Bachelor's Thesis

Information Technology

Embedded Systems

2016

Timo Jääskeläinen

CROSS-PLATFORM MOBILE APPLICATION DEVELOPMENT WITH XAMARIN.FORMS

OPINNÄYTETYÖ AMK | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikka | Sulautetut Järjestelmät

2016 | 44

Ohjaaja: Yliopettaja TkL Jari-Pekka Paalassalo

Timo Jääskeläinen

ALUSTARIIPPUMATON PUHELINSOVELLUSKEHITYS XAMARIN.FORMS-ALUSTALLA

Tämän opinnäytetyön tavoitteena oli tutkia ja esittää tapoja kehittää alustariippumattomia puhelinsovelluksia käyttäen Xamarin.Forms-alustaa. Tämä teknologia oli valittuna koska Xamarin antaa sovelluskehittäjille mahdollisuuden luoda natiivisovelluksia Windows Phone-, iOS- ja Android-mobiilialustoille nopeammin kuin koskaan.

Tässä opinnäytetyössä esitettiin Xamarin.Forms-alustan tarjoamia ratkaisuja kehittämällä esimerkkisovelluksen Android-, iOS- ja Windows Phone-mobiilialustoille.

Yhteenvedossa todetaan Xamarin.Forms-alustan antavan sovelluskehittäjille hyvät mahdollisuudet koodin uudelleenkäyttämiselle eri mobiilialustojen välillä. Työssä luotu esimerkkisovellus todistaa tämän väitteen.

ASIASANAT:

mobiilisovelluskehitys, alustariippumaton kehitys, Xamarin, ohjelmistokehitys

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Embedded Systems

2016 | 44

Instructor: Principal Lecturer Jari-Pekka Paalassalo Lic.Tech.

Timo Jääskeläinen

CROSS-PLATFORM MOBILE APPLICATION DEVELOPMENT WITH XAMARIN.FORMS

The objective of this thesis was to cover ways of building cross-platform mobile applications using Xamarin.Forms platform. This technology was chosen as a topic because Xamarin gives developers the possibility to create native applications for Windows Phone, iOS and Android mobile platforms faster than ever before.

The goals of this thesis were to make an analysis of Xamarin.Forms framework and prove its capabilities by creating a prototype for Android, iOS and Windows Phone.

As a conclusion, it is stated that Xamarin.Forms has the power to provide developers with the advantage of increasing the amount of code-reuse between platforms. The prototype created proves this concept.

KEYWORDS:

mobile development, cross-platform, Xamarin, software development

CONTENT

APPENDICES

PICTURES

LIST OF ABBREVIATIONS

API	Application programming interface
CSS	Cascading style sheets
GUI	Graphical user interface
HTML	Hyper text markup language
IDE	Integrated development environment
IL	Intermediate language
JIT	Just-in-time
MVVM	Model-view-viewmodel
NFC	Near field communication
UI	User interface
UX	User experience
XAML	Extensible application markup language

1 INTRODUCTION

The objective of this thesis is to cover ways of building cross-platform mobile applications using Xamarin Forms. Over the years, developers have tried different approaches to create mobile applications. Modern web technologies such as HTML5 and JavaScript used to be popular because the apps ran within a native browser, thus implicitly providing cross-platform support. However, it turned out that HTML5 did not offer the smooth user experience that mobile app users were expecting. Other web based technologies appeared afterwards (Apache Cordova, Sencha Touch), but the expectations of high performance and good user experience were still unmet.

Xamarin is a framework that allows developers to create mobile applications for Android, iOS and Windows Phone using C#.

When building the user interface, the developers have the possibility to use native controls of each mobile platform. That makes it easier to customize the UI's look and feel for individual mobile platforms and to provide a native experience for the end user.

Xamarin enables developers to create mobile app prototypes rapidly and effortlessly. That's because Xamarin Forms applications are native and can benefit from other APIs and features of the underlying mobile platform. For example, an application made with Xamarin Forms can make use of PassKit, Storekit, CoreMotion and other similar features on iOS. The apps can also utilize Tiles on Windows Phone or Google Play Services and NFC on Android.

Furthermore, user interfaces can be created in two specific ways with Xamarin Forms. Firstly, programmers can make use of the API provided by the framework to build user interface views with a source code. Secondly, user interfaces can be created using Extensible Application Markup Language (XAML). XAML is a declarative markup language that is used in Xamarin to define the visual contents of a user interface. XAML is especially appropriate for use with the popular MVVM (Model-View-ViewModel) application architecture: XAML defines the View that is linked to ViewModel code through XAML-based data bindings.

With Xamarin Forms, developers can create applications with several pages. Pages can be considered as the application's screens. Depending on the mobile platform, a page can be a View Controller in iOS, an activity in Android, or a Page in Windows Phone.

The UI can be customized by using types of pages like content page, navigation page, tabbed page, carousel page, or master detail page.

Another advantage is using Xamarin Forms is the variety of layouts. A layout can be used as a container for both views and other containers. Types of layout include stack, absolute, relative, or grid layout.

Among the user interface elements that developers can use, Xamarin.Forms has over 40 views as elements. Developers can use the following controls for different interactions: Entry as a single-line form element to receive user input, Button to initiate a user command, Image to display images to the user, and ListView to display information in cells.

Overall, mobile app developers can create cross-platform applications easily and rapidly using Xamarin.Forms. More aspects of this platform will be presented in the next chapters of this thesis.

2 MOBILE PLATFORM

Mobile application development business is growing bigger and bigger every year. More and more, users are resorting to smaller devices for personal computing, which makes it necessary to have applications that provide a great user experience and high performance. People use smaller devices nowadays to do most of personal computing: for quick information, media consumption and social networking.

3 PROBLEM

The main problem that occurs due to a high usage of mobile devices is that applications need to work well on all mobile platforms. For programmers, it is very time consuming to take a working program and rewrite it in an entirely different programming language or port it to another operating system with a different application programming interface (API).

If programmers work on two separate platforms (for example iOS and Android), the applications tend to drift apart from each other. For each platform, programmers need to use a different language, different architecture and different conventions. That leads to further complications in the future when new features are added to the applications. However, if programmers use Xamarin, they can do all the future maintenance work, all the revisions and enhancements just once rather than twice.

4 CROSS-PLATFORM DEVELOPMENT

Cross-platform development refers to the development of mobile applications that can be used on multiple platforms. These include:

- The Apple family of iPhones and iPads, all of which run the iOS operating system
- The Android operating system, developed by Google based on the Linux kernel, which runs on a variety of phones and tablets.
- Microsoft’s Windows Phone.

Among these, Android and iOS are the most popular for users. Windows Phone is less used, but it is still quite popular in Finland. The ideal strategy for developers to tackle functionality issues with their apps is to target more than just one of these platforms. There are several inconveniences with cross-platform development. (Net Applications 2016)

4.1 Cross-platform development complications

4.1.1 Different user interface paradigms

The first obstacle that developers deal with when creating an application is that there are different standards for each mobile platform. To begin with, navigation around applications and pages works differently. Then, the presentation of data is made following different sets of conventions. There are different ways to invoke and display menus and different approaches to touch.

Developers need to make sure that the applications have the same kind of functionality on all platforms in order to satisfy the end users. Once a user is accustomed to interacting with applications on a certain platform, he will expect the applications to work in the same way on other platforms.

4.1.2 Different development environments

Another issue that comes to surface when doing cross-platform development is the fact that IDEs are different for all three mobile platforms. An IDE is a sophisticated integrated development environment. An integrated development environment (IDE) is a programming environment that has been packaged as an application program, which usually consists of a code editor, a compiler, a debugger, and a graphical user interface (GUI) builder. The IDE may be a standalone application or may be included as part of one or more existing and compatible applications.

There are different IDEs for every mobile platform:

- iOS: Xcode
- Android: Android Studio
- Windows Phone: Visual Studio

4.1.3 Different programming interfaces

A third obstacle for developers is the existence of different APIs for every platform. There might be similar types of user-interface objects on all three platforms, but they have different names.

As an example, every platform has different names for a component that allows displaying text in the user interface:

- iOS: UILabel
- Android: TextView
- Windows Phone: TextBlock

4.1.4 Different programming languages

The forth important issue is the differences between programming language. Each platform is associated with a specific language:

- iOS: Objective-C or Swift

- Android: Java
- Windows Phone: C#

This issue means that companies need to find resources to hire specialized developers for each platform. Xamarin.Forms comes as a solution to this problem.

4.2 Xamarin platform

There are various Xamarin libraries developers can use to ease their work. For example:

- Xamarin.Mac for developing Mac applications.
- Xamarin.iOS for developing iOS applications.
- Xamarin.Android for developing Android applications.

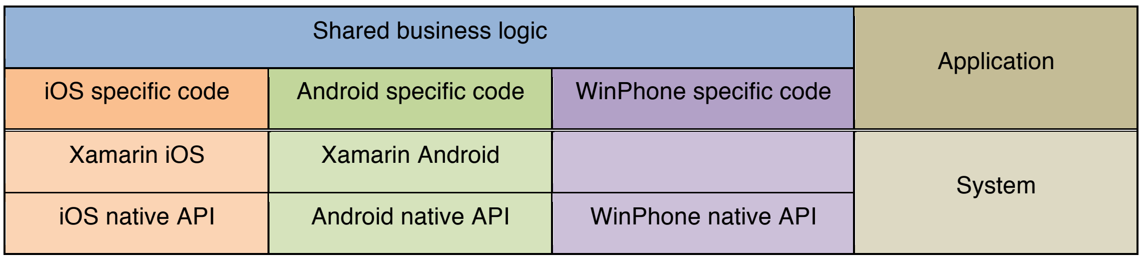
These libraries make up the Xamarin platform. They consist of .NET versions of the native Mac, iOS, and Android APIs. To target the native APIs of these platforms, developers can write applications in C# while also having access to the .NET Framework class library. (Xamarin 2016)

Another useful tool is Xamarin Studio, an IDE that runs on both the Mac and PC. It allows you to develop Phone and Android applications on the Mac and Android applications on the PC

Visual Studio can be used with Xamarin libraries to develop Mac, iPhone, Android, as well as Windows and Windows Phone applications. Still, Mac and iPhone development also requires a Mac with Xcode and Xamarin Studio installed and connected through a local network with the PC.

4.3 Sharing code

Xamarin allows developers to share code among the applications.



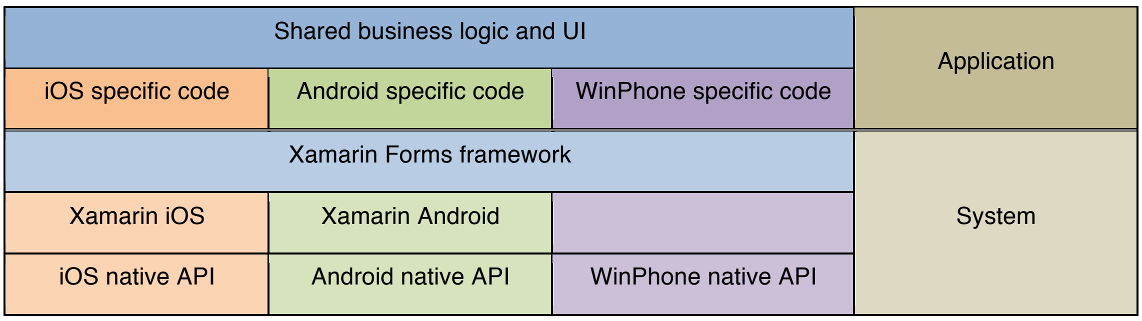
Picture 1. Xamarin based solution architecture

When the iPhone app is built, the Xamarin C# compiler makes use of the Apple compiler on the Mac to generate native iPhone machine code just like the Objective-C compiler. The advantage here is that the calls from the app to the iPhone APIs are the same as if the application had been written in Objective-C.

Regarding the Android app, the Xamarin C# compiler generates IL, which runs on a version of Mono on the device alongside the Java engine, but the API calls from the app are the same as though the app were written in Java. (Xamarin 2016)

4.3.1 Xamarin.Forms

Xamarin.Forms was introduced on May 28, 2014 as part of a collection of improvement to the Xamarin platform dubbed Xamarin 3. With Xamarin.Forms, developers can write UI code that can be compiled for all three mobile platforms: iPhone, Android, and Windows Phone.



Picture 2. Xamarin.Forms based solution architecture

As Charles Petzold mentions in his book Creating Mobile Apps with Xamarin.Forms, "Xamarin.Forms is an API that virtualizes the user-interface paradigms on each platform. Button, Switch, and Slider all have different appearances on the three phones because they are all rendered with the object specific to each platform". (Petzold 2016, 10)

Xamarin.Forms API contains a lot of functionality, which is expanded by third party plugins. Things such as persistent storage, location services, localization, telephony services can all be managed via a cross-platform API. However, some platform specific functionality must be managed separately for each platform. In this case, Xamarin Forms API provides a possibility to do it using DependencyService, where user defines cross-platform interface for a given feature and then implements this interface separately for each platform in their respective platform specific projects.

Xamarin.Forms UI controls are customizable to some extent, but for deeper customization user can create platform specific custom renderers, which allow native level UI customization.

Applications that rely on heavily custom graphical user interface and complex touch interactions should not use Xamarin Forms.

Developers use Xamarin.Forms for quick prototyping, but having in mind that they can continue using the Xamarin.Forms features to build the whole application.

4.3.2 Comparison to other cross-platform development solutions

For the comparison four of the currently most popular cross-platform mobile development solutions are chosen: PhoneGap, Appcelerator, ReactNative and Xamarin. Each solution has its own strengths and weaknesses and there is no tool that is best in for every type of project. The platform should be chosen depending on app performance requirements, UX/UI experience and the programming language of choice.

4.3.2.1 PhoneGap

PhoneGap allows developing mobile applications using JavaScript, HTML and CSS. The web based application is packed inside a native application shell, that uses embedded web browser instance to display the web content inside of the application. PhoneGap provides a bridge between JavaScript API and the native API, so the JavaScript code is able to use platform native functionality.

The advantage of PhoneGap is that it supports very wide variety of mobile platforms. This is due to the fact that most smartphone platforms nowadays support running browser instance inside of an application, which allows running web applications inside of a native application shell.

There are certain disadvantages in using PhoneGap based approach. One of them is HTML rendering performance on mobile devices. Another one is the fact that native UI components often have certain behaviors that are difficult to replicate precisely in web applications. Many users tend to notice these slight differences and might perceive them as a bug or bad application design. (Optimus Information 2015)

4.3.2.2 Appcelerator and ReactNative

Both Appcelerator and ReactNative allow writing cross-platform applications using JavaScript, while using native UI components for rendering at runtime. These applications come with an embedded JavaScript engine that interprets the JavaScript code at runtime and framework bridges the JavaScript code with the native API's of the platform that it is running on.

The fact that Appcelerator and ReactNative based solutions use native UI components gives it a certain advantage compared to PhoneGap based solutions. There's no longer need to render complex web application using HTML and CSS, so rendering performance is improved. Also since Appcelerator and ReactNative apps use native UI components, the user experience is potentially more familiar compared to UI components which are built with web technologies.

Having a JavaScript engine run inside of a native application shell on a mobile device, interpreting a complex JavaScript based application doesn't come without challenges. Launch time of the application becomes longer due to the need of starting up JavaScript engine and the underlying bridging frameworks. In addition, potential memory management issues tend to me a lot more difficult to debug than using native approach. (Optimus Information 2015)

4.3.2.3 Xamarin with Xamarin.Forms

When using Xamarin, the applications are normally written using C# or F# programming languages. The code is then compiled to each platforms native code either at compile time for iOS and Windows Phone, or using JIT compilation for Android. The Xamarin platform provides access to all native functionality and the user interface is rendered using native components.

Xamarin based solutions have performance that is equivalent to the native code, while allowing developers to share a significant portion of their codebase between different platforms. Main drawback of using Xamarin is that it's a software layer built on top of the native frameworks, which means that developers might have to deal with the potential bugs in Xamarin framework in addition to bugs in their own software and in the native frameworks.

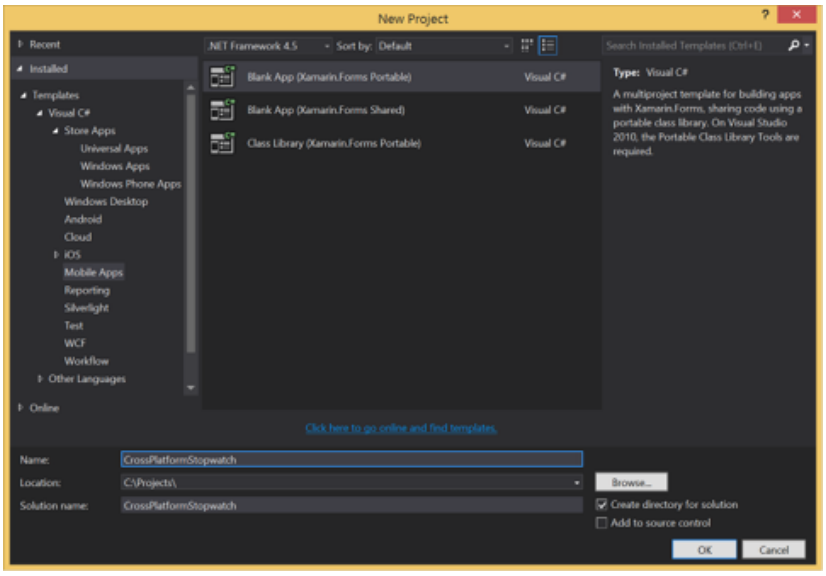
5 BUILDING CROSS-PLATFORM APPLICATION WITH XAMARIN.FORMS

This chapter introduces a sample project that includes several parts of a typical Xamarin.Froms mobile application:

- A multi-page user interface with tab/pivot navigation
- Various user interface components, such as text, buttons, lists and images
- Persistent storage
- Event handling
- Data binding

5.1 Creating a new solution

To create a new cross-platform mobile application using Xamarin Forms, start Visual Studio and select File -> New -> Project. From the project templates select Visual C# -> Mobile Apps -> Blank App (Xamarin.Forms Portable), type a name for the project and click OK.



.NET Framework 4.5 · Sort by: Default

Blank App (Xamarin.Forms Portable) · Visual C#

Type: Visual C#

A multiproject template for building apps with Xamarin.Forms, sharing code using a portable class library. On Visual Studio 2010, the Portable Class Library Tools are required.

Name: CrossPlatformStopwatch

Location: C:\Projects\

Solution name: CrossPlatformStopwatch

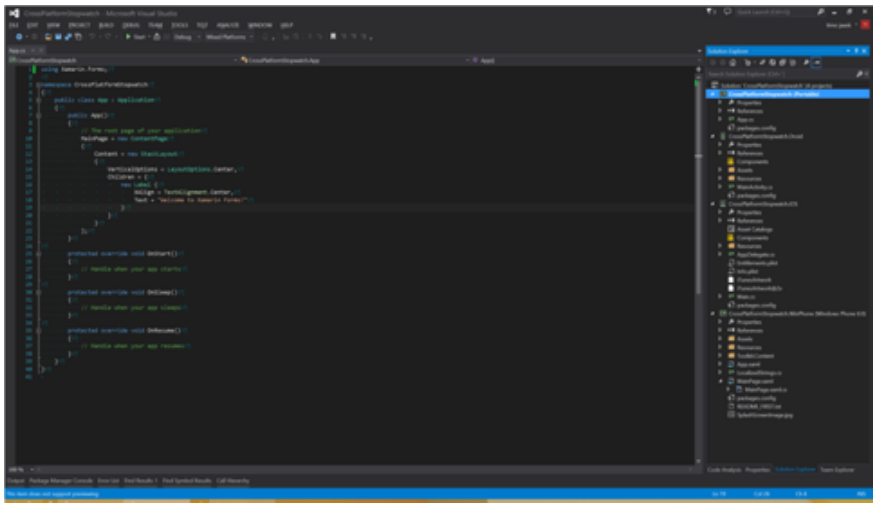
[✓] Create directory for solution

[ ] Add to source control

Picture 3. New project dialog

5.2 Xamarin.Forms solution structure

Visual Studio creates the basic Xamarin Forms solution structure that includes iOS, Android, Windows Phone as well as a Portable Class Library projects. The Portable Class Library project contains all the cross-platform code for the application, which includes application business logic and the user interface. Platform specific project (iOS, Android and Windows Phone) should contain only the platform specific could that could not otherwise be made using cross-platform API. Therefore, DependencyService implementations as well as custom renderers go into platform specific project. DependencyService and custom renderers are outside of the scope of this thesis.

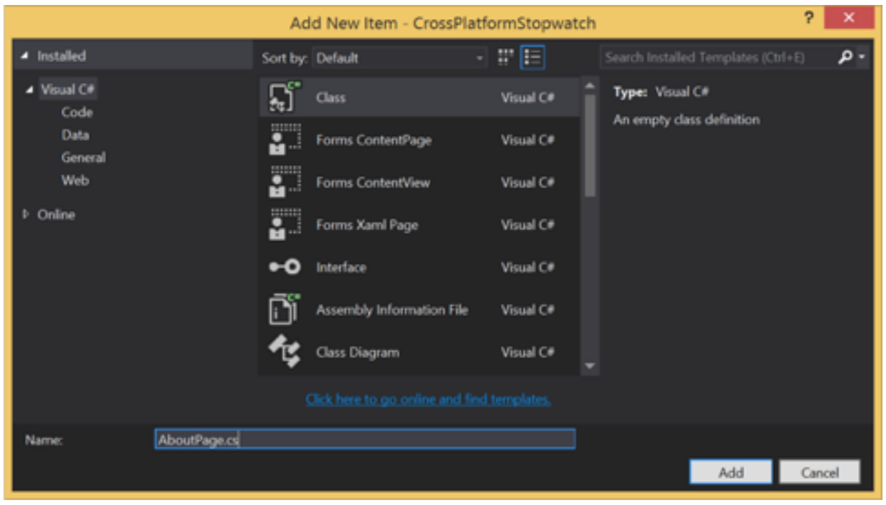


```
using Xamarin.Forms;

namespace CrossPlatformStopwatch
{
    public class App : Application
    {
        public App()
        {
            // The root page of your application
            MainPage = new ContentPage
            {
                Content = new StackLayout
                {
                    VerticalOptions = LayoutOptions.Center,
                    Children = {
                        new Label {
                            XAlign = TextAlignment.Center,
                            Text = "Welcome to Xamarin Forms!"
                        }
                    }
                }
            };
        }

        protected override void OnStart()
        {
            // Handle when your app starts
        }

        protected override void OnSleep()
        {
            // Handle when your app sleeps
        }

        protected override void OnResume()
        {
            // Handle when your app resumes
        }
    }
}
```

Picture 4. Newly created Xamarin.Forms project

5.3 Creating a basic UI

In the portable project, create new folder named Pages. In the new folder, create three classes named StopwatchPage, HistoryPage and AboutPage.


Add New Item - CrossPlatformStopwatch
Installed
Visual C#
Code
Data
General
Web
Online
Class
Forms ContentPage
Forms ContentView
Forms Xaml Page
Interface
Assembly Information File
Class Diagram
Visual C#
Search Installed Templates (Ctrl+E)
Click here to go online and find templates.
Add
Cancel

Sort by: Default

Type: Visual C#

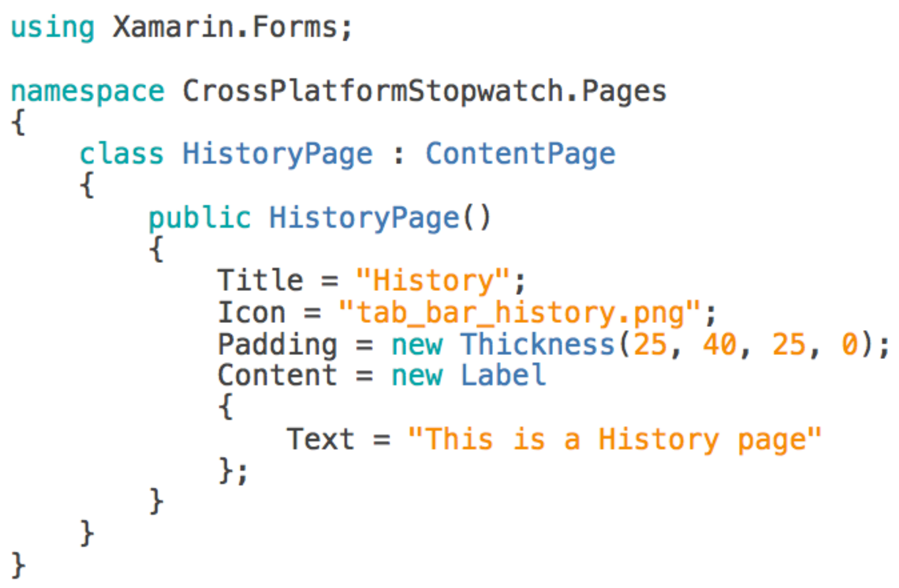
An empty class definition

Name: AboutPage.cs

Picture 5. New class dialog

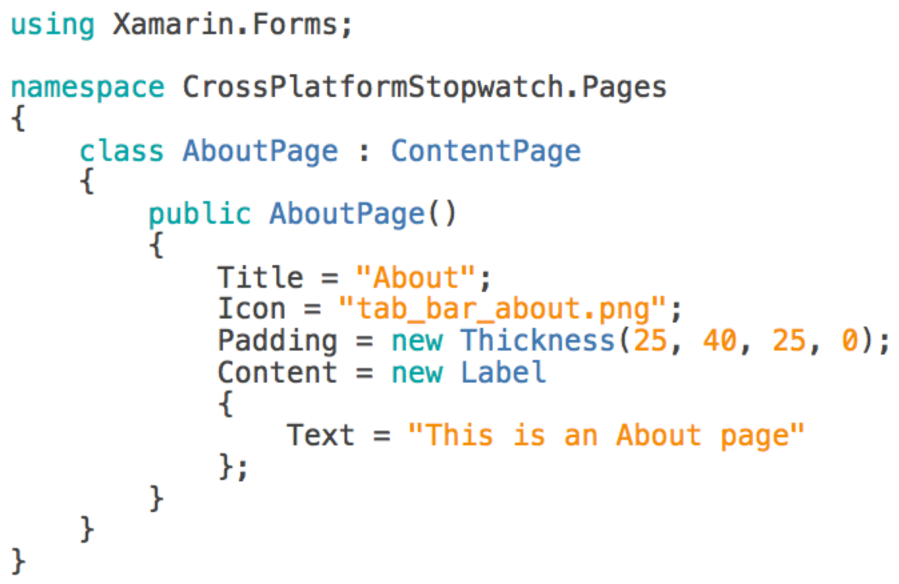
```
using Xamarin.Forms;

namespace CrossPlatformStopwatch.Pages
{
    class StopwatchPage : ContentPage
    {
        public StopwatchPage()
        {
            Title = "Stopwatch";
            Icon = "tab_bar_stopwatch.png";
            Padding = new Thickness(25, 40, 25, 0);
            Content = new Label
            {
                Text = "This is a Stopwatch page"
            };
        }
    }
}
```

Picture 6. Initial StopwatchPage code



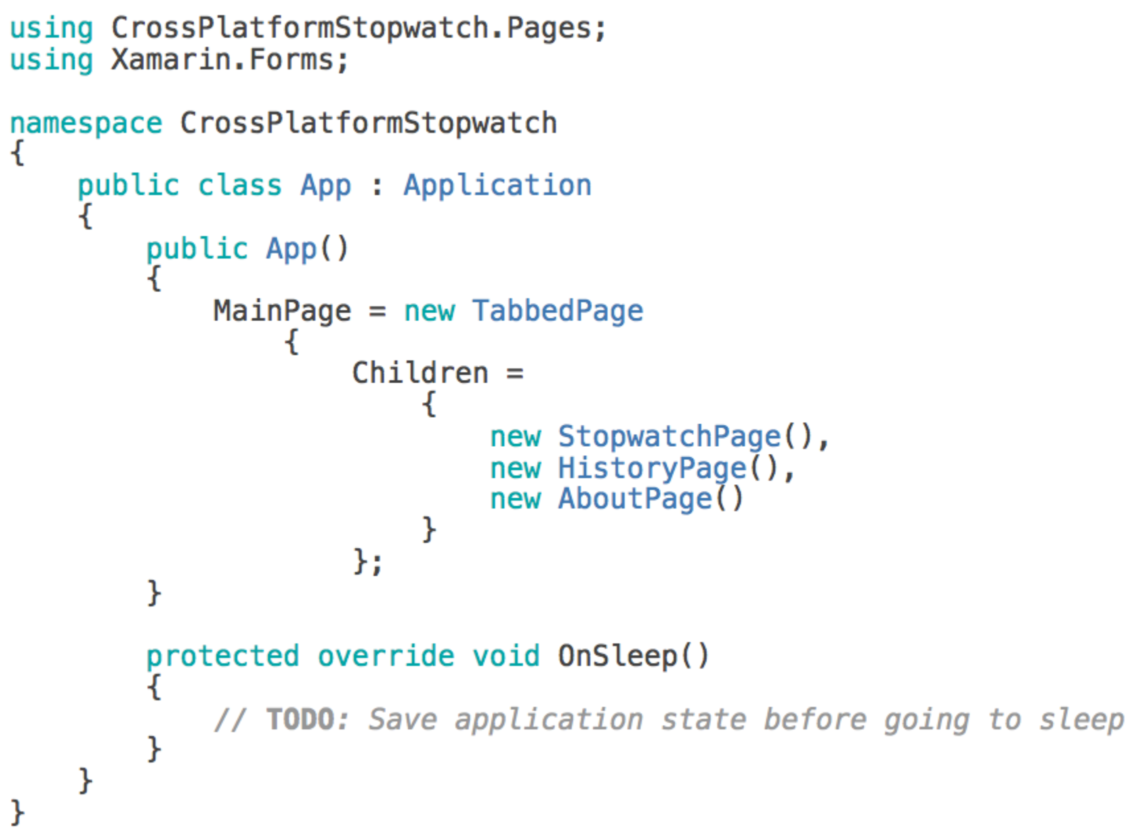
```
using Xamarin.Forms;

namespace CrossPlatformStopwatch.Pages
{
    class HistoryPage : ContentPage
    {
        public HistoryPage()
        {
            Title = "History";
            Icon = "tab_bar_history.png";
            Padding = new Thickness(25, 40, 25, 0);
            Content = new Label
            {
                Text = "This is a History page"
            };
        }
    }
}
```

Picture 7. Initial HistoryPage code



```
using Xamarin.Forms;

namespace CrossPlatformStopwatch.Pages
{
    class AboutPage : ContentPage
    {
        public AboutPage()
        {
            Title = "About";
            Icon = "tab_bar_about.png";
            Padding = new Thickness(25, 40, 25, 0);
            Content = new Label
            {
                Text = "This is an About page"
            };
        }
    }
}
```

Picture 8. Initial AboutPage code



```
using CrossPlatformStopwatch.Pages;
using Xamarin.Forms;

namespace CrossPlatformStopwatch
{
    public class App : Application
    {
        public App()
        {
            MainPage = new TabbedPage
                {
                    Children =
                        {
                            new StopwatchPage(),
                            new HistoryPage(),
                            new AboutPage()
                        }
                    };
        }

        protected override void OnSleep()
        {
            // TODO: Save application state before going to sleep
        }
    }
}
```

Picture 9. Initial App class code

Adding iOS tab bar icons: in the CrossPlatformStopwatch.iOS, right click on Resources folder and select Add —> Existing Item…

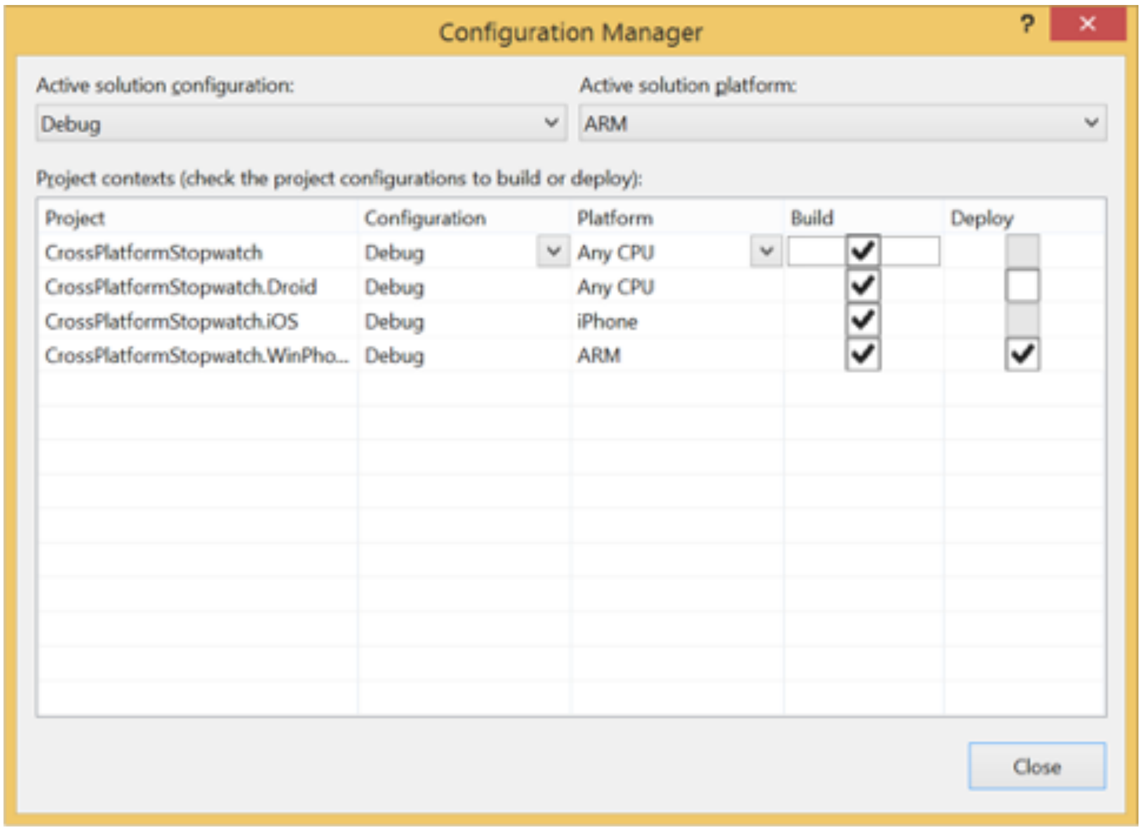
Add files tab_bar_stopwatch@2x.png, tab_bar_history@2x.png and tab_bar_about@2x.png

[or equivalent files of size approximately 50x50 pixels]

5.4 Testing the application

Set the appropriate build configuration for CrossPlatformStopwatch solution. Build —> Configuration Manager…

Check the Build checkbox next to every project in the solution.


Configuration Manager

Active solution configuration: Debug

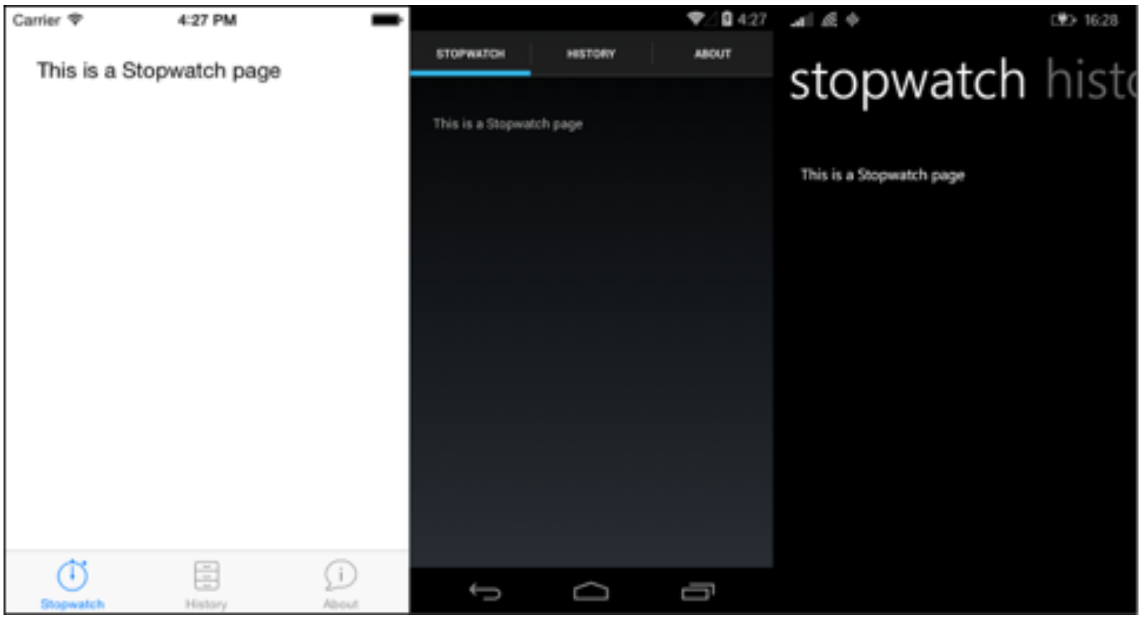
Active solution platform: ARM

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build	Deploy
CrossPlatformStopwatch	Debug	Any CPU	✓	
CrossPlatformStopwatch.Droid	Debug	Any CPU	✓	☐
CrossPlatformStopwatch.iOS	Debug	iPhone	✓	
CrossPlatformStopwatch.WinPho...	Debug	ARM	✓	✓

Picture 10. Build configuration manager

To run the application on a given platform, first you have to set the platform as a startup project. For example, to test on Windows Phone, right click on CrossPlatformTimer.WinPhone project and select Set as StartUp Project. Now you can run the application by selecting Debug —> Start Debugging.



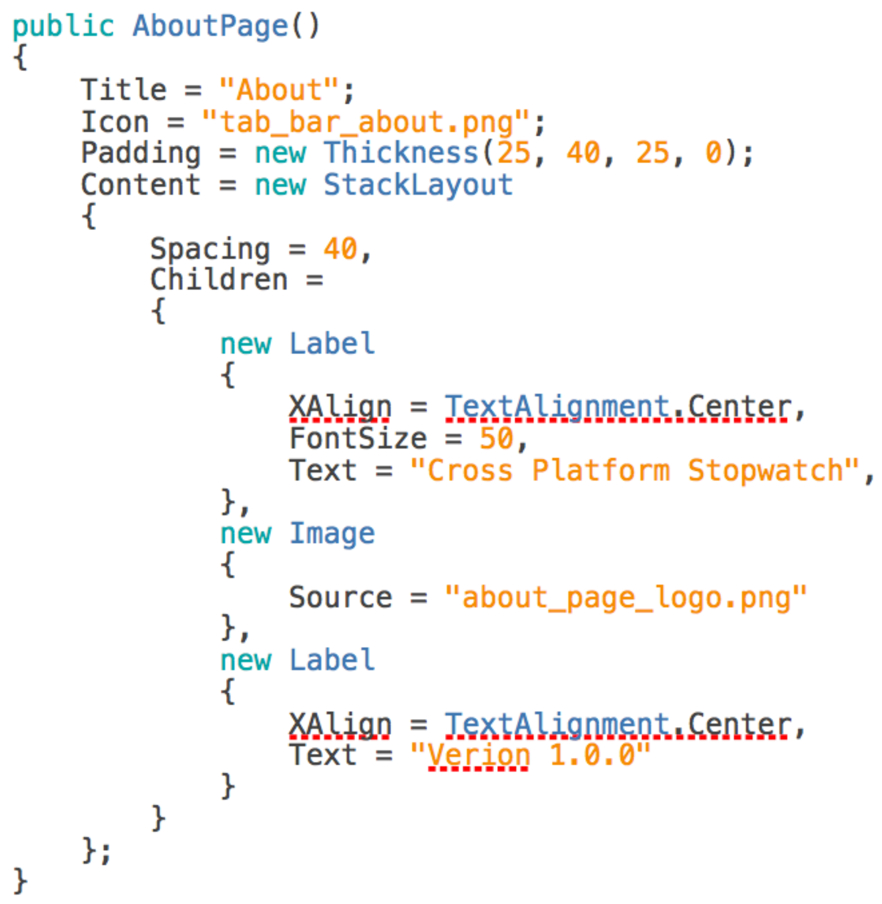
Picture 11. Stopwatch page rendered on three different platforms

The user interface looks completely different on each platform, even though it was produced from exactly the same code. This is because Xamarin.Forms uses different native implementations for each platform and is the strength of the Xamarin platform. Writing the user interface code only once, Xamarin gives up three user interface implementations that look familiar to the users and look according to the design trends of each platform.

5.5 Creating an About Page

The About page is going to contain the title of the application, application logo as well as the version number of the application. Xamarin.Forms' Label control is going to be used to display the application title and version, and Image control to display the application logo. A StackLayout container control is going to be used to position these controls on the page, stacked vertically, one after another.

Change the Content of the AboutPage from Label to StackLayout containing two Labels controls and one Image control.

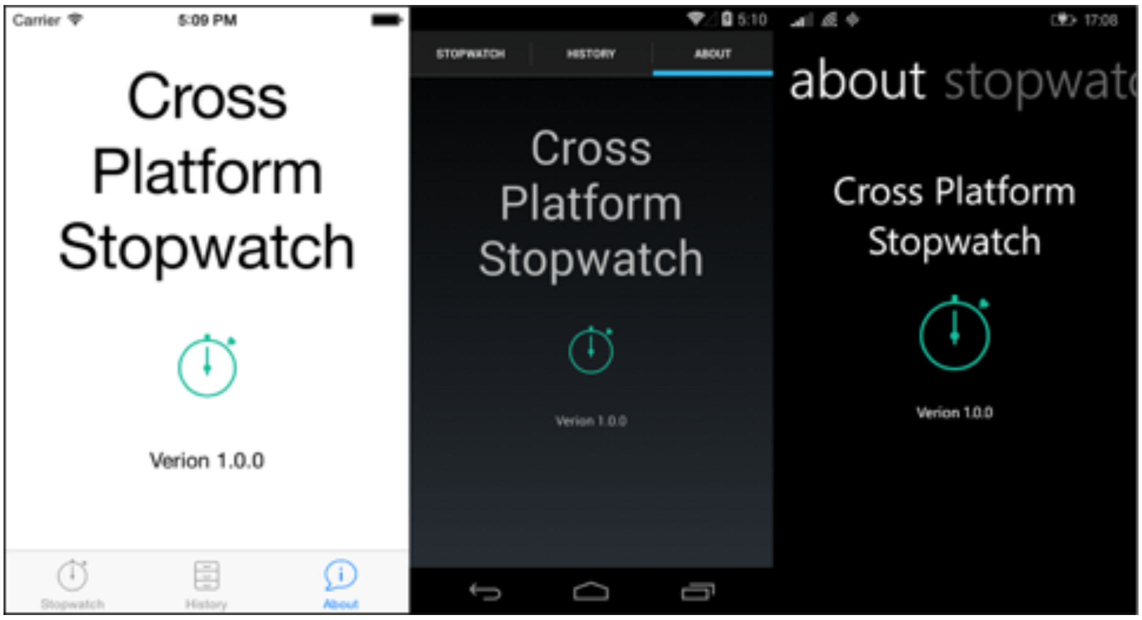


```
public AboutPage()
{
    Title = "About";
    Icon = "tab_bar_about.png";
    Padding = new Thickness(25, 40, 25, 0);
    Content = new StackLayout
    {
        Spacing = 40,
        Children =
        {
            new Label
            {
                XAlign = TextAlignment.Center,
                FontSize = 50,
                Text = "Cross Platform Stopwatch",
            },
            new Image
            {
                Source = "about_page_logo.png"
            },
            new Label
            {
                XAlign = TextAlignment.Center,
                Text = "Verion 1.0.0"
            }
        }
    };
}
```

Picture 12. Finished AboutPage code

5.6 Testing the About Page

The About page can now be tested by running the application on each platform and navigating to the About tab.



Picture 13. AboutPage rendered on three different platforms

5.7 Creating a Stopwatch Timer Page

The stopwatch can be started, stopped and reset by user. The user will be able to see the time that has passed since the stopwatch was started. Therefore, the Stopwatch page is going to host a passed time value (Label) as well as Start/Stop and Reset buttons (Button). The stopwatch logic will be separated from the user interface into its own class StopwatchViewModel and it will implement INotify-PropertyChanged to allow data binding. Data binding will be used to update the passed time value. The interface of StopwatchViewModel will be as follows:

Events:

- TimerRunningChanged(bool isRunning)

Properties:

- (TimeSpan) Time

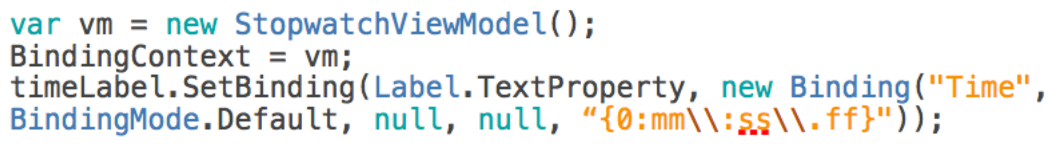
Methods:

- (void) StartStop()

- (void) Reset()

The full code listing of StopwatchViewModel is shown in appendix 4.

The Time property of the StopwatchViewModel contain the passed time of the stopwatch and will be displayed on the StopwatchPage using data binding:



```
var vm = new StopwatchViewModel();
BindingContext = vm;
timeLabel.SetBinding(Label.TextProperty, new Binding("Time",
BindingMode.Default, null, null, “{0:mm\\:ss\\.ff}"));
```

Picture 14. Binding time property to time label

The full code listing of StopwatchPage is shown in appendix 1.

5.8 Cross-Platform Presistent Storage

Portable Class Library does not have support for many platform specific features, such as geolocation, persistent storage, accelerometer, etc. Normally these parts of the application would have to be coded separately for each platform. However, there are a lot of community built plugins for Xamarin, that help to avoid writing a lot of device specific code separately for each platform. One of the websites containing such plugins is James Montemagno Xamarin.Plugins repository on GitHub (https://github.com/jamesmontemagno/Xamarin.Plugins). One of the plugins that it provides is called a Settings plugin, which basically provides access to device’s local storage directly from the portable class library, without having to write a single line of platform specific code. Installation of the plugin is straightforward:

- In Visual Studio, select Tools —> NuGet Package Manager —> Package Manager Console
- In the console make sure that default project is set to “CrossPlatformStopwatch”
- Install the Storage plugin by typing “Install-Package Xam.Plugins.Settings” and

pressing Enter key

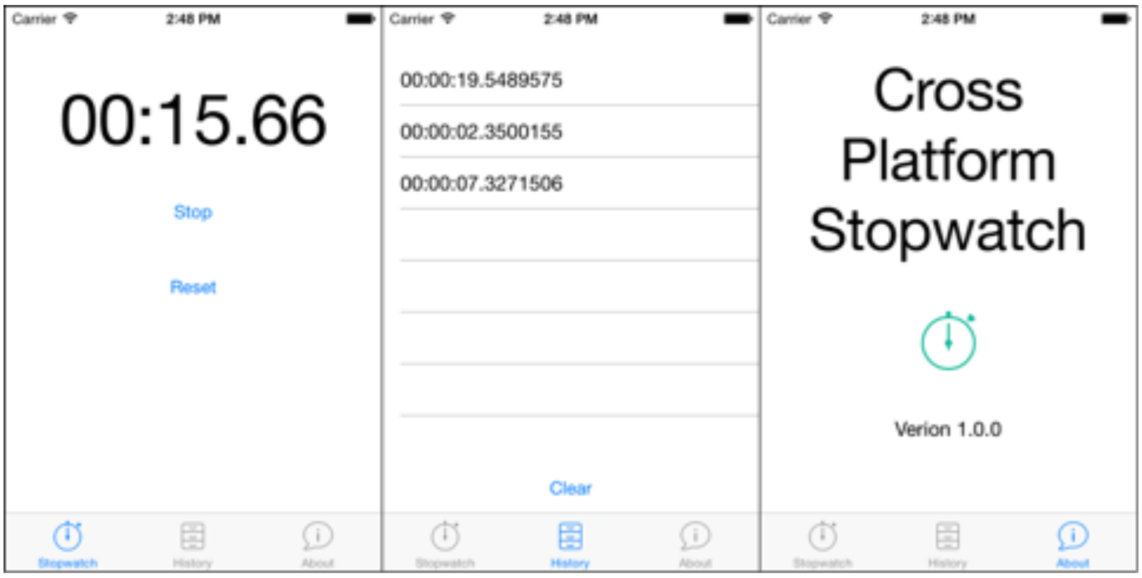
- Repeat the last step for each project in the solution

5.9 Adding the application logo

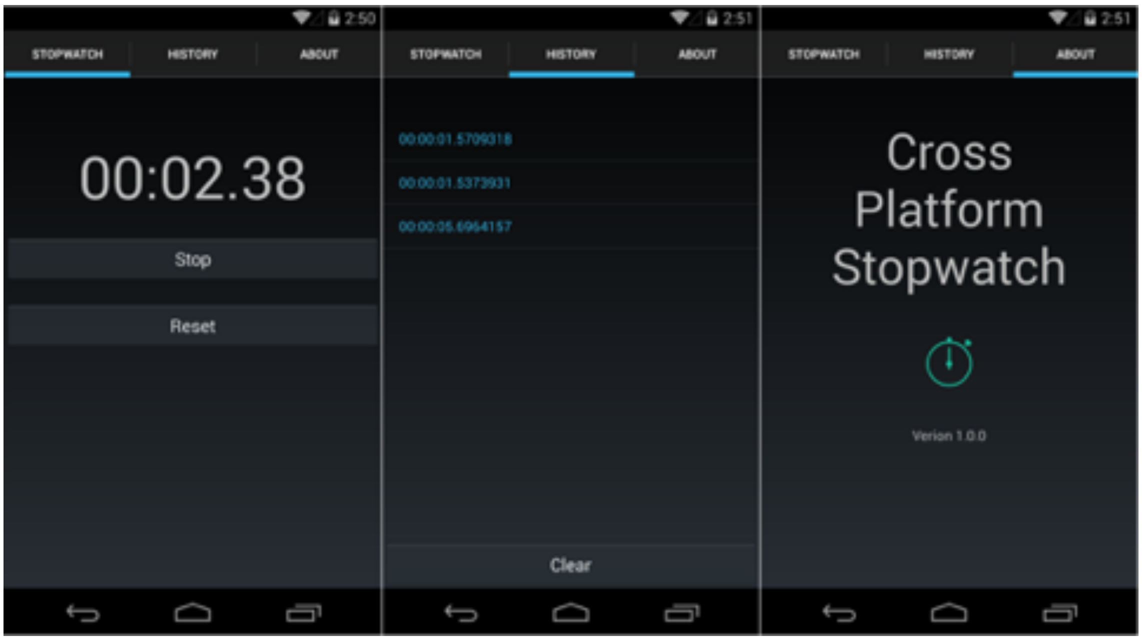
Add Stopwatch application logo about_page_logo.png to each project by right clicking the appropriate folder in each platform and selecting Add —> Existing Item. On iOS add the image to the Resources folder. On Android, the Resources —> drawable folder, or the appropriate folder for your test device's resolution. On Windows Phone add the image directly to the project root folder.

5.10 100% Shared Code

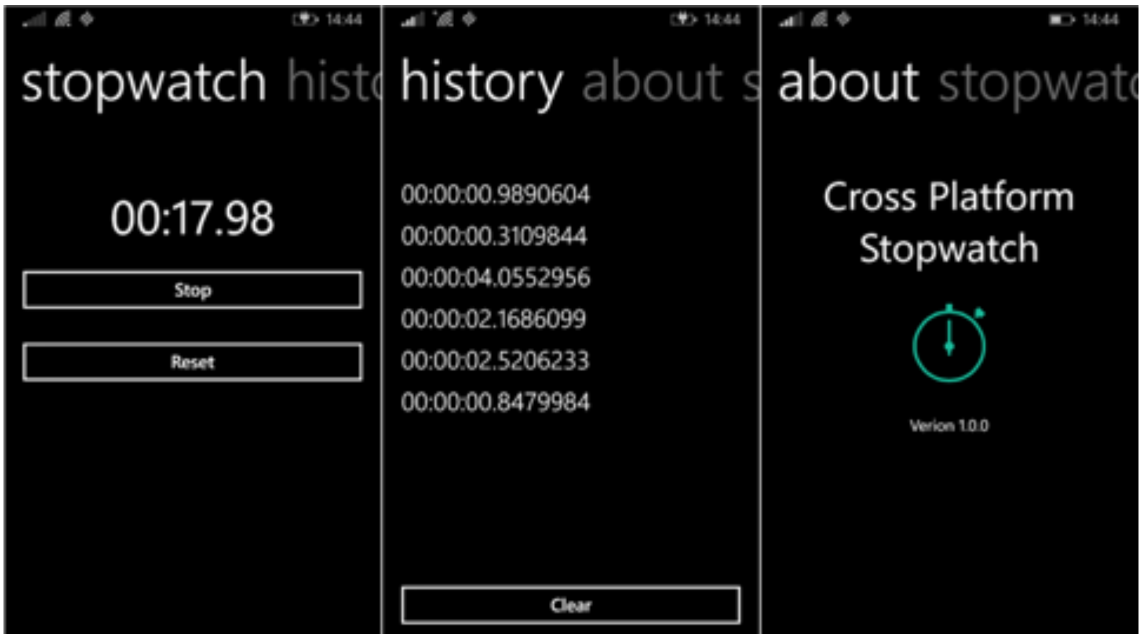
In CrossPlatformStopwatch there was no need to write any platform specific code – all of the application code was shared. This is the ultimate goal of using cross-platform frameworks like Xamarin.Forms. However, in most consumer grade highly polished applications writing some platform-specific code is almost always necessary. Often the user interface needs to be tweaked for each platform separately to provide a desired look.



Picture 15. The entire app rendered on iOS



Picture 16. The entire app rendered on Android



Picture 17. The entire app rendered on Windows Phone

6 CONCLUSION

Nowadays people are using their mobile phones for most of their personal computing. The expected user behavior is that the application they are using will work in the same way on any mobile platform. This brings a great challenge to developers who are trying to build apps with a great user experience, look & feel on all three platforms: iOS, Android and Windows Phone.

To tackle this problem, the developers can now use Xamarin.Forms. Xamarin Forms is a framework from Xamarin that allows developers to reduce the amount of platform specific UI code required when creating cross platform mobile applications.

The main idea behind Forms is to provide developers with the advantage of increasing the amount of code-reuse between platforms.

The purpose of this thesis was to show that Xamarin.Forms can be a great way to build beautiful, performant native apps for iOS, Android, and Windows.

REFERENCES

Eberhardt, C. 2015. Retrospective On Developing An Application With React Native. Consulted 10.1.2016 http://blog.scottlogic.com/2015/03/26/react-native-retrospective.html

Eisenman, B. 2016. Learning React Native: Building Native Mobile Apps with JavaScript. USA: O'Reilly Media

Hermes, D. 2015. Xamarin Mobile Application Development: Cross-Platform C# and Xamarin.Forms Fundamentals. USA: Apress

Johnson, P. 2015. Cross-platform UI Development with Xamarin.Forms. UK: Packt Publishing

Net Applications 2016. Mobile/Tablet Operating System Market Share. Consulted 5.1.2016 https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1

Optimus Information 2015. Cross-Platform Framework Comparison: Xamarin vs Titanium vs PhoneGap. Consulted 10.1.2016 http://www.optimusinfo.com/blog/cross-platform-framework-comparison-xamarin-vs-titanium-vs-phonegap/

Petzold, C. 2016. Creating Mobile Apps with Xamarin.Forms. USA: Microsoft Press

Ramanujam, P. 2015. PhoneGap: Beginner's Guide, Third Edition. UK: Packt Publishing

Snider, E. 2016. Mastering Xamarin.Forms. UK: Packt Publishing

Xamarin 2016. Part 1 – Understanding the Xamarin Mobile Platform. Consulted 5.1.2016 https://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/part_1_-_understanding_the_xamarin_mobile_platform/

StopwatchPage.cs source code

```
using Xamarin.Forms;


namespace CrossPlatformStopwatch.Pages
{
    class StopwatchPage : ContentPage
    {
        public StopwatchPage()
        {
            Title = "Stopwatch";
            Icon = "tab_bar_stopwatch.png";
            Padding = new Thickness(0, 60, 0, 0);
            var timeLabel = new Label
                {
                    XAlign = TextAlignment.Center,
                    FontSize = 60,
                    Text = "00:00.00"
                };
            var startStopButton = new Button
                {
                    Text = "Start"
                };
            var resetButton = new Button
                {
                    Text = "Reset"
                };
            Content = new StackLayout
                {
                    Spacing = 20,
                    Children =
                        {
                            timeLabel,
                            startStopButton,
                            resetButton
                        }
                    };
            var stopwatch = new StopwatchViewModel();
            BindingContext = stopwatch;
            timeLabel.SetBinding(Label.TextProperty, new Bind-
ing("Time", BindingMode.Default, null, null,
"{0:mm\\:ss\\.ff}"));
            stopwatch.TimerRunningChanged += (isRunning) =>
startStopButton.Text = isRunning ? "Stop" : "Start";
```

```
            startStopButton.Clicked += (sender, e) => stop-
watch.StartStop();
            resetButton.Clicked += (sender, e) => stopwatch.Re-
set();
        }
    }
}
```

HistoryPage.cs source code

```
using Xamarin.Forms;


namespace CrossPlatformStopwatch.Pages
{
    public class HistoryPage : ContentPage
    {
        private ListView list;
        public HistoryPage()
        {
            Title = "History";
            Icon = "tab_bar_history.png";
            Padding = new Thickness(0, 40, 0, 0);
            list = new ListView();
            var clearButton = new Button
                {
                    Text = "Clear"
                };
            Content = new StackLayout
                {
                    Spacing = 20,
                    Children =
                        {
                            list,
                            clearButton
                        }
                    };
            clearButton.Clicked += (sender, e) => ChearHis-
tory();
        }
        protected override void OnAppearing()
        {
            base.OnAppearing();
            list.ItemsSource = LocalStorage.PreviousTimes;
        }
        private void ChearHistory()
        {
            LocalStorage.PreviousTimes = new ObservableCollec-
tion<TimeSpan>();
            list.ItemsSource = LocalStorage.PreviousTimes;
        }
    }
}
```

AboutPage.cs source code

```
using Xamarin.Forms;


namespace CrossPlatformStopwatch.Pages

{

    class AboutPage : ContentPage

    {

        public AboutPage()

        {

            Title = "About";

            Icon = "tab_bar_about.png";

            Padding = new Thickness(25, 40, 25, 0);

            Content = new Label

            {

                Text = "This is an About page"

            };

        }

    }

}
```

StopwatchViewModel.cs source code

```
using System;

using System.ComponentModel;

using System.Diagnostics;

using System.Runtime.CompilerServices;

using System.Threading.Tasks;

using Xamarin.Forms;



namespace CrossPlatformStopwatch {

    public class StopwatchViewModel: INotifyPropertyChanged

    {

        public event TimerRunningChangedEventHandler TimerRun-
ningChanged;

        public delegate void TimerRunningChangedEv-
entHandler(bool isRunning);



        private TimeSpan time;

        public TimeSpan Time

        {

            get { return time; }

            set
            {
                time = value;
                OnPropertyChanged();
            }
        }

        private bool timerRunning;

        public StopwatchViewModel()
        {
            Time = LocalStorage.LastTimeValue;
            MessagingCenter.Subscribe<App>(this, Constants.Mes-
sagingCenterAppOnSleepKey, (s) => SaveTimerValue());
        }

        public void StartStop()
        {
            timerRunning = !timerRunning;
            if (timerRunning)
            {
                RunTimerLoop();
            }
        }

        public void Reset()
        {
            var previousTimes = LocalStorage.PreviousTimes;
            previousTimes.Insert(0, Time);
            LocalStorage.PreviousTimes = previousTimes;
            timerRunning = false;
            Time = TimeSpan.Zero;
        }

        private async void RunTimerLoop()
        {
            var previousTime = Time;
            var stopwatch = new Stopwatch();
            stopwatch.Start();
            while (timerRunning)
            {
                await Task.Delay(1);
                Time = stopwatch.Elapsed + previousTime;
            }

            stopwatch.Stop();

        }


        private void OnTimerRunningChanged(bool isRunning)

        {

            if (TimerRunningChanged != null)

            {

                TimerRunningChanged(isRunning);

            }

        }


        private void SaveTimerValue()

        {

            LocalStorage.LastTimeValue = Time;

        }


        #region INotifyPropertyChanged implementation


        public event PropertyChangedEventHandler Property-
Changed;
```

```
        protected virtual void OnPropertyChanged([CallerMember-
Name] string propertyName = null)

        {

            if (PropertyChanged != null)

            {

                PropertyChanged(this, new PropertyChangedEven-
tArgs(propertyName));

            }

        }

        #endregion

    }

}
```

App.cs source code

```
using CrossPlatformStopwatch.Pages;

using Xamarin.Forms;



namespace CrossPlatformStopwatch {

    public class App : Application

    {

        public App()

        {

            MainPage = new TabbedPage

                {

                    Children =

                        {

                            new StopwatchPage(),

                            new HistoryPage(),

                            new AboutPage()

                        }

                    };

        }



        protected override void OnSleep()
        {
            // TODO: Save application state before going to
sleep
        }
    }
```