

project p125602

by Darren Thomas

FILE	8910_DARREN_THOMAS_PROJECT_P125602_78196_373453439.DOCX (4.93M)		
TIME SUBMITTED	21-APR-2016 01:55PM	WORD COUNT	16869
SUBMISSION ID	56201917	CHARACTER COUNT	99867



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

2016

An investigation in to the use of Xamarin Forms.

BY Darren Thomas

Supervisor: Kemi Ademoye

Second marker: Mike Dacey

DECLARATION

I, ...(FULL NAME)..... declare that I am the sole author of this Project; that all references cited have been consulted; that I have conducted all work of which this is a record, and that the finished work lies within the prescribed word limits.

This work has not previously been accepted as part of any other degree submission.

Signed :

Date :

FORM OF CONSENT

I(FULL NAME)..... hereby consent that my Project, submitted in candidature for the B.Sc.(Hons) Computing & Information Systems degree, if successful, may be made available for inter-library loan or photocopying (subject to the law of copyright), and that the title and abstract may be made available to outside organisations.

Signed :

Date :

(where necessary replace *B.Sc.(Hons) Computing & Information Systems* with your named award)

Contents

Contents	3
1 Introduction	7
1.1 About.....	7
1.2 Rationale	7
1.3 Aims and objectives	8
2 Literature review.....	9
2.1 Introduction	9
2.2 Why is mobile development important.....	9
2.2.1 Apps vs browsers	10
2.2.2 Market share.....	11
2.3 Cross platform solutions	12
2.4 Xamarin	13
2.4.1 Background	13
2.4.2 Mono.....	14
2.4.3 MonoTouch.....	14
2.4.4 Mono for Android	15
2.4.5 Xamarin	15
2.4.6 How does Xamarin work?	15
2.4.7 Why Xamarin is more powerful than web based solutions.....	16
2.4.8 Example Xamarin App – Snap Attack	16
2.4.9 Screenshots.....	17
2.4.10 User experience	19
2.4.11 Example Xamarin App 2 – EasyJet	19
2.4.12 Screenshots.....	20
2.4.13 User experience	23
2.5 Related work	23
2.6 Conclusion.....	25
2.7 Software testing.....	26
2.7.1 Introduction	26
2.7.2 What is software testing.....	26
2.7.3 Types of testing.....	26
2.7.4 Why do we need to perform software testing?	27
3 Methodologies	28
3.1 Research methods	28

3.2 Development methodology	31
3.3 Software	32
4 Design and implementation	33
4.1 Requirements	33
4.2 Design	33
4.3 Evaluation of Azure	34
4.4 Setup of environment	35
4.5 Evaluation of Xamarin	36
4.6 Re-evaluation of design	37
4.7 Re-design	37
4.8 Implementation	38
4.8.1 Iteration 1 – Creating the pages	38
4.8.2 Iteration 2 – Database files	42
4.8.3 Iteration 3 – styles	44
5 Analysis	49
5.1 Analysis of results	49
Results of C.R.U.D operations	49
Create	50
Update	52
Read	54
Delete	55
Usability testing	56
5.2 Evaluation of aims and objectives	57
5.3 Changes to future attempts	57
6 Conclusion	59
7 Reflection	60
References	61
Appendices	63
Scenario 1 – Registration – User has not Registered	64
Scenario 2 – Login – User has registered previously	68
Scenario 3 – Initial creation of details – User logged in successfully	70
Scenario 4 – Edit details – User logged in successfully/User has previously entered details	72
Storyboard	74

Table of figures

Figure 1 App vs browser	9
Figure 2 Time in various apps	10
Figure 3 Preference for apps.....	10
Figure 4 Market share	11
Figure 5 Types of Xamarin.Forms pages(23).....	38
Figure 6 Types of Xamarin.Forms layouts(23)	38
Figure 7 Xaml Page.....	39
Figure 8 Xaml code behind.....	39
Figure 9 C# Code Page	40
Figure 10 CarouselPage.....	40
Figure 11 Code errors	42
Figure 12 ISQLite interface.....	43
Figure 13 SQLite on Android	43
Figure 14 SQLite on iOS.....	44
Figure 15 Home icon	45
Figure 16 Team icon	45
Figure 17 Edit icon.....	45
Figure 18 App Icon	45
Figure 19 Splash Screen	45
Figure 20 Home Screen iOS.....	46
Figure 21 Home Screen Android	46
Figure 22 Player Search Screen iOS	46
Figure 23 Player Search Screen Android	46
Figure 24 View Player Screen iOS	47
Figure 25 View Player Screen Android.....	47
Figure 26 Select Player To Edit Screen iOS.....	47
Figure 27 Select Player To Edit Screen Android	47
Figure 28 Edit Player Details iOS	48
Figure 29 Edit Player Details Android	48
Figure 30 Create New Player iOS	48
Figure 31 Create New Player Android.....	48
Figure 32 Test create	50
Figure 33 Validation error.....	50
Figure 34 Create details	51
Figure 35 Created player.....	51
Figure 36 Update player.....	52
Figure 37 Update details	52
Figure 38 Updated player	53
Figure 39 Read operation.....	54
Figure 40 Selected read player	54
Figure 41 Delete player	55
Figure 42 Selected player for delete	55
Figure 43 Player deleted	56

Table of tables

Table 1 Landing Screen	17
Table 2 Home Screen	18
Table 3 Initial Game Screen	18
Table 4 Home Screen	20
Table 5 Flight Tracker	20
Table 6 Date Picker	21
Table 7 Fly to (Text Input)	21
Table 8 Settings.....	22
Table 9 Info	22

1 Introduction

1.1 About

This project aims to produce a strong understanding of how a mobile application can be written once and transferred on to multiple platforms using Xamarin. This technique has become known as the 'Write once, Run anywhere' approach. Due to the massive popularity of mobile applications, practically every business requires a presence in at least one of the app stores, if not all. In order to have an app in all of them would require massive resources in terms of developers with all the relevant coding skills or a massive outlay in cost to outsource the project to suitable development teams. However, there are now more and more opportunities arising for the production of a multiplatform application, one that requires the developer only to master one set of skills that can be ported to multiple technologies. This project will summarise the various methods available and focus on the use of Xamarin as a solution to create an application for testing. Once created the application will be analysed and compared on the various platforms and the results critically analysed.

The main focus of this project will be to create a cross-platform application in order to allow the comparison of how the layout renders directly from Xamarin, and if there are any differences, how they can be addressed in order to produce a uniform look between platforms, native differences aside, such as menu icons. Once the look has been unified as far as possible, the performance will be compared using simulation tools such as Xamarin Android Player to compare relevant rival models ability to run the application. The main testing will be usability testing carried out by the developer in a cognitive walkthrough technique. This will be done by means of approaching the UI as a new user and attempting to decipher whether the UI is intuitive and easy to gain the required information.

1.2 Rationale

The reason this project is being carried out is that the need for a suitable solution to cross-platform application development is needed. This project looks at one of the methods available for doing this. Much of the similar work that has been carried out is becoming dated now that Xamarin Forms 2.0 has been released which solves some of the arguments against Xamarin being too heavy weight and not being able to provide cross-platform User Interface(UI) support. Another major argument was that it was a large investment in financial terms due to the pricing structure in comparison to the amount of time saving due to the failed promise of large percentages of code reuse, and now that Microsoft has taken over and open-sourced the project this argument has been reduced.

1.3 Aims and objectives

The aim of this project is to investigate the use of Xamarin forms in order to create an application that will run as identically as possible on multiple platforms. The created application will allow the developer to:

- Analyse and compare how the application renders on the various technologies.
- Analyse and compare the behavioural differences of the platforms.
- Test the performance on the various technologies from a user perspective.
- Ensure the business logic of the application is stable.
- Test the user experience on the various technologies.
- Critically analyse the results in order to decide whether Xamarin provides a suitable solution for cross-platform application creation.

2 Literature review

2.1 Introduction

The focus of this project is to explore the creation and usability testing of cross-platform mobile applications developed with Xamarin. Therefore this section will begin with a background of cross platform development with a major focus on Xamarin. The latter part of the section will review the testing approach used in development.

2.2 Why is mobile development important

Since the iPhone was launched in 2007 mobile computing has become a massive industry. As much as 76% of UK adults own a smartphone(1) and mobile based online sales have jumped to a staggering 88.7% in Europe 2015, with UK sales figures jumping from £8.41bn to £14.95bn in the year from 2014 to 2015. With such high figures, online businesses take mobile computing incredibly seriously and it is all about accessibility, research shows that 57% of users will abandon a site if it takes longer than three seconds to load, so having the flashiest, best laid out website with state of the art features may not be the best approach as it does not matter how good the content is, if it takes too long to load it will not get seen, backed by the fact that google stated in April 2015 that they will be prioritising mobile optimised sites, those that once sat at the top of the rankings may lose their positions simply by not supporting mobile technology(2). As important as being mobile compatible is, there are still more things that these businesses can do to be their users number one choice, and that includes having a dedicated mobile app. The five-year report produced by flurry analytics(3) analyses time spent on mobile devices by the average user in the us, and between January and march of 2014 the report shows that apps dominate the mobile web and their use has increased by 6% on the previous year and equates to 2 hours 19 minutes of in –app time out of a 2 hour and 42-minute online session.

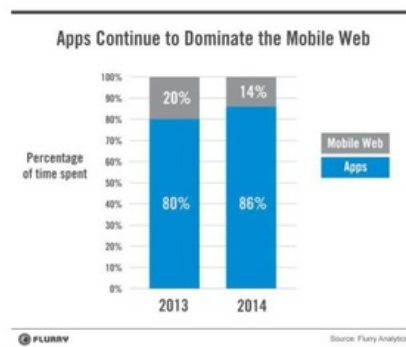


Figure 1 App vs browser

2.2.1 Apps vs browsers

Unsurprisingly mobile app development is a rapidly growing industry, with more and more big name companies producing apps to gain the lion's share of the market. The following chart shows the usage broken down in to categories such as gaming, social media and entertainment.

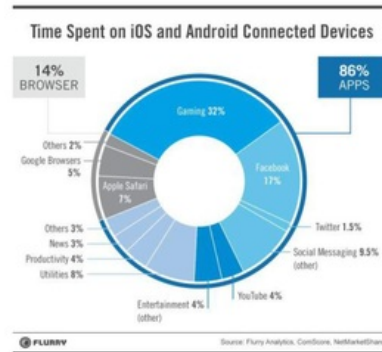


Figure 2 Time in various apps

From this we can deduce that users now prefer using a native app to access the content rather than to access it via a browser, whereas previously users had favoured browsers over dedicated apps as shown by Keynote systems in a survey produced for adobe systems Inc. in October 2010 revealing how most categories were preferred to be used via browsers

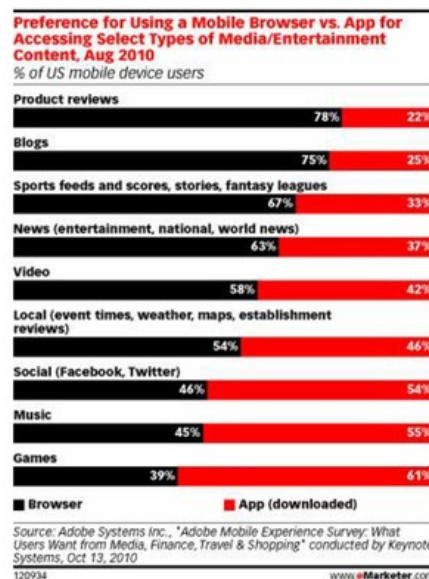


Figure 3 Preference for apps

Having an app means that the company can provide the user with a tailored experience that will be consistent to the device they are using, making it a more natural and enjoyable experience that becomes second nature, by hosting the content in an app rather than a browser the content can often be delivered quicker than via a browser where the creator has little or no control over the speed and features of the device.

2.2.2 Market share

Looking at figures for operating systems it can be seen that Android has the largest share of the UK market with 55.5%. Apple takes second place with 37.8%, followed by Windows with 6.2% and BlackBerry with just 0.3%.

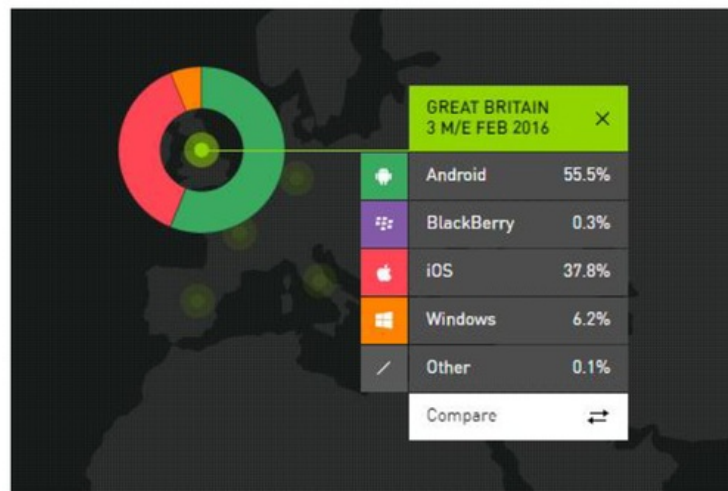


Figure 4 Market share

It is believed that part of the reason Android has such a large share is due to the amount of different devices it can run on, any hardware company can produce a phone or tablet and run android on it which leads to more choice in screen size, quality and budget. Whereas Apple do not allow third party vendors to install their operating system, so while some people may simply not like this kind of restrictive dictation over the devices, others may simply be limited by the need for a larger screen or a cheaper retail price. Part of this argument can be seen to be true in the last quarter of 2014 when Apple released the iPhone 6 and 6 plus. Apple reported a record number of customers switching from Android, in particular Samsung devices to Apple.

These figures lead to the decision to target the two major mobile operating systems, Apple's iOS and Google's Android. The small market share held by Windows Phone may only be worth targeting if Xamarin produces the application well enough without the need for extra time dedicated to it.

2.3 Cross platform solutions

As can be seen apps prove to be the future of mobile devices, and as previously shown there is more than one platform to cater for. The problem of producing multiple apps then arises, this is not only time consuming, but costly as businesses must maintain a development team for each platform targeted in order to produce, test and maintain each app.

Currently Android applications are written in Java and Windows phone apps use C#/VB and XAML while historically iOS apps were developed in Objective-C and are currently in a transitional period of switching to a new language developed by Apple known as Swift. While many apps on the AppStore still exist in C and possibly are still maintained in C, new apps will tend to be developed in Swift resulting in a need to hire developers that can either programme in multiple languages or retrain current developers in the appropriate languages, either of which can result in another costly solution to the problem.

One possible alternative may be to create cross platform applications, apps that are written in a universal medium that can be accessed by all operating systems, these can then be hosted on the appropriate app store and downloaded as standard. Although developing a cross platform app will save time and money when compared with developing in multiple languages, it could present other problems such as performance issues or a limited feature set as depending on the solution implemented the developer will either be working with web based technologies or working on an abstraction layer that will provide the access to the underlying body of code. However, if the situation is suitable, taking into consideration all factors such as performance and features, ease of development and finance, cross-platform development can prove to be an invaluable resource for eliminating the creation, testing and maintenance of duplicate applications.

There are three main types of solution available to the developer, they are web based solutions where a mobile optimised website will be created. Hybrid web solutions where a mobile optimised website is essentially wrapped up in an app and gives some access to features such as GPS and cameras. The last option is cross-compilers, where a unifying base language is used to write the app and it compiles down to the native languages, this is the category that Xamarin falls into and will be used as it is believed that this is the most powerful solution that businesses will be likely to desire.

As can be seen the dominant operating systems are Android and iOS so these will be the target for most businesses providing a mobile application, and therefore will be the focus of this project.

In order to determine how effective a cross-platform solution can be, it has been decided to produce an application using one of the available approaches in order to compare and test its features to determine if the 'write once, run anywhere' objective can be achieved, or whether the apps produced will be too different to be classed a viable solution. A point to bear in mind may be that the developers will want the applications to look native, not identical, as each operating system has its own characteristics. An example could be that the default background on iOS is white, whereas Android's is black, or the way a text entry field or button renders on each OS is unique to themselves, and users of Android will not want to be faced with the styles of an iPhone and vice versa.

2.4 Xamarin

The approach chosen for the project has been to use Xamarin, the reason for this is that unlike many other solutions such as Sencha and Cordova, Xamarin is not a web based solution. The web based solutions offer the developers a way of using web skills and technologies such as HTML5, CSS and JavaScript to produce a web site that will effectively be packaged in an application. The benefits of this are the relatively lower amount of skill sets needed to develop the app, the drawbacks being that it will not produce an application that will look or feel native to the device. An application created this way will also not have the performance of a native device or access to some of the device features such as accelerometers or cameras. In selecting Xamarin to produce the application the developer is offered the productivity of the .NET framework, the use of C# as a shared coding language, for which there is massive support and large amounts of developers, along with many other benefits such as native performance as C# is generally better performing than JavaScript, access to garbage collection, again improving performance with ease of implementation and direct access to all the available API's of the relevant platforms.

2.4.1 Background

The .NET framework is a 'development and runtime infrastructure'(4) and increases programmer productivity by providing a set of classes called the framework class library(FCL) that can be accessed by programmers using certain different languages such as C#, F# and VB to name a few. The resulting interoperability saves development of common programming scenarios such as data access, networking and encryption by providing these common classes in the FCL. Another major component of the framework is the common language runtime(CLR) that effectively acts as a virtual machine for the developed software application to reside in, thus producing a safe environment with handling for common problematic scenarios such as memory management and error handling,

reducing development by abstracting the need to cater for all possible combinations of hardware and software environments(4).

The .NET framework began life in the late 1990's and as Microsoft started to release details about this in 2001 Miguel de Icaza started to wonder how he could bring these ideas to Unix.

2.4.2 Mono

Mono was first announced in 2001 at the O'Reilly conference when the originally small team headed by Miguel de Icaza of Ximian realised that they could not cope with the needs of such a potentially large project and so decided to make it open source. Their original idea was to enable the use of the .NET libraries in Unix and create a developer platform for Linux desktop applications, but this quickly escalated to the idea of coding an API once and being able to use it in all languages resulting in the beginning of real cross-platform application development(5).

Ximian was later acquired by Novell in 2003 in order to 'expand Linux solutions and open source commitment'(6). Mono 1.0 was then released on the 30th June 2004.

Since its inception mono has grown immensely and now incorporates many other technologies developed by Microsoft which have been submitted to the ECMA for standardization(7).

MonoDevelop is the IDE that originated in 2003 as the platform to produce mono applications on Linux, but eventually by version 2.2 expanded to other platforms such as windows and mac.

With these problems addressed another platform arose in 2007 when the iPhone was launched, as it grew in popularity there was the desire to produce applications for this also but a major problem faced developers as .NET applications compile to machine code dynamically at runtime by means of Just-In-Time(JIT) compilation, this was not permitted by Apple under their strict licensing and operating terms. However, mono has use of a technology known as Ahead-Of-Time(AOT) that compiles the code before it is run.

2.4.3 MonoTouch

In 2009 MonoTouch was released which was a Software Development Kit(SDK) that allowed the use of C# and .NET in creating native apps for iOS. It was a port of mono specifically to create apps for the iPhone. Later in 2010 v2 added support for the iPad and by v5 in 2011 it supported most of the iOS API's. Although MonoTouch enabled the use of non-native language to create the apps it did not help with creation of a cross platform app as there was no way to share the code.

2.4.4 Mono for Android

In 2011 mono for android was released, initially named MonoDroid. In the same year Attachmate Corp purchased Novell. Mono for android brought the same kind of .NET productivity to android that they did with MonoTouch two years earlier, but again it only allowed the developers to use C# instead of the native java to create an application and didn't provide any form of cross platform support.

2.4.5 Xamarin

In 2011 Xamarin was created after Attachmate acquired Novell and laid off all the Mono staff, Miguel de Icaza evolved Xamarin from the open source Mono Project in the 'search for a better way of creating mobile apps' (8). The mono project itself had evolved from the desire to port the .NET framework to Unix. Xamarin successfully ported .NET to both iOS and Android by allowing the common C# core to be shared with the various platforms, and MonoTouch became Xamarin.iOS, Mono for Android became Xamarin.Android and the development IDE MonoDevelop became XamarinStudio. XamarinStudio can be used on both Mac OSX and Windows OS, but cannot produce apps for iOS on a windows device and cannot produce Windows apps on a Mac, in order to write for all platforms, the developer must have a Windows machine, or environment in order to run VisualStudio and a Mac accessible over a network in order for the windows machine to pass the code to, for the Mac to compile.

2.4.6 How does Xamarin work?

Xamarin works by allowing the developer to write code in C#, a modern language with modern constructs such as Lambdas, LINQ and Generics, that compiles down to the native code. It does this by working with a set of base class libraries(BCL) from the .NET framework by the name of Xamarin Mobile Profile, these are packaged in the Mono Framework under the MonoTouch.dll and Mono.Android.dll libraries and provide access to invoke practically the entire SDK of each platform, ensuring that using Xamarin is as close to writing a native application as possible by eradicating any limitation of features that can only be accessed by the native language. This can allow for up to 90% of the code written in Xamarin to be common to all the platforms, meaning only a small amount of specific user interface related code would need to be produced for each operating system. All this is achieved with a unique set of bindings that link to the native software development kit(SDK) of the platform. As with C#, these bindings are strongly typed which makes them easier to type and read, allowing the developer to reduce coding time by reducing errors, again strongly aided by the ability to use IDE's with features such as IntelliSense and code completion. Xamarin enables the developer to use existing libraries written in multiple languages for multiple scenarios. As well as the masses of

libraries written by other developers Xamarin allows provides to the .NET framework's vast library reducing the need for rewriting common functions like network support and serialization. Another major feature is Xamarin.Mobile, a single API designed to provide access to resources common to all three platforms. When coding is complete Xamarin will compile the applications down to a .APP file for IOS and a .APK file for Android, meaning that it will be practically impossible to tell that the application was developed with a cross platform tool(9).

2.4.7 Why Xamarin is more powerful than web based solutions

There are four factors that ultimately define a native app and make it stand out from non-native apps. They are User experience, the design of the app, how reliable it is and how well it performs and it is the platforms UI, API's and processing power that determine how these perform uniformly throughout previous apps, apps that were created before any attempts at cross-platform were viable and gave the user the native experience they recognise and expect(10).

In order for an application to feel native it needs to be fast, powerful and familiar, the only true way to achieve this is to write the code in the native language so that the developer can reach the relevant features. Controls such as iOS UITableViews and Android List-Views, Xamarin allows these to be accessed via C# by creating a TableView that compiles down to the appropriate API and so provides the developer with access to all the properties of the relevant object. Access to every API available in the native language means that Xamarin developers have the same power as native developers, so there is no need to omit features such as fingerprint authentication or location detection. Every feature that can be implemented in a native language can be implemented via C#'s binding to the API. The third major requirement is power and speed, as applications become more complex so does the data they retrieve, in order to fetch and display complex data patterns quickly and successfully while a user is on the go, possibly over a mobile network, the application needs to be as fast as possible. With no extra layers, such as web views or browsers to abstract the performance from the application the performance becomes native(10).

2.4.8 Example Xamarin App – Snap Attack

Snap attack by Microsoft Studios is a word based game similar to Scrabble that has been built using Xamarin. Initially Microsoft Studios produced the app for windows 8 and Windows Phone, but soon realised for the online game to become a massive success it needed to be on all platforms, so that anyone could play with their friends. First they created a Java version for Android and an Objective-C version for iOS, very quickly they realised the logistical nightmare they had created for themselves, they could not add new features to the game as the code was becoming too complex and even just maintaining the game while operating system updates were released was becoming unachievable.

The company then decided to try Xamarin and created one application, written in C# that can be deployed to all the different platforms. The developers loved the implementation as they were able to code in the latest version of a popular language, using their favoured IDE with all the tools they require, such as ReSharper(11)(12).

2.4.9 Screenshots

The following screen shots show how the app rendered the same screens on two different devices with different platforms.

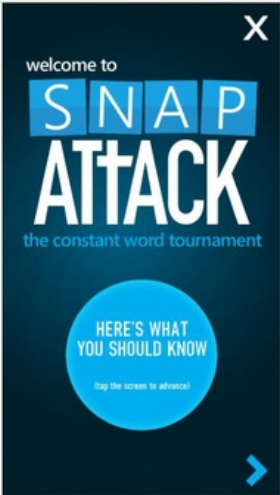
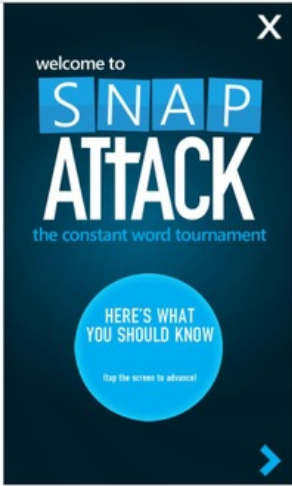
iPhone 6S	Samsung Galaxy S3 Mini
Landing Screen	
	

Table 1 Landing Screen

iPhone 6S	Samsung Galaxy S3 Mini
Home Screen	

Table 2 Home Screen

iPhone 6S	Samsung Galaxy S3 Mini
Initial Game Screen	

Table 3 Initial Game Screen

2.4.10 User experience

The screenshots clearly demonstrate the renders are virtually identical, considering the Android device is a lot older and less powerful, the experience on both was incredibly similar, there was obvious slight performance difference with the iPhone but it is a much more powerful device. The fact that with two such different devices the app can render and perform almost identically is a testament to the Xamarin team, and shows that the system can be used to create a more than viable solution. The developers now have an app that looks, feels and performs the same on all app stores, the fact that they only have to maintain one version is a massive plus point for Xamarin and the developers. Whether the results would be as mirror perfect with a more performance intensive application or a more graphically demanding game, remains to be seen, but for the context it is intended, Xamarin provides an excellent solution.

2.4.11 Example Xamarin App 2 – EasyJet

A more demanding concept for Xamarin use is a business application, and EasyJet has over 65 million passengers a year, 95% of these check in online and many will be using the app. These passengers, unlike game players, depend upon the app to provide them with an easy and reliable service in order to complete their business or pleasure. So the app has to be solid, stable, secure and performant.

Not only are EasyJet using Xamarin in order to reach the millions of different customers with different devices, but they also realise their staff prefer to use different devices, each having their own favourite platform. So in order to enable EasyJet to continue to be a modern employer they realised they must embrace the growing trend of Bring Your Own Device (BYOD), where the employees can bring their own tablets, phones or laptops to work to use without worry of compatibility or security. EasyJet reached out to Xamarin partner EPAM to create a cross-platform application that the customer-hosts can use to ascertain information pertaining to the customers flight, including delays and locations.

The EasyJet application uses Microsoft Azure as a backend and Xamarin Insights during development ensuring that all parts of the system are compatible and related.

2.4.12 Screenshots

The following screen shots show how the app rendered the same screens on two different devices with different platforms.

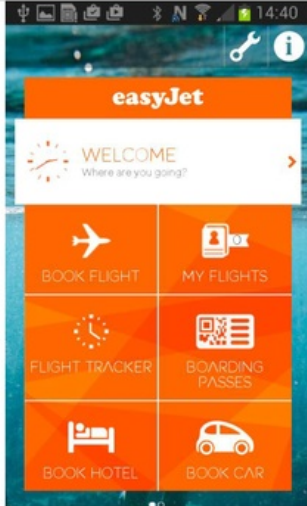
iPhone 6S	Samsung Galaxy S3 Mini
Home Screen	
	

Table 4 Home Screen

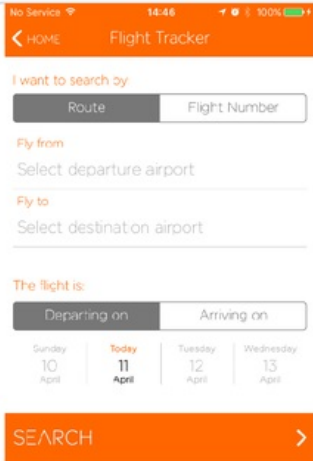
iPhone 6S	Samsung Galaxy S3 Mini
Flight Tracker	
	

Table 5 Flight Tracker

iPhone 6S	Samsung Galaxy S3 Mini
Date Picker	
	

Table 6 Date Picker

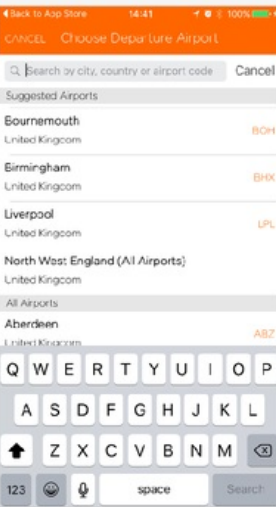
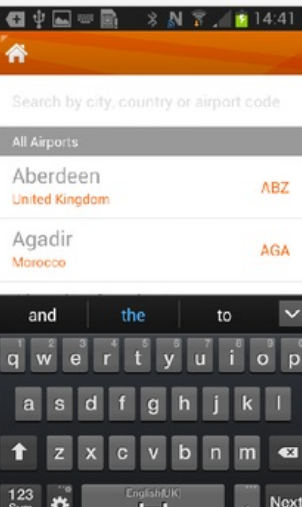
iPhone 6S	Samsung Galaxy S3 Mini
Fly to (Text Input)	
	

Table 7 Fly to (Text Input)

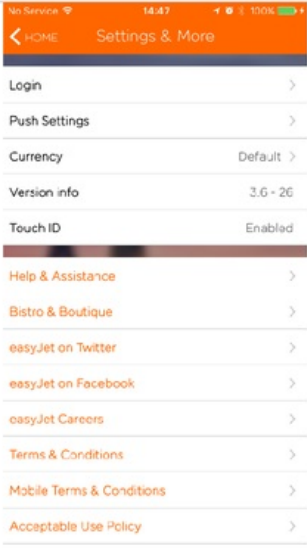
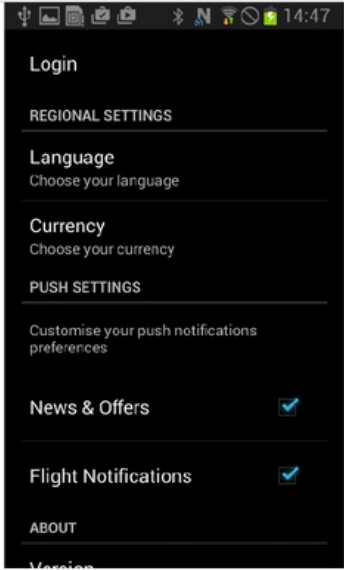
iPhone 6S	Samsung Galaxy S3 Mini
Settings	
	

Table 8 Settings

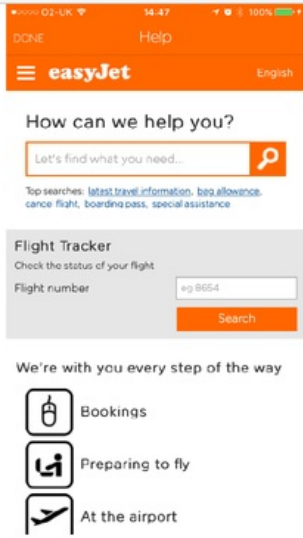
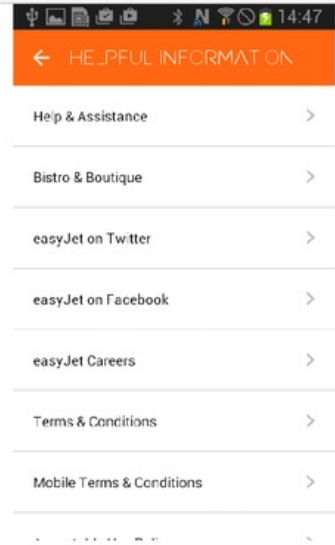
iPhone 6S	Samsung Galaxy S3 Mini
Info	
	

Table 9 Info

2.4.13 User experience

As can be seen in these screen shots the application renders quite differently on both devices this time, due to the fact that this application is more data centric it uses a lot more of the platform controls like tables, date pickers, text inputs and so on rather than a custom designed interface for the game. As this is a business app it needs to be easy familiar etc

2.5 Related work

In the paper (13) an application is created in each of the various solutions, the research found that if performance and user experience is key to the requirements then cross-compiled applications or even interpreted applications produce the best results, whereas if ease of development is key then a web based solution should be taken.

Dickson(14) found in his 2013 research that although appropriate in the right use case, Xamarin did not always provide the perfect solution. Particularly he found that the user should be heavily invested in Microsoft and its technologies in order to provide business value. Dickson had created an application in Xamarin that relied heavily on the hardware features of the devices and wanted to test how easily Xamarin made the access to these features. His findings were that certain aspects were easier than others, namely Bluetooth on the iPhone was hard to implement, but notes that due to advancements since his attempts, the resolution should have been made easier. A major noting that Dickson seems to have in common with many is that the developers own perception of 'write once, run anywhere' can be exaggerated and lead to disillusion compared with the reality of the findings. It has been seen, especially in the earlier research that the promise does not quite live up to the expectation, but it is thought that with the advancement in Xamarin Forms that could change. Dickson also proves that implementing features such as GPS and Bluetooth need not be so problematic to the developer since the addition of the Xamarin Component Store allows developers to obtain plugins that have been written by other users of Xamarin and placed on the store for reuse by developers either for free or under license by purchasing at a predetermined price. These plugins can implement such things as GPS modules to save the developer rewriting and testing common components.

The paper (15) by Carlos Sirvent Mazarico and Marc Campillo Carrera aims to test Xamarin against PhoneGap and looks to be an excellent reference for this project, however the researchers ran into many problems installing and setting up Xamarin as they found, as have many others have, that it does have quite a demanding requirement for its environment. They identify that performance is better with native than web based, but fall short on the realisation of the effort to create a Xamarin application in comparison to a PhoneGap application. This outcome stems from the underlying fact

that Xamarin is producing near native applications not web based as PhoneGap is doing. After struggling and overcoming many of the obstacles the researchers still found many problems that they were running into and stated that they found a lack of quality resources to rely on in solving them. Xamarin does have a large community following and although this paper is a year old at the time of writing this project, there would still have been a large community. The paper identifies the problem Xamarin addresses, cross-platform development, but seems to underestimate the amount of work that will still need to go into production of a near native app. The paper documents the ease of using drag and drop techniques with Xamarin and realises this leads to certain limitation when Xamarin does not force declaration of permissions, essentially sacrificing functionality over ease of application, which is what Xamarin aims to do, make it easier to produce. It is believed that had they searched a little further they would find that they could address any such problems with target specific code, for instance if it is limiting a feature in iOS, they could have written platform specific code for that feature and integrated it in the iOS project of the solution. Some of the other problems the researcher encountered include creation of menus and implementation of maps, they state on several occasions that the code and examples found on the Xamarin website are 'completely wrong' and that they had to do further research in order to get some parts working, while leaving other problems unsolved. They next document very little in the way that Xamarin attempts to share code by use of either PCL's or shared libraries and swiftly move on to list more problems.

In summary this could have been an excellent comparison paper, but it seems that the researcher underestimated the work needed to produce a Xamarin application. In the conclusions it is stated that Xamarin falls short of their promise and that it would be easier to create a native app, although it seems if they have such a hard time creating a Xamarin application they would possibly find creating native ones even harder, unless they are stating that experienced developers would find it easier to create native apps, or that the companies should stick to creating a native app, but then they would need a specialist in the language for the appropriate app, possibly two or even three to create the various apps for the various platforms, whereas they could have the one specialist in Xamarin who could solve the problem as it was intended, with one cross-platform app.

Overall it is believed that they may have expected too much and may not be conversant in programming languages, therefore finding adapting to Xamarin a lot harder than an experienced programmer would, an Objective-C or Java programmer should find the transition to C# a relatively painless experience and not give up and resort to a more limited solution quite so easily.

An excellent paper by Norbert Haberl entitled 'Cross Platform Development Possibilities and drawbacks of the Xamarin platform'(16) discusses many cross-platform solutions and focuses largely

on Xamarin giving great detail over its many pros and cons, pointing out that Xamarin not only provides a coding solution for the cross-platform approach, but also backs it up with many other features that enhance the experience of being involved in the software development lifecycle, effectively offering an all-in-one, in-house solution that companies could heavily invest in, this will only become more established and add more functionality with the backing it will gain with the takeover by Microsoft. Some of these features include integrated testing and monitoring tools. Testing of the shared business logic in the code can be dealt with easily by the standard built-in unit tests, the paper explains greatly how the platform specific parts of the code, such as the User Interface or any custom features need to be dealt with differently as they need to be compiled down to their native language in order to be tested. Xamarin has dealt with this by supporting NUnit as an engine for testing, this can be incorporated in the solution and run in the compiled application. An excellent solution with the only drawback being that it cannot produce live feedback as it is then running in the emulator and decoupled from the IDE where debugging could take place. Xamarin has even made it possible to test the UI with the use of the Xamarin Test Cloud, which allows the apps to be tested on a multitude of devices with automated user interaction. The support does not end once the app is released as they provide a vast system of analytical tools to monitor the success of the application. In all, Haberl has produced a paper that served as an excellent reference tool for the creation of this project with in depth research into the various options available he had clearly looked into the problem with great detail.

2.6 Conclusion

After reviewing other researchers work the conclusion that Xamarin can definitely provide a viable solution can be drawn despite some common draw-backs. Most of the negative points are rooted in high expectations, misunderstandings and an unfamiliarity with the language. Other problems will have been rectified in newer releases of the software, including the addition of Xamarin Forms, especially version 2.0.

The problem of creating a cross-platform application is not easily solved, and everyone has different ideals and views, so producing a perfect one size fits all solution may never become a reality while the different vendors continue to use different languages. The ultimate solution would be a unified language, but is highly unlikely to materialise due to company investments, politics and prejudices.

Another massive factor since any of the reviewed work has been carried out is the fact that Microsoft has now taken over Xamarin. This can only be good for the company as it will have access to Microsoft's massive resources and will get further entwined in its software architecture, especially products like the .NET Framework and Azure.

2.7 Software testing

2.7.1 Introduction

An equally important part of the project will be the software testing of the application. This will encompass the use of black box testing by a user in the form of a cognitive walkthrough.

2.7.2 What is software testing

Software testing is the process of running a program in order to find defects. As stated by Joseph R Horgan and Aditya p Mathur(17) 'it is believed that there is an important relationship between the estimation of reliability of a program, its structure and the amount of testing it has been subjected to'.

2.7.3 Types of testing

There are many types of software testing methodologies and most can be categorised, this section will attempt to define some of the more common techniques and identify the methods used in this project.

The two broadest categories for software testing may be thought of as manual or automated testing. Manual testing will involve human testers physically testing the features of the system to verify expected results, such as button clicking or form filling, and can be carried out without specialist knowledge of the system. Automated testing will involve testers using other software to create test programs that will execute on the system or parts of it in order to determine success.

2.7.3.1 Black box testing – may also be known by other names such as functional testing or specification testing. This method is based on the user testing the functionality of an application without any knowledge of the systems workings. It is not necessary for the tester to know how the application is written or even the language in which it is written, just what it is intended to do. An example of black box testing may be entering text into a form, or pressing a button and observing the result and comparing it with what is expected. This can be carried out at any point during the development life cycle.

2.7.3.2 White box testing – also known as glass/clear box testing relies on the tester having a working knowledge of the program or application undergoing testing. It is the testing of the coding itself as to whether it meets the requirements and that the structure is sound. Sections such as code paths will be tested in order to validate that the expected outcome is achieved. Methods and objects can also be verified as to whether they will perform as expected. Examples may include running a test script involving specific values and conditions to test whether the code follows the correct path for valid or invalid entries of data such as entering a registration process if the users selects register, rather than

log in or if an invalid email address is entered, the system will not continue to register or login. Can be performed at any point but the sooner the better as it can help catch major flaws and reduce code.

2.7.4 Why do we need to perform software testing?

Software testing is needed in order to provide better applications. As software is written by humans who are inevitably likely to make mistakes, it is necessary to test for these mistakes in order to prevent submission of code containing errors which could be detrimental to the business or its customers. In business, if someone is paying for a service, they expect that service to be deemed worthy of the price paid. They would not be happy to accept a product that is broken upon delivery, especially if the damage may result in costing the business even more in losses resulting from this damage. It is expected that the product will be quality tested before release, and with software the quality testing will involve a thorough testing processes to ensure beyond all reasonable doubt that every effort has been made to provide the best possible solution.

Other than striving to provide the best quality product to the customer, the testing process can produce many other benefits such as opportunities for code reuse or reduction, resulting in lighter, more responsive applications. However, software testing does not provide full protection as it can only find bugs that are present, it cannot find errors due to lack of code which may result in errors or security flaws.

3 Methodologies

3.1 Research methods

For the successful creation of this project it is essential that research methodologies are considered and followed, incorrect use of the methodologies would be detrimental to the outcome of the project. The methodology used will define the way we go about discovering the knowledge in a systematic manor enabling guidelines to be set and followed in order to provide an organised structure, thus

Determining whether or not a successful outcome can be achieved.

The first step in creating the project and successful hypothesis is to consider the use of the five W's, what? Why? Who? Where? And when? By correctly considering these factors a previously unobtainable hypothesis can be whittled down to a more effective statement providing an achievable.

For this project they can be considered as follows:

- **What** – are cross platform applications as effective as native applications? By narrowing this section down to a single sentence it allows the author to hone in on what is important to the project, fine tuning the research down to an area that can be realistically achieved in the given time frame and show whether the produced application is able to do the task it was intended for.
- **Why** – the research is being carried out as part of the dissertation project for the final year of a bachelor's degree. The subject was chosen as it is of interest to the author who would like to continue further into mobile application development as it is his belief that mobile internet, whilst being a relatively new technology is fast becoming the main method of access to information, not just on the move but in general for domestic consumers.
- **Who** – the project will develop a small database system for a children's football club. At present the information is kept in an excel spread sheet and messages are sent out via group text on a mobile phone. Ideally the developed database app will allow the coaches/admin to add, view and edit information. Currently the two coaches both have different mobile platforms, one IOS and one Android and the parents of the children have a variety of different platforms, therefore the use of Xamarin as a cross platform solution will be ideally suited to the situation. The participants can then test the developed application and results can be analysed.

- **Where** – the project will be developed for university of wales trinity saint David – Swansea, and physical development will take place on campus and at the author's home address with testing taken place at Penclawdd football ground.
- **When** - the project is being developed for the academic year of 2015-16.

Using the five W's as a tool to achieve a hypothesis it has been possible to decide upon the following

Can Xamarin produce usable cross platform software of a high enough standard to be accepted by usability testers.

In selecting a methodology for the project the research onion has been used, the research onion, developed by Saunders et al highlights the different steps to be taken in developing a research methodology, by working from the outside in it can provide a solution for most situations likely to be encountered(18).

The first stage to be considered is the philosophy stage and the two opposing sides to the onion are positivism and phenomenology. Since the philosophies relating to phenomenology are better suited to issues dealing with human behaviour and work well with interviews and interpretations of attitudes of varying opinions we will not be using this approach. The philosophy chosen will be of a positivistic manor as this searches for a truth demonstrating that there will be an objective, consistent reality that is measurable with the use of statistics. This suits the approach well as we will be able to physically see whether the application works and can measure its performance against a known datum such as a native interpretation of the same application. We can also receive performance related results in the form of feedback from participants and benchmark tests. The fact that it is an objective philosophy helps ascertain impartiality to either method ensuring that an unbiased result is obtained.

The next stage in the onion is the approach stage and the options available are deductive or inductive. As the stage name suggests this is the way that the project will be approached. Deductive approaches, also known as the top down approach involve starting with a general theory about the idea and working towards a more specific target. The general theory will be taken, analysed and reconsidered until a hypothesis can be produced, this again will be analysed and reanalysed as we test and observe the results, further specifying the hypothesis each time until a result is obtained that can be tested against specific data. Inductive approaches, or bottom up approach, as they are also known work in the opposite manor, starting with a particular observation and working towards

a generalised theory about the findings. The researcher may be given a specific observation and will look for patterns in the way it has occurred, from the patterns a hypothesis could be developed and this can lead to theories being created about the original observation.

From the brief description of each approach we can see that the deductive method is a much more scientific approach that will use targeted data such as specific numbers and variables and record results in an organised, structured, qualitative manner, placing strict controls on the tests to ensure clinically produced results, independent from bias can be obtained time and time again, allowing for replication to be introduced, further enhancing the pool of data and allowing for a generalised overall result to be confirmed. Whereas inductive approaches are less clinical, gathering information from individual's thoughts and opinions and so can vary greatly in many aspects even of the same objective. There is no call for such large amounts of data to be collected as averages and generalisations are not considered when dealing with human emotion rather than scientific fact. Therefore, the data collected will be of a quantitative nature and the researcher themselves will gain a close understanding of the research and may even be considered in the results. From this we can deduce that a deductive approach will be followed. (20)

Thirdly we reach the strategy layer and it can clearly be seen there is a relationship formed from the earlier choices, the positivistic approach leans us toward the experimental or survey strategy. Other options being more suited toward cultural and ethical research, such as case studies, which look at real people in real situations and ethnography, which looks at people in their natural surroundings. Action research looks at specific problems in specific situations so is not suitable, leaving archival which collects data from previous records and grounded theory that collects the data in order to produce a theory rather than testing one. Therefore, we are left with the survey strategy which works by collecting data by survey, be it by post, electronic or in person, this is however not wholly what we are looking to achieve as we wish to produce a product to test whether something performs as well as something else, leading us directly to the experimental approach. The experimental strategy examines the results of an experiment and compares them to the expected results, it is a solid, scientific strategy that concentrates on given factors and their relationships enabling the researcher to see the relationship between cause and effect. It is often used in many areas of research and is ideal for the research being carried out for this project.

The next stage is the time horizons stage, here we are faced with two choices, cross-sectional and longitudinal. Since the longitudinal horizon receives its data from results that are collected over a period of time it is suited to studying behavioural effects and evaluating how things develop and change over a longer period of time. In contrast we have the cross-sectional horizon, otherwise

known as the snapshot collection, this is performed over a short period of time. It uses pre-established data collected at a certain time, it focuses on a particular event at a particular time, ideally suited for collecting and analysing the data from a test case event such as the performance test of a mobile application. As there is no direct link between the rest of the layers and the selection of this layer the choice needs to fit the situation even more closely and the cross-sectional horizon perfectly fits the situation in question.

Finally, we reach the core of the onion and are driven into the techniques we will use to collect and analyse the data for the project by our choices in the previous layers. In order to produce a true, valid and reliable conclusion the choices made in the previous layers must be appropriate to each other, as they will guide our present methods. We must now classify our data into one of two categories, primary or secondary. Primary data can be collected by us or by others during their own research, but must remain unprocessed and raw in our use of it. It can be such data as the results of our survey or a survey conducted by a referenced party that we wish to consider. Secondary data consists primarily of data that has been processed previously, such as the conclusions derived by someone else's research or study, the conclusion they come to from the results they obtain and their analysis of said data.

3.2 Development methodology

It is suggested to use software development methodologies in order to produce a finished article in less time with less problems resulting in better value for the customer and better satisfaction for the producer.

There are many different development methodologies available to use, each with their own benefits and disadvantages. Due to the nature of this project it will not stick vigorously to any given methodology, but rather take sections from several different versions.

The waterfall method for example, works well for projects that have a strong predefined idea of what is required, and this project fits in with that requirement. It also helps find errors early, and the earlier any errors are eradicated, the less the cost to the project. It has well defined stages that need to be reviewed before progressing to the next stage which provides excellent organisation and work flow. However, in developing the project problems may arise with some of the features and result in a need to alter the requirements, this will be hard with the waterfall method which would also allow us to over run time spent on developing features and not be left with enough time for the rest of the system(21). Whereas an agile method using time boxing will allocate us a certain amount of time for developing stages and others, that must be adhered to so that it cannot overrun, so by creating priorities for the requirements early on we can decide which features to concentrate on and

implement first, and which can be dropped as remaining time lessens. Also as some requirements are removed it may become apparent that others should be added which can only be catered for by an agile method(22).

3.3 Software

In order to produce a cross platform application with Xamarin there are a few requirements.

Number one will be to have access to Xamarin which requires a costly subscription, for any commercial activity this would have to be factored into the cost of the operation. For the purpose of this project a subscription has been obtained due to the student status of the author. Xamarin allows development in either its own IDE Xamarin studio for windows and mac or the visual studio IDE for windows. For this project visual studio will be used due to its familiarity. In order to use visual studio with Xamarin a hefty extension is needed which will allow the compiling to take place. By using visual studio on windows the developer is limited to creation of windows phone and android applications, in order to develop iOS applications, the developer must have access to a mac running at least OSX Yosemite that can be accessed over a network such as Wi-Fi. This is required as the iOS SDK contains the iOS simulator and along with XCode is needed to compile the iOS application and both of these are only available on mac OSX.

4 Design and implementation

This project aims to test Xamarin's functionality in creating a cross-platform application and so this section will detail how the application was created, techniques used in creation, implementation and considerations made and problems encountered.

4.1 Requirements

- User should be able to quickly gain access to player's details
- User should be able to easily browse through all players
- User should be able to add new players
- User should be able to remove players
- User should be able to edit existing player's details
- Application should store player's details
- Application should be easy to navigate
- Application should have native feel and performance

The database should contain the following items

- Unique id
- First name
- Last name
- Contact number
- Profile picture
- Email address

The above requirements are what is required from the application in order to demonstrate a functional application capable of basic C.R.U.D operations.

4.2 Design

The original intent for the application was to test how well Xamarin Forms produces two apps on different platforms by writing just one piece of code. This would be accomplished by creating an application and analysing the finished product to see what differences are produced in the different platforms. An example of this would be how navigation is implemented in the shared code and how it relates to the produced native result.

To achieve the required result, it was decided to create an application with a login and registration screen, including a 'My Account' page. It was decided that this would show all the required features of C.R.U.D operations. They would include registering for an account, which would prove the Create aspect. The 'my account' page would demonstrate the Update operation by displaying the users details and allowing them to be edited and saved. Delete would be handled by the ability to remove an account. This was planned to take place over a network environment and it was decided to use Microsoft Azure to host the backend of the application by creating a database and hosting it in the service.

4.3 Evaluation of Azure

To use Azure for this purpose it is required to create an account, there are several pricing plans available to choose from, but for the purposes of this application a free trial was undertaken as it provides the user with the ability to create up to ten free services which can still be accessed after the 30-day free trial is completed. The idea of Azure is to combine many different services and contain them all under one roof. It offers the ability to create data storage, host websites, virtual machines such as servers and much more. The service that is appropriate to this project is the Mobile Services section which in theory allows the user to easily implement a backend for their application. However, the first problem arises when following documentation and tutorials for this part of the development. Microsoft has recently updated Azure and has superseded Mobile Services with App Services. Therefore, the documentation that exists for the deployment to the service is no longer relevant and certainly not best practice if trying to create a modern service as the service is already out-dated and soon to be discontinued. Currently they are in the transitional period where neither service is easily achieved by a newcomer, a major challenge is the fact that a lot of the documentation instructs the user on how to achieve a set task, but when it is attempted, the features are not present. Eventually it was discovered that Microsoft had updated the portal, the area where all the user's services are contained. The new portal however does not contain the same features as the old portal and so certain tasks and services are therefore unavailable. A search of many websites, tutorials and forums leads to many similar frustrated users, whereas there is often a wealth of information to be gained by other experienced users, at this early stage there does not appear to be many 'gurus' and information that is offered, is often undermined or counteracted by another's offerings. Every attempt to setup and connect to the service would end in a similar manor, where the service seems to be up and running but will not connect to transfer data. One of the features of the service is the ability to monitor traffic, and it could be seen that traffic was eventually hitting the service, but not when or where expected as even after manually creating the table in a feature named 'EasyTables', data was not being passed into the table. Subsequent attempts at

hitting the server after one strike would not result in another, even though no settings had been changed in between. The other option from using EasyTables is to follow the steps to create a database, then a table, then a connection string and then download a service that can be executed in Visual Studio in order to publish the site to the server, there were many different documented methods for doing this and each one would appear to work, resulting in the I.P Address being reached with a friendly message stating that the service is up and running. Unfortunately, many failed attempts occurred when trying to pass data to or from the server. A client application can also be downloaded in order to verify and connect to the service, but this also failed to connect. After several more failed attempts and using up the ten available free slots it was decided to remove the remote database from the equation and focus on the creation of the application with a local database.

4.4 Setup of environment

Using Xamarin Forms to create a mobile app seems in theory to be a great idea, but is not without some difficulties and limitations. This section will document the experience and highlight problems encountered, limitations found and note reoccurring errors.

In order to use Xamarin there is a somewhat complicated process to follow in order to setup the environment and it is quite a demanding and particular system to setup. A basic setup can be achieved fairly quickly with the right equipment, but in order to maintain reliability care should be taken.

There is a choice of IDE available to use in order to create the applications. They are Visual Studio 2013 and above for PC and Xamarin Studio for both PC and Mac, however the targets achievable differ depending on the platform being used.

If Xamarin Studio is being used on a Mac, the user can develop in Xamarin.iOS, Xamarin.Android, Xamarin.Mac and Xamarin.Forms, however the Xamarin forms can only be used for iOS and Android, Windows Store and Windows Phone applications cannot be developed on a Mac.

If Xamarin studio is being used on a PC then Xamarin.iOS and Xamarin.Mac are completely unavailable, Xamarin.Android can be used and Xamarin.Forms can be used only to develop Android applications.

Visual Studio on a PC allows for Xamarin.Android to be developed. It does not allow Xamarin.Mac at all, it will allow development of Xamarin.iOS as long as a Mac is available on a network and is running XCode in order to compile the Xamarin.iOS to native code. Similarly with Xamarin.Forms, iOS

can be targeted as long as there is a networked Mac available. All other targets can be achieved with Visual studio in Xamarin.Forms including Windows store and Windows Phone.

In conclusion to reach the maximum number of targets a PC with Visual Studio and a networked Mac available will achieve the highest rate.

The required operating systems are recommended to be the latest available – Windows 10, OSX 10.11, iOS 9.2 and Android 6.0.

The PC used in this development was a custom built pc with an Asus P8-Z68 pro motherboard, a intel I7 2.6K processor, a Samsung 850 256GB SSD and 16GB Ram. The operating system is Windows 7 which inhibited the ability to develop Windows based applications, in order to produce for Windows the operating system should be Windows 10, it was decided against upgrading due to personal choice and the fact that Windows Phone only has a small market share and so didn't warrant the change.

The networked Mac used in production was a 2014 MacBook Air 11, with an I7 processor, 8GB ram and a 512GB SSD running OSX 10.11.

Visual studio 2015 community edition with update is the chosen IDE and XCode 7.1 was installed on the Mac. Xamarin Android player was used as the Android emulator of choice due to Xamarin's recommendation that the shipped Google emulator included in the IDE is too slow for practical use.

Setup and installation of the environments on all equipment went smoothly with no problems and practice was able to begin immediately.

4.5 Evaluation of Xamarin

Xamarin.iOS and Xamarin.Android are fairly well established and have a substantial amount of support available in the shape of documents, tutorials and forums. Xamarin Forms however is a newer technology, especially in the version of 2.1, a fairly minor but quick update in comparison to the substantial version 2.0. Again having a young and developing technology to work with leaves the developer somewhat isolated when it comes to queries and error messages. The support that is available does appear to be of a high standard though, with two of the top names in the technology, Craig Dunn and James Montemagno answering many of the questions. Unfortunately, a lot of the answers provided often result in the same conclusion, that it is just not possible with Xamarin Forms at present, but may be available in the future.

4.6 Re-evaluation of design

On beginning development of the login system it became apparent that developing a custom system was kind of like reinventing the wheel. Xamarin Forms exists to make life easier in developing applications and as such has many plugins, components and modules that can be easily implemented into an application, Xamarin.Auth being one of these. Data protection is a major problem for companies storing people's data. In order to create a realistic login solution, that can securely access and store users details without risk of intrusion would be a major task. One that would be recommended to 'pass the buck' with in order to avoid implication if anything goes wrong, if a user's details are gained by an error in the company's security they can be held liable, therefore creating an inadequate solution to a problem that has already been solved did not seem realistic, and so it was decided against creating a login system. Not only does Xamarin.Auth abstract the login system, but it also offers the users a choice of logging in with various well established systems such as Facebook and Google. It even solves the problem of maintaining sessions so the user doesn't have to keep logging in all the time.

4.7 Re-design

In order to maintain the theme of the football club it was decided to create an application based around a coaches note book, where the player's details can be quickly and easily accessed and further features could be added in future. The main reason for creating the application was to test the C.R.U.D operations and so as long as this was achievable and could be demonstrated as working then the dressing of it is irrelevant, any further features implemented would just be a continuation of the same theory executed differently.

The structure of the app is a home screen that opens up to a carousel of the players, where the user can simply swipe sideways to navigate through the players list. Each screen would contain the users name, contact details and a profile picture. There would be three tabs on the screen for navigating to other sections. The first is the home screen where the user lands upon opening the app and contains the aforementioned carousel. The next tab will be a search list so that the user can scroll through a list of the players or use a search bar at the top of the page in order to find the required player quickly. Tapping on the desired player will then take them to the full details page. Both of these pages will demonstrate the Read operation of the C.R.U.D requirements. The final tab will be the edit tab which will take the user to a list of the players that will give access by means of tapping the players name, to their details in an editable format. The page displays a series of text boxes that are prefilled with the current information. The user is able to make changes and hit save, where the Update process takes place. Underneath the save button is a delete button that will allow the user

to remove a player, and in doing so implementing the Delete method. Back one screen via the back button at the top of the screen, demonstrating use of different navigation techniques, is the list the user originated from, which also has a 'Add new player' button. This takes the player again to the edit player page, but this time the details are all blank, enabling the user to add a new player, demonstrating the Create function.

4.8 Implementation

4.8.1 Iteration 1 – Creating the pages

The first stage was to select the appropriate type of page to use for the various screens.

There are several types of page to choose from in Xamarin Forms, they are shown in Figure 5



Figure 5 Types of Xamarin.Forms pages(23)

The root page in this project, that all pages sit on top of is a tabbed page, this allows for navigation via tabs on the screen.

The home screen for the application is a carousel page which allows swiping between child pages.

The player details page is a navigation page which means that it has the ability to manage navigation like adding the back feature.

The remaining pages are all content pages, these contain a view such as a ListView, ScrollView or StackLayout. Figure 6 Shows the various types of layout.

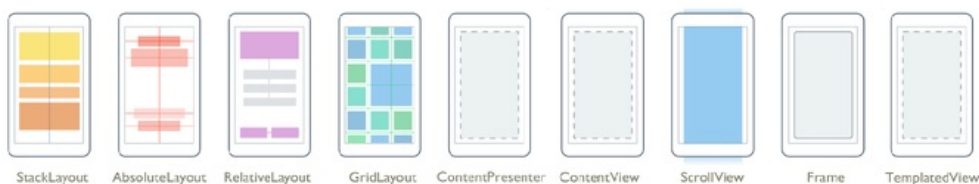


Figure 6 Types of Xamarin.Forms layouts(23)

In order to create a page in visual studio the process is to right click the solution and click add new item. The wizard then appears where cross platform is selected which results in an offering of Forms ContentPage, Forms ContentView or Forms Xaml Page. There are two main ways of creating a page in Xamarin Forms, they are to use a code file or a Xaml file. The code file uses standard C# code to create the file as shown in Figure 7. a Xaml file uses Xaml language to create the layout similar to Xml, and has a .cs file similar to ASP web pages, which holds the 'code behind' as shown in Figure 8 & Figure 9.

Personal preference led to most of the pages being developed in a code format, although Xaml does give slightly more freedom to create custom pages as there are more elements and parameter available to adjust. One page has been created in Xaml for demonstration purposes and to create a custom layout.

```
<?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="TeamMateApp.Views.ViewPlayerDetails"
             Title="{Binding Firstname}">
  <ContentPage.Content>
    <StackLayout x:Name="Details">
      <Label Text="{Binding Email Address}" HorizontalTextAlignment="Center"/>
      <Label Text="{Binding Firstname}" HorizontalTextAlignment="Center"/>
      <Label Text="{Binding Surname}" HorizontalTextAlignment="Center"/>
      <Label Text="{Binding Contact Number}" HorizontalTextAlignment="Center"/>
      <Label Text="{Binding ContactNumber}" HorizontalTextAlignment="Center"/>
      <Image Source="{Binding Image}" Aspect="AspectFit" HeightRequest="150" WidthRequest="150"/>
    </StackLayout>
  </ContentPage.Content>
</ContentPage>
```

Figure 7 Xaml Page

```
namespace TeamMateApp.Views
{
    public partial class ViewPlayerDetails : ContentPage
    {
        //create model for binding the data
        public Player PlayerModel
        {
            get
            {
                return (Model.Player)BindingContext;
            }
            set
            {
                BindingContext = value;
                OnPropertyChanged();
            }
        }

        //pass in the model
        public ViewPlayerDetails(Player _playerModel)
        {
            PlayerModel = _playerModel;
            InitializeComponent();
        }
    }
}
```

Figure 8 Xaml code behind

```

namespace TeamMateApp.Views
{
    public class SearchViewPage : ContentPage
    {
        //create instance of list to use on this ViewPage
        List<Player> searchResults;
        //create instance of searchbar
        SearchBar searchbar;
        public SearchViewPage()
        {
            //set the title and icon for the page
            Icon = "Profile.png";
            Title = "Team";

            //initialize the view
            searchResults = new List<Player>();
            //initialize the searchbar and set place holder
            searchbar = new SearchBar()
            {
                Placeholder = "Search...",
            };

            //add search to the type
            searchbar.TextChanged += (sender, e) => searchResults.FilterPlayers(searchbar.Text);
            //add search button to the button
            searchbar.SearchButtonPressed += (sender, e) => {
                searchResults.FilterPlayers(searchbar.Text);
            };

            //add the elements to the page
            Content = new StackLayout()
            {
                Children = {
                    searchbar,
                    searchResults
                }
            };
        }
    }
}

```

Figure 9 C# Code Page

To create the home page a ContentPage was chosen which needed to be converted to a CarouselPage, which is a simple process of changing one line of code. The process is the same for all page types.

Public Class HomePage : ContentPage is changed as seen in Figure 10.

```

namespace TeamMateApp.Views
{
    //set to carousel page to enable swiping between data
    public class HomePage : CarouselPage
    {
        //create instance of list to fill template
        List<Player> players;
        public HomePage()
        {
        }
    }
}

```

Figure 10 CarouselPage

The two player details pages are relatively similar in appearance with the exception of the image, but one was created in code, the other in Xaml, there really was no need for this here other than demonstration, but the point is that they could have both been made exactly the same with either method as they are fairly basic layouts. One issue worth noting is that there are no checkboxes or radio buttons for Xamarin Forms, in order to create them it would be necessary to get into the platform specific code. On the discovery that there was no checkboxes or radios, it was decided to use a switch, but then it was discovered that there was no way to add a label to the switch as it doesn't have its own implementation of a label. It is possible to add a standard label next to the

switch, but in order to place it so it looks naturally attached to the switch became a lot of work and would possibly have meant changing the whole page to a Xaml page just to customise the label location. Another feature that is not readily available is the ability to use a media picker. An example of this arose when adding the image field to the database, in itself simple enough, but as there is no media picker immediately available through Xamarin Forms you are required to use third party plugins, and although the Xamarin.Mobile plugin is the supposedly best way to access the commonly used API's such as location and camera, it is still a plugin and from experience it would be preferable to exclude the use of the feature than rely on possibly buggy third party plugins. So there was a need to find an easy way to save the user having to enter the file name of the image manually, and since it is a sample project just demonstrating the use of an image it was deemed unnecessary to spend extra time trying to implement a full media picker, instead a simple place holder image is being used, but it wasn't acceptable to have just one image for all the players, there should be at least one for male and one for female. A solution considered was to implement a dropdown list or picker with the choice of pictures linked to the file name appearing, but again there is no way to two way bind the picker element to the selected value and the image path. The entry box could not be left empty as the validation checks all boxes are complete, it is only simple validation checking for null or empty, more complex validation, even the use of a must contain @ symbol was considered and again decided that it did not warrant extra time to create when in real use, there would be the use of more complex validation. The solution chosen eventually was to use placeholder text in the box so the user can either leave it or at least see the format to change it.

A major bug that caused delays at first is the 'initialise component' error, as seen in Figure 11.

This first occurred from nowhere, one minute the code was working fine and then without any changes being made, the error appears. After many hours searching and discovering many others with the same problem, it was found that sometimes it would go with a clean and rebuild, but most times it would not. However most times it would still build and run. There were other similar errors that would popup with the same effect, stating that the type cannot be found in the namespace, but these were all in Xaml files, there was however one or two that would appear in code files, mainly APP() could not be found, but these would usually go with several rebuilds and a restart.

```

//pass the model into the constructor when crea
public EditOrAddPlayerPage(Player _playerModel)
{
    PlayerModel = _playerModel;
    InitializeComponent();
}

//validate all fields are filled
public bool CheckPlayer()
{
    if (string.IsNullOrEmpty(PlayerModel.EmailA
        string.IsNullOrEmpty(PlayerModel.Surname

```

Figure 11 Code errors

4.8.2 Iteration 2 – Database files

The next stage in development was to create the backend files. As a real online database is not being used there needs to be a way to create a database for the application to use. In order to do this a NuGet package called `sqlite-net-pcl` has to be referenced. SQLite is a lighter version of SQL that is ideal for use on mobile devices. Both Android and iOS handle the database connection differently, rather than delve into the native code it is possible to implement an interface to handle it. This is done by creating the interface `ISQLite` (Figure 12) in the shared code and then creating the files `SQLite_Android` (Figure 13) and `SQLite_iOS` (Figure 14) in the platform specific folders.

As the database interaction is being performed by the native code a dependency service must be used in order to access the API. The following line of code can be seen in the figures, providing this service:

```
[assembly: Dependency(typeof(TeamMateApp.Droid.SQLite_Android))]
```

The database is then stored in the `Environment.SpecialFolder` in a `Personal` folder to prevent anything else accessing it. To access the database from the controller the following code is used to create a reusable connection string:

```
_connection = DependencyService.Get<ISQLite>().GetConnection();
```

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  // add using sqlite
7  using SQLite;
8
9  namespace TeamMateApp
10 {
11     public interface ISQLite
12     {
13         SQLiteConnection GetConnection();
14     }
15 }
16

```

Figure 12 ISQLite interface

```

//add using:
using System.IO;
using Xamarin.Forms;
using SQLite;
//add dependency in order to use the native api
[assembly: Dependency(typeof(TeamMateApp.Droid.SQLite_Android))]

namespace TeamMateApp.Droid
{
    public class SQLite_Android : ISQLite
    {
        public SQLite_Android()
        {
        }

        //create implementation of ISQLite interface
        #region ISQLite implementation
        public SQLiteConnection GetConnection()
        {
            var filename = "database.db3";
            var documentsPath = System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal);
            var path = Path.Combine(documentsPath, filename);
            var connection = new SQLite.SQLiteConnection(path, true);
            return connection;
        }
        #endregion
    }
}

```

Figure 13 SQLite on Android

```

// add using:
using System.IO;
using Xamarin.Forms;
using SQLite;

//add dependency in order to use the native api
[assembly: Dependency(typeof(TeamMateApp.iOS.SQLite_iOS))]

namespace TeamMateApp.iOS
{
    public class SQLite_iOS : ISQLite
    {
        public SQLite_iOS()
        {
        }

        //create implementation of ISQLite interface
        #region ISQLite implementation
        public SQLiteConnection GetConnection()
        {
            var filename = "database.db3";
            var documentsPath = System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal);
            var libraryPath = Path.Combine(documentsPath, "..", "Library");
            var path = Path.Combine(libraryPath, filename);
            var connection = new SQLite.SQLiteConnection(path, true);
            return connection;
        }
        #endregion
    }
}

```

Figure 14 SQLite on iOS

Once the database connection files had been completed it was necessary to implement the database model, this is simply a standard model. It was also necessary to provide a method of inserting the data into the database on each device it is tested on to avoid crashing on first run due to the lack of table. To implement this, the table is created every time the app is run by means of the following line of code in the controller.

```
_connection.CreateTable<Player>();
```

The standard C.R.U.D operations are also housed in the controller – PlayerManager.

4.8.3 Iteration 3 – styles

As an iOS application starts it provides a splash screen, this is a good opportunity for the companies to advertise a flashy logo and fancy graphics. Android does not natively use a splash screen, it is possible to enforce one, but as the idea is to make the application provide a native experience it was decided against implementing one. Xamarin automatically creates a default splash screen in iOS and so it is only a matter of replacing this with the custom graphic of choice. This is done by editing the LaunchScreen_Storyboard in iOS designer in Visual Studio. To do this, the file can be found in the resources folder in the solution explorer, double clicking on it then launches the iOS designer with the file loaded ready for editing, while the designer is open it is possible to create a custom app icon also, again this is only required for iOS as Android do not use them, other than the App Icon.

Home



Figure 15 Home icon

Team



Figure 16 Team icon

Edit



Figure 17 Edit icon

App Icon



Figure 18 App Icon

Splash Screen

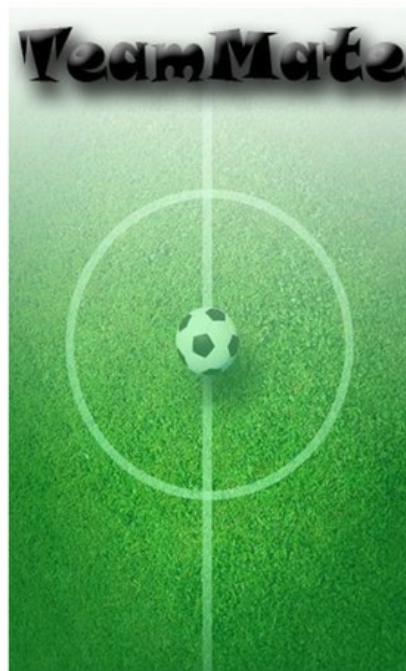


Figure 19 Splash Screen

Figure 15, Figure 16 and Figure 17 were created from Icons8(19) they allow free use as long as they are referenced, or a license can be purchased.

Figure 18 and Figure 19 were created in Photoshop.



Figure 20 Home Screen iOS

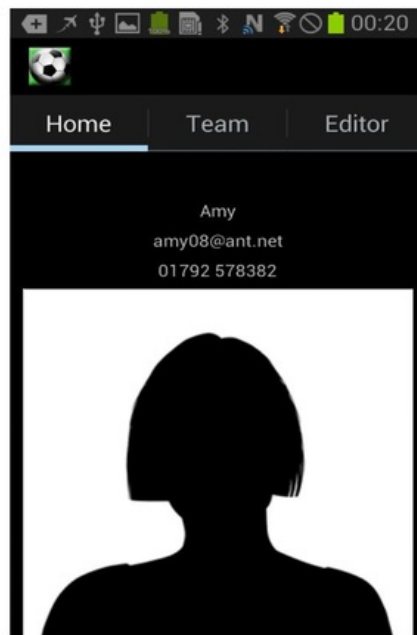


Figure 21 Home Screen Android

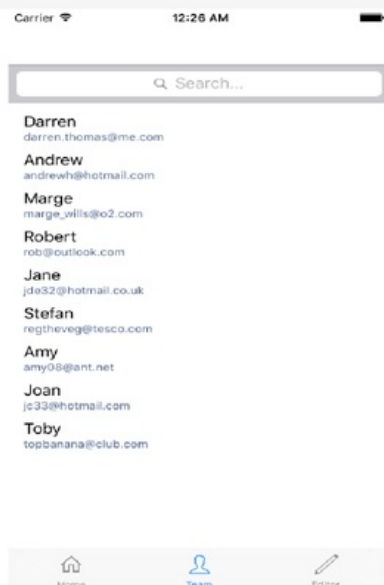


Figure 22 Player Search Screen iOS

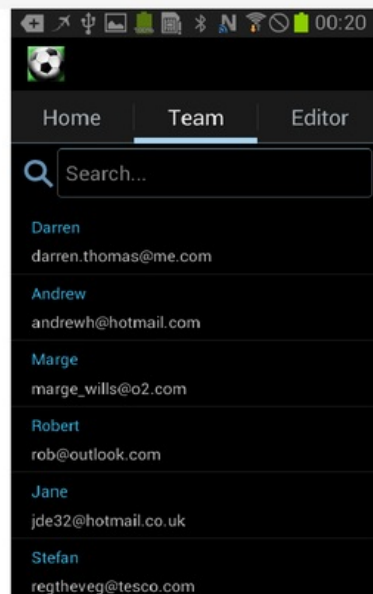


Figure 23 Player Search Screen Android

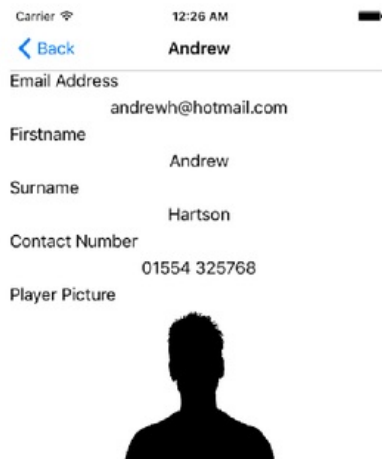


Figure 24 View Player Screen iOS

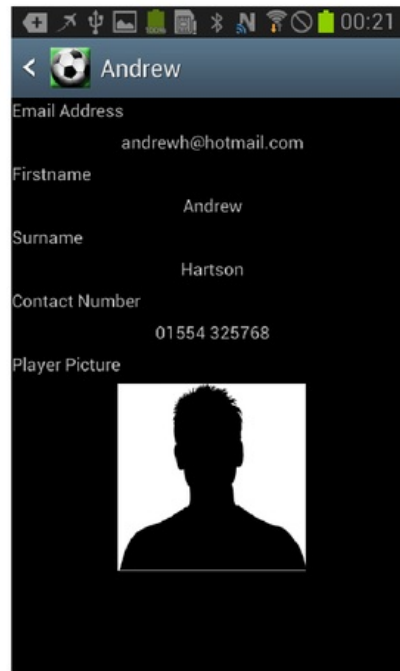


Figure 25 View Player Screen Android

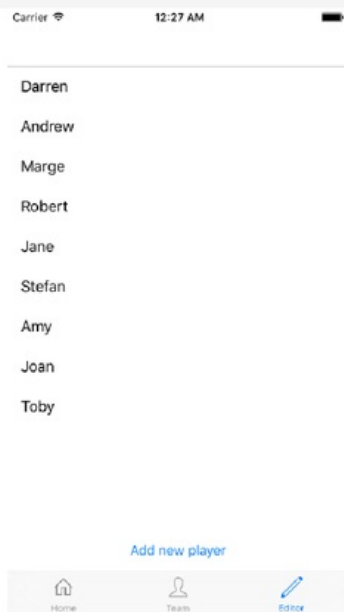


Figure 26 Select Player To Edit Screen iOS

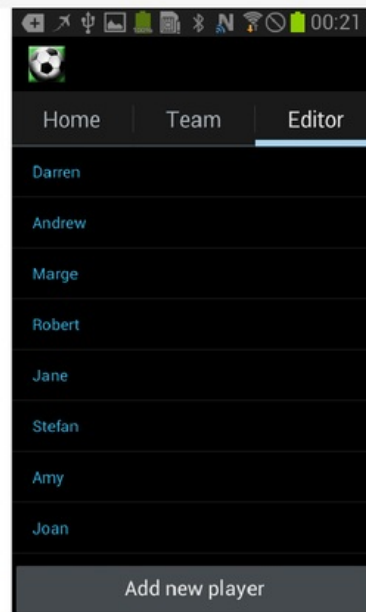


Figure 27 Select Player To Edit Screen Android



Figure 28 Edit Player Details iOS

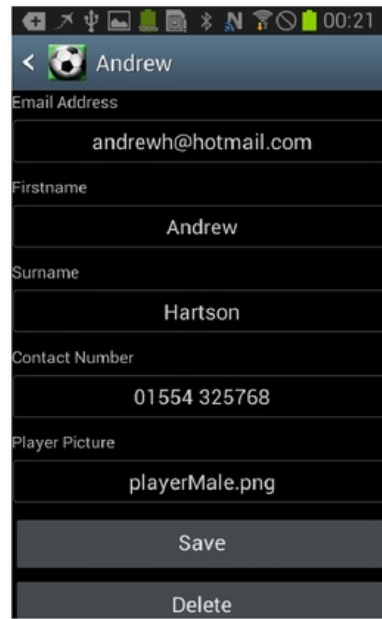


Figure 29 Edit Player Details Android

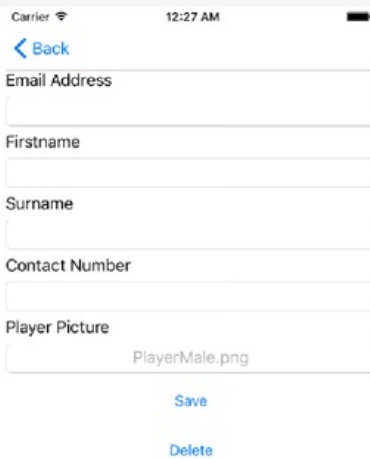


Figure 30 Create New Player iOS

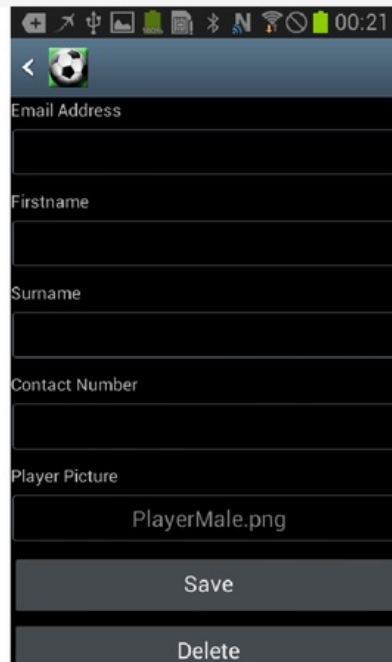


Figure 31 Create New Player Android

5 Analysis

On completion of the project and analysing the results it has shown that Xamarin can produce a suitable application that can run equally well on both platforms. It is a suitable method of producing an outcome that could not have been achieved otherwise. To be able to take knowledge of only the C# language, and produce two applications for the most popular mobile devices available, with limited learning time is great testimony to the open source developers that put their time and effort into such a project. The application has very basic functionality, mainly due to error of time management and un avoidable circumstances, but still serves to prove what could be achieved with a little more time. The application achieves everything it set out to do, but possibly in a slightly more basic version than was originally intended. It would have been preferred to have incorporated security and networking, but as previously stated problems demonstrated, time was lost and functionality had to be reduced and re-evaluated. In real application simple plugins would solve the issues such as authorisation. The fact that such a basic attempt can produce results that are as performant and acceptable as the equivalent would be in native code, to the average user in particular shows that Xamarin could become a lot more popular in the future. Since starting the project Microsoft has acquired Xamarin and removed the cost factor from the equation. Thoughts for the future are that with the backing of Microsoft and also the ability of anyone to be able to use it for free, Xamarin will continue to grow in strength and popularity.

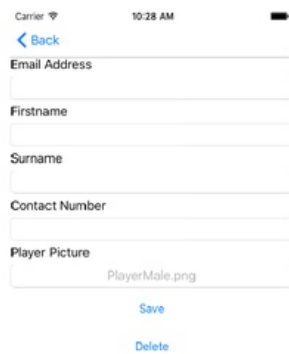
5.1 Analysis of results

Results of C.R.U.D operations

The following section show the results obtained while performing C.R.U.D operations,

The first operation is the creation of a new player on iOS.

Create



Carrier 10:28 AM

< Back

Email Address

Firstname

Surname

Contact Number

Player Picture

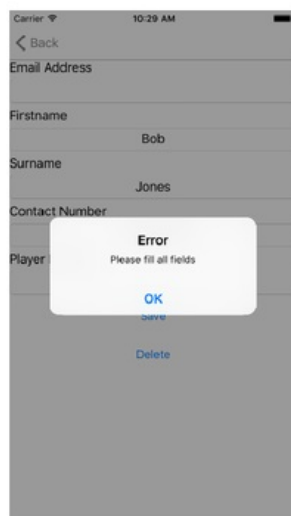
PlayerMale.png

Save

Delete

In order test create a new player the edit button is clicked at the bottom of the page, this navigates to the blank player details form in order to create a new player.

Figure 32 Test create



Carrier 10:29 AM

< Back

Email Address

Firstname Bob

Surname Jones

Contact Number

Player

Error

Please fill all fields

OK

Save

Delete

The fields are then filled and for test purposes one field is The validation prevents a blank field from being submitted and presents a display alert indicating the problem. left blank.

Figure 33 Validation error

Carrier 10:38 AM

< Back Bob

Email Address Bob@me.com

Firstname Bob

Surname Jones

Contact Number 0123234345

Player Picture PlayerMale.png

Save

Delete

The problem field is then filled as shown.

Figure 34 Create details

Carrier 10:37 AM

Darren

Andrew

Marge

Robert

Jane

Stefan

Amy

Joan

Toby

Bob

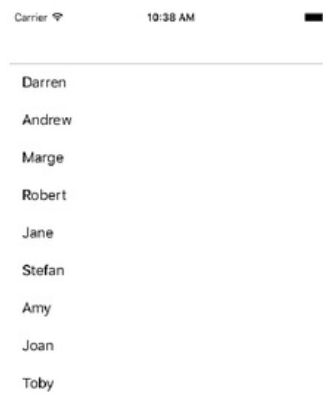
Add new player

Home Team Editor

The save button is pressed and the navigation returns to the team list with the newly created player.

Figure 35 Created player

Update



To test the update method a player is selected from the team edit list.

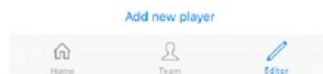


Figure 36 Update player



The player selected is darren, and the where clause is implemented in order to display the found record in the player edit screen.

Figure 37 Update details

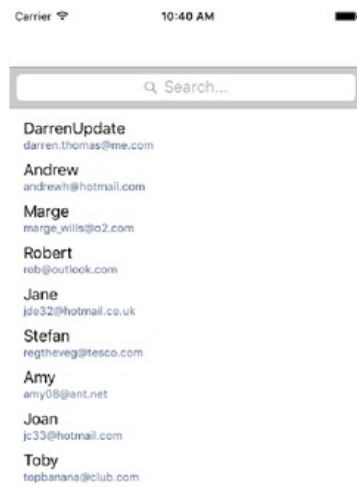


The firstname field is updated with the value Darren -> DarrenUpdate and the save button is clicked.

The save method is called and passed the players Id, therefore the update clause is called instead of the Insert clause. The details are save and the application returns to the player list.

Figure 38 Updated player

Read



In order to test the read operation a player is selected from the team list.

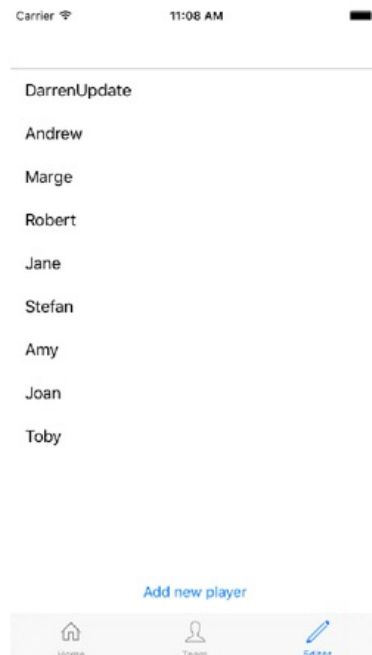
Figure 39 Read operation



The selected player is DarrenUpdate in order to demonstrate the update clause had fully implemented also, the where clause in the query then selects the corresponding player and presents the player view screen with the correct player.

Figure 40 Selected read player

Delete



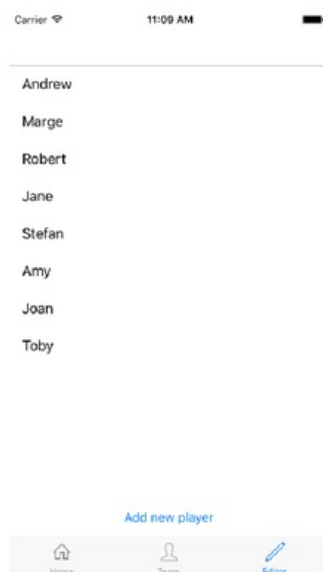
In order to test the delete method
DarrenUpdate is selected from the list.

Figure 41 Delete player



The player details are then displayed in the
edit screen.

Figure 42 Selected player for delete



The delete button is pressed, calling the delete method with the players Id, the remove method is applied and the screen returns to the list, minus the deleted player.

Figure 43 Player deleted

All test returned positive and each test executed as expected. Xamarin Forms incorporates the use of SQLite making database connection and querying a simple process.

Usability testing

The user interface testing was completed by the developer in a Cognitive Walkthrough approach.

This allowed the developer to approach the testing with the effect of exploratory learning, such as a new user to a web site or application would be, as these applications do not usually come with comprehensive documentation. This approach worked well for this project as it is a solo effort, it reduced the need to rely on other testers. The basic interface and application also leant themselves well towards the approach.(24)

It was noted how native the application felt, the user could pick up either device and would simply know what to do with no further explanation needed as all controls and interfaces are using the platforms own styles.

A method of attacking the buttons was also implemented, along with random presses and clicks in the hope of finding any bugs, glitches or errors, the Samsung physical device did crash with random

clicks, but the iPhone 6s simulator on the Mac did not. The Samsung is an older device, being a Galaxy S3 Mini running Android 4.1.4. It is believed that this would add to the cause of the crash, being a less powerful machine, whereas the emulator is running on a powerful machine and emulating a powerful machine. On reopening the application, it worked again as expected, but did crash on more random clicking.

A possible cause of the crashing on random clicks is that for simplicity it was decided to leave out the `async` and `await` calls on the processes which would tie up resources, this along with the omission of a lock to prevent multiple attempts on the database, could have caused the crashes. In a repeat attempt, a lock would be put on each of the database methods, and correct use of asynchronous calls would be used.

In effect the following testing has been completed, testing of the business logic with C.R.U.D operations, usability testing, system testing showing that the application does work as a whole and acceptance testing in the form of the user is satisfied that the application does what it is meant to.

5.2 Evaluation of aims and objectives

After completing the application, it can be seen that the application works just as well on both platforms. The layout although not identical, is as would be expected. It was discovered that the elements rendered exactly as they should on both platforms, which have an inherently different feel to them due to the way the natural layout of the platform is laid out. It would be easily possible to make the application look identical on both platforms by simply changing certain styles, but in order to do that, the native look and feel would be lost, certainly for one of the platforms as certain features would have to be removed and changed in order to balance out. All aspects of the database creation, read, update and delete work as expected.

5.3 Changes to future attempts

If the project was to be undertaken again, it would be to create the application integrated with a web service such as Azure, despite the rocky start with the platform it is believed that it will be a major concern in mobile computing and would be a high priority for future reference.

A larger application would be created with more visual styling, but the native looks and feels would remain intact as it is believed that that is part of what makes Xamarin a success is the ability to mimic a native device. It would be interesting to see a larger scale project and how well Xamarin performs under high demand.

The major feature that would be implemented would be the inclusion of white box testing such as unit testing. During the research for the project it was found that Xamarin has an excellent test

platform incorporated into the system, with the integration of NUnit. Xamarin has developed its own form of testing known as the test cloud, this enables testing on thousands of devices virtually. In order to perform the tests the developer would intergrate UITest into the project by simply adding a test project to the solution. The tests can then be intergrated as development is carried out, producing an excellent environment to practice TestDriven Development. This would again enhance the code by encouraging leaner error free code to be created from the developer. UITest is basically a C# front end on the Calabash framework, a framework developed specifically for cross platform development, it consists of the popular NUnit with NuGet packages on top to perform the cross platform operations. Once UITest has been incorporated into the iOS project via NuGet, a reference is added to the Droid project and a few lines of code are added to the startup files. All that remains to be done is to add the `StyleId=""` property to each user interface item and UITest can find them to perform the tests. Tests can be written as a user would interact, for example tapping buttons and swiping and screenshots can be taken as they are carried out. Normal unit tests can also be added to test the logic of the application and the whole thisng can be tested locally or uploaded to the test cloud where it can be simulated on thousands of selected devices.

6 Conclusion

The initial stage of the project was to plan the application, this went well with the inclusion of use case documentation, requirements captures and such helping to map out the plan.

The next process was the literature review, this was where time started to escape the developer and the proposed plan started to unravel. Personal circumstances led to a lack of time invested in the project, and a stricter adhesion to time limits set out in the initial plan would have helped immensely, but were unobtainable.

Stage three was the selection of the research methodologies and an appropriate method was chosen from each layer on the onion to enable the project to run smoothly, including features such as the outlook to be taken, the approach and the time management. Some of these features were not maintained as well as they should have been and the results can be seen to reflect this.

The following step was the development methodology which involved setting out the steps for the production, including arranging the time boxes and iterations.

The penultimate stage is the design and implementation stage which completely fell out of the timeboxing, resulting in major last minute changes which could only be achieved through the agile process.

Finally, are the conclusions which document the outcome of the project.

In all it is greatly believed that had the same task been set and use of the native language was the requirement, the results would have been nowhere near completed for two applications, possibly not one. This demonstrates that Xamarin can produce an acceptable result in excellent time.

7 Reflection

Whilst reviewing the literature it was noted that people had been struggling with the Xamarin platform. It was seen as inexperience on their part by this author and believed that they had quit too early. On reflection it can be understood that the frustrations of working with new technology, particularly open source projects can lead to abandonment. It can now be recognised that earlier admission to failure of some parts of the project, resulting in re-evaluation or abandonment of the section in question, rather than a stubborn attempt to complete it, would have resulted in a better project. By sticking to the timeboxes the previous authors managed to complete the realistic tasks set out in good time. A major realisation in the importance of sticking to the timeboxes produced in agile development has been obtained and is possibly the biggest lesson to be taken away from the project. Although technically the logic side of the application has been achieved a major change had to be made to the project with the loss of networking. If time had been greater a more advanced set of testing would have been maintained also.

In conclusion the project has not gone as well as had hoped, several lessons have been learned regarding time management and dedication to planning, but personal circumstances often lead to inevitable distractions from the set route. Although the project was something of great interest to the author, not only the idea but the language used was all favourable, the ability to stick to such a demanding routine was unobtainable at the present time. It is felt that the knowledge gained can and will be taken further and a subsequent attempt maybe made for personal gain.

Prior to undertaking the project, it was not understood that the process of research and development methods served any real purpose due partly to lack of knowledge on the subject. On completion it can be seen that they are of massive importance, but require a great amount of dedication in order to adhere to them when working on a solo project. It shall be a target for future events to try and incorporate the knowledge gained from the project, not only technically with programming languages, but to use the techniques and opportunities learned and observed here with regards to research and planning

References

1. Mobile Consumer 2015 : The UK cut Game of phones. . 2015.
2. Why Having a Mobile-Friendly Site May Be More Important Than Ever | OPEN Forum. [online]. [Accessed 6 November 2015]. Available from: <https://www.americanexpress.com/us/small-business/openforum/articles/google-search-mobile-friendly-smartphones/>
3. Flurry Insights | Flurry. [online]. [Accessed 6 November 2015]. Available from: <http://www.flurry.com/blog/flurry-insights>
4. THUAN THAI, Hoang Lam. .*Net Framework Essentials*. O'Reilly Media Inc, 2003. ISBN 9780596005054.
5. ICAZA, Miguel de. Mono early history. [online]. Available from: <http://lists.ximian.com/pipermail/mono-list/2003-October/016345.html>
6. Novell Press Releases | Novell. [online]. [Accessed 5 April 2016]. Available from: https://www.novell.com/PressRelease/press_archive.jsp?year=2003
7. FAQ: General | Mono. [online]. [Accessed 5 April 2016]. Available from: <http://www.mono-project.com/docs/faq/general/>
8. We knew there had to be a better way to build mobile apps - Xamarin. [online]. [Accessed 5 April 2016]. Available from: <https://www.xamarin.com/about>
9. Introduction to Mobile Development - Xamarin. [online]. [Accessed 29 March 2016]. Available from: https://developer.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_development/
10. SYKES, Hollister B. The anatomy of a. . 2006. Vol. 293, no. 1986, p. 275–293.
11. MICROSOFT. Case Study - Snap Attack. .
12. Gaming and Xamarin. [online]. [Accessed 11 April 2016]. Available from: <https://www.xamarin.com/customers/gaming>
13. DELIA, Lisandro, GALDAMEZ, Nicolas, THOMAS, Pablo, CORBALAN, Leonardo and PESADO, Patricia. Multi-Platform Mobile Application Development Analysis. . 2015. P. 0–5.
14. DICKSON, Jared. Xamarin Mobile Development. *Technical Library* [online]. 2013. Vol. Paper 167. Available from:

<http://scholarworks.gvsu.edu/cgi/viewcontent.cgi?article=1167&context=cistechlib>

15. APPS, Cross-platform, CARLOS, Author and MAZARICO, Sirvent. Comparison between Native. . 2015. No. February.
16. FH, D I and HABERL, Norbert. University of Applied Sciences Cross Platform Development Possibilities and drawbacks of the Xamarin platform Table of Content Table of Content. . 2015. No. August.
17. HORGAN, Joseph R. and MATHUR, Aditya P. *Software Testing and Reliability*.
18. SAUNDERS, By Mark and TOSEY, Paul. The Layers of. *Rapport*. 2013. Vol. 14, no. 4, p. 58–59.
19. *Icons8* [online]. Available from: <https://icons8.com/>
20. KOTHARI, C. R. *Research Methodology: Methods and Techniques*. 2004. ISBN 9788122415223.
21. waterfall methodology. [online]. Available from: <https://www.quora.com/What-are-the-pros-and-cons-of-the-waterfall-and-agile-scrum-project-management-approach>
22. *Agile Software Development Quality Assurance* [online]. Idea Group Inc (IGI), 2007. [Accessed 25 May 2015]. ISBN 1599042185. Available from:.
23. Xamarin.Forms Pages - Xamarin. [online]. [Accessed 21 April 2016]. Available from: <https://developer.xamarin.com/guides/xamarin-forms/controls/pages/>
24. Usability Evaluation with the Cognitive Walkthrough. [online]. [Accessed 21 April 2016]. Available from: http://www.sigchi.org/chi95/proceedings/tutors/jr_bdy.htm

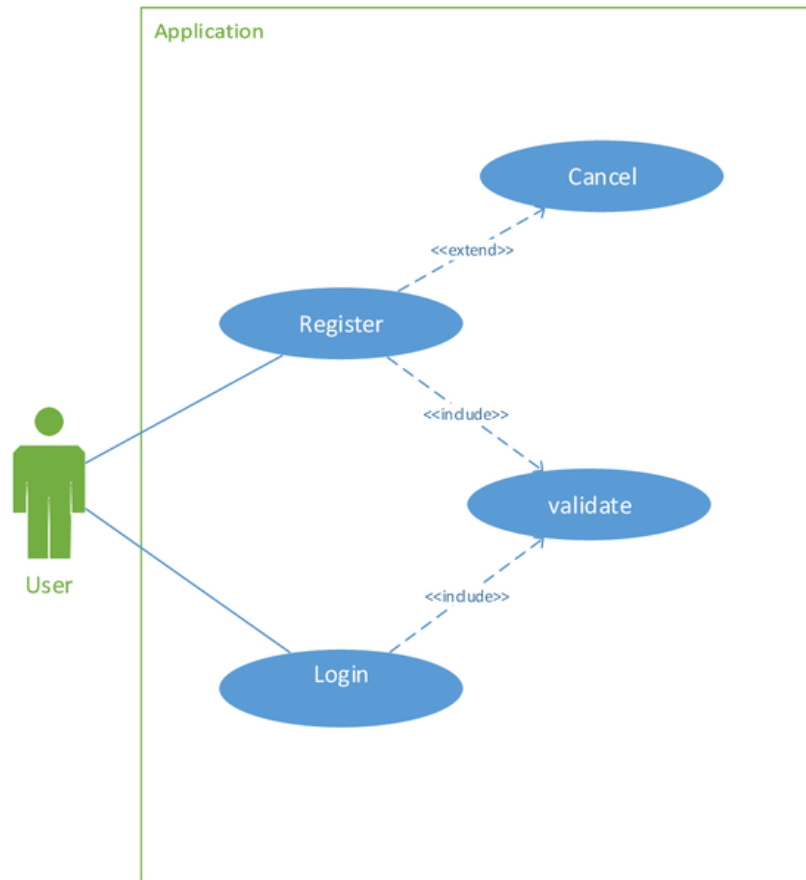
Appendicies

The following section contains the documentation created for the first system design, involving the login system, and consists of use case diagrams, documents and testing documents.

Scenario 1 – Registration – User has not Registered

Successful	Unsuccessful Validation failure/client	Unsuccessful Validation failure/server	Unsuccessful Early exit
User opens application.	User opens application.	User opens application.	User opens application.
Splash screen displayed while application loads.	Splash screen displayed while application loads.	Splash screen displayed while application loads.	Splash screen displayed while application loads.
Login screen displayed.	Login screen displayed.	Login screen displayed.	Login screen displayed.
User has no credentials and o clicks register.	User has no credentials and so clicks register.	User has no credentials and so clicks register.	User has no credentials and so clicks register.
User enters details in form.	User enters details in form.	User enters details in form.	User clicks cancel.
User clicks submit button.	User clicks submit button.	User clicks submit button.	Application returns to login screen.
Validation performed/passed.	Validation performed/failed.	Validation performed/passed.	
Application checks database for existing account.	Error message alerts user to unsuccessful fields.	Application checks database for existing account.	
Data entered into database.	No data added to database.	Matching details found.	
Confirmation given and application returns to login screen.	Application waits for re-entry of valid data or cancel button to be clicked.	Error message alerts user to unsuccessful fields.	
		No data added to database.	

Scenario 1 – Use case diagram



Scenario 1 – Use case document

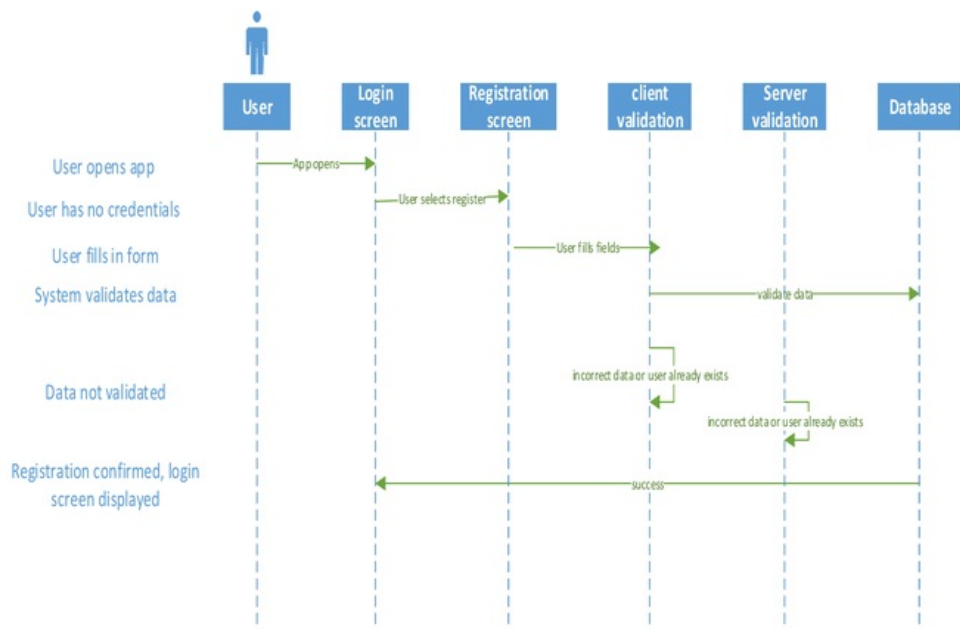
Use case name: Register
Stakeholders and interests: User, database
Brief description: User visits app for first time and wishes to register.
Trigger: User opens the application.
Normal flow of events: 1. Splash screen is displayed while application loads. 2. Login screen is displayed. 3. User has no credentials to login with so clicks register. 4. Registration page is displayed. 5. User fills in username field. 6. User fills in password field. 7. User fills in password comparison field. 8. User fills in email field. 9. User clicks submit button. 10. Application validates fields client side. (S1) 11. Validation is successful. 12. Application validates fields server side. (S2) 13. Data entered in database. 14. Registration confirmed and login screen displayed.
Sub flow of events: S1 1. At least one field does not validate. 2. Application highlights the fields with errors 3. User makes required changes. 4. Return to step 9. S2 1. Username or email is already registered 2. Application highlights the fields with errors 3. User makes required changes. 4. Return to step 9.

Alternative/exceptional flows:

S1, 1a

1. User clicks cancel button.
2. Application returns to login blank login screen.
3. Return to step 2.

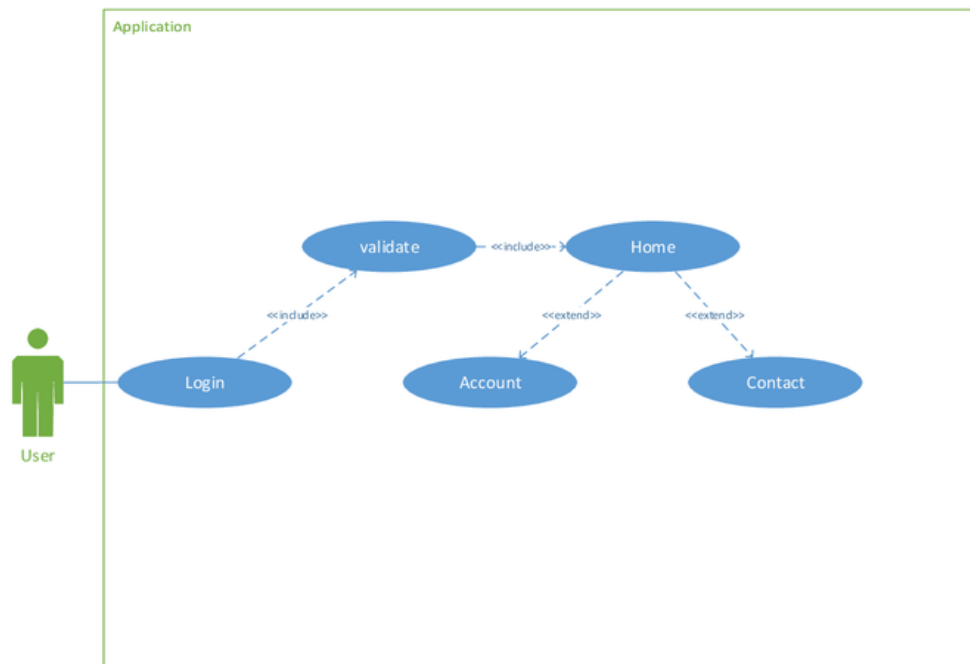
Scenario 1 – Sequence diagram



Scenario 2 – Login – User has registered previously

Successful	Unsuccessful Validation failure/client	Unsuccessful Validation failure/server
User opens application.	User opens application.	User opens application.
Splash screen displayed while application loads.	Splash screen displayed while application loads.	Splash screen displayed while application loads.
Login screen displayed.	Login screen displayed.	Login screen displayed.
User enters username and password.	User enters username and password.	User enters no username or password.
User clicks login button.	User clicks login button.	User clicks login button.
Validation performed/passed.	Validation performed/failed.	Validation performed/passed.
Application checks credentials in database.	Error message alerts user to unsuccessful fields.	Application checks credentials in database.
Match found.	Application waits for re-entry of valid data or cancel button to be clicked.	No match found.
Application logs in user and displays home page.		User not logged in and error message displayed.

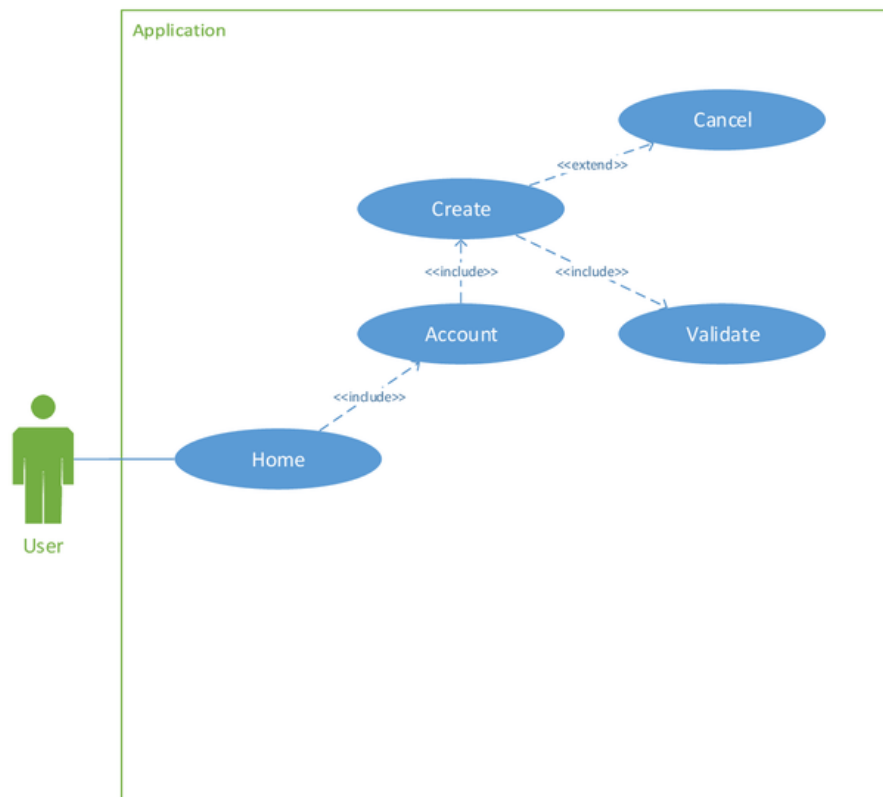
Scenario 2 – Use case diagram



Scenario 3 – Initial creation of details – User logged in successfully

Successful	Unsuccessful Validation failure/client	Unsuccessful Validation failure/server	Unsuccessful Early exit
User selects account page.	User selects account page.	User selects account page.	User selects account page.
User enters credentials in appropriate data fields.	User enters credentials in appropriate data fields.	User enters credentials in appropriate data fields.	User enters credentials in appropriate data fields.
User clicks submit button.	User clicks submit button.	User clicks submit button.	User clicks cancel.
Validation performed/passed.	Validation performed/failed.	Validation performed/passed.	Application returns to home screen.
Application checks credentials in database.	Error message alerts user to unsuccessful fields.	Application checks credentials in database.	
No match found.	Application waits for re-entry of valid data or cancel button to be clicked.	Matching details found.	
Data entered into database.		Error message alerts user to unsuccessful fields.	
Confirmation given and application returns to account screen.		Application waits for re-entry of valid data or cancel button to be clicked.	

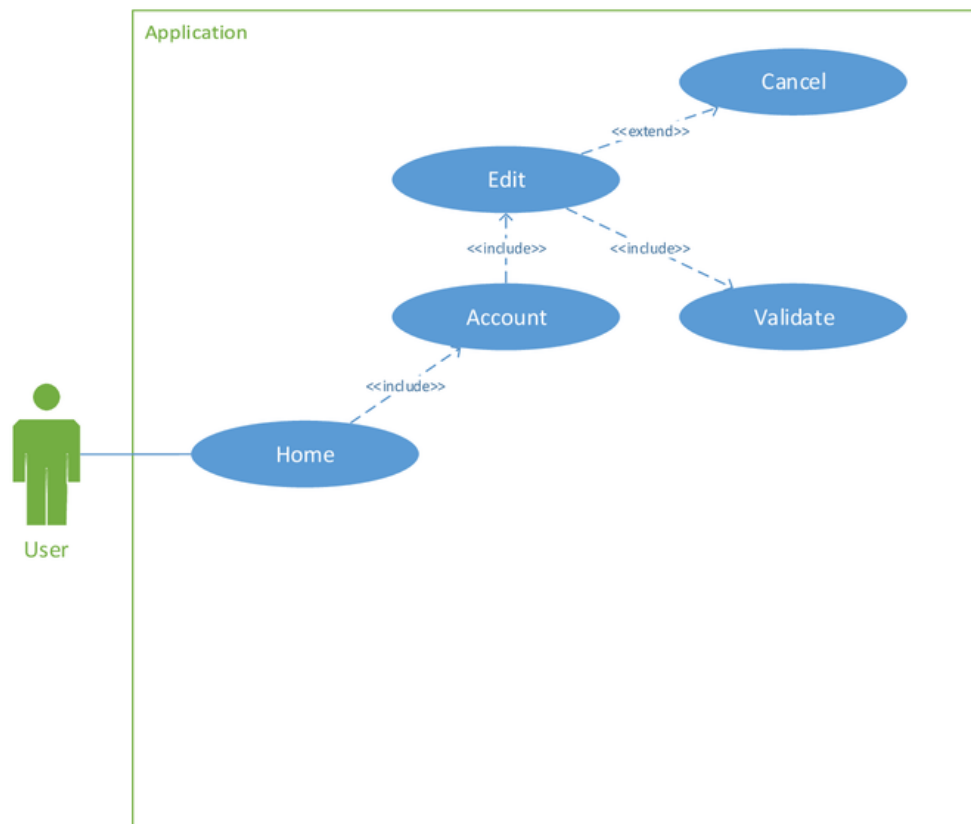
Scenario 3 – Use case diagram



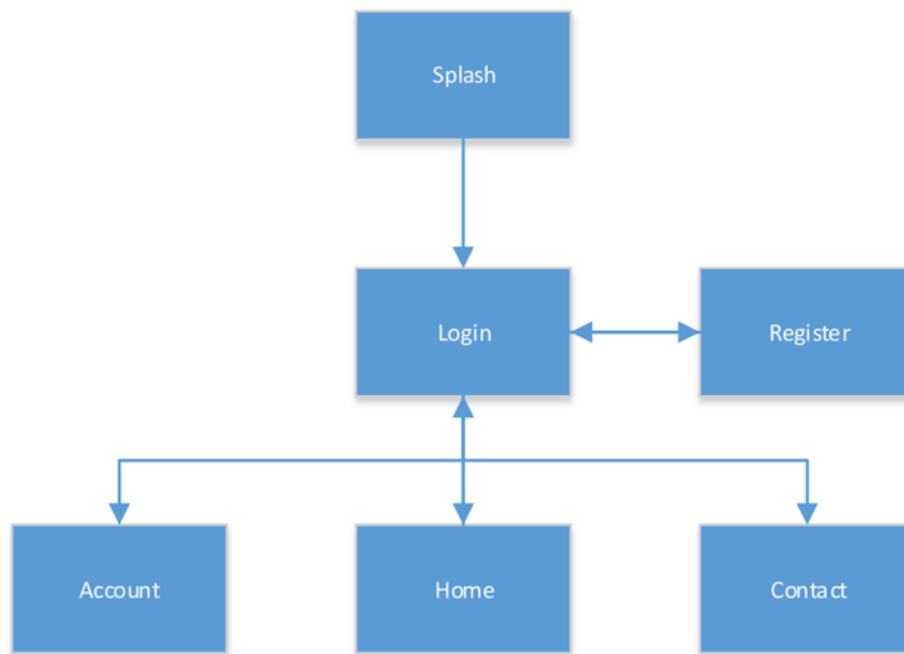
Scenario 4 – Edit details – User logged in successfully/User has previously entered details

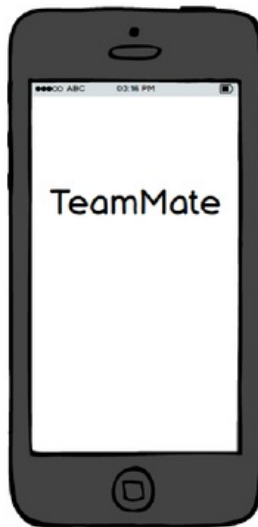
Successful	Unsuccessful Validation failure/client	Unsuccessful Validation failure/server	Unsuccessful Early exit
User selects account page.	User selects account page.	User selects account page.	User selects account page.
Application displays fields prefilled with user's details.	Application displays fields prefilled with user's details.	Application displays fields prefilled with user's details.	Application displays fields prefilled with user's details.
User changes data in selected fields.	User changes data in selected fields.	User changes data in selected fields.	User changes data in selected fields.
User clicks submit button.	User clicks submit button.	User clicks submit button.	User clicks cancel.
Validation performed/passed.	Validation performed/failed.	Validation performed/passed.	Application returns to account screen.
Application checks credentials in database.	Error message alerts user to unsuccessful fields.	Application checks credentials in database.	
Data entered into database.	Application waits for re-entry of valid data or cancel button to be clicked.	No match found. (for user id)	
Confirmation given and application returns to account screen.		Error message alerts user to unsuccessful fields.	
		No data added to database.	
		Application returns to account screen.	

Scenario 4 – Use case diagram



Storyboard





Splash screen – Application start-up

On starting the application, a brief splash screen containing the application name will display while the application initialises.



Login page – Initial screen

On first run the application will load the login screen.

If the user has a username and password, they can be entered and the user can login.

The user has the option to select 'remember me' which will allow the application to bypass this screen on future visits.

If the user does not have a username and password, they must click register.



Home page – Landing screen if remember me selected.

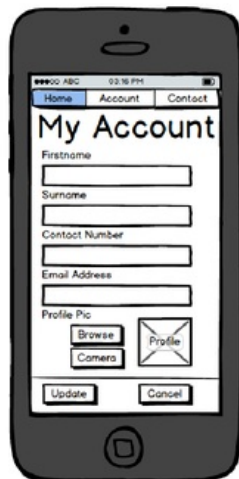
Once the user has logged in the home screen will be displayed.

If the user has previously selected 'Remember me', then this screen will be the first screen displayed, after the splash screen.



Register page – In order to login the user must have created an account.

If the user has not got an account when faced with the login screen, they may select 'Register' which will load the form to create a new account.



Account page – Selectable from the top bar menu.

Once logged in the user can reach this page via the top menu.

The account page is where the user can add personal details, including a profile picture.



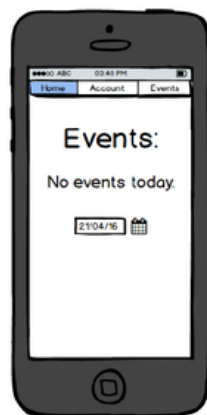
Contact page – Selectable from the top bar menu.

Once logged in the user can reach this page via the top menu.

The contact page allows the user to contact the application admins, via phone or email.

A list of the admins will be displayed with phone numbers that can be pressed.

A combo box of the admins will also be present in order for the user to select the appropriate receiver along with a text input field for the message.



Events page – Selectable from the top bar menu.

Once logged in the user can reach this page via the top menu.

The events page lists the current day's events.

The page also contains a date picker to search various dates.