition Width="25\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<!--И пять строк:

Заголовок

Кнопка

Подписи start-timer-stop

Значения текущего таймера

Список предыдущих замеров

Здесь высота \* у последней строки означает, что она будет занимать все доступное пространство,

которое не занято другими строками

-->

<RowDefinition Height="100"/>

<RowDefinition Height="100"/>

<RowDefinition Height="50"/>

<RowDefinition Height="40"/>

<RowDefinition Height="\*" />

</Grid.RowDefinitions>

<!--Заголовок приложения, свойство Row задает номер строки, в которой заголовок будет располагаться

А свойство ColumnSpan=3 - то, что заголовок занимает все три столбца

-->

<Frame Grid.Row="0" Grid.ColumnSpan="3" BackgroundColor="#2196F3" Padding="24" CornerRadius="0">

<Label Text="Timer App" HorizontalTextAlignment="Center" TextColor="White" FontSize="36"/>

</Frame>

<!--Кнопка START Строка 1, столбец 1 -->

<Button Grid.Row="1" Grid.Column="1" x:Name="btnStart" Text="Start" FontSize="30" MinimumHeightRequest="100" CornerRadius="50" />

<!-- Подписи к результатам измерений -->

<Label Grid.Column="0" Grid.Row="2" FontSize="20" HorizontalTextAlignment="Center" VerticalTextAlignment="End" Text="Start"/>

<Label Grid.Column="1" Grid.Row="2" FontSize="30" HorizontalTextAlignment="Center" Text="Timer"/>

<Label Grid.Column="2" Grid.Row="2" FontSize="20" HorizontalTextAlignment="Center" VerticalTextAlignment="End" Text="Stop"/>

<!-- Сюда будут выводиться измерения -->

<Label Grid.Column="0" Grid.Row="3" FontSize="20" HorizontalTextAlignment="Center" x:Name="lblStartTime"/>

<Label Grid.Column="1" Grid.Row="3" FontSize="30" HorizontalTextAlignment="Center" x:Name="lblTimer"/>

<Label Grid.Column="2" Grid.Row="3" FontSize="20" HorizontalTextAlignment="Center" x:Name="lblStopTime"/>

<!-- Последняя строка, здесь будет отображаться история измерений в прокручиващемся списке -->

<ScrollView Grid.Row="4" Grid.ColumnSpan="3" BackgroundColor="#ebebeb" Margin="0, 30, 0, 0">

<StackLayout x:Name="scrollLayout" Orientation="Vertical" VerticalOptions="Start"/>

</ScrollView>

</Grid>

При этом содержимое элемента ContentPage должно сохраниться без изменений, так как оно связано с индивидуальными параметрами вашего проекта.

Окно предпросмотра будет выглядеть следующим образом:



Рисунок 15 Главное окно приложение в режиме предпросмотра

Вы так же можете скомпилировать и запустить программу удобным для вас способом.

## Обработка событий действий пользователя

После того как мы отредактировали внешний вид, необходимо добавить обработку действий пользователя.

Для этого аналогично WPF нужно создать обработчик события клика по кнопке. Внутри элемента Button вводим Clicked и в выпавшем меню выбираем – New Event Handler

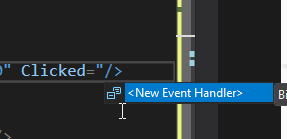


Рисунок 16 Создание обработчика события при помощи автодополнения

После чего Visual Studio создаст пустой обработчик события btnStart\_Clicked. Привязка события в MainPage.xaml:

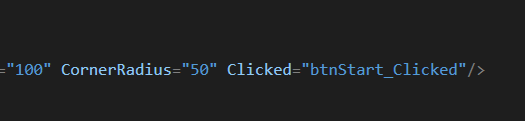


Рисунок 17 Привязка обработчика события в XAML

Созданный обработчик события в MainWindow.xaml.cs:

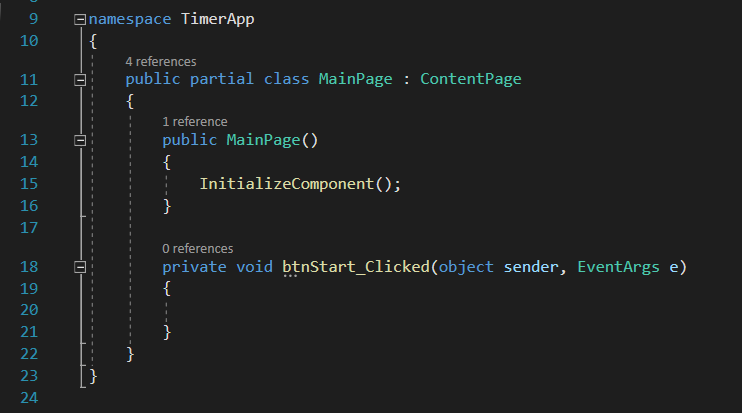


Рисунок 18 Обработчик события в исходном коде cs

В созданном обработчике события можно написать любой код, который должен вызываться при нажатии на кнопку мыши. Так как мы разрабатываем приложение-таймер, то необходимо выполнить следующие действия:

1. Получить текущее время, запомнить его и отобразить на экране приложения в lblStartTime
2. Запустить программный таймер, который будет с заданным интервалом обновлять текст в lblTimer на экране приложения
3. Установить признак того, что таймер запущен и поменять текст кнопки со Start на Stop

Затем, при повторном нажатии на кнопку необходимо выполнить:

1. Проверить признак того, что таймер запущен
2. Остановить программный таймер для того, чтобы текст lblTimer больше не менялся
3. Записать в lblStopTime текущее время

### Получение текущего времени

Для того, чтобы узнать текущее время необходимо использовать класс DateTime и его статическое свойство Now. Это свойство возвращает текущее время ПК или смартфона. Время запуска таймера необходимо сохранить в глобальной переменной типа DateTime, так как это значение понадобится для вычисления времени прошедшего с момента старта. Затем, для того чтобы его вывести в Label это время нужно преобразовать в строку с указанием формата преобразования. Так как нам не нужно отображать год месяц и день, то выполнить это можно при помощи следующей строки формата: «hh:mm:ss.fff». Тогда весь код целиком будет выглядеть следующим образом:

startTime = DateTime.Now;

lblStartTime.Text = startTime.ToString("hh:mm:ss.fff");

### Программный таймер

Наша программа-таймер должна с определенным (например, раз в 50мс) интервалом обновлять текст на экране, который будет содержать время прошедшее с момента запуска таймера. Для этого в Xamarin используется класс Device и его метод StartTimer. В качестве параметров этот метод принимает период сработки и название функции, которая будет вызываться с этим периодом. Эта функция должна возвращать значение типа bool, которое будет означать: нужно ли вызывать функцию в следующий раз. Проще говоря, до тех пор, пока функция возвращает true – таймер тикает, а как только она вернет false – таймер остановится.

Следовательно, необходимо объявить новую функцию, и передать ее название в метод StartTimer. А также объявить глобальную переменную isRunning которая будет определять запущен таймер или нет.

Для того, чтобы вычислить сколько времени прошло с момента старта таймера, нужно из текущего времени (DateTime.Now) вычесть время старта (startTime) и для вывода на lblTimer преобразовать его в строку.

Итоговый код:

bool isRunning = false;

DateTime startTime;

private void btnStart\_Clicked(object sender, EventArgs e)

{

if (isRunning)

{

isRunning = false;

}

else

{

startTime = DateTime.Now;

lblStartTime.Text = startTime.ToString("hh:mm:ss.fff");

isRunning = true;

Device.StartTimer(new TimeSpan(0, 0, 0, 0, 50), onTimer);

}

}

private bool onTimer()

{

lblTimer.Text = (DateTime.Now - startTime).ToString("hh\\:mm\\:ss\\.fff");

return isRunning;

}

Теперь программу уже можно запустить и попробовать понажимать на кнопку Start. Таймер будет запускаться и в главном окне программы будет отображаться прошедшее время, а по повторному нажатию на кнопку Start – останавливаться.

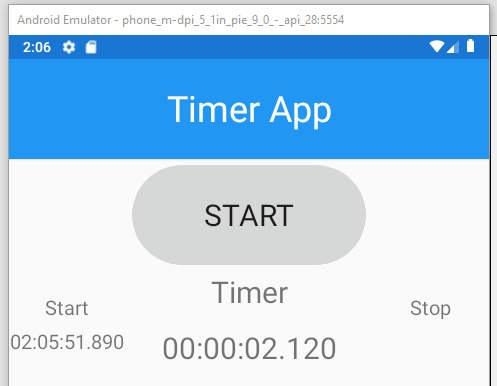


Рисунок 19 Пример запуска приложения

После этого нужно добавить остальные функции:

* Смена текста на кнопке в зависимости от того запущен таймер или нет (Start\Stop)
* Вывод времени останова таймера в lblStopTime
* При старте программы в конструкторе MainPage вывести в лейблах нулевые значения, чтобы они не были просто пустыми

Попробуйте выполнить это самостоятельно. А если не получится, то вот готовое решение:

bool isRunning = false;

DateTime startTime;

public MainPage()

{

InitializeComponent();

lblStartTime.Text = "00:00:00.000";

lblStopTime.Text = "00:00:00.000";

lblTimer.Text = "00:00:00.000";

}

private void btnStart\_Clicked(object sender, EventArgs e)

{

if (isRunning)

{

btnStart.Text = "Start";

lblStopTime.Text = DateTime.Now.ToString("hh:mm:ss.fff");

isRunning = false;

}

else

{

btnStart.Text = "Stop";

startTime = DateTime.Now;

lblStartTime.Text = startTime.ToString("hh:mm:ss.fff");

lblStopTime.Text = "";

isRunning = true;

Device.StartTimer(new TimeSpan(0, 0, 0, 0, 50), onTimer);

}

}

private bool onTimer()

{

lblTimer.Text = (DateTime.Now - startTime).ToString("hh\\:mm\\:ss\\.fff");

return isRunning;

}

### Отображение истории замеров

Так как при использовании таймера может пригодиться отображение истории замеров, а не только текущие показания, то давайте добавим эту функцию.

В XAML разметке мы уже добавили для этого компонент под названием ScrollView. Для того чтобы в нем отображалась история, мы должны при каждом останове таймера создавать набор из трех Label (start, timer, stop) и помещать их в scrollLayout. Вынесем это в отдельную функцию:

private void insertPreviousMeasurement(string start, string stop, string time)

{

Label label1 = new Label()

{

Text = start,

TextColor = Color.DarkGray,

FontSize = 18

};

Label label2 = new Label()

{

Text = time,

TextColor = Color.Gray,

FontSize = 25

};

Label label3 = new Label()

{

Text = stop,

TextColor = Color.DarkGray,

FontSize = 18

};

StackLayout stackLayout = new StackLayout()

{

Children = { label1, label2, label3 }

};

stackLayout.Orientation = StackOrientation.Horizontal;

stackLayout.Spacing = 20;

stackLayout.HorizontalOptions = new LayoutOptions(LayoutAlignment.Center, true);

stackLayout.VerticalOptions = new LayoutOptions(LayoutAlignment.Center, true);

scrollLayout.Children.Insert(0, stackLayout);

}

Перед добавлением в scrollLayout мы объединяем этот набор из трех ярлыков в одну группу путем помещения их в горизонтальный stackLayout.

Затем вызов этой функции нужно добавить в обработчик нажатия на кнопку.

### Полный код класса MainPage:

public partial class MainPage : ContentPage

{

bool isRunning = false;

DateTime startTime;

public MainPage()

{

InitializeComponent();

lblStartTime.Text = "00:00:00.000";

lblStopTime.Text = "00:00:00.000";

lblTimer.Text = "00:00:00.000";

}

private void btnStart\_Clicked(object sender, EventArgs e)

{

if (isRunning)

{

btnStart.Text = "Start";

lblStopTime.Text = DateTime.Now.ToString("hh:mm:ss.fff");

isRunning = false;

insertPreviousMeasurement(lblStartTime.Text, lblStopTime.Text, lblTimer.Text);

}

else

{

btnStart.Text = "Stop";

startTime = DateTime.Now;

lblStartTime.Text = startTime.ToString("hh:mm:ss.fff");

lblStopTime.Text = "";

isRunning = true;

Device.StartTimer(new TimeSpan(0, 0, 0, 0, 50), onTimer);

}

}

private bool onTimer()

{

lblTimer.Text = (DateTime.Now - startTime).ToString("hh\\:mm\\:ss\\.fff");

return isRunning;

}

private void insertPreviousMeasurement(string start, string stop, string time)

{

Label label1 = new Label()

{

Text = start,

TextColor = Color.DarkGray,

FontSize = 18

};

Label label2 = new Label()

{

Text = time,

TextColor = Color.Gray,

FontSize = 25

};

Label label3 = new Label()

{

Text = stop,

TextColor = Color.DarkGray,

FontSize = 18

};

StackLayout stackLayout = new StackLayout()

{

Children = { label1, label2, label3 }

};

stackLayout.Orientation = StackOrientation.Horizontal;

stackLayout.Spacing = 20;

stackLayout.HorizontalOptions = new LayoutOptions(LayoutAlignment.Center, true);

stackLayout.VerticalOptions = new LayoutOptions(LayoutAlignment.Center, true);

scrollLayout.Children.Insert(0, stackLayout);

}

}

## Сборка дистрибутива программы

Для того, чтобы написанную программу можно было распространять и устанавливать на смартфоны - необходимо подготовить так называемый дистрибутив или, по-другому, установочный файл. Для операционной системы Android дистрибутивы имеют формат APK.

Для создания дистрибутива необходимо перевести проект в режим Выпуска, выбрав профиль сборки Release:

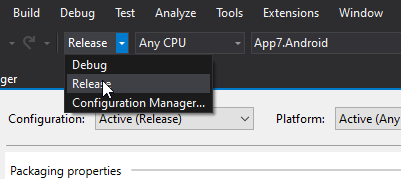


Рисунок 20 Переключение режима сборки

А затем щелкнуть правой кнопкой по проекту Android и выбрать пункт меню Archive:

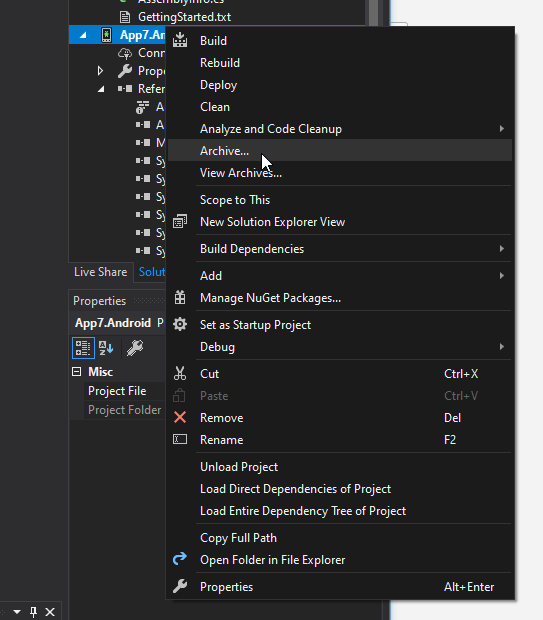


Рисунок 21 Запуск процесса архивирования

Это запустит процесс сборки APK:

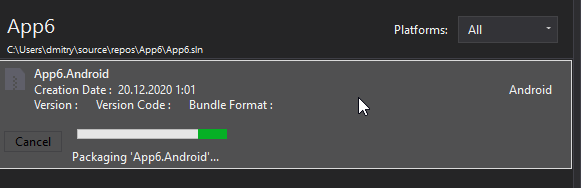


Рисунок 22 Идет архивация приложения

После того как архив программы собрался его необходимо подготовить к распространению. Для этого нужно нажать на кнопку Distribute.

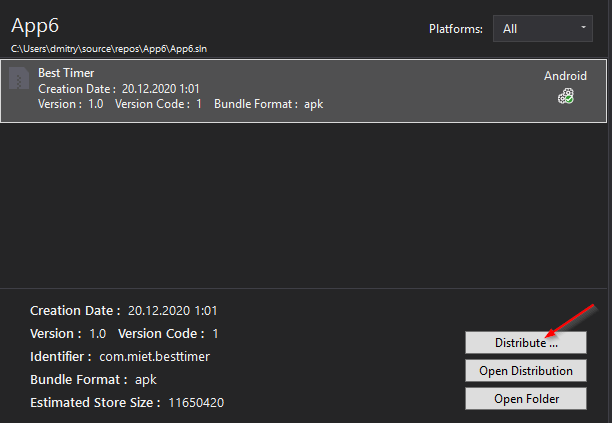


Рисунок 23 Подготовка приложения к распространению

В окне выбора типа распространения приложения будет доступно два варианта:

* AD-HOC – распространение приложения в виде APK файла
* Добавление в магазин приложений Google Play

Так как у нас еще нет учетной записи разработчика приложений Google Play, мы выберем первый вариант:

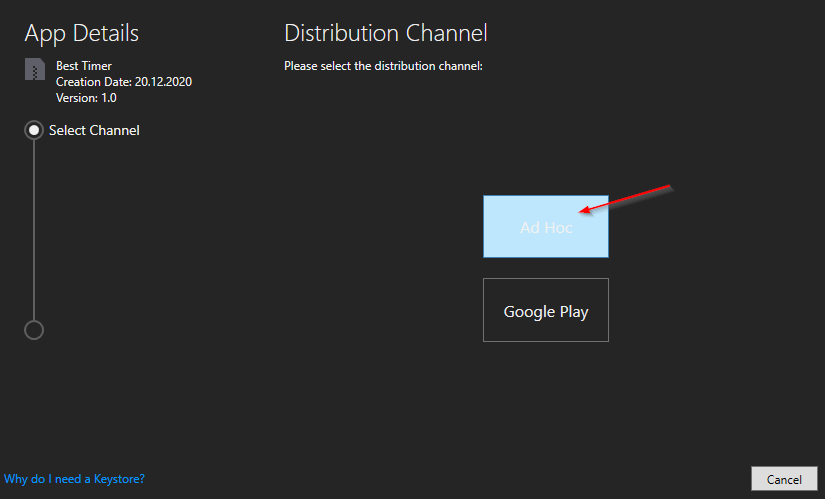


Рисунок 24 Выбор типа распространения приложения

Затем архив приложения необходимо заверить цифровой подписью. А для этого эту подпись нужно создать. Нажимаем на кнопку со знаком плюс:

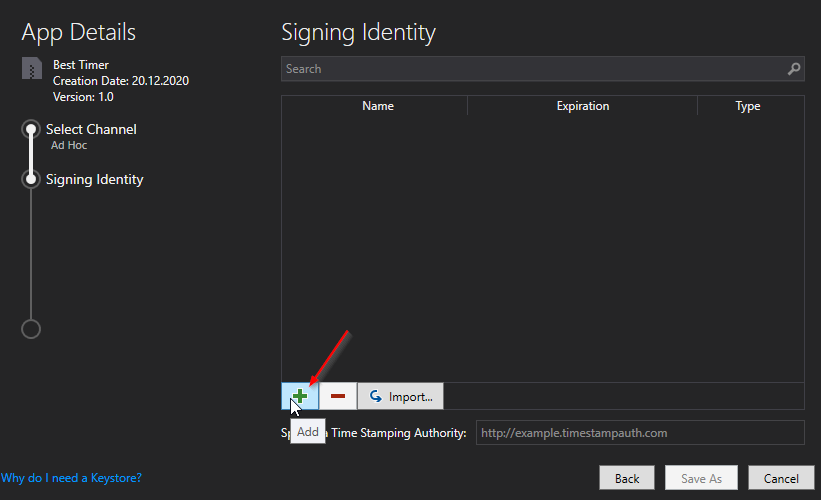


Рисунок 25 Создание цифровой подписи

И в следующем окне заполняем выделенные поля для создания нового хранилища цифровых подписей. Поля можно заполнить любыми значениями, но пароль нужно запомнить.

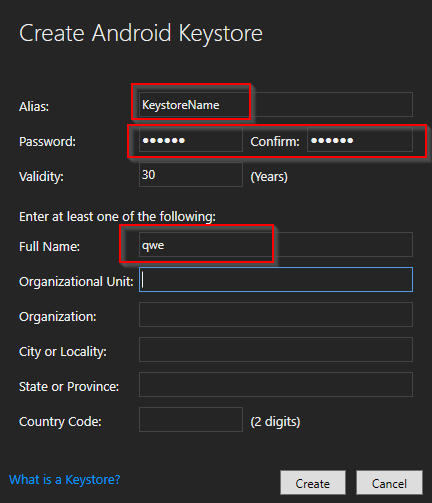


Рисунок 26 Окно создания цифровой подписи

После создания подписи выделяем ее в списке и нажимаем на кнопку Сохранить Как:

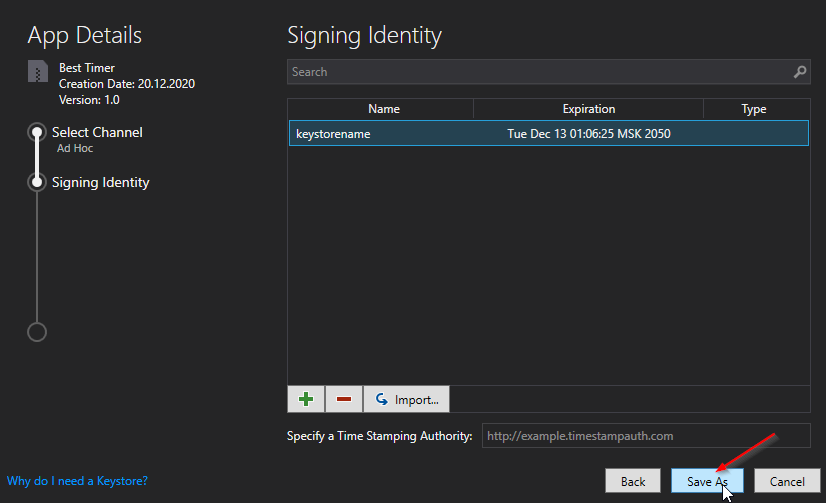


Рисунок 27 Почти все готово

Вводим пароль для цифровой подписи, созданной ранее:

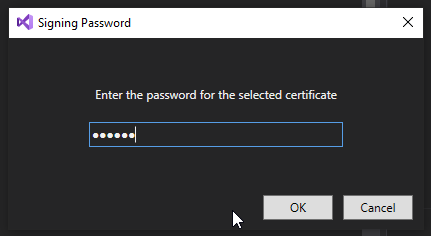


Рисунок 28 Ввод пароля

И получаем на выходе файл приложения APK который можно установить на любой смартфон.