Bachelor's thesis

Information Technology

Internet Technology

2016

Denis Anchugov

CROSS-PLATFORM MOBILE SOFTWARE DEVELOPMENT

Technology Overview and a Practical Example

TURUN AMMATTIKORKEAKOULU

TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR´S THESIS | ABSTRACT
TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Internet Technology

10 March 2016 | Number of pages: 58

Supervisor: Patric Granholm

Denis Anchugov

CROSS-PLATFORM MOBILE SOFTWARE DEVELOPMENT

The vast diversity of portable devices has increased the need for an easy and unified approach to build mobile software. The purpose of this work is to give an overview of technologies widely used to deliver cross-platform software solutions. This was achieved by studying two of the most common cross-platform frameworks – *Xamarin.Forms* and *Cordova*.

The first part of the work introduces the problem of mobile software development. The second part describes the underlying technology of the cross-platform solutions and compares them to the native application development in terms of benefits and drawbacks. The third part provides a practical example of building an app using the two cross-platform solutions, demonstrating the workflow of development and a toolset used.

KEYWORDS:

Mobile software development, cross-platform software, C#, JavaScript, Xamarin, Cordova, Ionic

CONTENTS

PICTURES

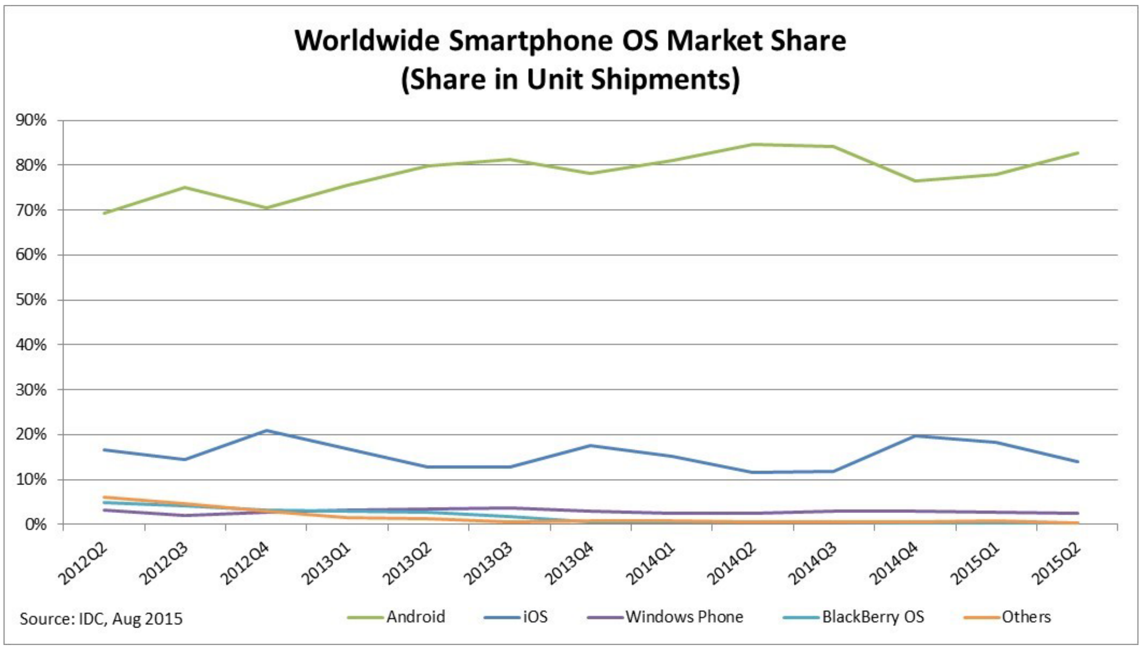
LIST OF ABBREVIATIONS

API	Application Programming Interface
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheet
JVM	Java Virtual Machine
MSIL	Microsoft Intermediate Language
IDE	Application Development Environment
CLR	Common Language Runtime
UI	User Interface
ECMA	European Computer Manufacturers Association
SPA	Single Page Application
DOM	Document Object Model

1 OS VENDORS AND DEVELOPMENT CHALLENGES

According to the International Data Corporation (IDC, 2015), three major mobile operating systems share the majority of the market:

1. iOS – developed by Apple.
2. Android – an open source project supported by Google.
3. Windows – developed by Microsoft.



Picture 1. Worldwide Smartphone OS Market Share (IDC, 2015).

Since all three platforms have been developed by different vendors and have appeared at different times, they differ not only in structure but also in the kind of technologies used to build applications. Although the most commonly used programming languages are *object oriented*, the differences in platform frameworks make applications written for one platform incompatible with the others.

In order to reach as many users as possible, companies and individuals that develop software need to create the same application three times – each time targeting a different platform. This means that the development cost essentially triples even though the functionality of the apps is identical.

Such approach is known as *native* application development. Native applications are created with the use of application frameworks provided by the operating system software vendors.

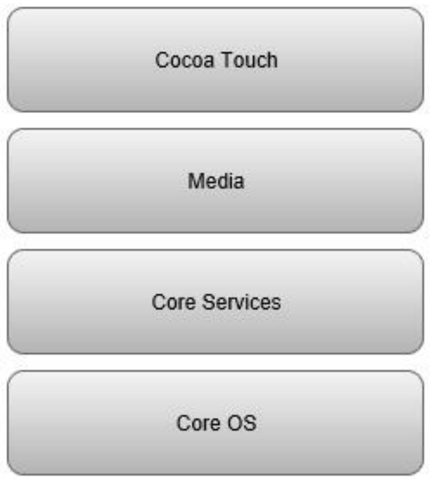
1.1 Platform Architectures

This section briefly describes the platform architectures of mobile operating systems in order to highlight why they are incompatible with each other.

1.1.1 iOS

Applications for iOS devices are written either in the Objective-C or Swift programming languages. The code written in these languages is compiled directly into a native code for a specific processor architecture, without the use of any intermediate languages. This is an important distinction among the languages used in other platforms.

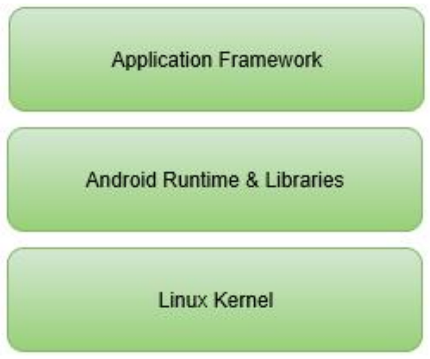
Platform Application Programming Interfaces (APIs) are delivered to developers through platform frameworks. These frameworks present themselves as layers, with complex and detailed components at the lower levels and abstract components at the higher levels (Apple, 2014).



Picture 2. iOS platform architecture, modified from (Apple, 2014).

1.1.2 Android

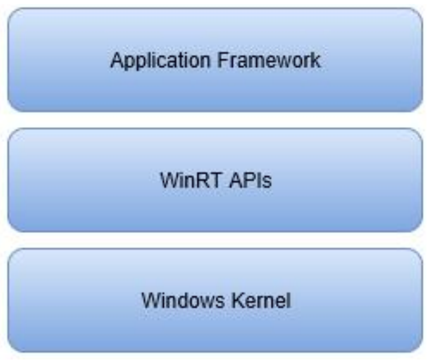
Java is the most common programming language for developing Android applications. Unlike Objective-C or Swift, Java does not compile to native code at once. Instead, Java source code compiles into an intermediate language, *byte code,* which can only be executed by a Java Virtual Machine (Oracle, 2011). There is a variety of Java Virtual Machines (JVMs) for different platforms. In case of Android, it is either Dalvik or Android Runtime (Android Project, 2016).



Picture 3. Android platform architecture, modified from (Android Team, 2016).

1.1.3 Windows Phone

Applications for Windows Phone are built using Microsoft's .NET framework with C# as the preferred programming language. Similar to Java, C# compiles into an intermediate language, *Microsoft Intermediate Language* (MSIL), which is then executed by the Common Language Runtime (CLR). *MSIL* and *byte code* share the same concept but are not compatible with each other (Microsoft, 2016).



Picture 4. Windows Phone platform architecture, modified from (Sharma, 2013).

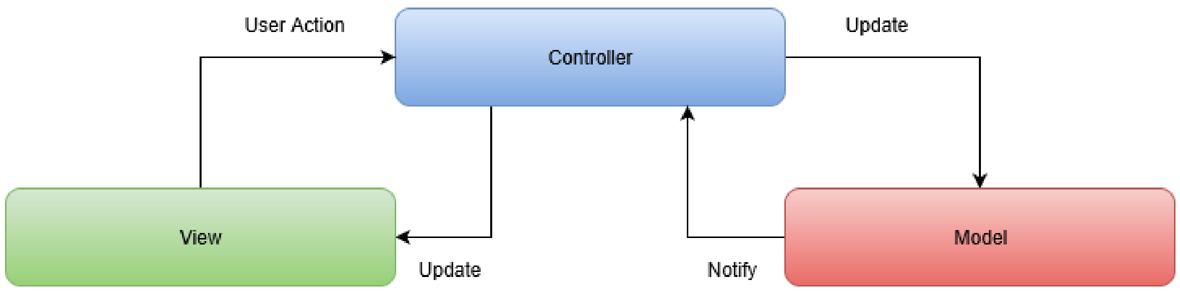
1.2 Application Architectures

Although developers are not limited in ways of organizing their code, several architectural patterns became a de-facto standard for building mobile applications. These patterns aim to facilitate the complexity of the application presentation and business logic. Two particularly common architectural patterns used in software development are **MVC** and **MVVM** patterns.

1.2.1 MVC (Model-View-Controller)

The most common architectural pattern that is used to build applications for iOS and Android is the MVC pattern (Apple, 2016). Trygve Reenskaug, a Swedish computer scientist, first introduced the pattern in the 70s in order to tackle the complexity of manipulation on large quantities of data (Reenskaug, 2010). The pattern consists of three object types:

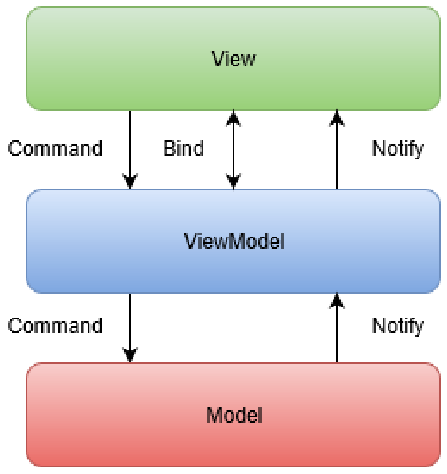
1. *View* – an object that is responsible for presenting information to the user and gathering user input.
2. *Model* – an object that encapsulates essential business domain rules.
3. *Controller* – an object that facilitates the communication between the *View* and the *Model*.



Picture 5. Interaction of MVC components, modified from (Apple, 2012).

1.2.2 MVVM (Model-View-ViewModel)

The MVVM is an architectural pattern most commonly used in Windows applications. The pattern was formulated by Ken Cooper and Ted Peters while working at Microsoft, and was first announced by John Gossman in his blog in 2005 (Gossman, 2005). The purpose of the *View* and the *Model* in MVVM are the same as in MVC, however, the *Controller* is now replaced with a *ViewModel*. The *ViewModel* is practically a representation of the *View* in terms of actions and properties. The elements in the *View* bind to the corresponding properties of the *ViewModel* and *ViewModel* updates the *View* whenever its state changes.



Picture 6. MVVM pattern, modified from (Microsoft, 2012).

2 CROSS-PLATFORM SOLUTIONS

Cross-platform mobile software has been a hot topic for several years now. Being able to create apps that could run on the three major platforms with the use of existing skills would decrease the development and maintenance cost of these apps significantly.

At this point, the two most popular frameworks for developing cross-platform mobile apps are **Cordova** and **Xamarin** and both frameworks have different approaches to run across platforms.

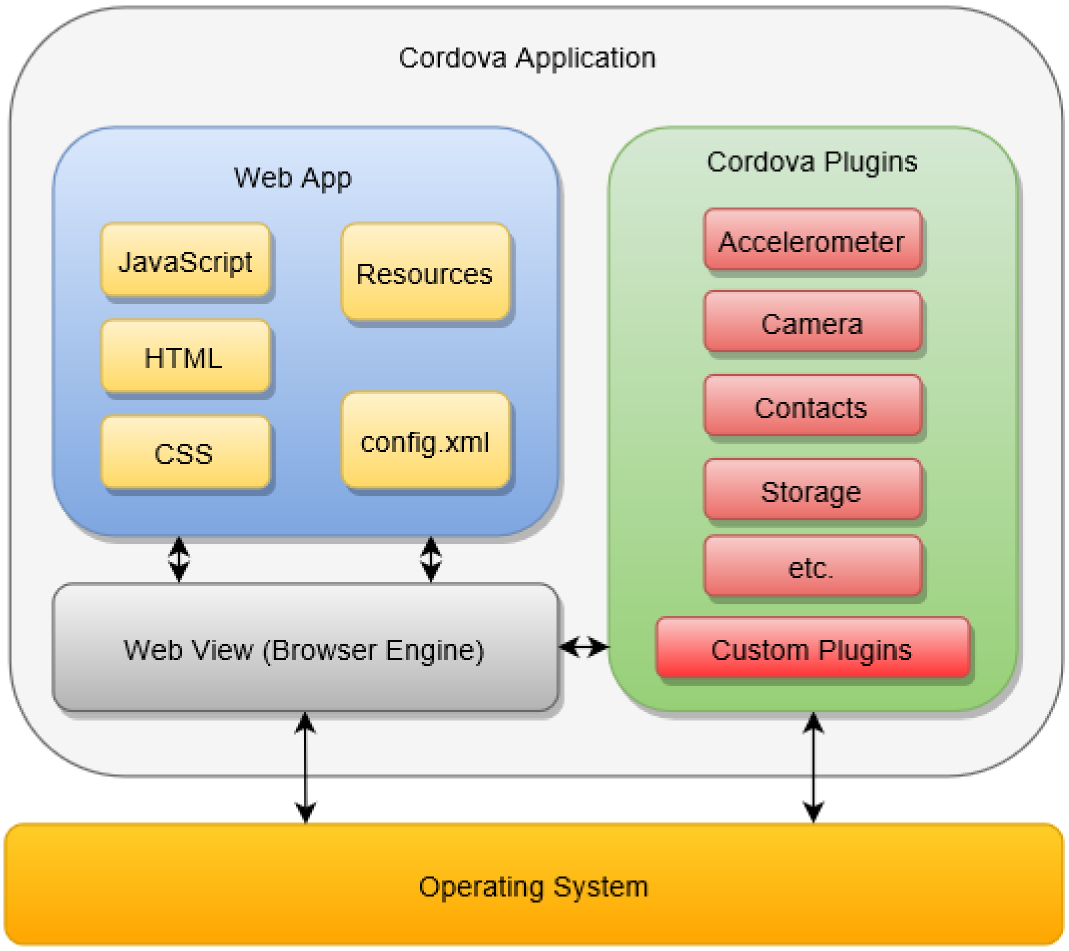
2.1 Cordova

Cordova (Apache, 2016) is a framework that enables developers to use their web programming skills (JavaScript, HTML, CSS) in order to build mobile apps. Apps that are built with Cordova are quite often called *hybrid* apps.

A *hybrid* application is essentially a web page that is contained locally on the users' phone (instead of being loaded from the server in a normal web page scenario) and executed in a built-in web browser engine. Browsers on each of the platforms conform to the ECMA standard in order to be able to render web pages; this makes it possible to share the code between the platforms.

Another important distinction between regular web pages and hybrid apps is that a hybrid app is wrapped in a platform-specific application executable that can be distributed through the platform's application store. The wrapper also permits access to the phone's hardware APIs (such as camera, accelerometer or storage) through a layer of plugins written in the platform specific language (C#, Swift, Objective-C, Java, among others). This capability is the reason for the name "*hybrid*".

There is a variety of frameworks that are built on top of Cordova and extend its functionality. Ionic framework is one such framework. It provides a set of tools and reusable components to be used when building apps and it uses AngularJS – a JavaScript framework for building single page frontend applications (SPA).



Picture 7. Cordova platform architecture, modified from (Apache, 2016).

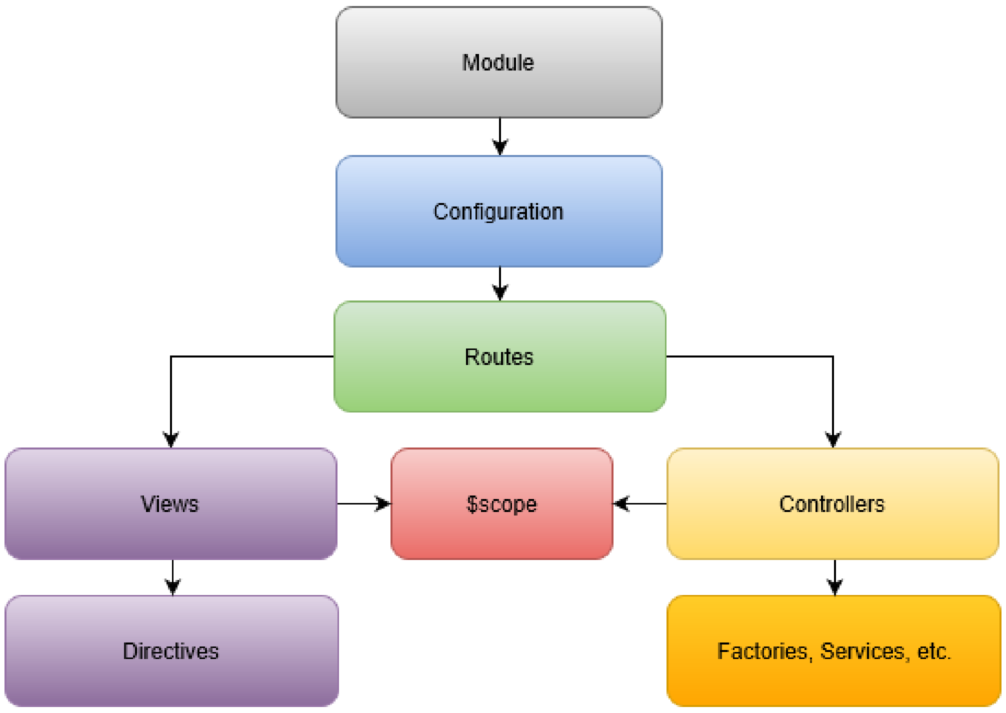
2.1.1 AngularJS

AngularJS is a well-known and highly popular framework for frontend development. It helps building complicated applications by utilizing a variation of MVVM pattern and other implementation patterns such as *factory* and *dependency injection*.

AngularJS' main component types are:

- *Controllers* – they provide a set of methods to manipulate the state of the *View*.
- *Views* – they are HTML templates that bind to a controller and are updated whenever the corresponding *Controller* updates the state.

- *Directives* – they are special attributes that can extend the functionality of regular HTML DOM elements.
- *Services* – they are objects that provide the functionality to be used within controllers. Usually these objects encapsulate the application's business logic.



Picture 8. Angular architecture, modified from (Wahlin, 2013).

2.2 Xamarin

Xamarin is a framework and a set of tools based on an open source, cross-platform version of Microsoft's .NET framework called Mono (Xamarin, 2016).

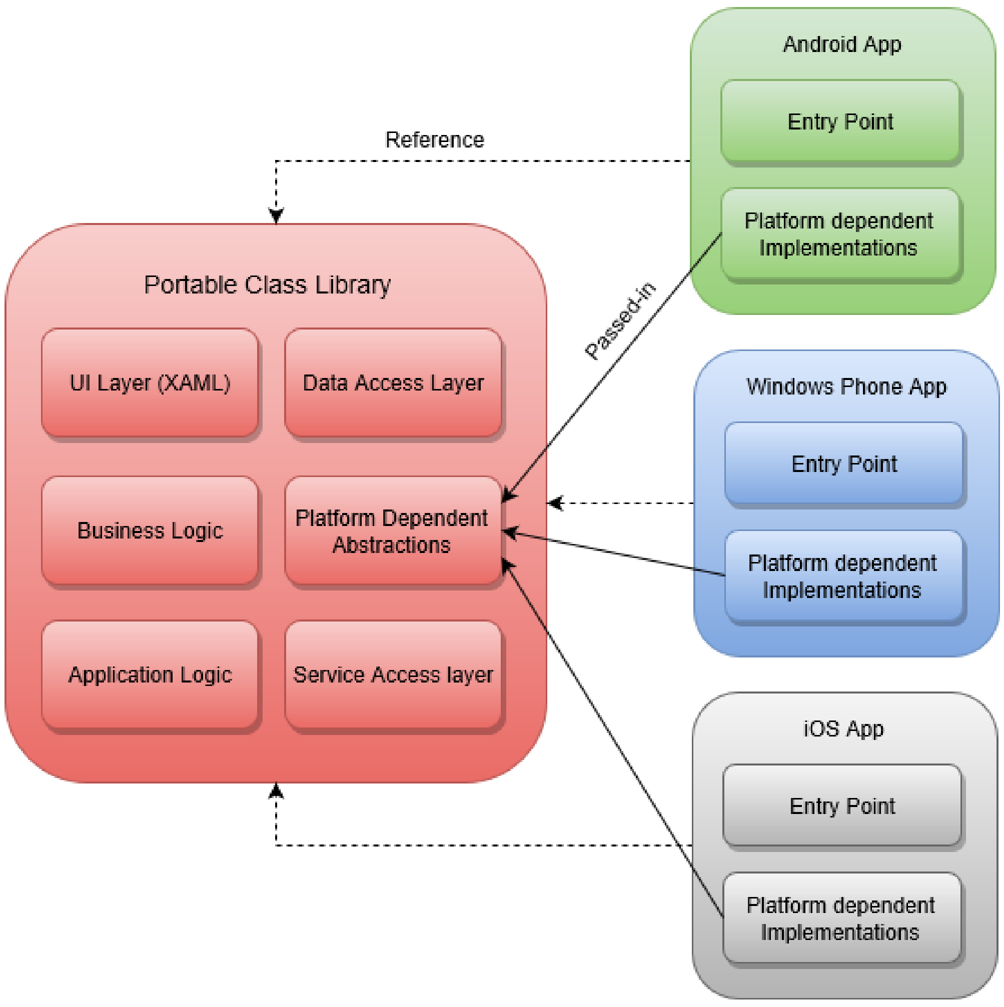
Xamarin apps are written in C# and work similar to native Windows Phone apps (refer to section 1.1.3). There is, however, a difference in compilation and execution process on different platforms (Xamarin, 2016):

- iOS – the C# source code is compiled directly into the processor instructions. No IL code is generated, hence, no runtime is involved in the execution process.

- Android – the C# source code is compiled into IL code and packaged along with Mono Runtime, which is an open source version of Microsoft's .NET CLR. The IL is then executed by the Mono Runtime.
- Windows Phone – works just like a Windows Phone app in a native scenario. The IL is executed by the Microsoft's .NET CLR.

There are several approaches of building apps with Xamarin. This work is particularly focusing on **Xamarin.Forms** since it permits the most code reuse.

Xamarin.Forms application consists of a platform specific executable and a shared library that contains the application's business logic and platform dependent abstractions. Platform specific executable contains an entry point and the implementation of platform dependent abstractions.



Picture 9. Xamarin.Forms solution architecture, modified from (Xamarin, 2016).

2.3 Solutions Comparison

When it comes to choosing a development approach, it is important to understand its limitations. This section covers the benefits and drawbacks of native, hybrid and Xamarin application development.

2.3.1 Native Application Development

Pros:

- Gestures – native apps support multi touch input and are capable of recognizing gestures like swipe and pinch out of the box.
- Animations – complicated animations can be created. This is a crucial factor if applications have to be visually rich and complicated.
- Hardware API access – native application frameworks provide easy to use access to hardware APIs, such as camera, accelerometer or storage.
- Documentation – the software vendors extensively document each native framework.

Cons:

- Skills – different platforms rely on a different set of technologies; time is required in order to learn to use them.
- Time/money – as mentioned before, the development time triples because three different apps have to be created. This factor turns out to be the most crucial when choosing a development approach.

2.3.2 Hybrid Application Development

Pros:

- Use of existing skills – because hybrid apps are built around web technologies, it is easier for web developers to leverage their existing skills in order to deliver mobile applications.
- Time/money – since the code base is shared, less time is required for developing the applications.

Cons:

- Hardware API access – because of an extra layer of browser engine, access to hardware APIs becomes complicated. Each time it is required to use a hardware API – a plugin has to be used. A plugin, written in platform specific language, exposes hardware APIs to be used from the browser's JavaScript engine. There is a variety of plugins created by the community, but quite often these plugins are poorly maintained.
- Animations – application developers are limited in the kind of animations they can have in their apps because browser engines are not as efficient as native presentation technologies. This is usually not a concern if the applications are not required to have sophisticated animations.
- Gestures – the only gesture that is supported out of the box is a regular tap gesture. In order to leverage more gestures in hybrid applications a plugin has to be used.
- Performance – again, because of the layer of the browser engine, apps consume more resources and may behave unresponsively.

2.3.3 Xamarin Application Development

Pros:

- Use of existing skills – Xamarin apps are built with C#, which is among the most popular programming languages and is often used for building server applications.
- Time/Money – similar to the hybrid approach, the code base is shared and less time is required for developing the applications.
- Compiled into native code – unlike the hybrid approach, C# code is compiled into native code. This makes apps as efficient as native apps.

Cons:

- Animations – because of the differences in presentation technologies, it is harder to make animations that would look good on each of the platforms.

- Gestures – not all the gestures are supported by Xamarin.Forms framework.

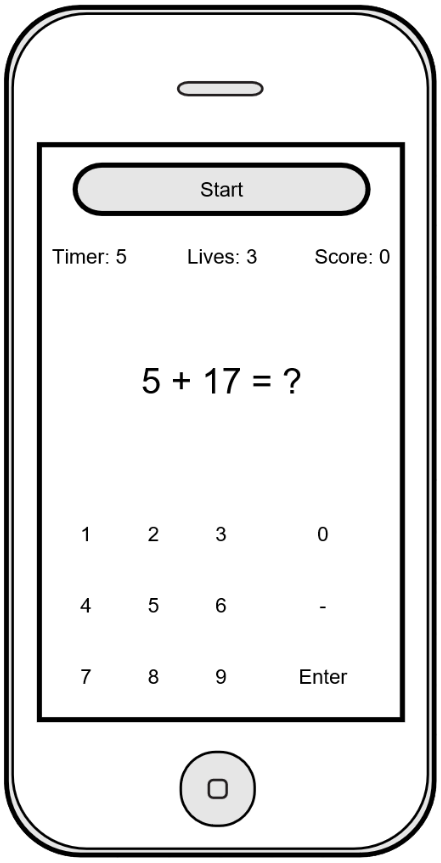
2.3.4 Cost

Hybrid frameworks that exist today are free to use and it has been recently announced that the cost for developing Xamarin apps has been dropped (Xamarin, 2016). This means that there is no cost difference when using either of the approaches anymore.

3 PRACTICAL EXAMPLE

3.1 The Application

The example application named “Train Your Brain” is a simple problem-solving game. The player has to solve mathematical problems in a limited amount of time. Each correct answer increases the player’s score and complexity of the subsequent expressions. Each incorrect answer subtracts a “life” from the player stats. Once the player reaches zero lives – the game is over.



Picture 10. Application UI mockup, created with (Moqups, 2016).

The timer starts to count once the user presses the “Start” button. The player has five seconds to give an answer by using a numeric pad on the bottom of the screen. Once the answer is submitted and evaluated – another expression is deployed. If the game is over, the timer stops and the “Game Over” message appears.

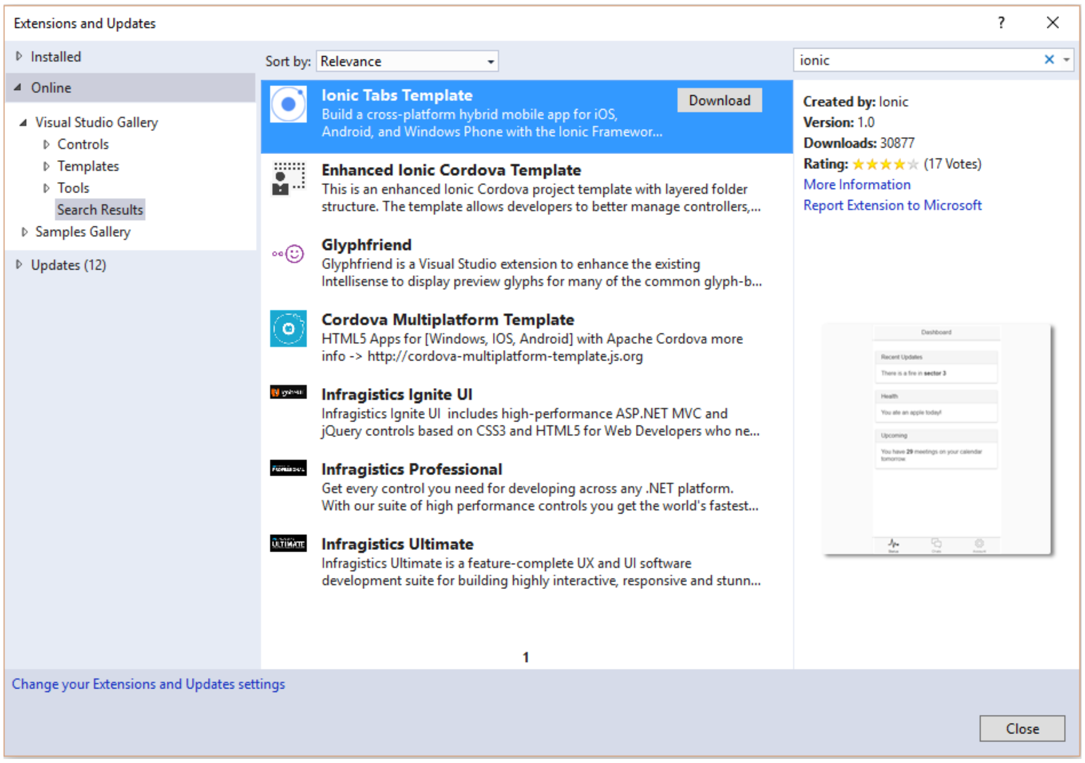
3.2 Hybrid Approach

The following section describes the development environment and implementation of the example application using *hybrid* development approach.

3.2.1 Development Environment

Practically, a simple text editor and a command line are sufficient to build hybrid applications; however, we believe that Visual Studio IDE provides greater support and streamlines the development process.

Visual Studio supports a variety of project templates, including a template for applications based on Ionic framework. The template can be downloaded from the Visual Studio extension gallery (Microsoft, 2015), or directly from the Visual Studio's "*Extensions and Updates*" window. After the installation, the Ionic project template will be available under the "*New Project*" window.



Ionic Tabs Template
Build a cross-platform hybrid mobile app for iOS, Android, and Windows Phone with the Ionic Framewor...

Enhanced Ionic Cordova Template
This is an enhanced Ionic Cordova project template with layered folder structure. The template allows developers to better manage controllers,...

Glyphfriend
Glyphfriend is a Visual Studio extension to enhance the existing Intellisense to display preview glyphs for many of the common glyph-b...

Cordova Multiplatform Template
HTML5 Apps for [Windows, IOS, Android] with Apache Cordova more info -> http://cordova-multiplatform-template.js.org

Infragistics Ignite UI
Infragistics Ignite UI includes high-performance ASP.NET MVC and jQuery controls based on CSS3 and HTML5 for Web Developers who ne...

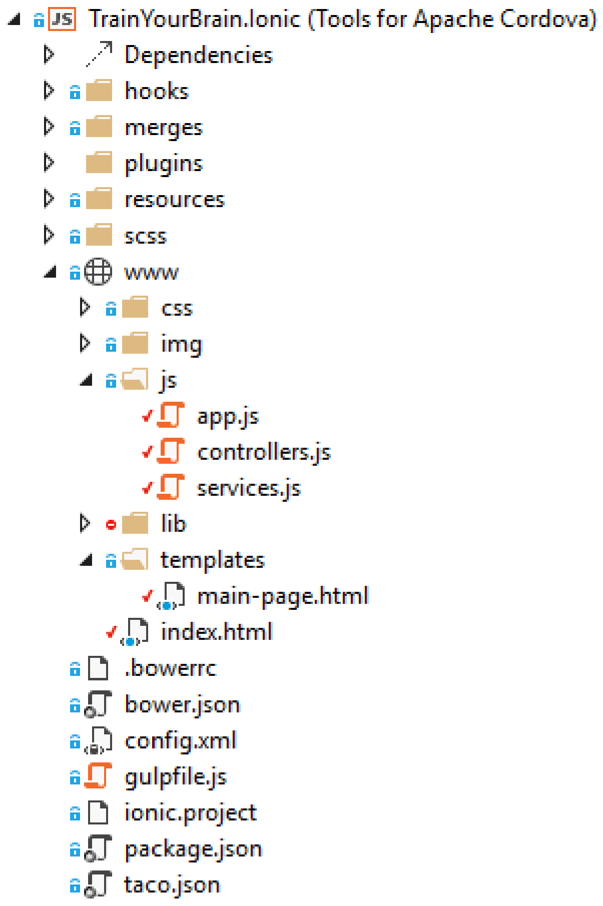
Infragistics Professional
Get every control you need for developing across any .NET platform. With our suite of high performance controls you get the world's fastest...

Infragistics Ultimate
Infragistics Ultimate is a feature-complete UX and UI software development suite for building highly interactive, responsive and stunn...

Created by: Ionic
Version: 1.0
Downloads: 30877
Rating: ★★★★☆ (17 Votes)
More Information
Report Extension to Microsoft

Picture 11. Search result for Ionic project template.

After the creation of the project from Ionic template, Visual Studio generates a basic project structure.



```
TrainYourBrain.Ionic (Tools for Apache Cordova)
    Dependencies
    hooks
    merges
    plugins
    resources
    scss
    www
        css
        img
        js
            app.js
            controllers.js
            services.js
        lib
        templates
            main-page.html
        index.html
    .bowerrc
    bower.json
    config.xml
    gulpfile.js
    ionic.project
    package.json
    taco.json
```

Picture 12. Project structure as seen from Solution Explorer in Visual Studio.

3.2.2 Ionic Project Structure

- *merges folder – it* contains platform specific code, typically, compatibility scripts and platform overrides.
- *plugins folder* – it contains native components that could be used in the application code.
- *www folder* – it holds application resources.
- *index.html* – it is the main view of the application. Cordova framework, application logic and styles are referenced from this file.
- *config.xml* – it contains configurations specific to each platform. It also holds references to plugins.

- *css folder* – it holds application style files.
- *img folder* – it holds visual assets that are used in the application.
- *js folder* – it contains JavaScript files with the application logic.
- *app.js* – it contains bootstrapping code and configures AngularJS application page routing.
- *controllers.js* – it holds AngularJS controllers.
- *services.js* – it holds AngularJS services.
- *templates folder* – it contains HTML files that represent the UI of the application. These files are loaded into the main view (*index.html*).
- *main-page.html* – it is the main template to be loaded into the main view.

3.2.3 Implementation

Following section describes the implementation of the example app using the *hybrid* approach.

- *app.js* (Appendix 1, Code Listing 1)

The code in this file is the bootstrapping code. It configures certain features of Cordova application and brings together *Controllers* and *Views*.

- *services.js* (Appendix 1, Code Listing 2)

This file contains the application logic that is agnostic to the presentation layer. In this file, we have defined a factory module that is going to be responsible for constructing *Expression* objects. *Expression* is an object that holds the operands and contains a method for formatting these operands into strings. It also contains a method for checking the results.

- *controllers.js* (Appendix 1, Code Listing 3)

Since we have only one *View* – we have created one *Controller*. The *scope* of this controller contains methods that the *View* uses to display the expression, and gather user input.

- *main-page.html* (Appendix 1, Code Listing 4)

This file contains HTML markup that represents the application UI. This *View* binds to a scope of the main *Controller*. The *View* is divided into three rows:

a. the first row displays user statistics, such as amount of lives, score and timer.
b. the second row displays the expression and the user input by concatenating them.
c. the third row is an input pad. It holds numeric buttons as well as buttons to submit the answer, erase characters, or insert the minus sign. Commands for these buttons are bound via AngularJS' *ng-click* directive with the corresponding number as a parameter.

- *index.html* (Appendix 1, Code Listing 5)

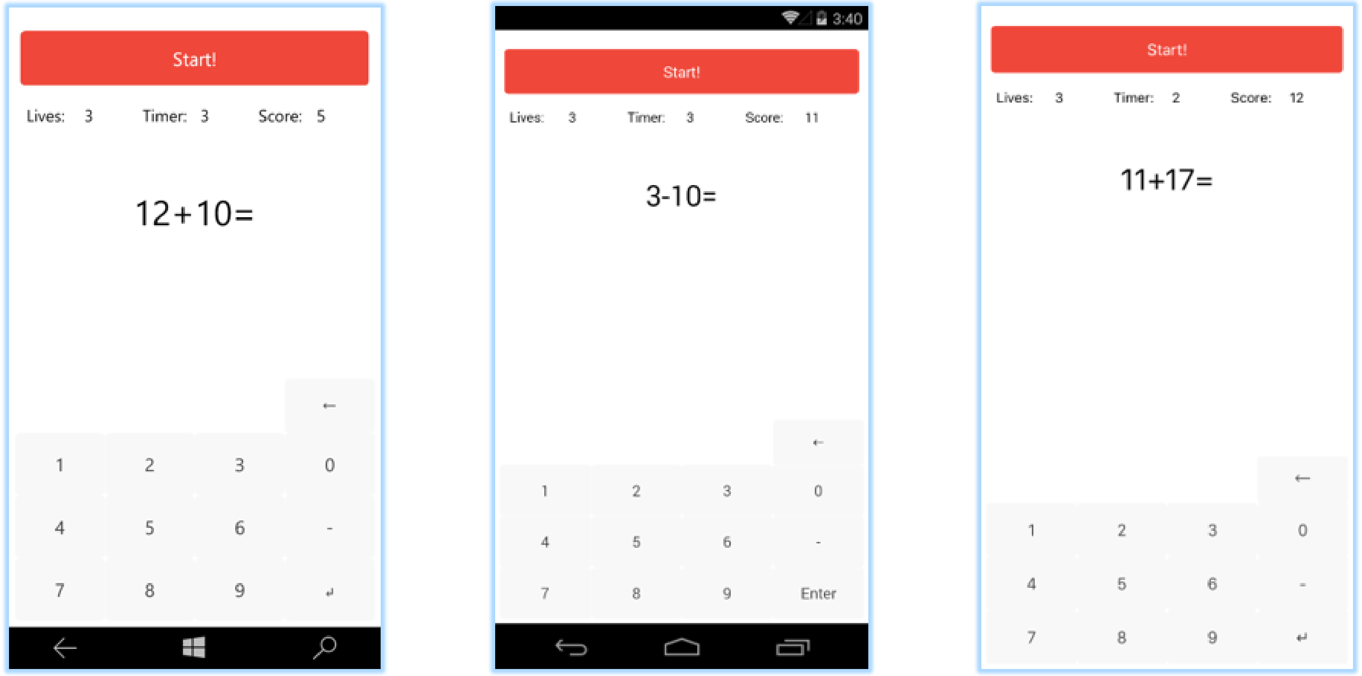
In this file, we have changed the title of the app and referenced *controllers.js* and *expressionFactory.js*. It also contains references for Cordova and Ionic itself. The application mainPage.html will be rendered within the `<body ng-app="trainBrain" >…</body>` tag.

- *style.css* (Appendix 1, Code Listing 6)

Most of the styles used in the app are default Ionic styles. We have applied a few minor visual improvements.

3.2.4 Results

All three operating systems can execute the application after compilation. The application looks similar on each of the platforms because no native components are used.



Picture 13. Ionic app running in WP, Android and iOS emulators.

3.3 Xamarin Approach

The following section describes the development environment and implementation of the example application using *Xamarin.Forms* development approach.

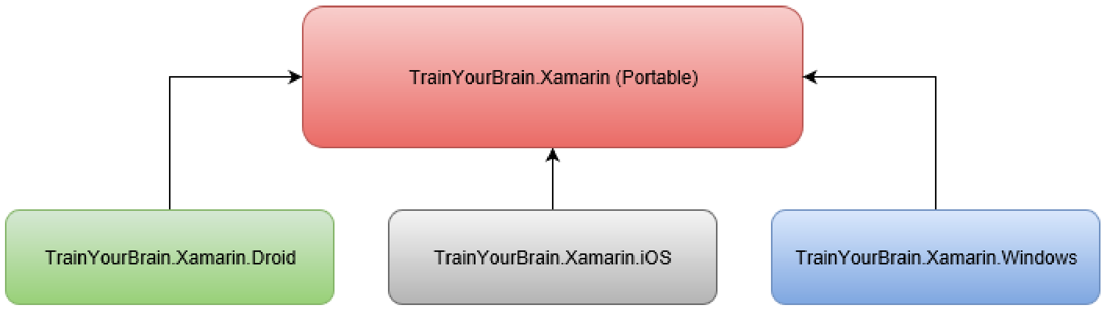
3.3.1 Development Environment

Similar to the case of Ionic, Visual Studio supports the development of Xamarin apps. Xamarin tools are installed along with the Visual Studio or separately as a plugin (Microsoft, 2016). There is one limitation when developing Xamarin apps on a PC; due to certain restrictions from Apple, in order to build and run the application for iOS, Apple Mac is required.

After creating the *Xamarin.Forms* project, the basic project structure is generated.

3.3.2 Xamarin.Forms Project Structure

The Xamarin.Forms solution consists of four projects: three projects corresponding to each platform plus a shared project. Each platform project contains platform specific bootstrapping code, assets, and resources.

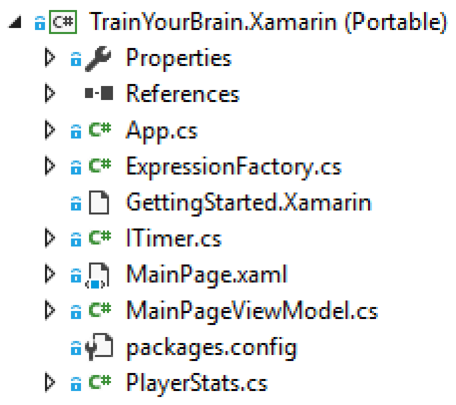

TrainYourBrain.Xamarin (Portable)
TrainYourBrain.Xamarin.Droid
TrainYourBrain.Xamarin.iOS
TrainYourBrain.Xamarin.Windows

Picture 14. Xamarin projects and their dependencies.

3.3.3 Implementation

The following section describes the implementation of the example app using *Xamarin.Forms*.

TrainYourBrain.Xamarin (Portable)



```
TrainYourBrain.Xamarin (Portable)
  Properties
  References
  App.cs
  ExpressionFactory.cs
  GettingStarted.Xamarin
  ITimer.cs
  MainPage.xaml
  MainPageViewModel.cs
  packages.config
  PlayerStats.cs
```

Picture 15. Shared project.

- *App.cs (*Appendix 2, Code Listing 1)

A C# class in this file is an entry point of the application. In this class, we have configured the main page of the application. This class is instantiated from the host platform project.

- *ExpressionFactory.cs* (Appendix 2, Code Listing 2)

The sole responsibility of this class is to create *Expression* objects.

- *Expression.cs* (Appendix 2, Code Listing 2)

It contains methods and fields to hold and operate on the expression operands and defines formatting rules.

- *ITimer.cs* (Appendix 2, Code Listing 3)

The Timer type provides timer functionality. However, this type cannot be used from this portable class library. Hence, we have created an `ITimer` interface that is implemented in each platform-specific project and injected into the *App* through the constructor.

- *PlayerStats.cs* (Appendix 2, Code Listing 4)

It contains the player stats properties. This class implements the `INotifyPropertyChange` interface, which enables the *View* to be notified whenever the player statistics change.

- *MainPageViewModel.cs* (Appendix 2, Code Listing 5)

This is a *ViewModel* for the main *View* to bind. It exposes properties and actions the main *View* uses.

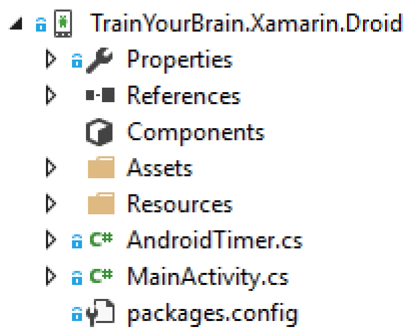
- *MainPage.xaml* (Appendix 2, Code Listing 6)

This XAML page describes the user interface of the app. The `Binding` keyword is used to bind to the *ViewModel's* properties and commands.

- *MainPage.xaml.cs* (Appendix 2, Code Listing 7)

This is a typically called "code behind". This partial class contains code that can access elements of the *View* and perform manipulations with them. Because the application uses MVVM pattern, this code has little to do with the application logic. The binding to the *ViewModel* is created in the class constructor.

Train.YourBrain.Xamarin.Droid



```
TrainYourBrain.Xamarin.Droid
  Properties
  References
  Components
  Assets
  Resources
  AndroidTimer.cs
  MainActivity.cs
  packages.config
```

Picture 16. Android project structure.

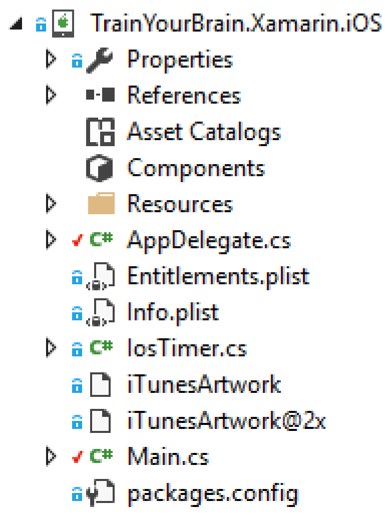
- *AndroidTimer.cs* (Appendix 2, Code Listing 8)

This is an Implementation of the `ITimer` interface. An instance of this class is passed during the instantiation of the App. The implementation is identical in case of Windows Phone and iOS.

- *MainActivity.cs* (Appendix 2, Code Listing 9)

In Android, an *Activity* is the main work unit that brings together the application business and the presentation logic. The *App* is instantiated from this class.

Train.YourBrain.Xamarin.iOS



```
TrainYourBrain.Xamarin.iOS
    Properties
    References
    Asset Catalogs
    Components
    Resources
    AppDelegate.cs
    Entitlements.plist
    Info.plist
    IosTimer.cs
    iTunesArtwork
    iTunesArtwork@2x
    Main.cs
    packages.config
```

Picture 17. iOS project structure.

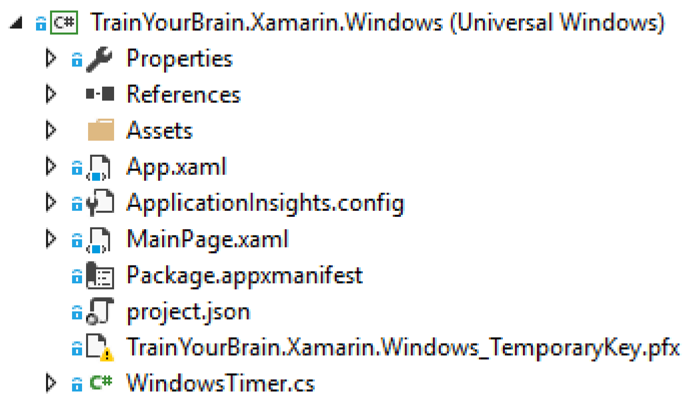
- *AppDelegate.cs* (Appendix 2, Code Listing 10)

In iOS, this class is responsible for the user interface initialization. The *App* is instantiated from this class.

- *Main.cs* (Appendix 2, Code Listing 11)

This is the application’s entry point which calls into the `AppDelegate`.

TrainYouBrain.Xamarin.Windows


TrainYourBrain.Xamarin.Windows (Universal Windows)
- Properties
- References
- Assets
- App.xaml
- ApplicationInsights.config
- MainPage.xaml
- Package.appxmanifest
- project.json
- TrainYourBrain.Xamarin.Windows_TemporaryKey.pfx
- WindowsTimer.cs

Picture 18. Windows project structure.

- *App.xaml/App.xaml.cs* (Appendix 2, Code Listing 12)

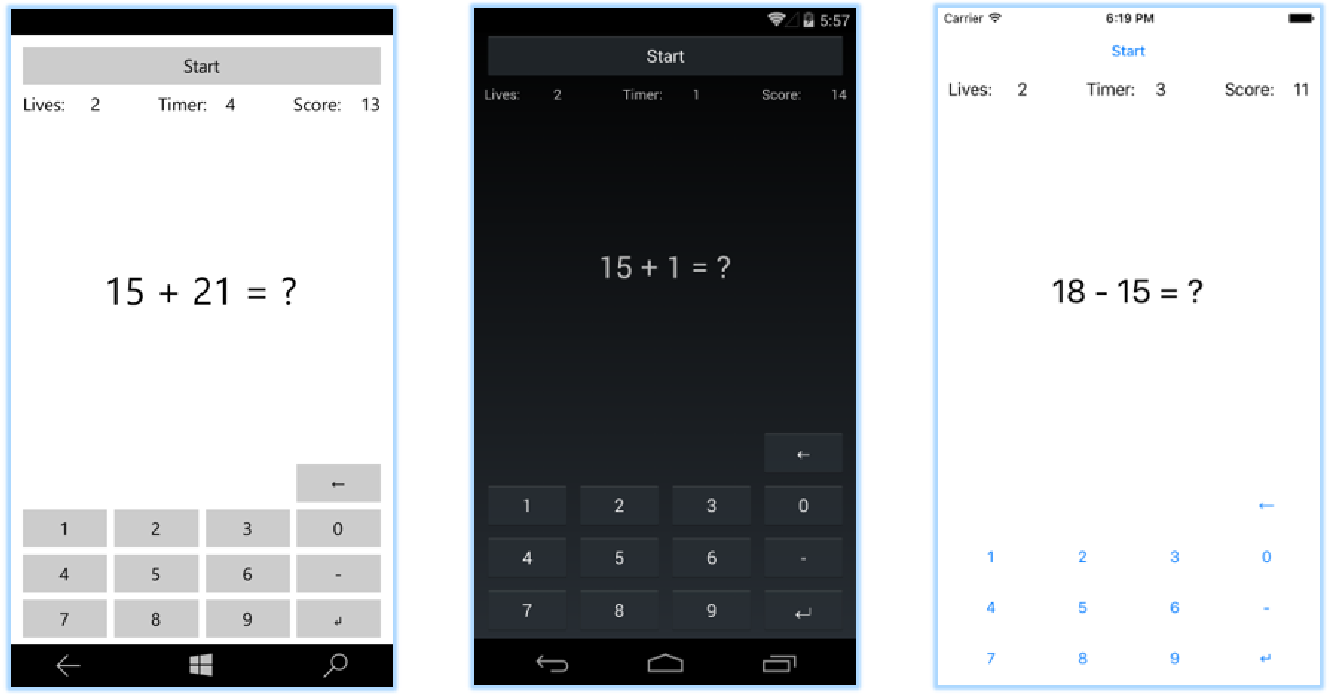
This is the native entry point of *Windows* apps. The *App* is instantiated from this class.

- MainPage.xaml (Appendix 2, Code Listing 13)

The *Page* from the shared project is rendered within this page.

3.3.4 Results

All three operating systems can execute the application after compilation. The application looks slightly different on each of the platforms because native UI components are used.



Picture 19. Xamarin.Forms app running in WP, Android and iOS emulators.

4 DISCUSSION AND CONCLUSION

Both Cordova and Xamarin allow building cross-platform mobile software solutions. However, there are compromises to be made when choosing one platform over the other.

Cordova, with addition of Ionic, allows rapid development due to a vast variety of ready-made components and the application architecture imposed by AngularJS. Applications with a simple UI can be delivered in a relatively short period of time. This approach is perfect for apps that do not require high performance since the browser engines are not as efficient as they would be if the app ran natively. The user interface of the app appears to be identical from platform to platform because no native UI components are used. Certain difficulties are present when it comes to accessing hardware capabilities. Although there are numerous native plugins available, it is not easy to work with them as it can be in the native case.

Xamarin.Forms apps are written in C# and because this language is strongly typed, it tends to be less error prone and easier to debug than JavaScript. However, usually C# includes more of "*boilerplate code*", meaning that the code is more verbose. This affects the speed of the development in the short term, but makes it easier to introduce changes in the long term. Since C# is compiled into native code of each platform, the apps use native components and have better performance. Because the code is separated in several platform projects, it is easier to do platform-specific customization. Similarly to the case of Ionic, there is a variety of components that are provided by the community and can be used within the app.

In summary, the decision of choosing an approach should be well considered based on the application's complexity and customer needs in terms of time and performance.

5 REFERENCES

Android Project, 2016. *ART and Dalvik.* [Online]
Available at: https://source.android.com/devices/tech/dalvik/
[Accessed 25 December 2015].

Android Team, 2016. *Google Code.* [Online]
Available at: http://androidteam.googlecode.com/files/Anatomy-Physiology-of-an-Android.pdf
[Accessed 10 March 2016].

Apache, 2016. *Cordova Documentation.* [Online]
Available at: https://cordova.apache.org/docs/en/latest/guide/overview/
[Accessed 10 January 2016].

Apple, 2012. *Model-View-Controller.* [Online]
Available at:
https://developer.apple.com/library/ios/documentation/General/Conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html
[Accessed 27 December 2015].

Apple, 2014. *iOS Technology overview.* [Online]
Available at:
https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html
[Accessed 12 January 2016].

Apple, 2016. *Concepts in Objective-C Programming.* [Online]
Available at:
https://developer.apple.com/library/ios/documentation/General/Conceptual/CocoaEncyclopedia/Introduction/Introduction.html#//apple_ref/doc/uid/TP40010810
[Accessed 27 December 2015].

Gossman, J., 2005. [Online]
Available at: http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx
[Accessed 15 December 2015].

IDC, 2015. *Smartphone OS Market Share.* [Online]
Available at: http://www.idc.com/prodserv/smartphone-os-market-share.jsp
[Accessed 27 December 2015].

Microsoft, 2012. *The MVVM Patttern.* [Online]
Available at: https://msdn.microsoft.com/en-us/library/hh848246.aspx?f=255&MSPPError=-2147217396
[Accessed 12 January 2016].

Microsoft, 2015. *Ionic Project Template.* [Online]
Available at: https://visualstudiogallery.msdn.microsoft.com/8fa5bff2-e023-4e13-8b36-0244e935fb7d
[Accessed 5 January 2016].

Microsoft, 2016. *Managed Execution Process.* [Online]
Available at: https://msdn.microsoft.com/en-us/library/k5532s8a(v=vs.110).aspx
[Accessed 9 March 2016].

Microsoft, 2016. *Xamarin Extension.* [Online]
Available at: https://visualstudiogallery.msdn.microsoft.com/dcd5b7bd-48f0-4245-80b6-002d22ea6eee
[Accessed 10 January 2016].

Moqups, 2016. *Moqups Designer.* [Online]
Available at: https://moqups.com
[Accessed 12 04 2016].

Oracle, 2011. *JVM Specification.* [Online]
Available at: https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-1.html#jvms-1.2
[Accessed 12 January 2016].

Reenskaug, T. M. H., 2010. [Online]
Available at: http://folk.uio.no/trygver/themes/mvc/mvc-index.html
[Accessed 15 December 2016].

Sharma, G., 2013. *C# Corner.* [Online]
Available at: http://www.c-sharpcorner.com/UploadFile/b5be7f/some-basic-differences-between-windows-store-app-windows-p/
[Accessed 10 March 2016].

Wahlin, D., 2013. [Online]
Available at: http://weblogs.asp.net/dwahlin/video-tutorial-angularjs-fundamentals-in-60-ish-minutes
[Accessed 8 March 2016].

Xamarin, 2016. *Application Fundamentals.* [Online]
Available at: https://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/sharing_code_options/
[Accessed 10 March 2016].

Xamarin, 2016. *Understanding the Xamarin Mobile Platform.* [Online]
Available at: https://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/part_1_-_understanding_the_xamarin_mobile_platform/
[Accessed 25 04 2016].

Xamarin, 2016. *Xamarin Platform.* [Online]
Available at: https://xamarin.com/platform
[Accessed 5 February 2016].

Xamarin, 2016. *Xamarin Store.* [Online]
Available at: https://store.xamarin.com/
[Accessed 10 March 2016].

APPENDIX 1: HYBRID APP IMPLEMENTATION

6.1 Code Listing: app.js

```
angular.module("trainBrain", ["ionic", "trainBrain.controllers", "trainBrain.factories"])

.run(function ($ionicPlatform) {
    $ionicPlatform.ready(function () {
        if (window.StatusBar) {
            window.StatusBar.styleLightContent();
        }
    });
})

.config(function ($stateProvider, $urlRouterProvider) {

    $stateProvider.state("main", {
        url: "/main",
        templateUrl: "templates/main-page.html",
        controller: "expressionController"
    });

    $urlRouterProvider.otherwise("/main");
});
```

6.2 Code Listing: services.js

```
angular.module("trainBrain.services", [])
       .factory("expressionFactory", expressionFactory);

var operators = ["+", "-"];

function expressionFactory() {
    var numberRange = 8;

    var createExpression = function () {
        var a = Math.floor(Math.random() * numberRange);
        var b = Math.floor(Math.random() * numberRange);
        var operator = operators[Math.floor(Math.random() * operators.length)];

        return new Expression(a, b, operator);
    };

    return {
        getExpression: function () {
            numberRange++;
            return createExpression();
        },
        reset: function () {
            numberRange = 8;
        }
    };
};

function Expression(a, b, operator) {

    function toString() {
        return a.toString() + operator.toString() + b.toString() + '=';
    };

    function checkAnswer(userAnswer) {
        return calculate() === userAnswer;
    };

    function calculate() {
        switch (operator) {
            case "+":
                return a + b;
            case "-":
                return a - b;
        }
    };

    return {
        checkAnswer: checkAnswer,
        toString: toString
    };
};
```

6.3 Code Listing: controllers.js

```
angular.module("trainBrain.controllers", [])
       .controller("expressionController", expressionController);

function expressionController($interval, $ionicPopup, expressionFactory) {
    var timer;
    var vm = this;
    vm.isInputDisabled = true;

    var timerTick = function () {
        vm.timer--;
        if (vm.timer === 0) {
            vm.lives--;
            vm.deployExpression();
            vm.timer = 5;
        }
    };

    var init = function () {
        vm.timer = 5;
        vm.score = 0;
        vm.lives = 3;
        vm.userInput = "";
        vm.isInputDisabled = false;

        vm.deployExpression();
    };

    vm.deployExpression = function () {
        if (vm.lives > 0) {
            vm.currentExpression = expressionFactory.getExpression();
        } else {
            vm.showAlert();
            $interval.cancel(timer);
        }
    };

    vm.eraseChar = function () {
        if (vm.userInput.length !== 0) {
            vm.userInput = vm.userInput.slice(0, -1);
        }
    };

    vm.appendChar = function (char) {
        vm.userInput = vm.userInput + char;
    };
    vm.checkAnswer = function () {
        var isCorrect = vm.currentExpression.checkAnswer(parseInt(vm.userInput));

        if (isCorrect) {
            vm.score++;
        } else {
            vm.lives--;
        }

        vm.timer = 5;
        vm.userInput = '';
        vm.deployExpression();
    };

    vm.start = function () {
        expressionFactory.reset();
        init(vm);
        if (timer) {
            $interval.cancel(timer);
        }
        timer = $interval(timerTick, 1000);
    };

    vm.showAlert = function () {
        var alertPopup = $ionicPopup.alert({
            title: "Game Over",
            template: "Your score: " + vm.score
        });

        alertPopup.then(function () {
            vm.isInputDisabled = true;
        });
    };
}
```

6.4 Code Listing: main-page.html

```
<ion-view view-title="Main" ng-controller="expressionController as ec">
    <div class="row-top">
        <div class="col">
            <div class="row">
                <button class="button button-block button-assertive"
                        ng-click="ec.start()">
                    Start!
                </button>
            </div>
            <div class="row">
                <div class="col">Lives:</div>
                <div class="col">{{ec.lives}}</div>
                <div class="col">Timer:</div>
                <div class="col">{{ec.timer}}</div>
                <div class="col">Score:</div>
                <div class="col">{{ec.score}}</div>
            </div>
        </div>
    </div>
    <div class="row-center expression">
        <h2>{{ec.currentExpression.toString() + ec.userInput}}</h2>
    </div>
    <div class="row-bottom fixed-bottom">
        <div class="row">
            <div class="col-25 col-offset-75 col-right button"
                 ng-disabled="ec.isInputDisabled"
                 ng-click="ec.eraseChar()">←</div>
        </div>
        <div class="row">
            <div class="button col"
                 ng-disabled="ec.isInputDisabled"
                 ng-click="ec.appendChar(1)">1</div>
            <div class="button col"
                 ng-disabled="ec.isInputDisabled"
                 ng-click="ec.appendChar(2)">2</div>
            <div class="button col"
                 ng-disabled="ec.isInputDisabled"
                 ng-click="ec.appendChar(3)">3</div>
            <div class="button col"
                 ng-disabled="ec.isInputDisabled"
                 ng-click="ec.appendChar(0)">0</div>
        </div>
        <div class="row">
            <div class="button col"
                 ng-disabled="ec.isInputDisabled"
                 ng-click="ec.appendChar(4)">4</div>
            <div class="button col"
                 ng-disabled="ec.isInputDisabled"
                 ng-click="ec.appendChar(5)">5</div>
            <div class="button col"
                 ng-disabled="ec.isInputDisabled"
                 ng-click="ec.appendChar(6)">6</div>
            <div class="button col"
                 ng-disabled="ec.isInputDisabled"
                 ng-click="ec.appendChar('-')">-</div>
        </div>
        <div class="row">
            <div class="button col"
                 ng-disabled="ec.isInputDisabled"
                 ng-click="ec.appendChar(7)">7</div>
            <div class="button col"
                 ng-disabled="ec.isInputDisabled"
                 ng-click="ec.appendChar(8)">8</div>
            <div class="button col"
                 ng-disabled="ec.isInputDisabled"
                 ng-click="ec.appendChar(9)">9</div>
            <div class="button col"
                 ng-disabled="ec.isInputDisabled"
                 ng-click="ec.checkAnswer()">↵</div>
        </div>
    </div>
</ion-view>
```

6.5 Code Listing: index.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1,
                                   maximum-scale=1,
                                   user-scalable=no,
                                   width=device-width">
    <title>Train Your Brain</title>

    <link href="lib/ionic/css/ionic.css" rel="stylesheet">
    <link href="css/style.css" rel="stylesheet">

    <script src="js/platformOverrides.js"></script>
    <script src="lib/ionic/js/ionic.bundle.js"></script>
    <script src="cordova.js"></script>

    <script src="js/app.js"></script>
    <script src="js/controllers.js"></script>
    <script src="js/expressionFactory.js"></script>
</head>
<body ng-app="trainBrain">
    <ion-nav-view></ion-nav-view>
</body>
</html>
```

6.6 Code Listing: style.css

```
.scroll-bar-indicator {
    display: none;
}

.fixed-bottom{
  position: absolute;
  bottom: 0;
  width: 100%;
}

h2 {
  text-align: center;
}

.expression {
    padding-top: 40px;
}
```

APPENDIX 2: XAMARIN APP IMPLEMENTATION

6.7 Code Listing: App.cs

```
using Xamarin.Forms;

namespace TrainYourBrain.Core
{
    public class App : Application
    {
        public App(ITimer timer)
        {
            var expressionFactory = new ExpressionFactory();
            var mainPageViewModel = new MainPageViewModel(expressionFactory, timer);
            MainPage = new MainPage(mainPageViewModel);
        }
    }
}
```

6.8 Code Listing: ExpressionFactory.cs

```
using System;

namespace TrainYourBrain.Core
{
    public class ExpressionFactory
    {
        int numberRange = 8;
        char[] operators = { '+', '-' };

        public Expression CreateExpression()
        {
            var random = new Random();

            var operand1 = random.Next(numberRange);
            var operand2 = random.Next(numberRange);

            var operatorSign = operators[random.Next(operators.Length)];

            numberRange++;
            return new Expression(operand1, operand2, operatorSign);
        }

        public void Reset()
        {
            numberRange = 8;
        }
    }

    public class Expression
    {
        int operand1;
        int operand2;
        char operatorSign;

        public Expression(int operand1, int operand2, char operatorSign)
        {
            this.operand1 = operand1;
            this.operand2 = operand2;
            this.operatorSign = operatorSign;
        }

        public bool CheckAnswer(int userInput)
        {
            return userInput == Calculate();
        }

        int Calculate()
        {
            switch (operatorSign)
            {
                case '+':
                    return operand1 + operand2;
                default:
                    return operand1 - operand2;
            }
        }
        public override string ToString()
        {
            return $"{operand1} {operatorSign} {operand2} = ?";
        }
    }
}
```

6.9 Code Listing: ITimer.cs

```
using System;

namespace TrainYourBrain.Core
{
    public interface ITimer
    {
        void Start(Action toExecute, TimeSpan period, TimeSpan dueTime);
        void Stop();
    }
}
```

6.10 Code Listing: PlayerStats.cs

```
using System.ComponentModel;
using System.Runtime.CompilerServices;
using TrainYourBrain.Core.Annotations;

namespace TrainYourBrain.Core
{
    public class PlayerStats : INotifyPropertyChanged
    {
        int lives;
        int score;
        int timerCount;

        public int Lives
        {
            get { return lives; }
            set
            {
                if (lives == value) return;
                lives = value;
                OnPropertyChanged();
            }
        }

        public int Score
        {
            get { return score; }
            set
            {
                if (score == value) return;
                score = value;
                OnPropertyChanged();
            }
        }

        public int TimerCount
        {
            get { return timerCount; }
            set
            {
                if (timerCount == value) return;
                timerCount = value;
                OnPropertyChanged();
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;

        [NotifyPropertyChangedInvocator]
        protected virtual void OnPropertyChanged([CallerMemberName]
                                                 string propertyName = null)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

6.11 Code Listing: MainPageViewModel.cs

```
using System;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using System.Windows.Input;
using TrainYourBrain.Core.Annotations;
using Xamarin.Forms;

namespace TrainYourBrain.Core
{
    public class MainPageViewModel : INotifyPropertyChanged
    {
        ExpressionFactory expressionFactory;
        ITimer timer;

        PlayerStats playerStats;
        Expression currentExpression;
        string currentExpressionString;
        bool isInputEnabled;

        public PlayerStats PlayerStats
        {
            get { return playerStats; }
            set
            {
                if (playerStats == value) return;
                playerStats = value;
                OnPropertyChanged();
            }
        }

        public bool IsInputEnabled
        {
            get { return isInputEnabled; }
            set
            {
                if (isInputEnabled == value) return;
                isInputEnabled = value;
                OnPropertyChanged();
            }
        }

        public string ExpressionString
        {
            get { return currentExpressionString; }
            set
            {
                if (currentExpressionString == value) return;
                currentExpressionString = value;
                OnPropertyChanged();
            }
        }

        public ICommand StartCommand { get; set; }
        public ICommand CheckAnswerCommand { get; set; }
        public MainPageViewModel(ExpressionFactory expressionFactory, ITimer timer)
        {
            this.timer = timer;
            this.expressionFactory = expressionFactory;
            StartCommand = new Command(StartRound);
            CheckAnswerCommand = new Command<int>(CheckAnswer);

        }

        void InitializePlayerStats()
        {
            PlayerStats = new PlayerStats()
            {
                Lives = 3,
                Score = 0,
                TimerCount = 5
            };
        }

        void CheckAnswer(int answer)
        {
            if (PlayerStats.Lives == 0) return;
            if (currentExpression.CheckAnswer(answer))
            {
                PlayerStats.Score++;
            }
            else
            {
                PlayerStats.Lives--;
            }
            DeployExpression();
            PlayerStats.TimerCount = 5;
        }

        void StartRound()
        {
            expressionFactory.Reset();
            IsInputEnabled = true;
            timer.Stop();
            InitializePlayerStats();
            DeployExpression();
            timer.Start(OnTick, TimeSpan.FromSeconds(1), TimeSpan.FromSeconds(1));
        }

        void DeployExpression()
        {
            if (PlayerStats.Lives != 0)
            {
                currentExpression = expressionFactory.CreateExpression();
                ExpressionString = currentExpression.ToString();
            }
            else
            {
                EndGame();
            }
        }
```

```
        void EndGame()
        {
            timer.Stop();
            IsInputEnabled = false;
            ExpressionString = $"Game over. Your score: {PlayerStats.Score}";
        }

        public event PropertyChangedEventHandler PropertyChanged;

        [NotifyPropertyChangedInvocator]
        protected virtual void OnPropertyChanged([CallerMemberName]
                                                 string propName = null)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propName));
        }
    }
}
```

6.12 Code Listing: MainPage.xaml

```xml
<?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="TrainYourBrain.Core.MainPage">
  <StackLayout VerticalOptions="FillAndExpand">

    <Button Text="Start" Command="{Binding StartCommand}" Clicked="Start_OnClicked"/>
    <StackLayout Orientation="Horizontal" VerticalOptions="StartAndExpand">
      <Grid HorizontalOptions="FillAndExpand">
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="*"/>
          <ColumnDefinition Width="*"/>
          <ColumnDefinition Width="*"/>
          <ColumnDefinition Width="*"/>
          <ColumnDefinition Width="*"/>
          <ColumnDefinition Width="auto"/>
        </Grid.ColumnDefinitions>
        <Label Grid.Column="0" Text="Lives:"/>
        <Label x:Name="Lives" Text="{Binding PlayerStats.Lives}" Grid.Column="1"/>
        <Label Grid.Column="2" Text="Timer:"/>
        <Label x:Name="Timer" Text="{Binding PlayerStats.TimerCount}" Grid.Column="3"/>
        <Label Grid.Column="4" Text="Score:"/>
        <Label x:Name="Score" Text="{Binding PlayerStats.Score}" Grid.Column="5"/>
      </Grid>
    </StackLayout>

    <Label Text="{Binding ExpressionString}"
           VerticalOptions="CenterAndExpand"
           HorizontalOptions="CenterAndExpand"
           FontSize="32"/>

    <Grid x:Name="InputGrid" VerticalOptions="EndAndExpand" >
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*"/>
        <ColumnDefinition Width="*"/>
        <ColumnDefinition Width="*"/>
        <ColumnDefinition Width="*"/>
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
      </Grid.RowDefinitions>

      <Label x:Name="AnswerField"
             FontSize="24"
             HorizontalTextAlignment="Center"
             Grid.Row="0"
             Grid.Column="0"
             Grid.ColumnSpan="3"/>
      <Button Text="←" Grid.Row="0" Grid.Column="3" Clicked="Erase_OnClicked"/>

      <Button Text="1" Grid.Row="1" Grid.Column="0" Clicked="AppendChar_OnClicked"/>
      <Button Text="2" Grid.Row="1" Grid.Column="1" Clicked="AppendChar_OnClicked"/>
      <Button Text="3" Grid.Row="1" Grid.Column="2" Clicked="AppendChar_OnClicked"/>
      <Button Text="0" Grid.Row="1" Grid.Column="3" Clicked="AppendChar_OnClicked"/>
      <Button Text="4" Grid.Row="2" Grid.Column="0" Clicked="AppendChar_OnClicked"/>
      <Button Text="5" Grid.Row="2" Grid.Column="1" Clicked="AppendChar_OnClicked"/>
      <Button Text="6" Grid.Row="2" Grid.Column="2" Clicked="AppendChar_OnClicked"/>
      <Button Text="-" Grid.Row="2" Grid.Column="3" Clicked="AppendChar_OnClicked"/>

      <Button Text="7" Grid.Row="3" Grid.Column="0" Clicked="AppendChar_OnClicked"/>
      <Button Text="8" Grid.Row="3" Grid.Column="1" Clicked="AppendChar_OnClicked"/>
      <Button Text="9" Grid.Row="3" Grid.Column="2" Clicked="AppendChar_OnClicked"/>
      <Button Text="↵" Grid.Row="3" Grid.Column="3" Clicked="SubmitAnswer_OnClicked"/>
    </Grid>
  </StackLayout>
</ContentPage>
```

6.13 Code Listing: MainPage.xaml.cs

```
using System;
using Xamarin.Forms;

namespace TrainYourBrain.Core
{
    public partial class MainPage : ContentPage
    {
        public MainPage(MainPageViewModel viewModel)
        {
            BindingContext = viewModel;
            InitializeComponent();
            this.Padding = new Thickness(10, Device.OnPlatform(20, 0, 10), 10, 5);

            foreach (View view in InputGrid.Children)
            {
                view.SetBinding(View.IsEnabledProperty, "IsInputEnabled");
            }
        }

        void AppendChar_OnClicked(object sender, EventArgs e)
        {
            AnswerField.Text += (sender as Button)?.Text;
        }

        void SubmitAnswer_OnClicked(object sender, EventArgs e)
        {
            if (AnswerField.Text.Length == 0) return;
            var viewModel = BindingContext as MainPageViewModel;
            viewModel?.CheckAnswerCommand.Execute(int.Parse(AnswerField.Text));
            ClearAnswerField();
        }

        void Erase_OnClicked(object sender, EventArgs e)
        {
            if (AnswerField.Text.Length == 0) return;
            AnswerField.Text = AnswerField.Text.Remove(AnswerField.Text.Length - 1);
        }

        void Start_OnClicked(object sender, EventArgs e)
        {
            ClearAnswerField();
        }

        void ClearAnswerField()
        {
            AnswerField.Text = String.Empty;
        }
    }
}
```

6.14 Code Listing: AndroidTimer.cs

```csharp
using System;
using System.Threading;
using TrainYourBrain.Core;

namespace TrainYourBrain.Xamarin.Droid
{
    public class AndroidTimer : ITimer
    {
        Timer timer;

        public void Start(Action toExecute, TimeSpan period, TimeSpan dueTime)
        {
            timer?.Dispose();
            timer = new Timer(state => toExecute(), null, dueTime, period);
        }

        public void Stop()
        {
            timer?.Dispose();
        }
    }
}
```

6.15 Code Listing: MainActivity.cs

```
using Android.App;
using Android.Content.PM;
using Android.OS;
using TrainYourBrain.Core;

namespace TrainYourBrain.Xamarin.Droid
{
    [Activity(Label = "TrainYourBrain.Xamarin", Icon = "@drawable/icon", MainLauncher =
true, ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]
    public class MainActivity :
global::Xamarin.Forms.Platform.Android.FormsApplicationActivity
    {
        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);

            global::Xamarin.Forms.Forms.Init(this, bundle);
            LoadApplication(new App(new AndroidTimer()));
        }
    }
}
```

6.16 Code Listing: AppDelegate.cs

```
using Foundation;
using TrainYourBrain.Core;
using UIKit;

namespace TrainYourBrain.Xamarin.iOS
{
    [Register("AppDelegate")]
    public partial class AppDelegate :
global::Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
    {
        public override bool FinishedLaunching(UIApplication app, NSDictionary options)
        {
            global::Xamarin.Forms.Forms.Init();
                     LoadApplication(new App(new IosTimer()));

            return base.FinishedLaunching(app, options);
        }
    }
}
```

6.17 Code Listing: Main.cs

```
using UIKit;

namespace TrainYourBrain.Xamarin.iOS
{
    public class Application
    {
        static void Main(string[] args)
        {
            UIApplication.Main(args, null, "AppDelegate");
        }
    }
}
```

6.18 Code Listing: Windows.App.cs

```
using System;
using Windows.ApplicationModel;
using Windows.ApplicationModel.Activation;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Navigation;
using XamarinForms = Xamarin.Forms;

namespace TrainYourBrain.Xamarin.Windows
{
    sealed partial class App : Application
    {
        public App()
        {
            Microsoft.ApplicationInsights.WindowsAppInitializer.InitializeAsync(
                Microsoft.ApplicationInsights.WindowsCollectors.Metadata |
                Microsoft.ApplicationInsights.WindowsCollectors.Session);
            this.InitializeComponent();
            this.Suspending += OnSuspending;
        }

        protected override void OnLaunched(LaunchActivatedEventArgs e)
        {

#if DEBUG
            if (System.Diagnostics.Debugger.IsAttached)
            {
                this.DebugSettings.EnableFrameRateCounter = true;
            }
#endif

            Frame rootFrame = Window.Current.Content as Frame;

            if (rootFrame == null)
            {
                rootFrame = new Frame();

                rootFrame.NavigationFailed += OnNavigationFailed;

                XamarinForms.Forms.Init(e);

                if (e.PreviousExecutionState == ApplicationExecutionState.Terminated)
                {
                    //TODO: Load state from previously suspended application
                }

                Window.Current.Content = rootFrame;
            }

            if (rootFrame.Content == null)
            {
                rootFrame.Navigate(typeof(MainPage), e.Arguments);
            }
            Window.Current.Activate();
        }
        void OnNavigationFailed(object sender, NavigationFailedEventArgs e)
        {
            throw new Exception("Failed to load Page " + e.SourcePageType.FullName);
        }

        private void OnSuspending(object sender, SuspendingEventArgs e)
        {
            var deferral = e.SuspendingOperation.GetDeferral();
            //TODO: Save application state and stop any background activity
            deferral.Complete();
        }
    }
}
```

6.19 Code Listing: Windows.MainPage.xaml

```
<forms:WindowsPage
    x:Class="TrainYourBrain.Xamarin.Windows.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:TrainYourBrain.Xamarin.Windows"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:forms="using:Xamarin.Forms.Platform.UWP"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

    </Grid>
</forms:WindowsPage>
```