Comparative analysis of Enterprise cross-platform mobile application development platforms

Ionel Pal

Submitted as part of the requirements for the degree
of MSc in Web Technologies
at the School of Computing,
National College of Ireland
Dublin, Ireland.

September 2016

Supervisor Mr. Vikas Sahni

Abstract

Over the last few years the mobile segment market has largely increased and challenge the enterprise to deliver high quality applications to support each user' mobile interface platform. Developing an application for each platform can be very expensive and time consuming. The best solution to implement an application that targets all mobile platforms are a cross-platform mobile implementation. This study presents a quantitative performance analysis between two of the most representative enterprise cross-platform frameworks, Apache Cordova and Xamarin. An Android an IOS prototype application is built identically with each framework and tested with Xamarin Test Cloud platform, on real mobile devices with different screen sizes and hardware performance. Performance properties between the relevant applications are compared and evaluated. The preliminary results will provide valuable information to the enterprise

Keywords: mobile cross-platform frameworks, Xamarin, Apache Cordova, Xamarin Test Cloud.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr.Vikas Sahni for his truly support, patience, motivation and immense amount of knowledge shared. He helped me successfully to complete this research paper from the beginning up to the end. During this period, he supervised and guide me how to stay on the right path and have the best approach for the thesis's implementation.

Likewise, I would like to express my gratitude to my wife Elena and to my kids Anthony and Hannah for the support during the dissertation.

Furthermore, I would like to thank to NCIRL staff and teaches for the support and guidance in the completion of my course.

Declaration

I confirm that the work contained in this MSc project report has been composed solely by myself and has not been accepted in any previous application for a degree. All sources of information have been specifically acknowledged and all verbatim extracts are distinguished by quotation marks.

Signed ..... Date .....

Ionel Pal

Contents

List of Tables

List of Figures

Listings

Chapter 1

Introduction

Cross platform mobile software development is a relative new area in mobile application, where its maturity is not reached yet. Nowadays, mobile devices have increased dramatically and any enterprise software development needs to have special considerations on the mobile software implementation. Getting the right mobile platform for an enterprise is a major decision that can affect the business. Taking a wrong decision may force them to start over again. The resulting delay and expenditure may cause the business to suffer as a result.

Enterprise organizations needs to react fast and efficiently to the business requirements. Maree et al. (2014) states that enterprise organizations are mainly focusing on performances and build for change, speed to the market together with a good user experience and low total cost of ownership, these are the most important requirements for an enterprise organization when implementing mobile software. By choosing the most appropriate mobile development tool, helps them to achieve successfully the above mentioned requirements.

These requirements can be implemented to a certain degree by the two most popular existing mobile frameworks, such as Xamarin and Apache Cordova (PhoneGap). According to current available research and as highlighted by Willocx et al. (2015) a quantitative comparison between these tools is rather limited.

The scope of this research project is to do a quantitative analysis on the performance properties between the above tools. The preliminary results will provide guidelines for the Enterprise organizations of what is the best mobile platform development tool for their full life-cycle mobile software implementation.

1.1 Hypothesis

How do the most popular enterprise mobile platform frameworks implement the business requirements driven by the users?

Are there still any differences regarding performance between Apace Cordova and Xamarin frameworks?

We will assume that with the latest features of the above named frameworks, each implementation can achieve similar results regarding performance, user experience and fast time-to-market.

1.2 Contribution

This project aims to compare performance properties of a mobile cross-platform enterprise application implemented similarly with Xamarin and Apache Cordova frameworks based on quantitative analysis.

These frameworks have huge potential for the developers already familiar with HTML/-JavaScript or C#. Microsoft has recently acquired Xamarin (Guthrie, 2016), promising great improvement to mobile developers. On the other hand, Apache Cordova is a free tool, open source with a large community support. From the search results, resulted that there is no evidence of an in-depth comparison of performance properties based on a quantitative analysis between these frameworks. This research paper is going to fulfill this hole.

In order to analyze the above performance properties, a prototype enterprise application will be developed separate with Xamarin and Apache Cordova. Each application prototype will share the same backend solution on Azure Mobile Apps, solution that will be developed and deployed with the .Net technologies. The name of the prototype application is *nCare* and it has an implementation architecture based on *Service as a Service* cloud model to the Irish Nursing Homes market and aims to implement communications between nursing homes and their customers.

1.3 Scope and Limitations

The most popular enterprise mobile cross-platform tools may include Telerik. Due to inaccessibility of the framework to te students access, it could not be included in this

study.

The preliminary results are based on a beta version of the *nCare* application development for each mobile platform. Look and feel of the applications are not the main considerations in the tools comparisons, the results of the performance properties are based on quantitative analysis only and limited to the Xamarin Test Cloud resources.

1.4 Organisation of the dissertation

The remainder of the document will outline the research and evaluation. Chapter 2 includes an overview of the mobile market and enterprise mobile development approaches. Chapter 3 outlines the technologies used in this research document. Chapter 4 covers the requirements analysis for an enterprise application. Further, in Chapter 5 the design of user interface for the application will be discussed based on the requirements analysis discussed in Chapter 4. Chapter 6 is divided in three sections. The first section describes the server-side implementation and setup of the cloud resources on Azure for the enterprise application. The second and third sections provides development details for an enterprise application implemented identically twice, with Apace Cordova and Xamarin. In Chapter 7 provides details about testing the applications and evaluates the results. Lastly, In Chapter 8 provides the conclusion.

1.5 Related Work

According to the database search results a comparative analysis of the performance properties based on quantitative analysis between Apace Cordova and Xamarin frameworks is rather limited.

Chapter 2

Literature Review

2.1 Overview

This sections describes the available literature regarding enterprise mobile cross-platform tools and development approaches for mobile applications. It provides an overview of the available researches and how these tools can help the enterprise to develop and maintain mobile applications that can cover the entire range of mobile handsets.

2.2 Mobile Market Overview

According to Chaffey (2016) mobile user is around 90% in the favor of using apps versus the web. Mobile usage trend has increasing dramatically every year. At present, according to Statista (2016) Google Play has 2.2 million of apps in store, followed by App Store, with 2 million and others, e.g., Microsoft, Azure, with a significant lower amount. Altogether, there are over five millions of apps worldwide.

IDC (2016) stated that in 2015 the worldwide smartphone market grew by 13% and it was dominated by Android with 82.8%, followed by IOS with 13.9% and 2.6% Windows Phone. Therefore, and as highlighted by Angulo and Ferre (2014) when the mobile market was similarly leaded by Android and IOS, enterprise organizations best choice would be to focus on the main users, in order to gain the biggest mobile market.

Zodik (2016) states that mobile market is in continue change and enterprise needs to adapt to the next generations of applications development, and to be able to produce cutting edge apps, which needs to adapt rapidly to time, situations and specific user.

2.3 Mobile Cross-Platform Development Approaches

There are many available cross-platform solution approaches, many still under research and development. However, Xanthopoulos and Xinogalos (2013) and Willocx et al. (2016) divides the cross-platform development approaches into Web, Hybrid, Runtimes and Source Code Translators.

2.3.1 Web Apps

Web apps are real websites, not mobile applications. They are running in the mobile browser and are combination of HTML5, JavaScript and CSS. Application are accessed over the URL and there are no specific mobile components installed. This type of application cannot have access to mobile device specific platform features, e.g., camera, GPS and calendar.

The performance, look and feel are inferior compare to other types of implementations. It requires to have Internet access and render the pages can be slow as they need to be downloaded from the Internet first.

The Web apps promise to stimulate the same functionality as the native applications. This promise can be achieved with a number of libraries, e.g., JQuery Mobile and Sencha Touch.

Likewise, W3C (2016) gives details about how HTML5 can access the same mobile hardware and software components through a number of APIs such as Camera, Calendar and Contacts and local storage.

2.3.2 Hybrid Apps

Hybrid apps are created using HTML5, CSS, JavaScript and a browser engine to render the HTML5 content into a native container on the device. This app is installed on the mobile device and can access the underlie of the device hardware and data access with some special APIs. In additional, they have access to the app stores, e.g., Google Play and App Store. A popular example is Apache Cordova which is the engine for many other popular mobile developing tools such as Ionic and PhoneGap.

Yunge et al. (2015) states that Hybrid apps are platform independent and they have the benefits of using interpreted code and as a result the same hybrid app can run of different devices in the same way as the Conventional apps. They have memory management and can perform to a certain degree similar to the native apps.

2.3.3 Runtimes and source code translators

Willocx et al. (2016) gives a clear insight of some of the most important tools that are implementing this approach, including Titanium Appcelerator and Xamarin.

Xanthopoulos and Xinogalos (2013) states that Titanium Appcelerator is a very popular development tool to create interpreted mobile apps, where the native code for each platform is generated automatically behind the scenes, away from the user interface and the developer focuses on user interface only, without writing any mobile platform specific code. In additional, Willocx et al. (2016) states that Titanium Appcelerator is completely written in JavaScript and at the runtime, the javaScript code is fully mapped to native components resulting into a native experience to the users. The down side is that Windows Phone is not supported and the native API is not fully supported.

Furthermore, Xamarin is a very popular Enterprise framework, Willocx et al. (2016) gives an insight of the Xamarin framework. They highlighted that the developers use C# programming language for the developing and Mono framework translates the source code at the runtime into an intermediary language, then into platform specific native code by JIT- compilation on Android and AOT on IOS. For Windows Phone the intermediary languages run on top of the .Net runtime.

2.4 Mobile Cross-platform Frameworks

Mobile cross-platform frameworks are tools that allow mobile developers to target multiple mobile platforms with the same code base. Implementing a cross-platform mobile application is favorite by enterprise compared to single implementation for each targeted mobile platform where the development and maintainability cost could be much higher. The concept is similar to Java Platform, "write once, run anywhere", which in mobile means that one base code project can be deployed successfully on any mobile platform and achieve the same result as single platform implementation.

Angulo and Ferre (2014) states that cross-platform development frameworks is a development approach which can save companies a significant amount of money over an mobile application life cycle, with mobile platform specific design and better user experience.

When implementing a mobile version of an application, Maree et al. (2014) states that enterprise organizations are mainly focusing on performance, good user experience and faster time-to-market.

According to the latest statistics Zain (2015) UK mobile market has already reached 72% of users accessing data from a variety of devices, e.g., tablets, smart phones. Developing and maintaining applications compatible with all available mobile devices can be expensive and time consuming for enterprise, if each application is developed and maintained individually for each mobile platform.

El-Kassas et al. (2016) highlight that by targeting all existing mobile platform with one base code, enterprise can maximize the return on the investment. They discussed the limitations of the available tools, such as user interface generation, and most importantly, that there is no possibility to reuse the code from an existing native applications to other platform, the code base has to be rewritten again by the developer.

According to Boushehrinejadmoradi et al. (2015), the main leading choices for creating cross-platform enterprise applications are web based frameworks, implemented with HTML and JavaScript, e.g., Apache Cordova, and native implementations frameworks, implemented in C#, e.g., Xamarin. As well known, these platforms have advantages and disadvantage. As highlighted by the Facebook co-founder, Mark Zuckerberg Olanoff (2012), counting too much on the power of HTML5 was his biggest ever mistake done in the early stage of the company. Mark highlighted that it was almost impossible to get the enterprise structure in place for the mobile version of Facebook application with hybrid implementation. Targeting all the mobile platforms was very hard and almost impossible. Moving the development structure to native implementation had increase dramatically his success and brought his company in top worldwide.

However, Malavolta et al. (2015) states that hybrid apps are first approach for companies like Adobe and IBM, and first choice for smaller companies or with a well-defined HTML and JavaScript development team.

2.4.1 Mobile Frameworks evaluation criteria

Mobile frameworks are broadly compared based on qualitative and quantitative performs criteria. According to the database, there is a lot of research available in the area of qualitative comparison between web and native mobile framework tools, from focusing mainly on user's experience as described by Angulo and Ferre (2014), to an in-depth comparison based on developer licensing, platform support, maintainability and scalability as presented by Cordeiro and Krempels (2013) and Dalmasso et al. (2013) which gives an analysis of the cross-platform framework requirements, from Rich User Interface to Security, Power consummation and Back-end communications.

In contrast to the qualitative comparison and according to the database findings, the amount of research based on quantitative analysis between enterprise mobile cross-platform frameworks is rather very limited.

Willocx et al. (2015) compared the performance properties results of an application created with Apache Cordova and Xamarin versus a native implementation of the same application. The comparison performance parameters between the frameworks were based on a quantitative analysis and are the app lunch time, memory, disk and CPU usage. The performance properties are measured locally with few high and low end versions of mobile devices.

Chapter 3

Research Background

3.1 Overview

The main goal of this project is to compare the performance properties based on quantitative analysis between Apache Cordova and Xamarin. This section will discuss the main components involved in this research area and define the measured parameters, with regard to an overall performance of an application. The performance properties will clearly define how the frameworks comparison will be based on.

Nowadays, when most of the enterprise have the IT infrastructure cloud based, the benefits of deploying the application on Microsoft Azure Mobile Apps Service will be discussed in details, also an in-depth analysis of the benefits of testing the applications with Xamarin Cloud Testing. Furthermore, an insight of Xamarin, Apache Cordova and Xamarin Cloud Testing Tool will be discussed in detail, explaining the architecture and the outcome benefits.

3.2 Problem Definition

Nowadays, enterprise needs to develop applications compatible with all mobile devices. The best solution that covers all mobile platforms are cross-platform development frameworks. Some of these frameworks are coming with few overhead penalties, such application size increase and some specific platform implementation may be necessary to be added, in order to the application to work properly on any platform.

Cloud computing is other popular approach that many enterprises have adopted in order to run efficiently their business. Cloud computing together with cross-platform mobile

frameworks can support to implement effectively business requirements, which usually focuses on good application performance, user experience, fast time to the market and low ruining cost and implementation. There are many developing frameworks and cloud providers which implement these requirements to a certain degree. Most popular enterprise mobile developing platforms are Xamarin and Apace Cordova. These frameworks are well integrated and fully supported by Microsoft Azure Mobile Apps.

3.3 Xamarin framework

Xamarin is an open source mobile development platform based on Mono runtime which provides resources to the developers to create native cross-platform applications. According to the Mono Project website, Mono is a free open source implementation of Microsoft's .Net Framework using the .Net CLR and the C# compiler (Mono, 2016). Xamarin was recently acquired by Microsoft and it is promising to enable further innovation and power tools to the mobile developers. It is fully integrated with Microsoft Azure, which makes Xamarin a very good choice for creating enterprise applications. The programing language is C# and supports development applications in IOS, Android, Windows Phone 8.1 and Windows Universal (UWP).

The Development Integrated Environment (IDE) is Xamarin Studio and Microsoft Visual Studio 2013 or 2015. Developer licence is not free and can be expensive. Xamarin Studio is fully integrated with Xamarin Test Cloud platform for testing mobile application. Xamarin is a commercial tool, with free trial licence available to the developers.

Xamarin provides IDE support for Android IOS project where the code is translated from C# into native version at the compile time, e.g., ObjectiveC and Java. They also provide support for Xamarin Forms projects, where the developers write code in a single sources folder which gets translates into all supported platforms, where a significant portion of code is shared across all supported platforms. This type of project might require some platform specific code to be written by the developer.

Reusable code is separated into class library and the application is following the separation responsibilities principle, by layering the application architecture and then moving the core functionality into reusable libraries that can be shared across all platforms as showing in Figure 3.1.

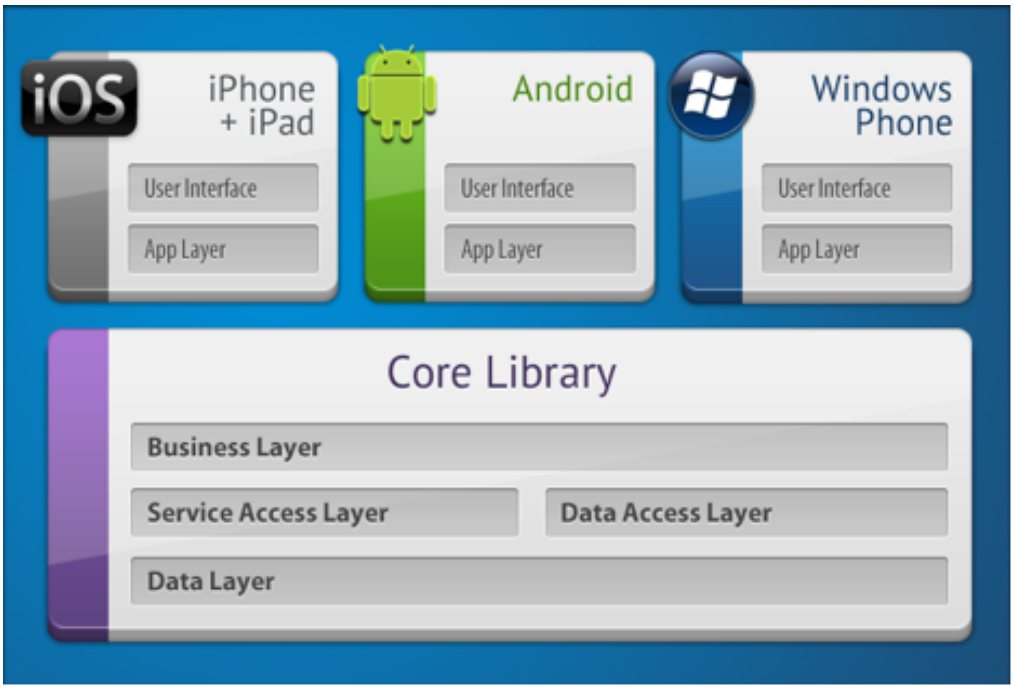


Figure 3.1: Reusable Code Architecture (Xamarin, 2016b)

The base code is shared in two ways, one with Asset Project (SAP) and one with Portable Class Library (PCL). The shared code is not platform specific code, e.g., talk to a web service, parses of data format, performs processing or logic. Code specific to platforms needs to be abstracted first in the share library before use in platform specific, e.g., access system information, access personal information or device files and folders.

3.3.1 Portable Class Library

Is the most popular available option to share base code across platforms. It centralize the code into a single Class Library that can be easily reference by others projects within and outside of the solution. Figure 3.2 presents the architecture of the PCL.

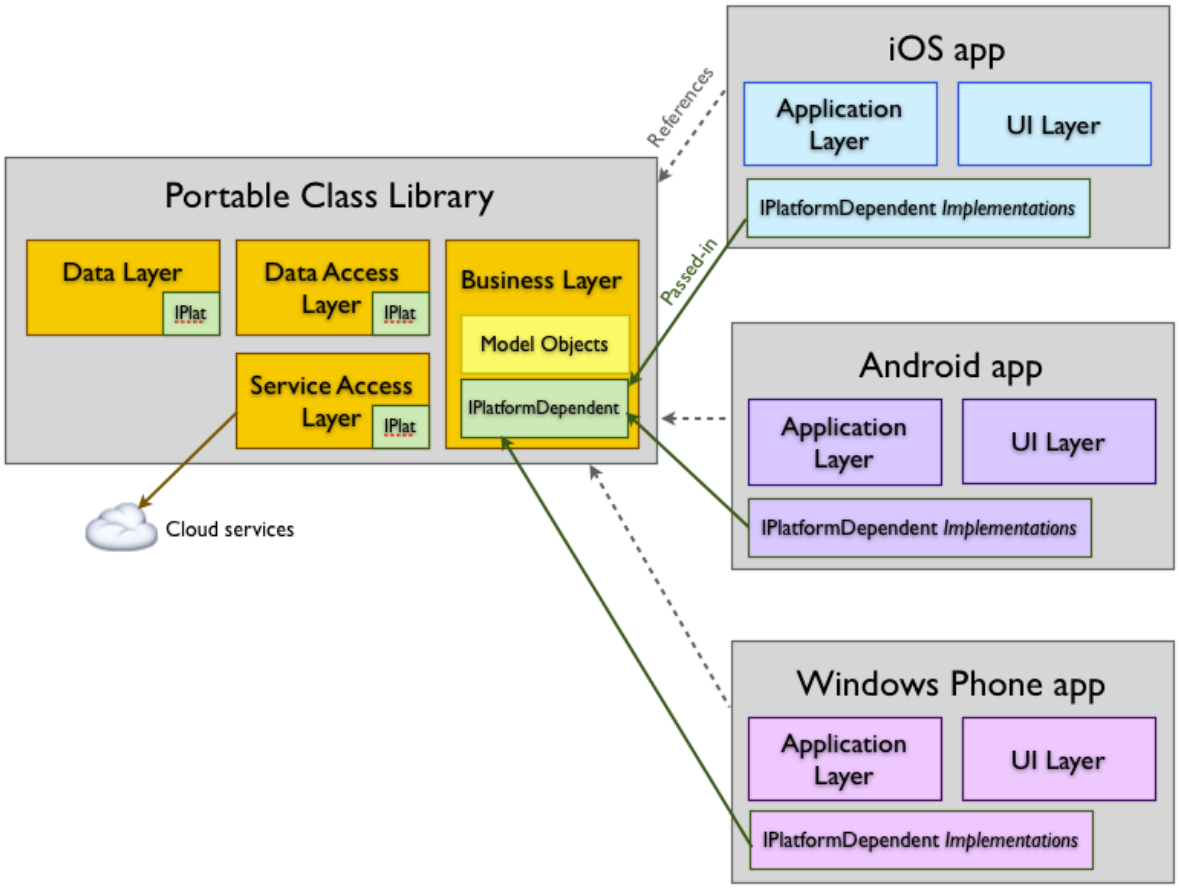


Figure 3.2: Portable Class Library Architecture (Xamarin, 2016d)

3.3.2 Shared Asset Project (SAP)

It is the second most popular option for sharing code. It contains a single folder with files which does not have any output in a Dynamic Link Library (DLL) at the compile time. It does not compile in its own, it just provides a set of directive which get enabled or disabled at the compile time. Figure 3.3 provides the architecture of SAP projects.

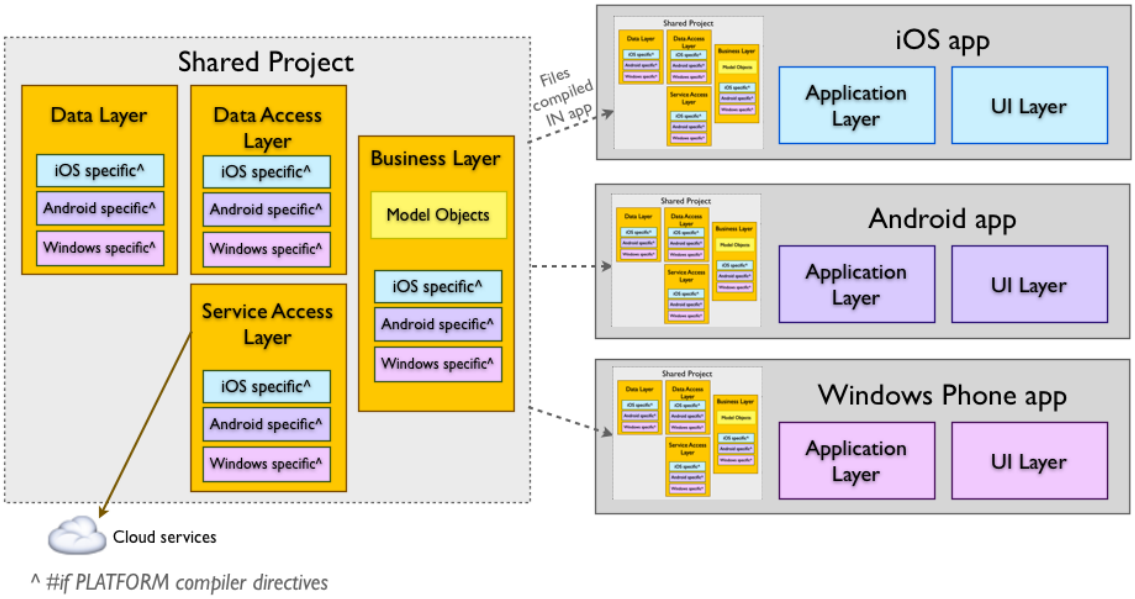


Figure 3.3: Shared Asset Projects Architecture (Xamarin, 2016e)

3.3.3 Architecture Xamarin Android

Xamarin Android applications are ruining in the Mono environment in the same time with Android's runtime virtual machine. The two informants are ruining on top of Linux Kernel which allows the developers to access the underline of the mobile systems. Figure 3.4 shows an overview of the architecture.

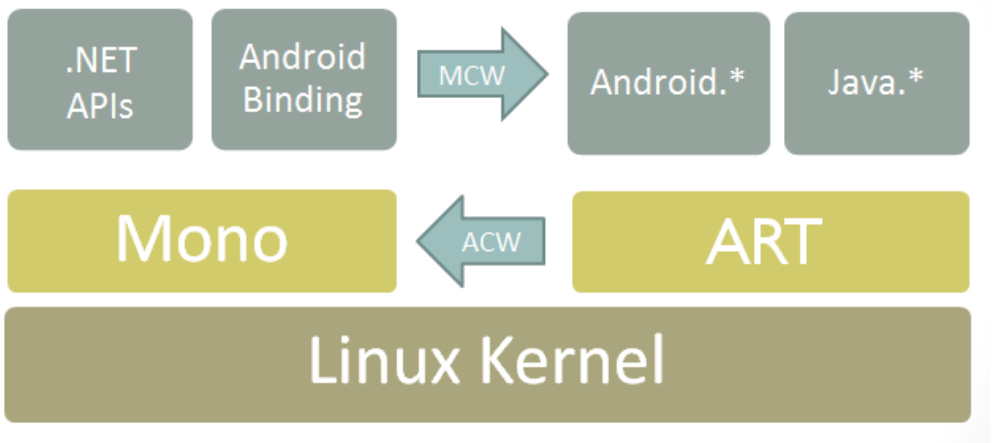


Figure 3.4: Android Application Architecture (Xamarin, 2016a)

3.3.4 Architecture Xamarin IOS

At the compile time C# code gets compiled into ARM assembly language which run side by side with Apple's Objective-C runtime. The two copies environments are ruining on top of UNIX kernel. Figure 3.5 shows an overview of the Xamarin IOS architecture.

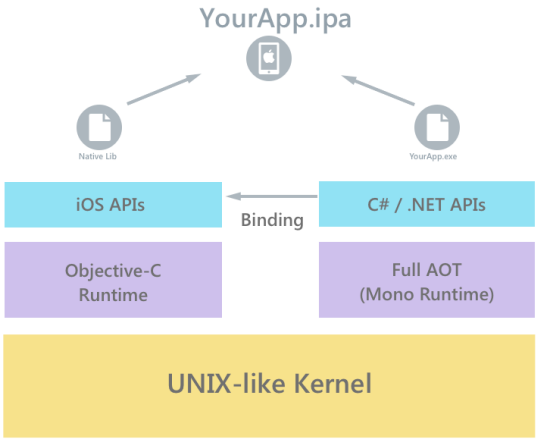


Figure 3.5: IOS Application Architecture (Xamarin, 2016c)

3.4 Apache Cordova framework

It is a very popular cross-platform mobile development framework which allows developers to use standard technologies like HTML5 and JavaScript. It is an open source and it has an very active developer community. According to the Wikipedia, it was originally created by Nitoby and the pouched by Adobe Systems in 2011 where it was renamed as Phone Gap and the released as an open source (Wikipedia, 2016b).

It is easy to learn and the developers can apply the web skills. It supports development on any Operating System and one source code can be deployed on any mobile platforms. Cordova creates a container where the written app resides, it compiles the web code into package files which is required by the app stores such as Google Play or App Store and makes the pages available to an interface web view.

The IDE to create Apache Cordova applications is Visual Studio 2015. Projects can be created also in Linux and Windows command line which requires a framework installation first.

3.4.1 Apace Cordova Architecture

Apace Cordova creates a single screen on the native mobile platform and use the native application web view to render the HTML/JavaScript pages for view. In other words, Cordova embeds mobile web browser inside native application in order to display the pages for the view and use plugins to access device capabilities. Figure 3.6 shows a high level diagram of a Cordova application architecture.

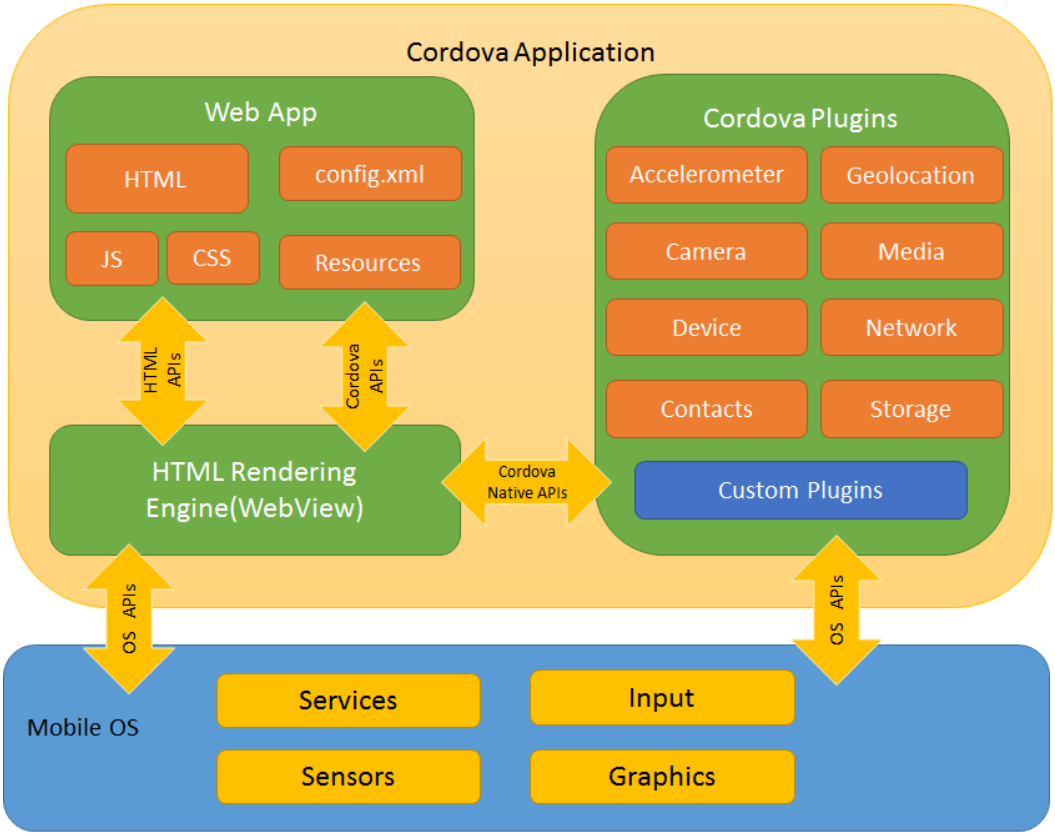


Figure 3.6: Cordova Application Architecture Cordova (2016)

3.4.2 Cordova Plugins

Cordova has many plugins avail be for the development, where the developer can access the device underline such as camera, network, GPS. The plugins are written in native code specific to each platform. There are two types of available plugins, one officially maintained by Cordova community and custom plugins usually available on the GitHub and they can be developed by any developer.

3.4.3 Cordova Performance

Cordova can access native device's underline with plugins. Access the underlie can slow down the performance of the application. Therefore is not a good option to develop applications that are using lots of CPU or GPU.

3.5 Mobile Platforms

This sections gives a short overview of the two most popular mobile platforms, in terms of market share, as resulted in Chapter 2.1. An mobile operation system will be hereby be referred as a platform.

3.5.1 Apache Cordova Android

As highlighted in Chapter 2.1, Android is one of the most popular mobile open source platform in the world. Wikipedia (2016a) states that Android was originally developed by Android Inc. and is currently owned by Google. It is based on Linux kernel and it is used in computers, TVs and mobile phones. It has a very good community support and it is available in over 70 languages. The latest release is Marshmallows. The main programing language is Java and supports multitasking and threading.

3.5.2 Apace Cordova IOS

IOS is the second most popular mobile platform, it was created by Apple Inc., it has its own ecosystem. It has a large variety of products such as the iPhone, iPad and iPod. The main programing language is Objective C and Swift. It is required to use only IOS SDK for developing apple applications which can be found only on Macintosh commuters. Apple (2016) states that there is $99 yearly development license for single developer and $299 for enterprise single membership to test and develop apple applications.

3.6 Microsoft Azure Mobile Apps

Microsoft Azure Mobile Apps is the new version of Azure Mobile Services, it is a fully managed Platform as a Service, mainly for Enterprise development and applications

integration. It offers scalability and global availability for mobile application development. It has full support to create cross-platform native mobile application with Xamarin or Apache Cordova. It enables a good number of features to mobile development, such as the followings:

- Push Notification
- Offline Support
- Data Access

Figure 3.7 provides a high level architecture of the Azure Mobile Apps container.

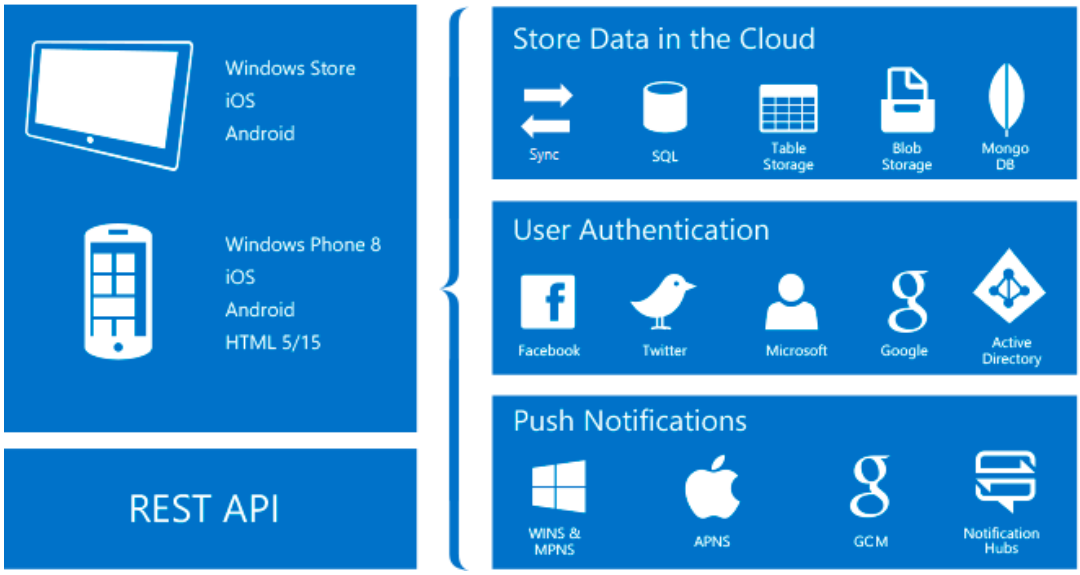


Figure 3.7: Azure Mobile Apps Architecture Hall (2016)

All the above mobile features together with the Azure Apps Services (e.g. Continue development, Isolated /Dedicated Environments) enhance the mobile development to a new level regarding performance and speed to the market of the mobile application.

3.6.1 Frameworks evaluation criteria

Framework evaluation criteria is based on an of measured parameters and are the following:

- Response time

  The Response time is an important parameter regarding user experience and it is how long it takes for an action to be completed, e.g., Application start-up,

loading a new page or complete a new task.

- Memory Usage

  It is defined by the amount of RAM memory is allocated by the application. Memory size can affect user experience on low end mobile devices. The memory is measured for an application start-up, navigation between pages and a maximum memory used at any stage by application.

- CPU usage

  The CPU is the percentage of the amount of total device CPU used by the application during measured period of time. It can be measured as an overall usage of the application or for a specific task, e.g., navigate between a page, application start-up.

- Battery consumption

  The battery consummation is the percentage of the battery drained by the application from the start-up and up to close.

3.6.2 Xamarin Test Cloud

Xamarin Test Cloud is a cloud-based platform service that provides an automate way of doing UI Acceptance Testing of mobile applications with hundreds of different mobile devices. It enables anyone to ensure that their application performs correctly and efficiently across a variety of devices with minimal effort. Additionally, because its cloud-based, the maintenance and procurement efforts are then removed from the test consumer, who can then focus on other efforts." (Xamarin Test Cloud, 2016a)

Developers can write tests in C# using the popular *NUnit* testing library or in Ruby with Calabash library, a popular library for Behavior Driven Development. Automated tests can be executed with any of the above libraries. Applications can be deployed for testing using the command line or the Xamarin Studio, which has fully integrated the Cloud Test in the IDE, where the applications can get tested with real devices on the cloud with only few clicks. The license is not free and has many subscription options available. Figure 8 shows the architecture of Xamarin Test Cloud.

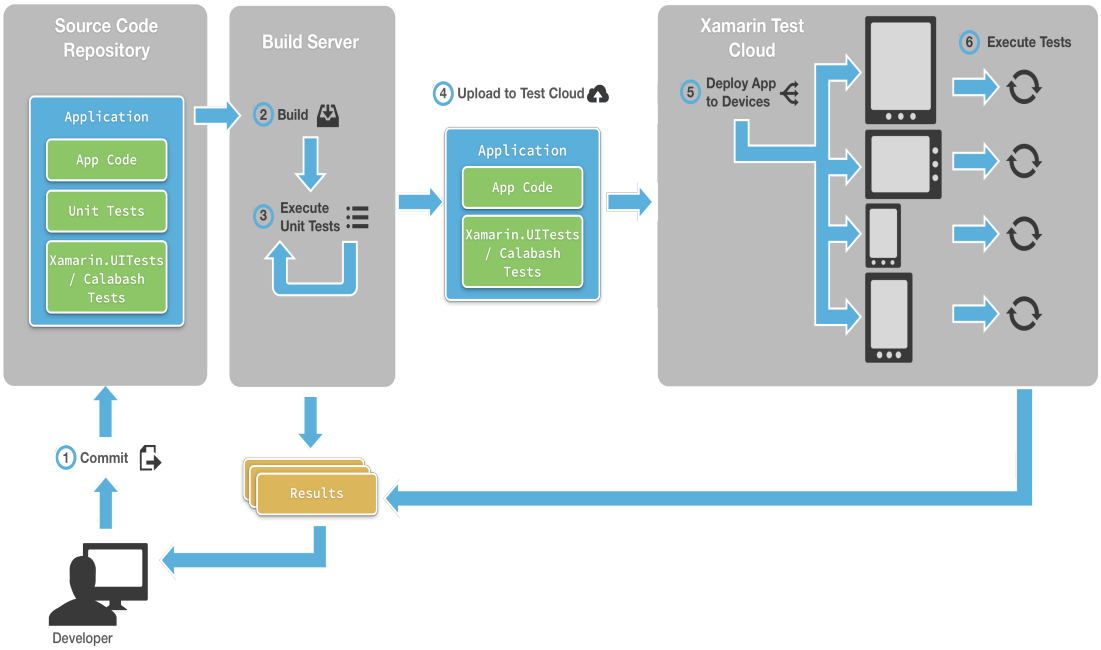


Figure 3.8: Xamarin Test Cloud Architecture Xamarin Test Cloud (2016b)

Xamarin Studio provides the best support to build test projects, which are based on the *Xamarin.UITest* framework, which is a framework that lets the developers to automate a test on a mobile device. It is based on *Calabash* and allows developers to write and execute tests in C# and NUnit. Xamarin Test Cloud requires to have implemented an automated *Xamarin.UITest* project before any app test is performed on the cloud with real devices.

The web console of the Xamarin Test Cloud, provides feedback regarding the UI functionality based on the written automated test and details regarding the app size, the amount of memory used during a test, step duration and the total time to complete the test from start to finish. Test log files are also available with full details regarding each step during the test.

Chapter 4

Application Requirements Analysis

4.1 Introduction - nCare Case Study

nCare is a start-up Enterprise company, looking to develop a mobile cross-platform application for nursing homes around Ireland based on Cloud Service as a Service model. After sign-up to this service, a nursing homes can provide daily updates and information about patients' status to their relatives, or any other general information that the care facility wants to pass over to their customers, e.g., a charity event, a Christmas party information.

At this stage the company is looking to implement a mobile cross-platform solution that covers the Android and IOS customers only, however a web interface for the nursing homes is on the table as the second priority of the implementation. Firstly, the company is looking to implement a prototype version of the application for Android and IOS, which can provide real results for analysis regarding application performance and running cost, thereafter to implement a final application based on this results.

4.2 Existing problem

Implementing mobile software for each mobile platform has a higher cost of developing and maintainability. After a research, nCare business management considered that the best available solution to implement the applications will be a cross-platform development approach with Cloud support for development testing and maintainability.

The last doubt they have, is to find out what is the best application development approach which will fully-feel successfully the business requirements between hybrid approach with Apace Cordova and native result with Xamarin.

In order to have an answer for the above doubt, an application prototype will be developed with identical design and functional requirements, with two of the most advanced existing cross-platform framework, Apache Cordova and Xamarin. The preliminary results will show-up which approach is more appropriate for the application development.

4.3 Project Scope

The scope is to create two application prototypes, first with Xamarin called *nCareXamarin* and secondly with Apace Cordova, called *nCareCordova* and to compare the prototypes based on quantitative performance analysis. The comparison results will suggest what will be the best approach for the final development.

In this project the "user" is defined as a client(s) which has a relative(s) reside in a nursing home and is using the *nCare* app to receive daily updates or any events information from that specific care facility. The application needs to be fully compatible with Microsoft Azure Mobile Apps, in order to gain the benefits discussed in Chapter 3.6. The application must include the following features:

- Authorize user
- View Messages only related to the authorized user
- View events related specific to the Nursing Home
- Application can work offline
- Message Push Notification
- Get direction to the Nursing Home where the relative of the user reside.

For the prototype version not all requirements are compulsory to be implemented, and for the final version they might change according to the final analysis of the business requirements. However, for accurate test result, the prototype applications need to be created identically between the implementations with identical functionality.

4.4 Functional Requirements

4.4.1 Overview

The nCare functional requirements are defined by the shareholders and covers all the needs that the users and the care facilities may need in a daily basis. From receiving messages related to a user, to crud functionality tasks relates to facility's admin and staff. Below are the details of each functional requirements.

4.4.2 Authentication

Users authentication is organized trough a third party provider, such as Goggle, Microsoft, Active Directory. User's email address, firstly needs to be added to a related patient resided in nursing home by the nursing home's facility's admin person, in order to be able to use the *nCare* mobile app, there is no option for a user to sign-up.

Nursing home specific requirements

This requirements are specific to nursing homes interface only, however they need to be considered as mobile applications and the nursing homes' web interface are sharing the same back-end and database. All facilities have a web interface to introduce patients data into the database, which will be accessed by the mobile users. Web access for the facilities care have different level of access. A facility admin has functionality rights to create, read, update and delete (CRUD) for nurses, patients, messages and events. Nurses can only add messages and send messages to a user. These messages are related to only one patient profile. Facility care staff authentication will be organized with Microsoft Active Directory, where they can use their work email address for authentication.

4.4.3 Messages

A message is defined as a summary of on overall health condition for a patient, and is updated daily by the nursing home. Users can only read messages related to their relative(s) with their mobiles. Nurses can only add messages to the system database.

4.4.4 Events

Events are defined as special messages which a facility care may need to send over to the users, e.g. a Christmas party or a charity day. This events can be send only by admin level access in the facility care, e.g., managers, general manager and owner.

4.4.5 Facility Care Directions

Users can get direction to the facility location from their mobile by using the provided nursing home location's address in GPS.

4.4.6 Admin requirements

An nursing home staff with admin level functionality access has access to create, read, update and delete (CRUD) for staff members, patients, messages and events.

4.4.7 Operational requirements

- Mobile cross-platform compatibilities
- Professional good look and feel

4.5 Use Case Diagrams

4.5.1 Class Diagram

Class diagram for the application is shown in Appendix C. The diagram is created in accordance with the nCare business requirements and it will be used to create application's domain model and the database by following strict the relationships.

4.5.2 Mobile User (Nursing homes' client)

Figure 4.1 shows the Use Case diagram for a user. A mobile user is able to login, view the messages, view events or location address specific related to his profile.

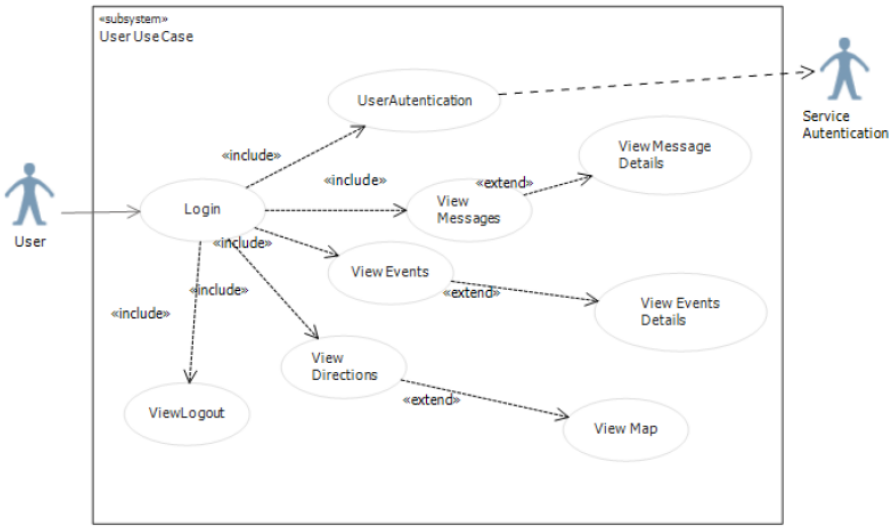


Figure 4.1: Use Case -Mobile User

4.5.3 Nursing Homes

Figure 4.2 shows the Use Case diagram for a Nursing Home, where an admin person has the authorisation level to add, edit and delete for messages, events and staff related specific to the location.

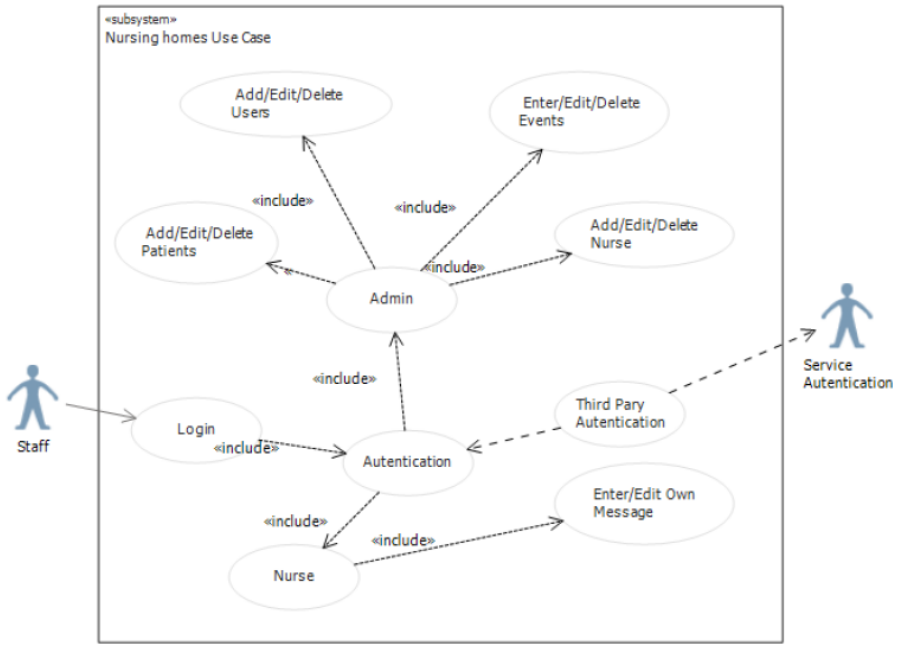


Figure 4.2: Use Case - Nursing Home

Chapter 5

Design

5.1 Overview

This section gives details of mobile user interface design. The mobile interface needs to have a professional look and feel, it is a requirement from the business owner and shareholders. In order to achieve this all the well-known design principles needs to be considered. Mobile interface design implementation is driven from the Mobile User use case diagram. Pages consistency trough all interfaces has be implemented. Similar design and functional requirements between the implementation with Xamarin and Apache Cordova must be implemented. This is required in order to be able to compare performance results between the implementations. The first step in the developing the application is to create a template for the application. The template includes navigation and all pages required by the Mobile Use Case diagram.

5.1.1 Application Icon

Application's icon is an image which is required by App Store for IOS and Google Play for Android apps. This image must be included in any release version before test the application on real device or publish the it to the store. The image requires different sizes which will be analyzed in Chapter 6.5. nCare application icon representation is showing in Figure.5.1. The icon is similar for Android and IOS.

Figure 5.1: Application Icon

5.1.2 Application Navigation

Mobile version of the *nCare* is going to be kept simple. It requires only few pages to implement all the functional requirements enumerated in Chapter 4.4, as a result a tab bar navigation pattern will be used for the navigation. Figure 5.2 shows layout details of the navigation bar template.

Figure 5.2: nCare - Navigation bar

5.1.3 Login page

The Login page is the landing page of the application were the user starts. This page is informative regarding login requirements. *nCare's* functional requirement requires that the user to have a valid email address registered with the nursing home before login. User authentication is validated by a third party e.g. Google. Figure 5.3. shows the layout the Login page.

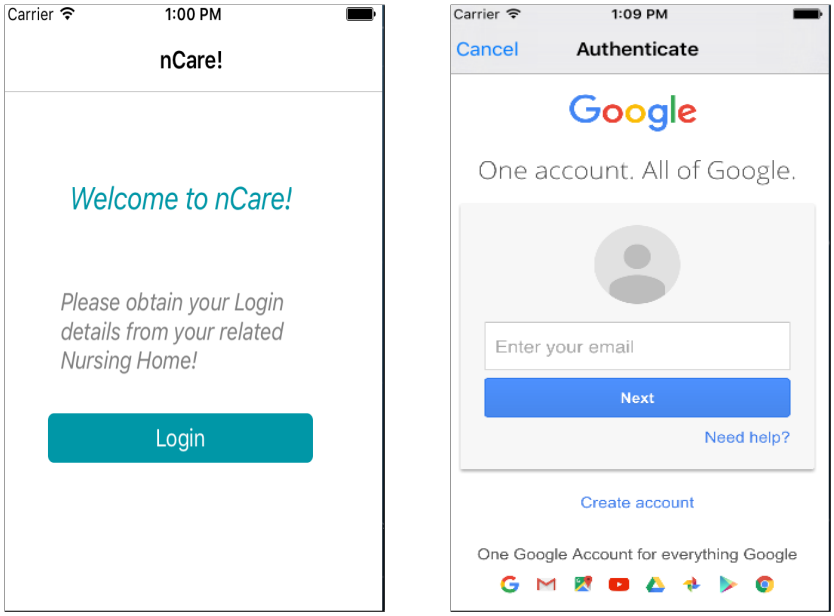


Figure 5.3: nCare - Login page

5.1.4 View Message and Message Detail page

After successfully authenticating, the Message page is the first page displayed to the user. This page contains a list of the latest messages related specific to the user. Each message is a link to the Message Detail page. The Message page provides a button to refresh the page for any new content. The Message Detail page has full details for a particular message from the list, including the name of the nurse which wrote the message.

Figure 5.4 shows the layout of the Message and Message Detail page.

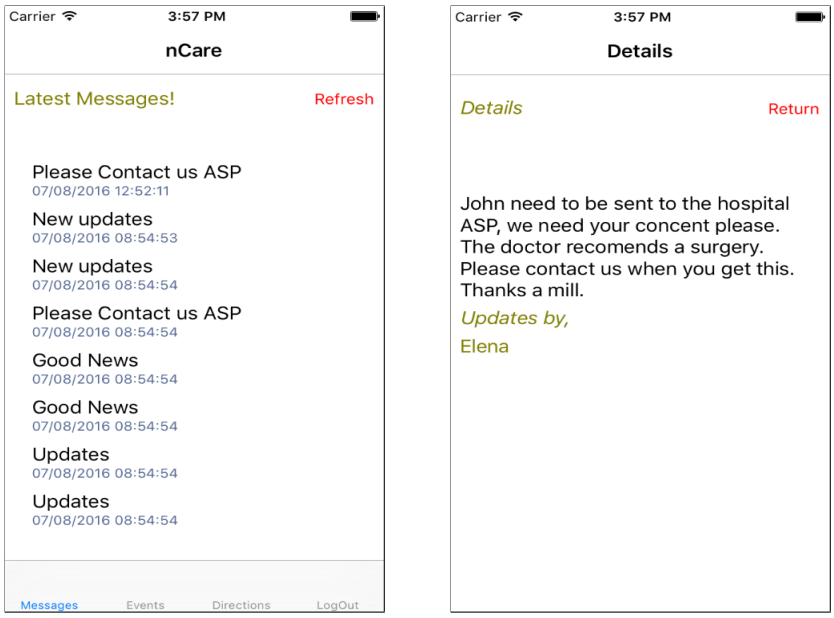


Figure 5.4: nCare - Message and Message Detail page

5.1.5 View Events and Event Detail page

This a page is dedicated to special events that a nursing home likes to organize, e.g. a Charity Event or a Christmas Party. Each event from the list is a link to the Event Detail page, where the user can read full details about an event. The Event page provides possibility to refresh the data if needed. The Event Detail page shows to the user full details about a specific event. Figure 5.5 shows the layout of the Event and Event Detail page.

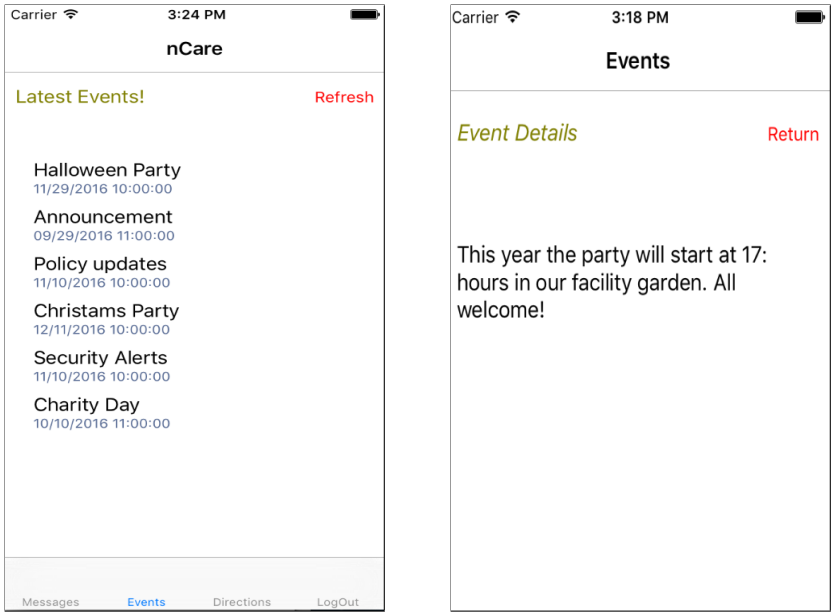


Figure 5.5: nCare - Event and Event Detail page

5.1.6 Direction Page

This page displays the Nursing home's address to the user right on the page, where with just one click the user can navigate to the location. This is very usefully when the user is driving. Figure 5.6 presents the layout display of the direction page.

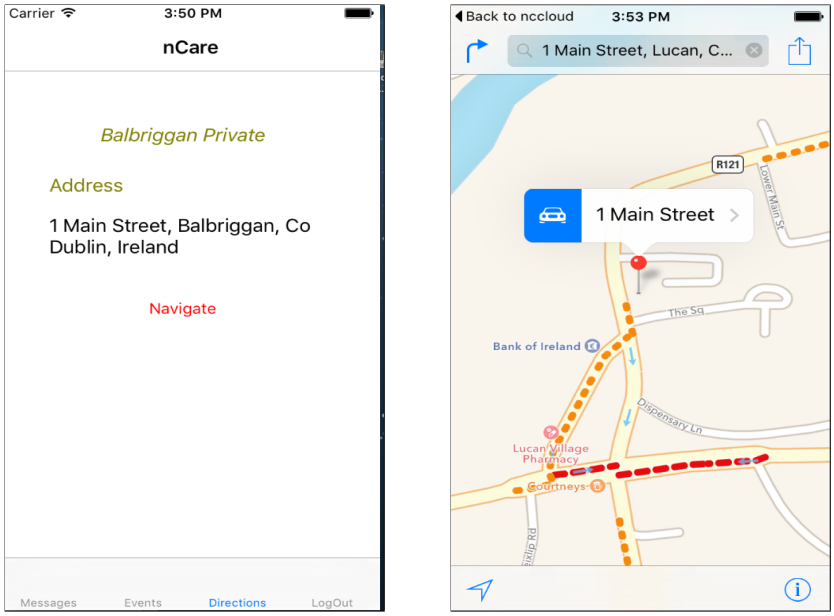


Figure 5.6: nCare - Direction display

5.1.7 Logout

This page is designed to fully log-out from the application. Full authentication will be required on return. Figure 5.7 presents the layout display of the logout page.

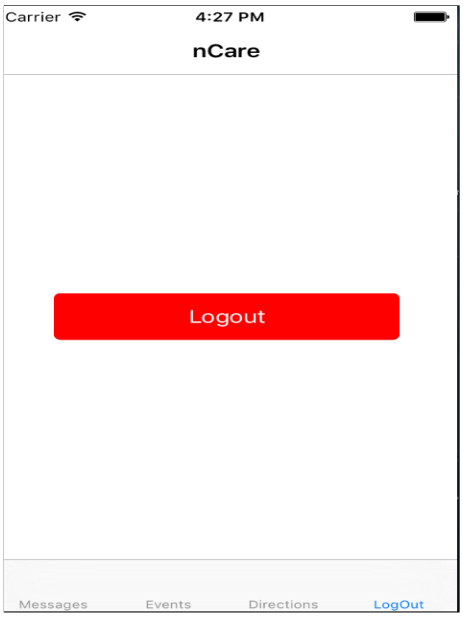


Figure 5.7: nCare - Logout page

5.2 Application Main Components

5.2.1 Application Backend solution

The backend solution will be deployed to the Azure Mobile Apps, which is a is a fully managed *Platform as a Service* cloud and a compressive mobile development platform for Enterprise applications. It offers full mobile capabilities to the developers, from push notifications to authentications and data access layer. The programming language can be in JavaScript or C# with .NET and Entity Framework. The nCare application's back-end will be fully integrated with Azure and will use .NET backend with Entity Framework.

5.2.2 Application Mobile Platforms

The application will be implemented separate with Xamarin and Apache Cordova with Mobile jQuery. For Xamarin implementation, the programming language is C# and

HTML5 with CSS3 and JavaScript for the Cordova implementation. These mobile platforms have full support from Azure platform for the development and production which can significantly speed-up the development and facilitate continuous development. Full details specific to each platform implementation will be available in Chapter 6.

5.2.3 SQL Server Database

Application's database will be SQL Server Database and will be hosted on the Microsoft Azure, in the same container with the backend solution. SQL Server Management Studio will be used to visualize and modify the database for the application development. Due to the cross-origin implications between Microsoft IIS Express local server and IOS application build and testing, which fully requires a Mac computer, an Azure cloud based database will be considered for the development of the application, in order to avoid complicate custom IIS Express local settings.

5.2.4 Domain Model

Represent the business logic for the applications. It contains behaviors of the objects and relationships between them. The model classes are defined form the UML diagram available in Appendix C.

5.2.5 Data Access Layer

Data access layer is used to store and retrieve data from the database. It has the responsibility to persist the data between the database and user. The Domain Model will includes the data access functionality.

5.2.6 Xamarin Test Cloud

A test project will be created for the Apace Cordova and Xamarin application implementation. Each application will be tested with an identical test on the Xamarin Test Cloud platform, which is a cloud based mobile center with thousands of mobile devices ready to test applications.

Chapter 6

Implementation

This section describes in details the implementation of the applications and the setup of the Cloud resources. Firstly, starting by creating the Cloud resources, followed-up with the implementation of the server-side solution and then separate front-end implementations of the application with Xamarin and Apache Cordova framework.

6.1 Cloud Resources

The cloud provider chosen by shareholders of the *nCare* was Microsoft Azure. Microsoft Azure provides cost effective IT infrastructure for many enterprise companies around the world. Recently they provide a new service called *Azure Mobile Apps*, a cloud service based platform which provides compressive support for mobile applications, e.g. Push Notifications, Application Insights and Content Delivery Network (CND). These resources are flexible regarding subscriptions and usage management.

For better management and performance, all the cloud resources are located into a single Resource Group, hosted into a single container. Keeping them together improves user experience and minimize any network legacy and data upload/ download charges. The Resource Group includes a Web Server, SQL Database and an App Services with Google enabled as the application authenticator. At present, as an application authenticator, Azure Services supports fully Google, Facebook, Twitter, Microsoft and Active Directory. For the developing and testing the applications Google is the only authenticator going to be implemented. Google is a very popular authenticator which supports common OAuth 2.0 scenarios (Google Developers, 2016). Google requires a developer account subscription which costs $25 (`https://console.developers.google.com`).

Before enable any authenticator provider in the cloud console portal, is required to have an *Client Id* and a *Client Secret Key*. All steps required to create the above resources are available in Appendix F.

Azure cloud resources are located is in the North Europe and for the development and testing duration are based on a *Pay-As-You-Go* subscription type, with 1 Core CPU and 1.75GB RAM with 10GB storage. However, for the production, those resources might be necessary to be redesigned accordingly to the business demand.

Those resources are initially necessary for the application to run on the cloud. However, Azure provides many resources that can be added in at any stage to improve the performance of the application and to increase user experience.

6.2 nCare Server-side solution implementation

Azure Mobile APP service has two implementations available to the developers, JavaScript based on NodeJS and .Net based on C#. JavaScript has more limited flexibility, where all development is implemented only in the cloud console compare to the .Net solution were the implementation can be developed locally in Visual Studio, then deployed on the cloud on the Mobile Apps. nCare solution for the server-side will be implemented with the .Net technologies and deployed to the Azure Mobile Apps for hosting and testing. The initial project implementation steps are available in Appendix D1. The main components of the solution implementation include the following: Domain Model, Data Access, Controllers and Database.

6.2.1 Domain Model and Data access representation

The Domain Model is an important part in the solution. The entities and the entities relationships must implemented based on the UML class diagram from Appendix C.

The model entities are the following: Location, Patient, Customer, Message and Event. Full relationships based on the UML diagram are available in Appendix D2 and implementation details of the Data Access class are available in Appendix D3 .

All the above entities are extending the *EntityData* class, which is designed specific to the Azure mobile applications. This class is required for push notifications implementations. It provides properties like time stamp and data version control which are usefully to synchronize mobile offline data with database. The *EntityData* class content is shown in Figure 6.3.

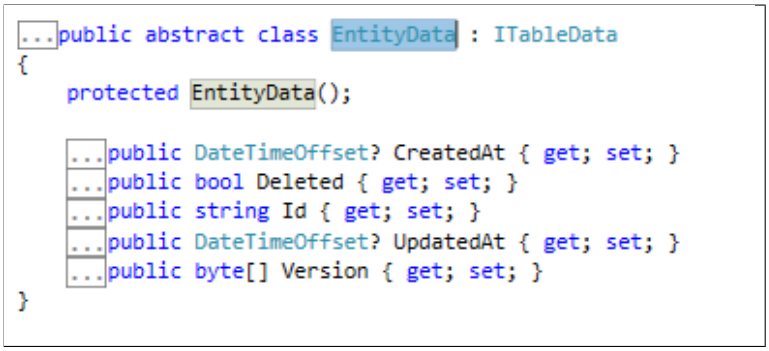


Figure 6.1: EntityData class

6.2.2 Controllers

The Controllers have the responsibilities to interact between user's view and the domain models. The controllers require the user to authenticate before any data request is processed. The methods are specific supporting the mobile solution only and have limited functionality implemented, e.g. a mobile user can not add a message or delete an event. Further implementation is required for the Web version which will be available into the nursing homes and part of another project.

In each controller, user's email is retrieved from the login authentication process. This was possible by implementing the *ClaimsPrincipal* class, which is a special class designed to do this in the Microsoft.Net framework. Listing 6.1 shows how to retrieve user's email from the authentication with Google.

```
1  //get user email address
2  private string GoogleSID() {
3    var principal = this.User as ClaimsPrincipal;
4    var sid = principal.FindFirst(ClaimTypes.NameIdentifier).Value;
5    return sid;
6  }
7  private async Task < string > GetEmailAddress() {
8    var credentials = await User.GetAppServiceIdentityAsync < GoogleCredentials > ( ↩
         Request);
9    return credentials.UserClaims.Where(claim = >claim.Type.EndsWith("/emailaddress")) ↩
         .First < Claim > ().Value;
10 }
```

Listing 6.1: Retrive User's Email address in Controller

After user's email address is retrieved, the next step in each controller is to query the database based specific on user's email address and on UML class diagram. All queries are processed on the server side, as a result, user experience cannot be affected

on mobile devices with low performance regarding CPU and memory. Full controllers implementation are available in Appendix D4.

6.2.3 Database Design

On publishing the solution to Azure, database concrete implementation is generated based on code first approach model. Entity Framework is responsible for the implementation of the database based on the entity models relationships. Full details of the Database implementation are available in Appendix E.

6.3 nCare Xamarin Forms Implementation

Currently after Microsoft has acquired the Xamarin company (Guthrie, 2016), Xamarin framework is available for free for a single developer or small teams with limited framework accessibility, and under subscription license for enterprise, with more details available at the following link (`https://store.xamarin.com/`). However, at the start of this research paper, Xamarin was available for free only one month and thereafter the paid versions were available at $999 a year for business and $1899 for enterprise. This application was developed under business subscription with full access to the framework resources and Xamarin Cloud Mobile Testing.

Xamarin framework can be installed in the Visual Studio 2013 or Visual Studio 2015. It has its own IDE available of Mac computers (Macintosh) only. If the framework is installed into the Visual Studio, it required a remote connection to a Mac computer for build IOS applications. Apple requires a developer subscription for testing and developing IOS applications, which costs $99 each year for a single developer, more details are available at the following link (`https://developer.apple.com/programs/enroll/`). Full details of how to setup the development environment are available in Appendix G1.

The solution build with Xamarin forms is called *nCareXamarin* and implements all the requirements specifications from Chapter 4 and has the UI implemented according to the design specifications from Chapter 5. Full implementation details of the solution are available in Appendix G2.

6.4 nCare Apache Cordova Implementation

Apache Cordova is a free mobile development tool with great community support. The development environment has been already installed with the Xamarin Forms with full details available in Appendix G1.

The solution build with Apace Cordova is called *nCareCordova* and implements similar to the *nCareXamarin* solution build with Xamarin Forms all the requirements specifications from Chapter 4 and has the UI implemented according to the design specifications from Chapter 5.

Full implementation details of the solution are available in Appendix H where is described in details the implementation of the application with Apache Cordova framework. Starting by setting up the development environment, creating the solution in Visual Studio followed by user interface implementation (UI) and adding JavaScript support for each page.

6.5 Preparing the applications for release

This is a very important part in the development process. This section describes in details how the applications implemented earlier in section 6.3 and 6.4 are prepared to be ready for testing and for publish to the stores. Testing the application on Xamarin Test Cloud platform with real devices requires that the application is compiled into release mode and have all security implemented. In other words, the application must have implemented the requirements for publishing on Google Play or App store and the file format is *".apk"* for Android and *".ipa"* for IOS application.

6.5.1 Preparing for release Xamarin Forms Android

The process has the followings steps:

- Add application's icon images in the Resources folder with all recommend sizes (Android API Guides, 2016).
- Disable debugging. This will prevent the user from trying to debug the application on the device.
- Specify application version number and the package name in Android Manifest file. The package name convention is to be unique and have the following format: *"com.organizationName.appName"* as showing in Figure 6.2.

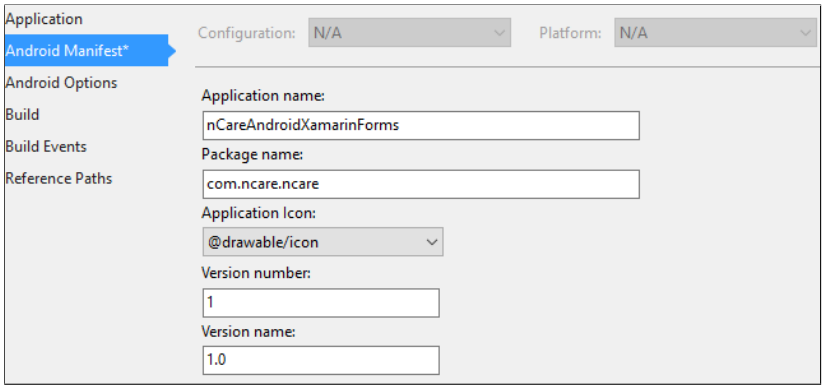


Figure 6.2: *nCare.Droid* Xamarin Forms Android Manifest release details

- Add a *private keystore*. A keystore is a security certificate created by a program tool called *Keytool* which is available in Java SDK. The key is valid only for one application and should be created only once. Listing 6.2 shows the commands for creating a keystore certificate.

```
1  C:\Windows\system32>cd C:\Program Files (x86)\Java\jre1.8.0_101\bin
2  C:\Program Files (x86)\Java\jre1.8.0_101\bin>keytool -genkey -v -keystore c:\ ↩
       nCare.keystore -alias nCareAndroidXamarinForms -keyalg RSA -keysize 2048 - ↩
       validity 50000
3  Enter keystore password:
4  Re-enter new password:
5  What is your first and last name?
6    [Unknown]: ionel
7  What is the name of your organizational unit?
8    [Unknown]: ncare
9  What is the name of your organization?
10   [Unknown]: ncare
11 What is the name of your City or Locality?
12   [Unknown]: dublin
13 What is the name of your State or Province?
14   [Unknown]: co
15 What is the two-letter country code for this unit?
16   [Unknown]: ir
17 Is CN=ionel, OU=ncare, O=ncare, L=dublin, ST=co, C=ir correct?
18   [no]: yes
19 Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA)  ↩
       with a validity of 50,000 days
20         for: CN=ionel, OU=ncare, O=ncare, L=dublin, ST=co, C=ir
21 Enter key password for <nCareAndroidXamarinForms>
22         (RETURN if same as keystore password):
23 Re-enter new password:
24 [Storing c:\nCare.keystore]
```

Listing 6.2: Create an Android keystore certificate

- Add the *keystore certificate* in the application. This is done by setting Visual Studio build Application in release mode, connect a real Android device for build, then go to tools tab and select publish and upload the keystore created in Listing 6.3 including passwords and alias name and password.

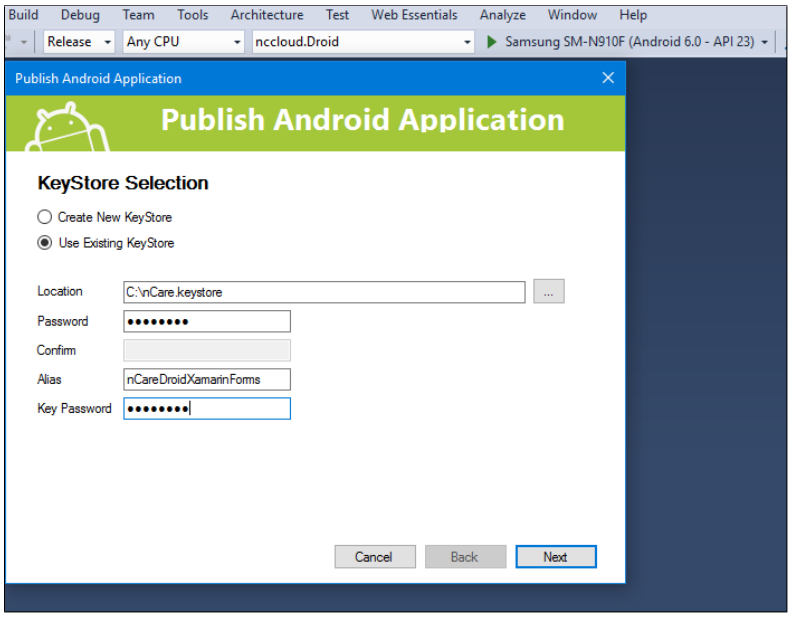


Figure 6.3: *nCare.Droid* Xamarin Forms keystore upload

- The application is fully ready to be published. The *com.ncare.ncare.apk* file is locate in the project solution folder in the *nCare.Droid* bin release folder. This file is final and is ready to submit to Google Play store for approval.

6.5.2 Preparing for release Xamarin Forms IOS

Preparing nCare.IOS for release is a more complicate process compare with Android version and it has the following steps:

- Add application's icon image in *nCare.IOS* Resources folder. IOS are very strict regarding images size and resolution, more information is available at the following link (Apple Developer, 2016).

- Create a provisioning profile. This is a complex process. It requires to have an apple developer licence and a developer certificated installed in the Mac computer. The steps are the following:

  - Login Apple developer console and create a certificate. To create a certificate requires two steps. The first is to go into the Mac computer, open the

Keychain Access and request a certificate from the *Certificate Authority* by following the wizard process and download it into the computer. The second step is to go to the Apple developer console and create a distribution certificate. This process requires to upload to Apple the certificate created earlier.

- Go to Provisioning profile and create an App Store distribution profile, follow the wizard and give a name. This name must be unique and represents the application file name on the App store. The format is *com.companyName.appName*.
- The last step is to download the certificate into the Mac computer. This can be also done from the Xcode. Figure 6.4 illustrates the distribution certificate ready for download.

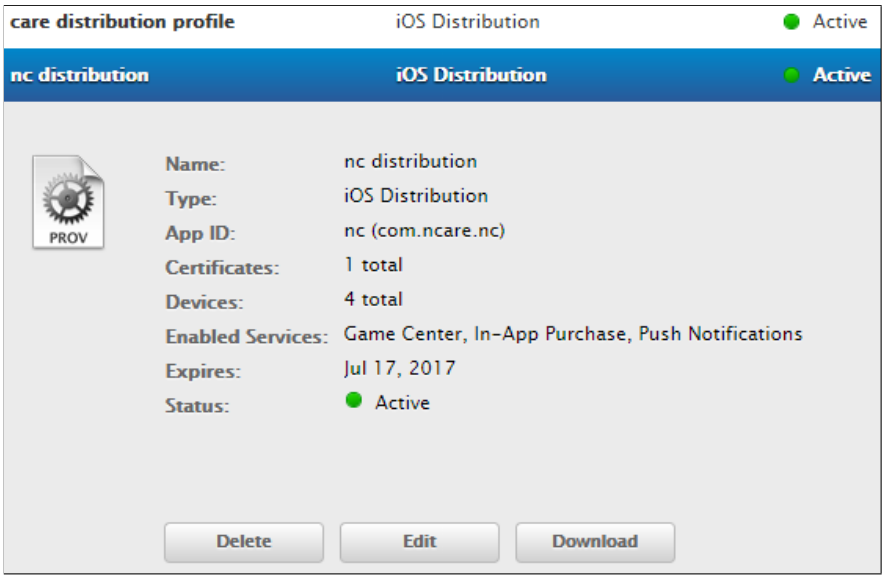


Figure 6.4: *nCare.IOS* Distribution Profile

- For testing the application locally with real IOS device, Apple requires to register each device first in the developer console before let you upload the app. To register a device is required the device's *UDID number*.
- The last step is to include the name of the provision profile in the application. This is done going in Visual Studio, in *nCare.IOS* project properties, go to *IOS IPA Options* tab and add into the *Package name* the provisioning profile implemented earlier as shown in Figure 6.5.

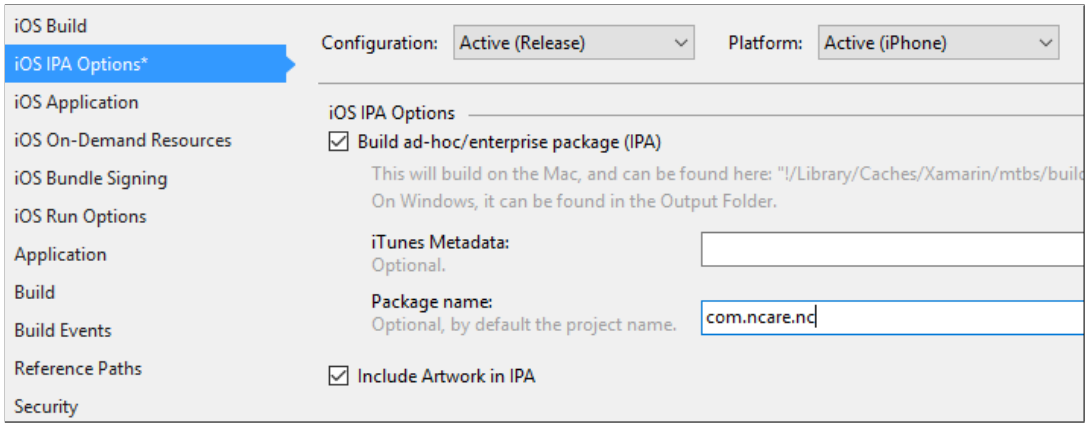


Figure 6.5: Add provisioning profile to *nCare.IOS*

- The last step is to connect an iPhone device with the Mac computer and build the application in release mode with the iPhone device selected. The device must be registered first in the Apple developer console. The application is now ready for testing with Xamarin Test Cloud or to be submitted to the App Store for publishing approval. The *com.ncare.nc.ipa* file is located in the application release iPhone folder. For the last step Xamarin Studio from the Mac computer provides an easier way of preparing the application for publishing and release with less possible compiling errors.

6.5.3 Preparing for release Cordova Android

Preparing the application for release requires the following steps:

- Include icon images in android sub-folder which is located in the solution's *res* folder with the image sizes recommended in section 6.5.1.
- In the *nCareCordova* solution, go to *config.xml* file and in the *Common tab* add a package name in similar format as the one showing in Figure 6.2, i.e *com.ncare.cordova*.
- Create a keystore by following the same steps as in Listing 6.2.
- Add the generated keystore key in the project. This is implemented by going into the solution and open the *ant.properties* file locate in the *res* folder and add into the file the *keystore file* details as showing below in Listing 6.3.

```
key.store=C:\\ncare_key.keystore
key.alias=com.ncare.andcordova
key.store.password=Mypasswordhere
key.alias.password=Mypasswordhere
```

Listing 6.3: nCareCordova keystore details in ant.properties file

- Connect an Android mobile device to the Visual Studio and build the application in the release mode with the connected device selected.
- The final Android version of the application called *com.ncare.andcordova.apk* is located in the bin/Release folder. This is the file which can be submitted for approval to Google Play store or to use for Xamarin Test Cloud.

6.5.4 Preparing for release Cordova IOS

Preparing the application for release requires the following steps:

- Include images in the *ios* sub-folder locate in the solution's *res* folder. The images must comply with the IOS requirements similar with IOS Xamarin Forms implementation from section 6.5.2.
- Create a provision profile. This can be implemented with similar steps as in section 6.5.2. The final implementation is shown in Figure 6.6.

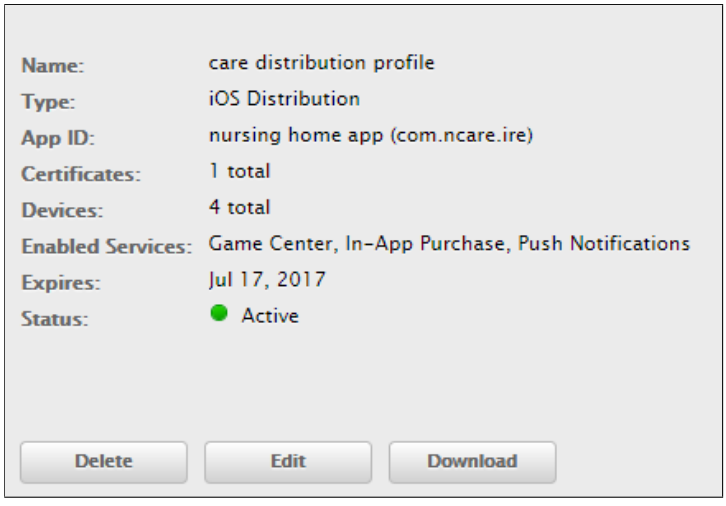


Figure 6.6: Add provisioning profile to *nCareCordova* project

- Add the provision profile in the project, which can be done by opening the *config.xml* file and add the provision profile *name* created earlier in *Common tab* as shown in Figure 6.7.

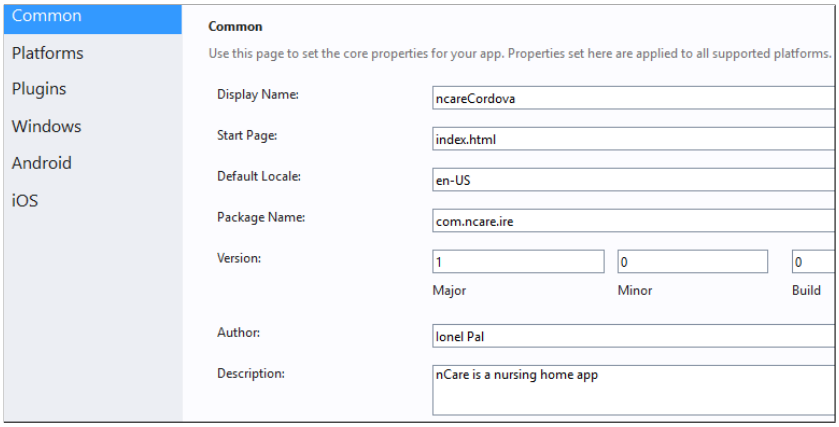


Figure 6.7: IOS provisioning profile to *nCareCordova* project

- Connect an iPhone device to the Mac computer, make sure that Visual Studio is connected with the Mac computer and build the application in the release mode with the iPhone device selected. In the bin folder is the location of the final file ready for the testing with real devices.

Chapter 7

Evaluation

In this chapter are evaluated the performance comparison of the properties based on quantitative analysis of Apace Cordova and Xamarin frameworks. Starting by planning the testing architecture in section 7.1, followed-up with the test implementation in section 7.2, gathering the results in section 7.3 and lastly by analyzing the results in section 7.4.

7.1 Testing Plan

The testing plan is to simulate real users using the applications implemented with Xamarin Forms and Apace Cordova in Chapter 6 and to analyze the performance metrics of the memory and CPU usage for loading the application and navigation between pages for each implementation. These can be achieved by implementing an automate test with *Xamarin.UITest* and Xamarin Test Cloud, an automate test that can be deployed to the cloud from the Xamarin Studio IDE. It is important that each application is tested identical with an identical test plan. The testing pages sequence are shown in Table 7.1.

1	Login Page
2	Authenticate with Google
3	Message Page
4	Message Detail Page
5	Event Page
6	Event Detail Page
7	Location Page
8	Logout page

Table 7.1: Application Testing plan sequence

Xamarin Test Cloud provides a very large variety of mobile device for testing based on subscriptions. However, due to the limitation of the existing student subscription, the student has chosen to test the applications with a small number of low and high-end devices, which covers a good area of most popular users' devices as follows:

Device Name	OS version	Memory	Processor
Samsung Galaxy S4	Android 5.0.1	2GB	1.5GHz
Samsung Galaxy S6	Android 5.1.1	3GB	2.1GHz
iPhone 5s	iOS 9.2	1GB	1.3GHz
iPhone 6s Plus	EiOS 9.3.3	2GB	1.84GHz

Table 7.2: Cloud mobile devices chosen for testing

For accurate testing results, each application will be tested on each mobile device (Table 7.2) for seven times. For each set of tests performed, an average and standard deviation will be calculated for each performance metric (Section 7.1.1). The tests results provide the data for the comparison between the frameworks.

7.1.1 Performance Metrics

The performance metrics that are the subject of the test results analysis and are based on the automate test implementation. The performance metrics are available on the Xamarin Test Cloud portal after each test completion. The performance metrics are defined as follows:

- *App size* - App size is the total size of the application measured in megabytes (MB). The application is uploaded in the release mode.

- *App loading* - App loading time is defined by how long it takes to the automate test to load the application after simulating a tapping gesture on the app icon and how much memory was required to complete this process. The loading time it is measured in seconds (sec) and the memory consumed is measured in (MB).
- *Page navigation* - Page navigation is defined by how long it takes to the automate test to navigate with the navigation bar from the *Direction* page to the *Logout* page and how much memory was necessary to complete this process. These pages were chosen because they have no dynamic content loading on the page at the page appearing. For accurate results it is important to avoid network loading time differences for dynamic pages which retrieves the content from other sources. The navigation time is measured in seconds (sec) and the memory also in (MB).
- *CPU usage* - Is defined the percentage of the total device's CPU used by the automate test to complete a specific given task. It is measured in percentage (%) .

7.2 Testing implementation

Testing the applications on Xamarin Test Cloud requires to have Xamarin UITest project implemented with Xamarin Studio on a Mac computer. However, due to different implementation language between the two solution, e.g., HTML/JavaScript versus C#, the tests were required to be written twice in order to work on the Xamarin Test Cloud. The first test targets the native UI for Android and IOS implemented in Xamarin Forms project (*nCareXamarin*), and second test targets the HTML elements from the mobile browser implemented in the Cordova project (*nCareCordova*). The test steps are identically implemented for each application. Full tests implementation details are available in Appendix B1 for application implemented with Xamarin and Appendix B2 with Cordova. Each solution is tested in on Xamarin Test Cloud for seven times on with each device enumerated in Table 7.2. The tests results are available in the next section.

7.3 Test Results

The following sections displays the findings of the data recorded for all tests completed in section 7.2 on Xamarin Test Cloud for both enterprise application, based on the metrics defined in section 7.1.1. The analysis of these findings will be discussed section 7.4. Each test was conducted on different dates and time and on different mobile devices

and platforms (Table 7.2). All data from the Xamarin Cloud Test portal was gathered on an Excel spreadsheet and is available in Appendix I2. Xamarin Test Cloud provides limited functionality regarding downloading test images, hence, as a viewing example, we have taken a screenshot which is displayed in Appendix I1

7.3.1 Launch Time

The measure time of the *app launch* and *navigation page* are the defined by how long takes to the automated test (section 7.2) on the Xamarin Test Cloud to complete these tasks and are available below in Table 7.3.

	Android				**IOS**			
	Galaxy S4	SD	*Galaxy S6*	SD	*iPhone 5S*	SD	*iPhone 6S-Plus*	SD
App Launch Time (sec)								
Xamarin	24.5	1.1	24.9	2.4	30.5	2.1	26.5	0.4
Cordova	20.3	1.0	22.3	2.8	32.9	3.5	27.7	1.9
Page Navigation Time (sec)								
Xamarin	5.8	0.2	10.1	0.7	3.1	0.4	2.9	0.1
Cordova	6.2	0.4	10.3	0.4	3.1	0.3	3.1	0.4

Table 7.3: App Launch Time

The average results show that for low-end Android devices, e.g., Galaxy S4, applications build with Cordova takes approximately 21% less time to load the application. However, for high-end Android devices, e.g., Galaxy S6, the results show no significant difference in the app loading time.

IOS results shows that there is no significant difference between an application built with Xamarin and Apace Cordova. A graph representation is available below in Figure 7.1.

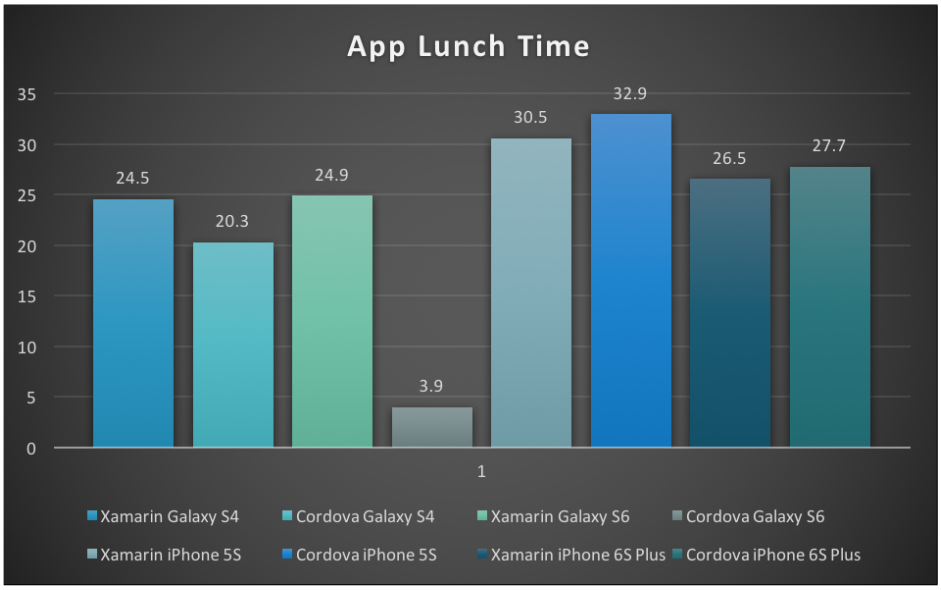


Figure 7.1: App Launch Time

According to the average results (Table 7.3), there is no significant difference for page navigation response time between an application built with Xamarin and Apace Cordova. Figure 7.2 illustrates a graph representation of the page navigation results.

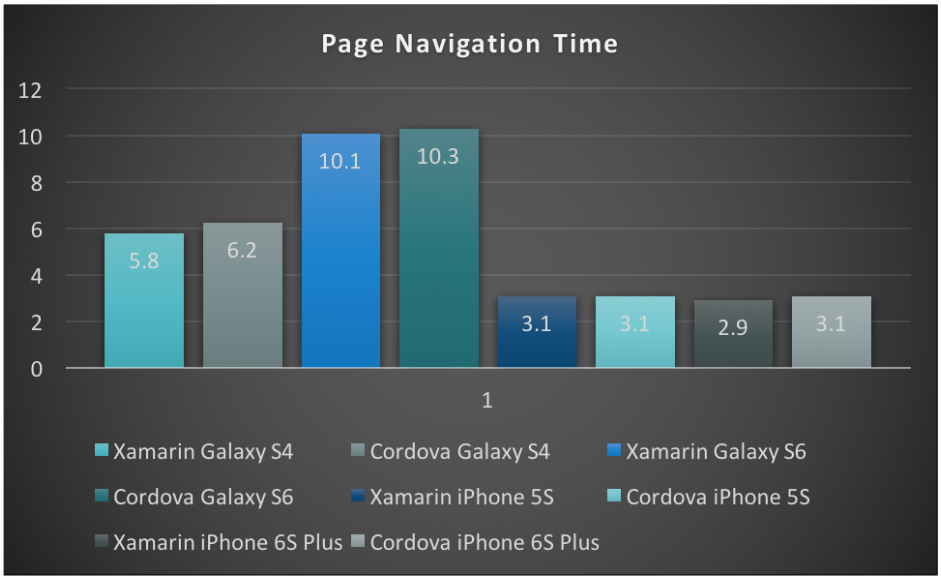


Figure 7.2: Page Navigation Time

7.3.2 Memory Consumption

Memory consumption average results (Table 7.4) for the *application launch* indicates that there is a clear difference for high-end Android devices were applications build with Xamarin framework consume approximately 21% less phone memory at the app launch. However, for low-end devices the results show that there is no significant difference between the applications tested.

Furthermore, the IOS results indicates insignificant difference ($< 7\%$) for high-end devices, e.g., iPhone 6S Plus, and small difference (approx 13%) for low-end devices, e.g. iPhone 5S. The *memory peak* results indicates that the highest memory consumptions for Android and IOS are recorded at the application launch.

	Android				**IOS**			
	Galaxy S4	SD	*Galaxy S6*	SD	*iPhone 5S*	SD	*iPhone 6S-Plus*	SD
Memory consumption at the application launch (MB)								
Xamarin	225.8	8.5	280.2	7.7	74.8	1.4	104.8	1.8
Cordova	225.5	20	340.6	11.0	66.2	2.5	98.0	1.3
Memory consumption at page navigation (MB)								
Xamarin	182.9	2.2	246.5	1.2	74.5	1.4	103.7	1.4
Cordova	199.0	3.6	276.5	4.2	66.0	2.6	97.5	1.4
Peak Memory consumption (MB)								
Xamarin	225.7	8.6	280.2	7.7	74.8	1.4	104.8	1.8
Cordova	222.2	20.3	340.6	11.0	66.2	2.5	98.0	1.3

Table 7.4: Memory consumption

Figure 7.3 illustrates a graph representation of the memory consumption records during the launch application.

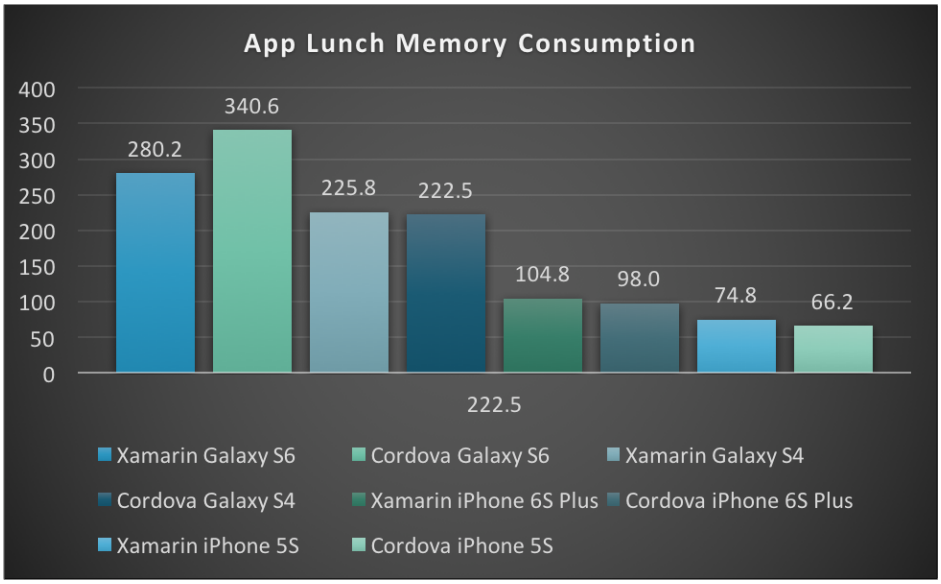


Figure 7.3: App Launch Memory Consumption

The average results (Table 7.4) for the memory consumption at *page navigation* indicates that high-end Android devices applications built with Apace Cordova consumes approximately 12% less memory compare with applications build with Xamarin. However, for low-end Android devices the memory consumption is approximately 9% greater compare with Xamarin.

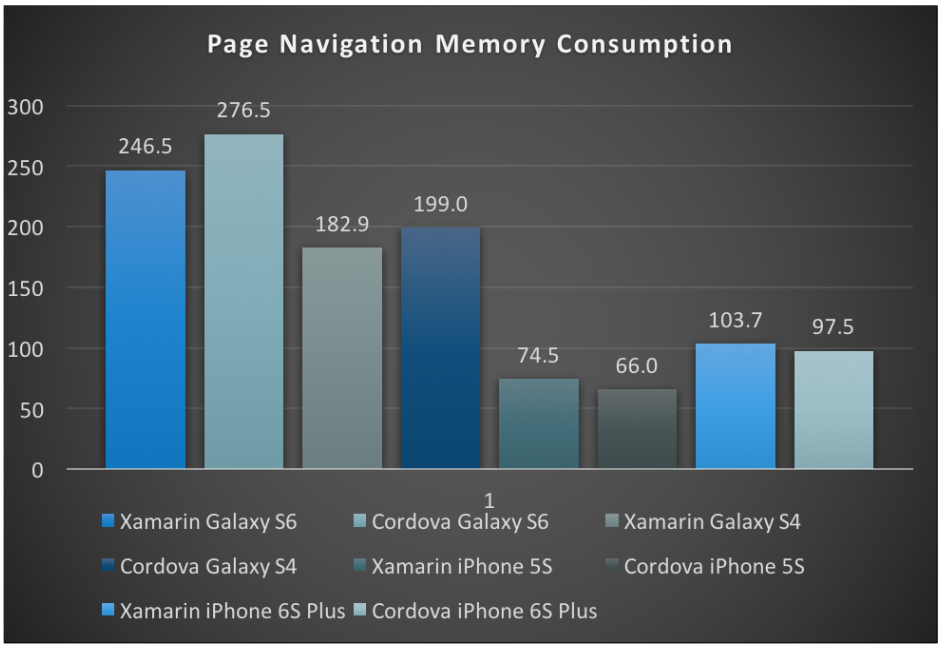


Figure 7.4: Page Navigation Memory Consumption

7.3.3 CPU usage

The average CPU usage (Table 7.5) shows that Xamarin Android applications have a higher CPU usage during the launch of application, compare to similar applications built with Apace Cordova. The results indicate approximately 37% more CPU usage for low-end Android device and 29% for the high-end.

	Android				**IOS**			
	Galaxy S4	SD	*Galaxy S6*	SD	*iPhone 5S*	SD	*iPhone 6S-Plus*	SD
CPU usage at app loading (%)								
Xamarin	102	7.1	92	10	27	3.3	17	0.9
Cordova	65	8.7	65	25	28	1.1	17	1.5
CPU usage at page navigation (%)								
Xamarin	27	3.8	27	12.1	5	1	3	1.2
Cordova	49	16.2	26	16.3	10	1.1	8	1.4

Table 7.5: CPU Usage

Figure 7.5 illustrates a graph representation of the CPU usage records during the application launch.

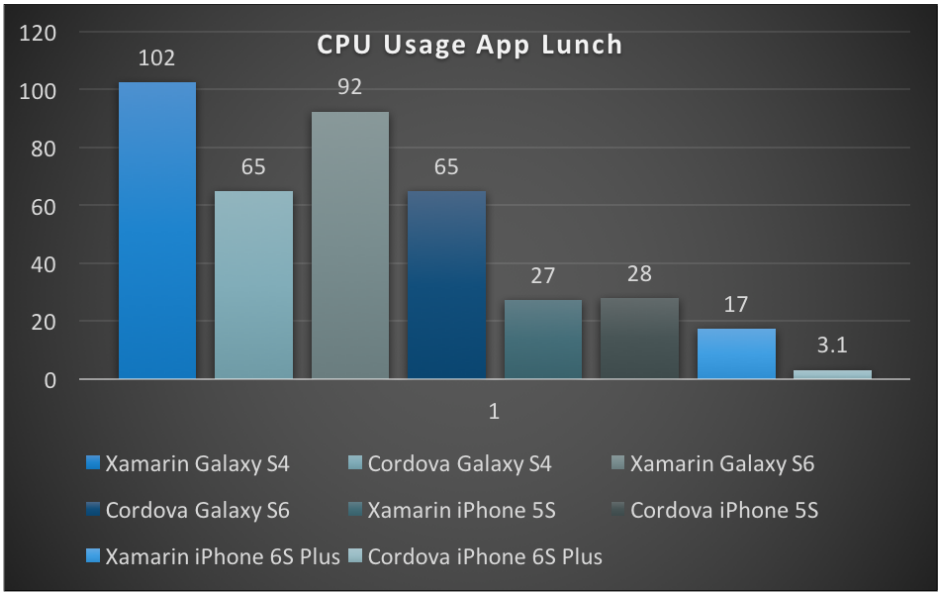


Figure 7.5: App Launch CPU Usage

The table (Table 7.5) shows that at the app launch, there is no significant difference on average CPU usage between the applications build with Xamarin and Apace Cordova.

Figure 7.6 illustrates a graph representation of the CPU usage records for the page navigation.

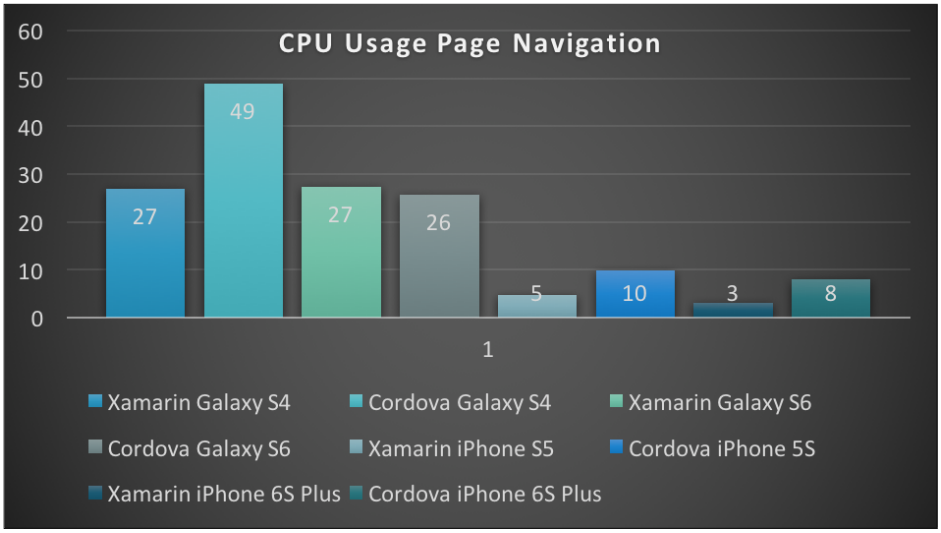


Figure 7.6: Page Navigation CPU Usage

7.3.4 App Size

According to the results (Table 7.6), application built with Xamarin have considerably bigger size compare with the same application build with Apace Cordova.

	Android	**IOS**
Xamarin	20.21	20.21
Cordova	3.93	3.93

Table 7.6: App Size (MB)

7.4 Android versus IOS

The aim of this section is to compare at a high level the performance properties test results recorded for low and high-end Android and IOS applications built identically with the same framework.

7.4.1 Xamarin Android versus Xamarin IOS

Figure 7.7 and Figure 7.8 shows clearly that the memory consumption is significantly bigger with over 200% during the launch applications and over 160% during the page navigation for Android applications compare to IOS. Likewise, CPU usage for Xamarin applications have an approximately 270% bigger usage compared to Cordova applications. Other properties have a lower significant difference.

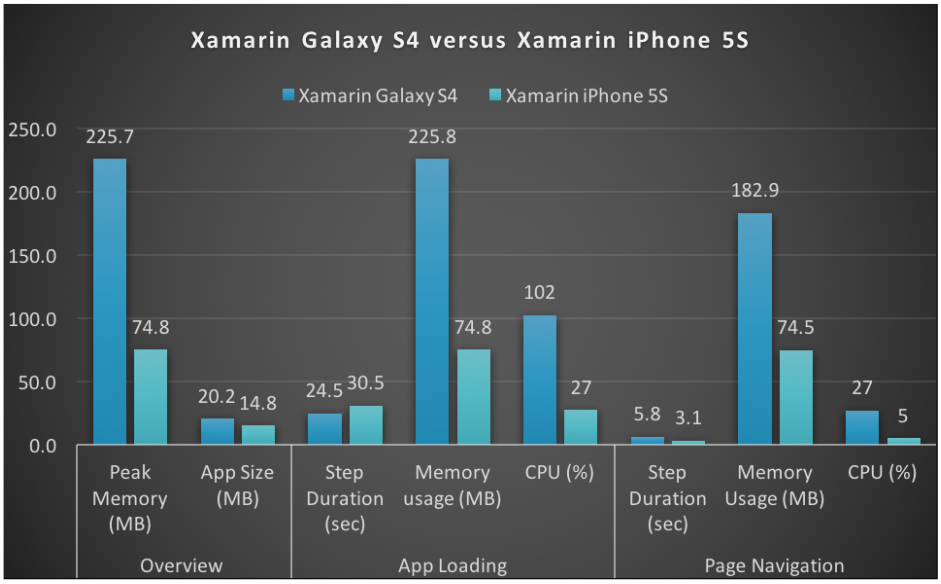


Figure 7.7: Samsung Galaxy S4 versus iPhone 5S

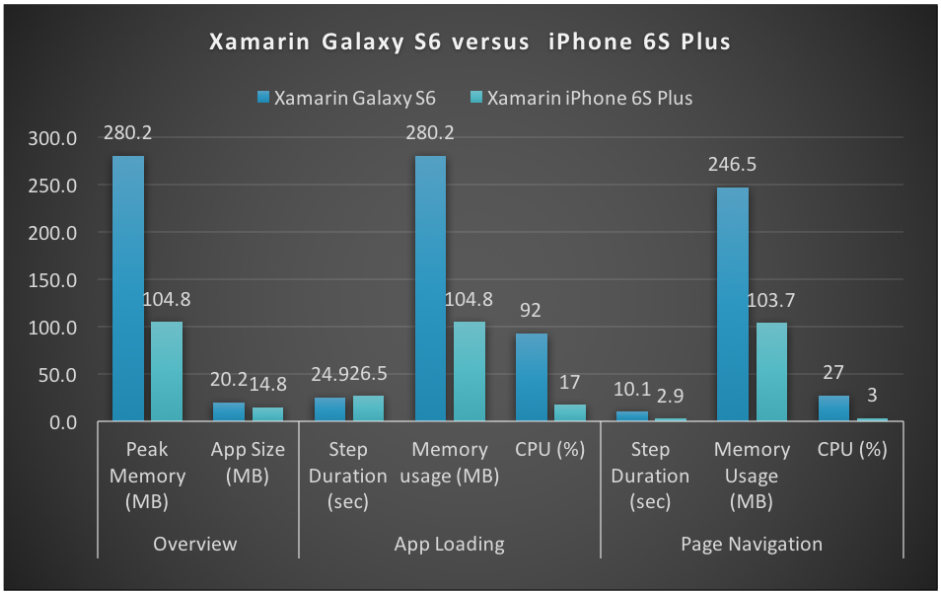


Figure 7.8: Samsung Galaxy S6 versus iPhone 6S Plus

7.4.2 Cordova Android versus Cordova IOS

Figure 7.9 and Figure 7.10 shows that Android applications built with Apace Cordova have approximately 258% more memory consumption during the app launch and 247% during the page navigation compare to the Cordova IOS applications. Others performance properties have a much lower difference recorded.

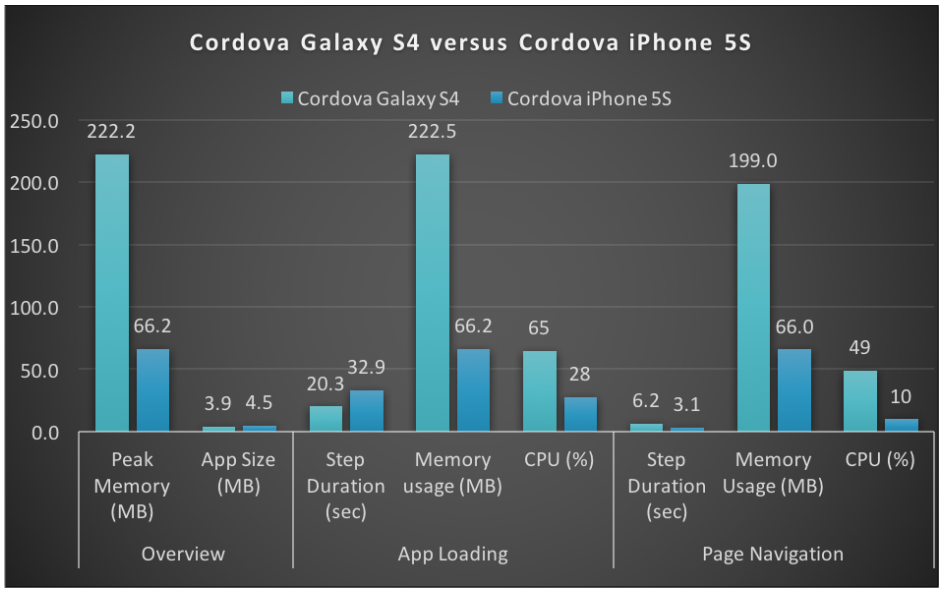


Figure 7.9: Cordova Galaxy S4 versus Cordova iPhone 5S

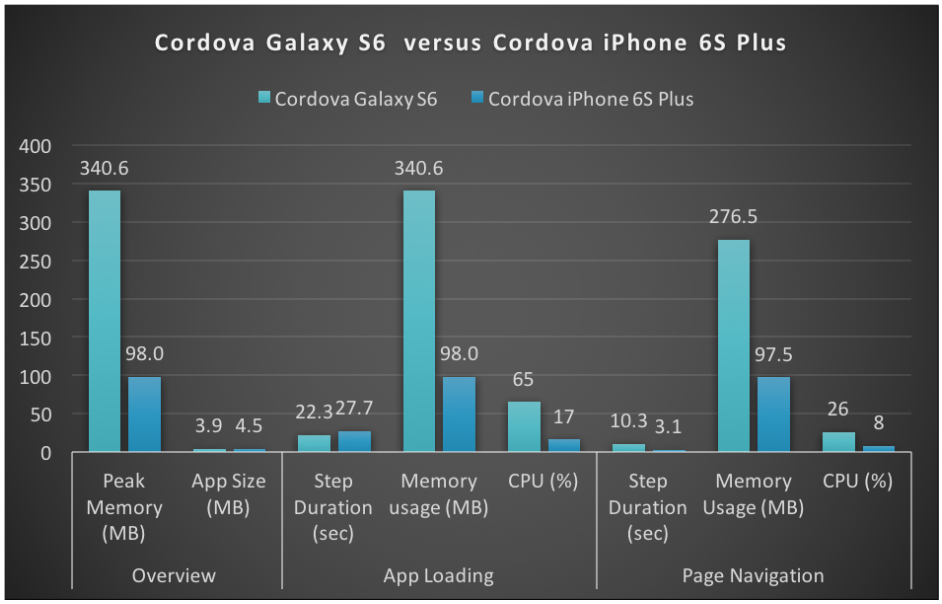


Figure 7.10: Cordova Galaxy S6 versus Cordova iPhone 6S Plus

7.5 Analysis of Results

From the test records analyzed in section 7.4 resulted that application implemented with Xamarin have a much larger application size compare to the same application implemented with Apache Cordova. However, the app size limit accepted by the Google Play is 100MB and 60MB for the App Store. Smart watches have a limit of 50MB.

The results show that Xamarin Android allocates a very large amount of memory for app launch and page navigation, with similar large amount of CPU. This is due to the *Mono Runtime* which runs on top of Xamarin framework and translates the C# into native code and due to the Android OS architecture. However, Xamarin IOS applications, which similarly have the *Mono Runtime*, have less memory allocated for app launch and page navigation. Therefore, the large amount of memory allocated to the processes are related to the Android OS architecture and not specific to the Xamarin framework.

Overall, the results show Apace Cordova recorded a higher memory allocation for low-end Android devices and Xamarin has a higher CPU usage for high-end Android. In contrast to the Android implementations, there is no significant performance differences between IOS applications build with Xamarin and Apace Cordova. However, Apace Cordova comes with a huge penalty regarding user interface (UI), where Xamarin framework implements native look and feel.

Chapter 8

Conclusions

The main objective of this research paper was to compare Xamarin and Apace Cordova mobile cross-platform frameworks based on quantitative analysis of performance properties. An enterprise identical prototype app has been produced with both frameworks and tested with Xamarin Test Cloud.

The performance results are compared with the same app on Android and IOS. The assessment shows firstly that Android application build with Apache Cordova framework have a lower performance on low-end Android devices compared with similar application built with Xamarin, and secondly on the high-end Android Xamarin records higher CPU usage. Furthermore, the results show that there is no significant difference for the IOS implementations.

One important aspect is that Xamarin implements native user experience in all platforms in contrast to web user experience on Apace Cordova implementations.

For the development process, the finding shows that Xamarin has a more enterprise orientation development approach, with many available resources for implementations and testing the applications. However, Apace Cordova is a free tool and it is a good resource to start for small companies or single developer.

This concludes that the first hypothesis question is true, the most popular enterprise mobile frameworks helps the enterprise to develop mobile applications in a relative short time, where one base development code can generate a mobile application for Android, IOS and Windows Phone.

Furthermore, our findings shows that the second hypothesis question is false, as there is a significant difference regarding performance between Apace Cordova and Xamarin frameworks, such as higher memory allocation recorded for low-end Android devices

built with Apace Cordova framework and a higher CPU usage for the Xamarin Android high-end.

Following to the results shown above, the recommendation to the enterprise is that the most popular mobile cross-platforms frameworks, e.g., Apace Cordova and Xamarin, have the capabilities to deliver rapid mobile applications into the market. Likewise, if enterprise choose to support the low Android devices, it is recommended to use Xamarin, otherwise either framework has similar results regarding performance parameters, therefore it is advisable to choose the framework according to the in-house development preferences.

The results are preliminary, but we believe they provide interesting insights to the enterprise mobile software development.

8.1 Further research

Further research work may include Windows Phone and test the applications locally.

Bibliography

Android API Guides. Supporting multiple screens — android developers. https://developer.android.com/guide/practices/screens_support.html, 2016. [Online; accessed 28-July-2016].

E. Angulo and X. Ferre. A case study on cross-platform development frameworks for mobile applications and ux. In *Proceedings of the XV International Conference on Human Computer Interaction*, Interacción '14, pages 27:1–27:8, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2880-7. doi: 10.1145/2662253.2662280. URL http://ezproxy.ncirl.ie:3776/10.1145/2662253.2662280.

Apple. Choosing a membership. https://developer.apple.com/support/compare-memberships/, 2016. [Online; accessed 20-May-2016].

Apple Developer. ios human interface guidelines, 2016. URL https://developer.apple.com/ios/human-interface-guidelines/graphics/app-icon/. [Online; accessed 20-July-2016].

N. Boushehrinejadmoradi, V. Ganapathy, S. Nagarakatte, and L. Iftode. Testing Cross-Platform Mobile App Development Frameworks (T). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 441–451, nov 2015. doi: 10.1109/ASE.2015.21.

D. Chaffey. Mobile Marketing Statistics compilation. http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics, 2016. [Online; accessed 5-July-2016].

J. Cordeiro and K.-H. Krempels, editors. *Web Information Systems and Technologies: 8th International Conference, WEBIST 2012, Porto, Portugal, April 18-21, 2012, Revised Selected Papers*, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-36608-6. doi: 10.1007/978-3-642-36608-6_8. URL http://dx.doi.org/10.1007/978-3-642-36608-6_8.

A. Cordova. Apache Cordova application architecture. https://cordova.apache.org/static/img/guide/cordovaapparchitecture.png, 2016. [Online; accessed 04-June-2016].

I. Dalmasso, S. K. Datta, C. Bonnet, and N. Nikaein. Survey, comparison and evaluation of cross platform mobile application development tools. *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 323–328, jul 2013. doi: 10.1109/IWCMC.2013.6583580. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6583580http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=6583580$\delimiter"026E30F$nhttp://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6583580.

W. S. El-Kassas, B. A. Abdullah, A. H. Yousef, and A. M. Wahba. Enhanced code conversion approach

for the integrated cross-platform mobile development (icpmd). *IEEE Transactions on Software Engineering*, PP(99):1–1, 2016. ISSN 0098-5589. doi: 10.1109/TSE.2016.2543223.

Google Developers. Google identity platfom. https://developers.google.com/identity/protocols/OAuth2, 2016. [Online; accessed 06-July-2016].

S. Guthrie. The Official Microsoft Blog. https://blogs.microsoft.com/blog/2016/02/24/microsoft-to-acquire-xamarin-and-empower-more-developers-to-build-apps-on-any-device/#sm.00001ljhix3rfbcvvztl8qc0zx8ag, 2016. [Online; accessed 06-June-2016].

A. Hall. What are Mobile Apps? https://acom.azurecomcdn.net/80C57D/cdn/mediahandler/docarticles/dpsmedia-prod/azure.microsoft.com/en-us/documentation/articles/app-service-mobile-value-prop/20160506100124/overview.png, 2016. [Online; accessed 13-June-2016].

IDC. Smartphone OS Market Share. http://www.idc.com/prodserv/smartphone-os-market-share.jsp, 2016. [Online; accessed 5-July-2016].

Lee Crossley. Cordova / phonegap map directions plugin for apache cordova 3.0.0. https://github.com/leecrossley/cordova-plugin-directions, 2016. [Online; accessed 28-July-2016].

I. Malavolta, S. Ruberto, T. Soru, and V. Terragni. Hybrid mobile apps in the google play store: An exploratory investigation. In *Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems*, MOBILESoft '15, pages 56–59, Piscataway, NJ, USA, 2015. IEEE Press. ISBN 978-1-4799-1934-5. URL http://ezproxy.ncirl.ie:3168/citation.cfm?id=2825041.2825051.

M. D. Maree, I. Strydom, and M. Matthee. Towards a Framework for the Achievement of Mobile Transformation in Enterprises. In *Proceedings of the Southern African Institute for Computer Scientist and Information Technologists Annual Conference 2014 on SAICSIT 2014 Empowered by Technology*, SAICSIT '14, pages 343:343—-343:351, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3246-0. doi: 10.1145/2664591.2664613. URL http://doi.acm.org/10.1145/2664591.2664613.

Mono. The mono runtime. http://www.mono-project.com/docs/advanced/runtime/, 2016. [Online; accessed 28-June-2016].

D. Olanoff. Mark Zuckerberg: Our Biggest Mistake Was Betting Too Much On HTML5. http://techcrunch.com/2012/09/11/mark-zuckerberg-our-biggest-mistake-with-mobile-was-betting-too-much-on-html5/, 2012. [Online; accessed 04-June-2016].

Paul P. Cordova-calabash-ios-plugin. https://github.com/paulpatarinski/Cordova-Calabash-iOS-Plugin, 2016. [Online; accessed 28-July-2016].

Statista. Mobile Marketing Statistics compilation. http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/, 2016. [Online; accessed 5-July-2016].

Taco. Getting started with taco cli. http://taco.tools/docs/remote-build.html, 2016. [Online; accessed 20-July-2016].

W3C. Device and APIs Working Group. https://www.w3.org/TR/dap-api-reqs/, 2016. [Online; accessed 07-June-2016].

Wikipedia. Android (operating system). https://en.wikipedia.org/wiki/Android_(operating_system), 2016a. [Online; accessed 30-June-2016].

Wikipedia. Apache cordova. https://en.wikipedia.org/wiki/Apache_Cordova, 2016b. [Online; accessed 02-July-2016].

M. Willocx, J. Vossaert, and V. Naessens. A quantitative assessment of performance in mobile app development tools. In *2015 IEEE International Conference on Mobile Services*, pages 454–461, June 2015. doi: 10.1109/MobServ.2015.68.

M. Willocx, J. Vossaert, and V. Naessens. Comparing Performance Parameters of Mobile App Development Strategies. In *Proceedings of the International Workshop on Mobile Software Engineering and Systems*, MOBILESoft '16, pages 38–47, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4178-3. doi: 10.1145/2897073.2897092. URL http://doi.acm.org/10.1145/2897073.2897092.

Xamarin. Android architecture. https://developer.xamarin.com/guides/android/under_the_hood/architecture/Images/architecture1.png, 2016a. [Online; accessed 03-July-2016].

Xamarin. Building cross platform applications overview. https://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/part_0_-_overview/Images/Layers2.png, 2016b. [Online; accessed 03-July-2016].

Xamarin. Ios architecture. https://developer.xamarin.com/guides/ios/under_the_hood/architecture/Images/iOS-arch-small.png, 2016c. [Online; accessed 03-July-2016].

Xamarin. Portable class library architecture. https://developer.xamarin.com/guides/cross-platform/application_fundamentals/pcl/introduction_to_portable_class_libraries/Images/image1.png, 2016d. [Online; accessed 04-July-2016].

Xamarin. Shared asset projects) architecture. https://developer.xamarin.com/guides/cross-platform/application_fundamentals/shared_projects/Images/SharedAssetProject.png, 2016e. [Online; accessed 04-July-2016].

Xamarin Test Cloud. Introduction to xamarin test cloud. https://developer.xamarin.com/guides/testcloud/introduction-to-test-cloud/, 2016a. [Online; accessed 04-July-2016].

Xamarin Test Cloud. Introduction to xamarin test cloud. https://developer.xamarin.com/guides/testcloud/introduction-to-test-cloud/Images/image02.png, 2016b. [Online; accessed 05-August-2016].

S. Xanthopoulos and S. Xinogalos. A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications. In *Proceedings of the 6th Balkan Conference in Informatics*, BCI '13, pages 213–220, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1851-8. doi: 10.1145/2490257.2490292. URL http://doi.acm.org/10.1145/2490257.2490292.

D. Yunge, P. Kindt, M. Balszun, and S. Chakraborty. Hybrid apps: Apps for the internet of things. In *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on*, pages 1175–1180, Aug 2015. doi: 10.1109/HPCC-CSS-ICESS.2015.292.

A. L. Zain. The UK Goes Mobile. http://www.comscore.com/Insights/Data-Mine/The-UK-Goes-Mobile, 2015. [Online; accessed 31-May-2016].

G. Zodik. Cognitive and contextual enterprise mobile computing: Invited keynote talk. In *Proceedings of the 9th India Software Engineering Conference*, ISEC '16, pages 11–12, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4018-2. doi: 10.1145/2856636.2876471. URL http://ezproxy.ncirl.ie:3776/10.1145/2856636.2876471.

Appendix A

Apace Cordova index.html

```
1
2  <--Login page--->
3  <div data-role="page" id="loginpage" class="ui-page">
4     <div data-role="header">
5        <h3>nCare</h3>
6     </div>
7  </div>
8  <---Message page --->
9  <div data-role="page" id="messagespage" class="ui-page">
10    <div data-role="header">
11       <h3>nCare</h3>
12       <div data-role="navbar">
13          <ul>
14             <li><a href="#messagespage" class="ui-btn-active ui-state-persist"> ↩
                   Messages</a></li>
15             <li><a href="#eventspage">Events</a></li>
16             <li><a href="#directionspage">Directions</a></li>
17             <li><a href="#logoutpage">Logout</a></li>
18          </ul>
19       </div>
20    </div>
21 </div>
22 <-- Message Detail page--->
23 <div data-role="page" id="mesgdetails" class="ui-page">
24    <div data-theme="a" data-role="header">
25       <h3>Details</h3>
26    </div>
27 </div>
28 <--Event page-->
29 <div data-role="page" id="eventspage" class="ui-page">
30 <div data-role="navbar">
31    <ul>
```

```
32      <li><a href="#messagespage">Messages</a></li>
33      <li><a href="#eventspage" class="ui-btn-active ui-state-persist">Events</a></ ↩
            li>
34      <li><a href="#directionspage">Directions</a></li>
35      <li><a href="#logoutpage">Logout</a></li>
36    </ul>
37 </div>
38 <div>
39 <---Event page-->
40 <div data-role="page" id="eventspage" class="ui-page">
41 <div data-role="header">
42    <h3>nCare</h3>
43    <div data-role="navbar">
44       <ul>
45          <li><a href="#messagespage">Messages</a></li>
46          <li><a href="#eventspage" class="ui-btn-active ui-state-persist">Events</a ↩
               ></li>
47          <li><a href="#directionspage">Directions</a></li>
48          <li><a href="#logoutpage">Logout</a></li>
49       </ul>
50    </div>
51 </div>
52 <div></div>
53 <---Event Detail page-->
54 <div data-role="page" id="eventdetails" class="ui-page">
55    <div data-theme="a" data-role="header">
56       <h3>Details</h3>
57    </div>
58 </div>
59 <--Direction page --->
60 <div data-role="page" id="directionspage" class="ui-page">
61    <div data-role="navbar">
62       <ul>
63          <li><a href="#messagespage">Messages</a></li>
64          <li><a href="#eventspage">Events</a></li>
65          <li><a href="#directionspage" class="ui-btn-active ui-state-persist"> ↩
               Directions</a></li>
66          <li><a href="#logoutpage">Logout</a></li>
67       </ul>
68    </div>
69 </div>
70 <---Logout page
71 <div data-role="page" id="logoutpage">
72    <div data-role="navbar">
73       <ul>
74          <li><a href="#messagespage">Messages</a></li>
75          <li><a href="#eventspage">Events</a></li>
```

```
76          <li><a href="#directionspage">Directions</a></li>
77          <li><a href="#logoutpage" class="ui-btn-active ui-state-persist">Logout</a ↩
                ></li>
78      </ul>
79    </div>
80  </div>
```

Appendix B

Applications Test implementation

B.1 nCareXamarin Android and IOS Automated Test

The test project for the *nCareXamarin* was created by adding a Xamarin.UITest project into the existing solution with the Xamarin Studio as is shown in Figure B.1.

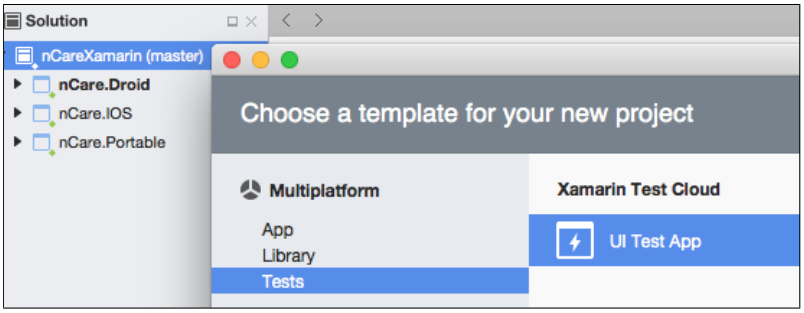


Figure B.1: Create a Xamarin UITest project

The project contains two files, one for witting the NUnit tests called *Test.cs* and *AppInitializer.cs* which includes a method that takes in the path location for the testing files if the testing application are not in the same solution. The NUnit test is written in the Test.cs file and covers all steps from Table 7.1. Full details of the test are available below in Listing B.1.

Xamarin UITest framework provides provide a great tool called *"Repl"* which is very useful to debug the test locally and complete the test steps. The Xamarin Test Cloud for IOS or Android project can be run directly from Xamarin Studio *Unit Tests window* as shown in Figure B.2.

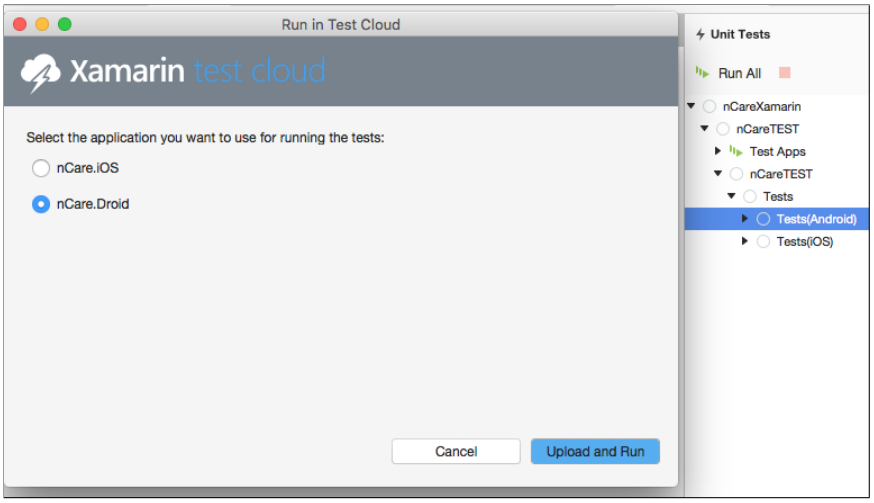


Figure B.2: Deploy a Xamarin app to Xamarin Test Cloud

The test project shown in Figure 7.2 will deployed each application (Android, IOS) seven times to the Xamarin Test Cloud platform for each mobile device from Table 7.2, counting a total of fourteen tests for IOS and fourteen tests for Android. Each time a device from Table 7.3 is selected in the cloud as showing in Figure B.3.

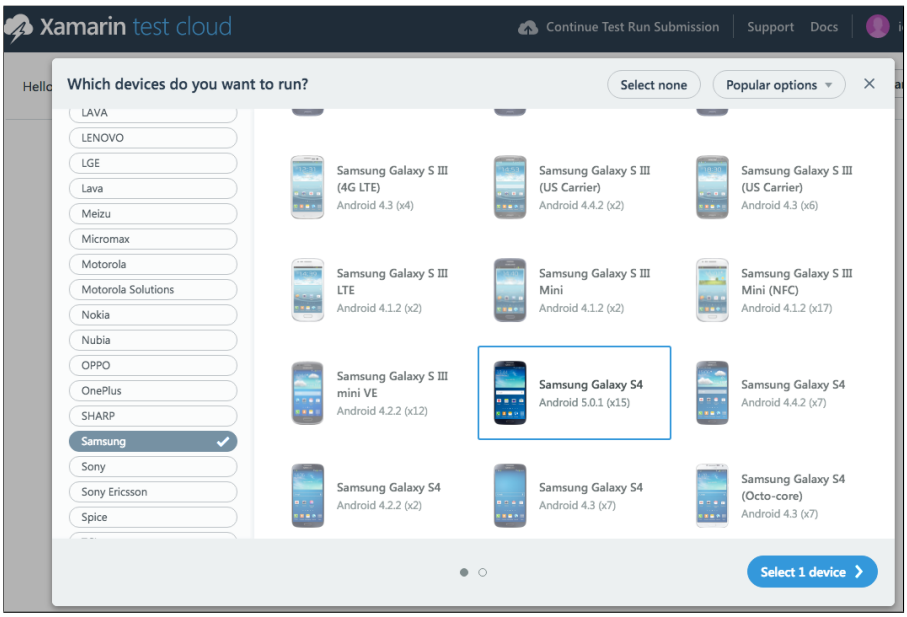


Figure B.3: Select a mobile device on Xamarin Test Cloud

For each test completed performance metrics are available in the cloud portal as showing in Figure B.4. Figure B.4 shows an example of the summary test result of an Android test. The cloud interface provides many others features, e.g., test log file, device log file which can be very useful for debugging.

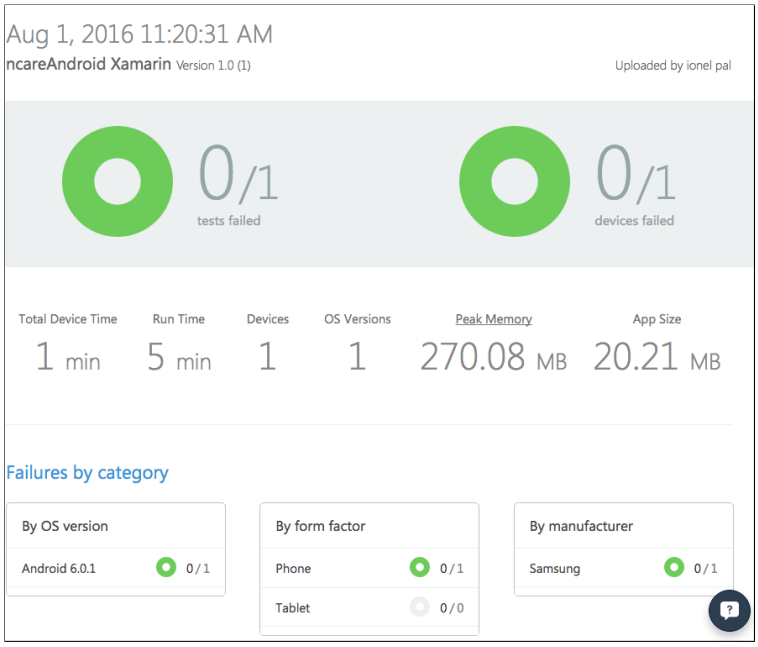


Figure B.4: Xamarin Test Cloud test result summary

```
1  using System;
2  using System.IO;
3  using System.Linq;
4  using NUnit.Framework;
5  using Xamarin.UITest;
6  using Xamarin.UITest.Queries;
7
8  namespace nCareTEST
9  {
10   [TestFixture(Platform.Android)]
11   [TestFixture(Platform.iOS)]
12   public class Tests
13   {
14     IApp app;
15     Platform platform;
16
17     public Tests(Platform platform)
18     {
19       this.platform = platform;
20
21     }
22
23     [SetUp]
24     public void BeforeEachTest()
25     {
26       app = AppInitializer.StartApp(platform);
27     }
```

```
28
29      //[Test]
30      //public void AppLunchXamarinForms()
31      //{
32      //  app.WaitForElement(c => c.Marked("Login"));
33      //}
34
35      [Test]
36      public void AppTestXamarinForms()
37      {
38        //app.Repl();
39        app.Screenshot("Login");
40        app.Tap(c => c.Marked("Login"));
41        app.Tap(c => c.WebView().Css("#Email"));
42        app.EnterText("xxxirex@gmail.com");
43        app.PressEnter();
44        app.Tap(c => c.WebView().Css("#password-shown"));
45        app.EnterText("Pioneer0123");
46        app.PressEnter();
47        app.WaitForElement(e => e.Marked("Refresh"));
48
49        if (platform == Platform.iOS)
50        {
51          app.WaitForElement(c => c.Class("UITableViewLabel").Index(0));
52          app.Screenshot("Message page");
53          app.Tap(c => c.Class("UITableViewLabel").Index(0));
54
55          app.Screenshot("Message Detail page");
56          app.Tap(c => c.Class("UIButtonLabel").Text("Return"));
57
58          app.Tap(c => c.Class("UITabBarButtonLabel").Text("Events"));
59          app.Screenshot("Event page");
60
61          //app.Tap(c => c.Class("UIButtonLabel").Text("Refresh"));
62
63          app.Tap(c => c.Class("UITableViewLabel").Index(0));
64
65          app.Screenshot("Event Detail page");
66          app.Tap(c => c.Class("UIButtonLabel").Text("Return"));
67          app.Tap(c => c.Class("UITabBarButtonLabel").Text("Directions"));
68
69          app.Screenshot("Direction page");
70          app.Tap(c => c.Class("UITabBarButtonLabel").Text("LogOut"));
71
72          app.Screenshot("Logout page");
73          app.Tap(c => c.Class("UIButtonLabel").Text("Logout"));
74        }
```

```
75      else {
76        app.WaitForElement(c => c.Class("TextView").Index(0));
77        app.Screenshot("Message page");
78        app.Tap(c => c.Class("TextView"));
79
80        app.Screenshot("Message Detail page");
81        app.Tap(c => c.Marked("Return"));
82        app.Tap(c => c.Marked("Events"));
83
84        app.Screenshot("Event page");
85        app.WaitForElement(c => c.Class("TextView"));
86        //app.Tap(c => c.Marked("Refresh"));
87        app.Tap(c => c.Class("TextView"));
88
89        app.Screenshot("Event Detail page");
90        app.Tap(c => c.Marked("Return"));
91        app.Tap(c => c.Marked("Directions"));
92
93        app.Screenshot("Direction page");
94        app.Tap(c => c.Marked("LogOut"));
95
96        app.Screenshot("Logout page");
97        app.Tap(c => c.Marked("Logout"));
98      }
99    }
100
101  }
102 }
```

Listing B.1: Xamarin Forms *nCareXamarin* solution Unit Test Implementation

B.2 nCareCordova Android and IOS Automated Test

Testing the *nCareCordova* applications with Xamarin Test Cloud has a quite similar implementation as Xamarin Forms discussed above in Appendix B1. Apace Cordova projects are not supported by Xamarin Studio IDE, however it is possible to create a *Xamarin.UITest* project with Xamarin Studio as shown in Figure B.5, and add the application release files created in section 6.5.3 and section 6.54 into the solution folder and add the path of the applications in the *AppInitializer* file as shown in Listing B.2.

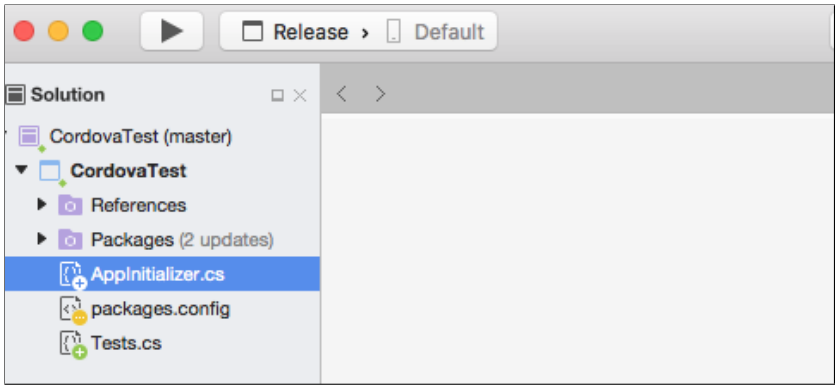


Figure B.5: Apache Cordova Xamarin Test project

Before copy the Apache Cordova IOS application into the test project, Xamarin Test Cloud requires that the application has installed the Calabash framework. Paul P (2016) has created a plugin to solve successfully this matter.

```
1  public class AppInitializer{
2    public static IApp StartApp(Platform platform)
3    { if (platform == Platform.Android)
4      { return ConfigureApp
5          .Android
6          //path to the Android app
7          .ApkFile("/Users/ionelpal/Desktop/CORDOVA/com.ncare.andrcordova.apk")
8          .StartApp();
9      }
10     return ConfigureApp
11       .iOS
12       //path to the IOS app
13       .InstalledApp ("/Users/ionelpal/Desktop/CORDOVA/com.ncare.ire.ipa")
14       .StartApp(
15       }
16   }
```

Listing B.2: Apache Cordova Test AppInitializer class

Full details of the *Test.cs* file from the *Xamarin.UITest* Cordova project are available below in Listing B.3.

The *CordovaTest* test project showing in Figure B.6 will deploy each application file (Android, IOS) seven times to the Xamarin Testing Cloud platform for testing each device showing in Table 7.3. Each time on the cloud interface a mobile device from Table 7.2 is selected as is showing in Figure B.3. The test results are available after each test completion. Figure B.4 is showing a summary of a test results.

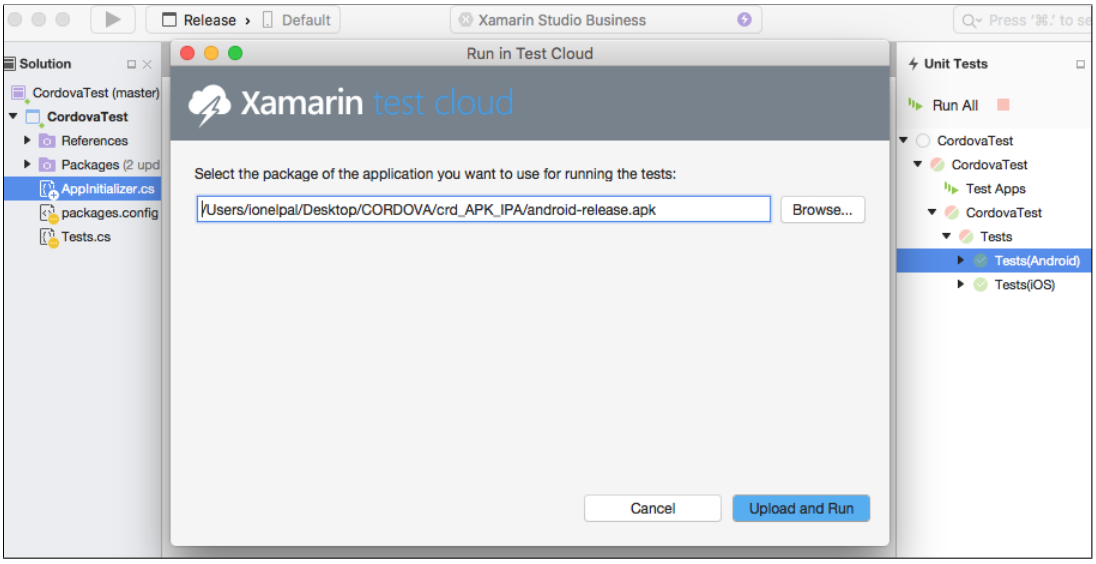


Figure B.6: Deploy a Cordova app to Xamarin Test Cloud

```
1  using System;
2  using System.IO;
3  using System.Linq;
4  using NUnit.Framework;
5  using Xamarin.UITest;
6  using Xamarin.UITest.Queries;
7
8  namespace CordovaTest
9  {
10   [TestFixture(Platform.Android)]
11   [TestFixture(Platform.iOS)]
12   public class Tests
13   {
14     IApp app;
15     Platform platform;
16
17     public Tests(Platform platform)
18     {
19       this.platform = platform;
20     }
21
22     [SetUp]
23     public void BeforeEachTest()
24     {
25       app = AppInitializer.StartApp(platform);
26     }
27
28     //[Test]
29     //public void AppLunchCordova()
```

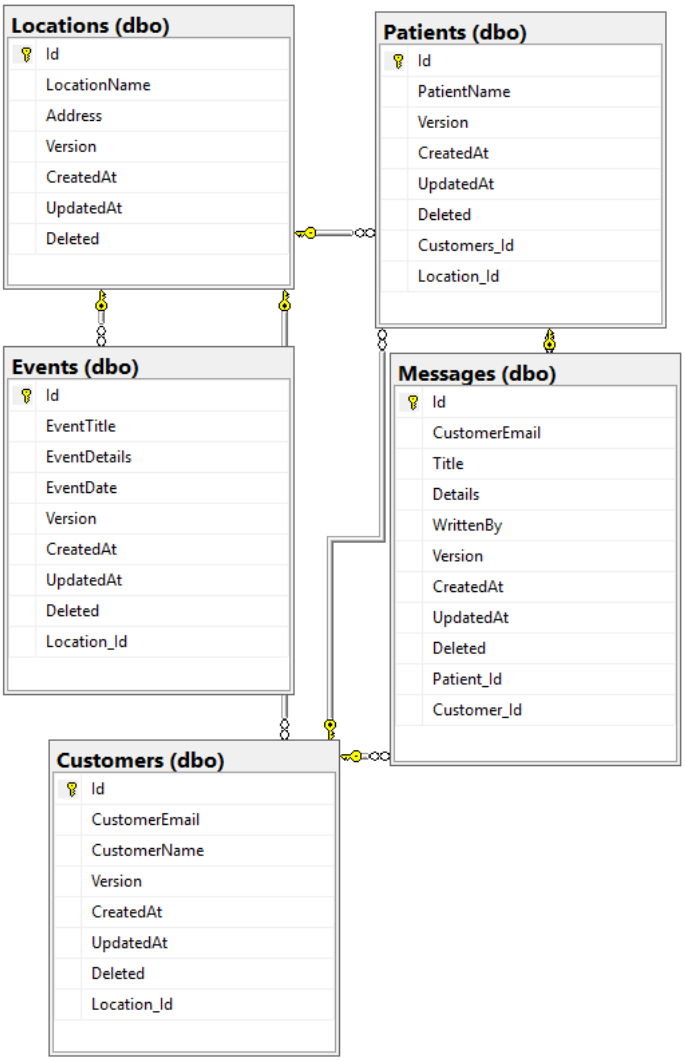
```
30      //{  //app.Repl();
31      //  app.WaitForElement(c => c.WebView().Css("#loginbtn"));
32      //}
33
34      [Test]
35      public void AppTestCordova()
36      {
37
38        //app.Repl();
39
40        if (platform == Platform.iOS)
41        { app.WaitForElement(c => c.WebView().Css("#loginbtn"));
42          app.Screenshot("Login page");
43          app.Tap(c => c.WebView().Css("#loginbtn"));
44          app.Tap(c => c.WebView().Css("#Email"));
45          app.EnterText("xxxirex@gmail.com");
46          app.PressEnter();
47          app.WaitForElement(c => c.WebView().Css("#Passwd"));
48          app.Tap(c => c.WebView().Css("#Passwd"));
49          app.EnterText("Pioneer0123");
50          app.PressEnter();
51
52          app.WaitForElement(c => c.WebView().Css("#refresh"));
53          app.WaitForElement(c => c.WebView().Css("a").Index(4));
54
55          app.Screenshot("Message page");
56          app.Tap(c => c.Css("a").Index(5));
57
58          app.Screenshot("Message Detail page");
59          app.Tap(c => c.Css("a").Index(0));
60          app.Tap(c => c.Css("a").Index(1));
61
62          app.Screenshot("Event page");
63          app.WaitForElement(c => c.WebView().Css("a").Index(4));
64          //refresh event page
65          //app.Tap(c => c.Css("a").Index(4));
66          app.Tap(c => c.Css("a").Index(5));
67
68          app.Screenshot("Event Detail page");
69          app.Tap(c => c.Css("a").Index(0));
70          app.Tap(c => c.Css("a").Index(2));
71
72          app.Screenshot("Direction page");
73          app.Tap(c => c.Css("a").Index(3));
74
75          app.Screenshot("Logout page");
76          app.Tap(c => c.Css("a").Index(4));
```

```
77      }
78      else {
79        app.WaitForElement(c => c.WebView().Css("#loginbtn"));
80
81        app.Screenshot("Login page");
82        app.Tap(c => c.WebView().Css("#loginbtn"));
83        app.Tap(c => c.WebView().Css("#Email"));
84        app.EnterText("xxxirex@gmail.com");
85        app.PressEnter();
86        app.WaitForElement(c => c.WebView().Css("#Passwd"));
87        app.EnterText("Pioneer0123");
88        app.PressEnter();
89        app.WaitForElement(c => c.WebView().Css("#refresh"));
90        app.WaitForElement(c => c.WebView().Css("a").Index(4));
91
92
93        app.Screenshot("Message page");
94        app.Tap(c => c.Css("a").Index(5));
95
96        app.Screenshot("Message Detail page");
97        app.Tap(c => c.Css("a").Index(0));
98        app.Tap(c => c.Css("a").Index(1));
99
100       app.Screenshot("Event page");
101       app.WaitForElement(c => c.WebView().Css("a").Index(4));
102       //refresh event page
103       //app.Tap(c => c.Css("a").Index(4));
104
105       app.Tap(c => c.Css("a").Index(5));
106
107       app.Screenshot("Event Detail page");
108       app.Tap(c => c.Css("a").Index(0));
109       app.Tap(c => c.Css("a").Index(2));
110
111       app.Screenshot("Direction page");
112       app.Tap(c => c.Css("a").Index(3));
113
114       app.Screenshot("Logout page");
115       app.Tap(c => c.Css("a").Index(4));
116
117     }
118   }
119 }
120 }
```

Listing B.3: Apache Cordova *nCareCordova* solution Unit Test Implementation

Appendix C

UML Diagram



Appendix D

nCareBackend Controllers implementation

D.1 nCare Backend Initial solution

Requires the following steps:

- Create a blank solution in Visual Studio called *nCareSolution*.
- In the solution, add a Cloud Azure Mobile App project called *nCareBackend*, in the project wizard, select hosted on Azure and connect to the newly created Mobile App service *nCare* and *nCaredb* database. Figure 6.2 shows *nCareSolution* initial solution.

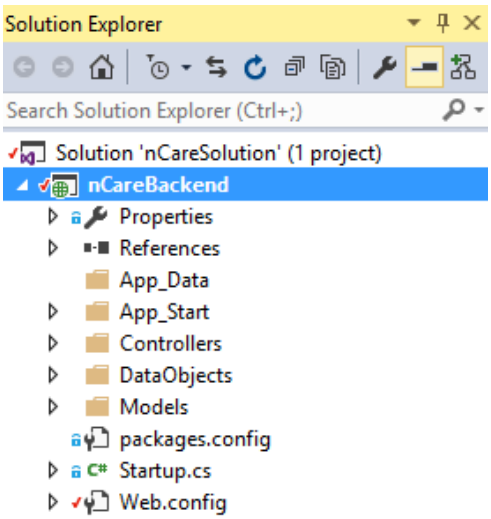


Figure D.1: nCareBackend initial solution

D.2 Domain Model Setup

The entities are a total of five, and their relationships are represented in the following listings:

1. Location

```
public class Location:EntityData
{
public string LocationName{get;set;}
public string Address{get;set;}

public virtual ICollection<Customer>Custommers{get;set;}
public virtual ICollection<Patient>Patients{get;set;}
public virtual ICollection<Event>Events{get;set;}
}
```

Listing D.1: nCareBackend Location Model

2. Patient

```
public class Patient:EntityData
{
public string PatientName{get;set;}
public virtual Location Location{get;set;}
public virtual ICollection<Message>Messages{get;set;}
public virtual Customer Customers{get;set;}
}
```

Listing D.2: nCareBackend Patient Model

3. Customer

```
public class Customer:EntityData{
public string CustomerEmail{get;set;}
public string CustomerName{get;set;}
public virtual Location Location{get;set;}
public virtual ICollection<Patient>Patients{get;set;}
public virtual ICollection<Message>Mesages{get;set;}
}
```

Listing D.3: nCareBackend Customer Model

4. Message

```
public class Message:EntityData
{
public string CustomerEmail{get;set;}
public string Title{get;set;}
public string Details{get;set;}
public string WrittenBy{get;set;}
public virtual Patient Patient{get;set;}
```

```
8 }
```

Listing D.4: nCareBackend Message Model

5. Event

```
1 public class Event:EntityData
2 {
3 public string EventTitle{get;set;}
4 public string EventDetails{get;set;}
5 public DateTime EventDate{get;set;}
6 public Location Location{get;set;}
7 }
```

Listing D.5: nCareBackend Event Model

D.3 Data access representation

Data access layer is represented by the *nccloudContext* class which extends the *DbContext* class. This class is an important part of the Entity Framework, it is a bridge between entities classes and the database. It represents a combination of the Unit of Work and Repository pattern. DbContext class is responsible with the followings:

- Mapping the entities to the database table.
- Convert LINQ queries to SQL
- Tracking all the changes between the entities and database
- In charge with data persistence and manage the relationships between the models.

Listing D.6 shows the *DbContex* Class representation.

```
1 public class nccloudContext : DbContext
2 {
3 private const string connectionStringName = "Name=MS_TableConnectionString";
4     public nccloudContext() : base(connectionStringName) { }
5   public DbSet<Customer> Customers { get; set; }
6   public DbSet<Event> Events { get; set; }
7   public DbSet<Location> Locations { get; set; }
8   public DbSet<Message> Messages { get; set; }
9   public DbSet<Patient> Patients { get; set; } }
```

Listing D.6: nCareBackend DbContext class

D.4 nCare backend solution Controller implementation

The controllers are the following:

- Message Controller is the first controller implemented and is designed to retrieve the messages from database based on user's email address only as shown in Listing D.7.

```
1  using System.Linq;
2  using System.Threading.Tasks;
3  using System.Web.Http;
4  using System.Web.Http.Controllers;
5  using System.Web.Http.OData;
6  using Microsoft.Azure.Mobile.Server;
7  using nccloudService.DataObjects;
8  using nccloudService.Models;
9  using System.Security.Principal;
10 using Microsoft.Azure.Mobile.Server.Authentication;
11 using System.Diagnostics;
12 using System.Net;
13 using System.Security.Claims;
14
15 namespace nccloudService.Controllers
16 {
17     [Authorize]
18     public class MessageController : TableController<Message>
19     {
20         protected override void Initialize(HttpControllerContext ↩
               controllerContext)
21         {
22             base.Initialize(controllerContext);
23             nccloudContext context = new nccloudContext();
24             DomainManager = new EntityDomainManager<Message>(context, Request);
25         }
26         //get user email address
27         private string GoogleSID()
28         {
29             var principal = this.User as ClaimsPrincipal;
30             var sid = principal.FindFirst(ClaimTypes.NameIdentifier).Value;
31             return sid;
32         }
33         private async Task<string> GetEmailAddress()
34         {
35             var credentials = await User.GetAppServiceIdentityAsync< ↩
                   GoogleCredentials>(Request);
36             return credentials.UserClaims
37                 .Where(claim => claim.Type.EndsWith("/emailaddress"))
38                 .First<Claim>()
39                 .Value;
```

```
40              // Debug.WriteLine(credentials.UserClaims);
41          }
42          // GET tables/Message
43          public async Task<IQueryable<Message>> GetAllMessage()
44          {
45            //  Debug.WriteLine("GET tables/TodoItem");
46              var emailAddr = await GetEmailAddress();
47              return Query().Where(item => item.CustomerEmail == emailAddr);
48          }
49
50          // GET tables/Message/48D68C86-6EA6-4C25-AA33-223FC9A27959
51          public async Task<SingleResult<Message>> GetMessage(string id)
52          {
53             // Debug.WriteLine($"GET tables/Message/{id}");
54              var emailAddr = await GetEmailAddress();
55              var result = Lookup(id).Queryable.Where(item => item.CustomerEmail  ↩
                    == emailAddr);
56              return new SingleResult<Message>(result);
57          }
58
59          // PATCH tables/Message/48D68C86-6EA6-4C25-AA33-223FC9A27959
60          public async Task<Message> PatchMessage(string id, Delta<Message> patch ↩
                )
61          {
62             // Debug.WriteLine($"PATCH tables/TodoItem/{id}");
63              var item = Lookup(id).Queryable.FirstOrDefault<Message>();
64              if (item == null)
65              {
66                  throw new HttpResponseException(HttpStatusCode.NotFound);
67              }
68              var emailAddr = await GetEmailAddress();
69              if (item.CustomerEmail != emailAddr)
70              {
71                  throw new HttpResponseException(HttpStatusCode.Forbidden);
72              }
73              return await UpdateAsync(id, patch);
74          }
75
76          // POST tables/Message
77          public async Task<IHttpActionResult> PostMessage(Message item)
78          {
79            //  Debug.WriteLine($"POST tables/TodoItem");
80              var emailAddr = await GetEmailAddress();
81              item.CustomerEmail = emailAddr;
82              Message current = await InsertAsync(item);
83              return CreatedAtRoute("Tables", new { id = current.Id }, current);
84          }
```

```
85
86          // DELETE tables/Message/48D68C86-6EA6-4C25-AA33-223FC9A27959
87          public async Task DeleteMessage(string id)
88          {
89             // Debug.WriteLine($"DELETE tables/TodoItem/{id}");
90              var item = Lookup(id).Queryable.FirstOrDefault<Message>();
91              if (item == null)
92              {
93                  throw new HttpResponseException(HttpStatusCode.NotFound);
94              }
95              var emailAddr = await GetEmailAddress();
96              if (item.CustomerEmail != emailAddr)
97              {
98                  throw new HttpResponseException(HttpStatusCode.Forbidden);
99              }
100             await DeleteAsync(id);
101             // return DeleteAsync(id);  }
102     }
103 }
```

Listing D.7: Message Controller

- The second controller, is the Events Controller. This controller retrieves all the events specific to a location, where the user belongs. Full implementation is shown in Listing D.8.

```
1  using System.Linq;
2  using System.Threading.Tasks;
3  using System.Web.Http;
4  using System.Web.Http.Controllers;
5  using System.Web.Http.OData;
6  using Microsoft.Azure.Mobile.Server;
7  using nccloudService.DataObjects;
8  using nccloudService.Models;
9  using System.Security.Principal;
10 using Microsoft.Azure.Mobile.Server.Authentication;
11 using System.Diagnostics;
12 using System.Net;
13 using System.Security.Claims;
14 namespace nccloudService.Controllers
15  {
16     [Authorize]
17     public class EventController : TableController<Event>
18     {
19         protected override void Initialize(
20             HttpControllerContext controllerContext)
21         {
22             base.Initialize(controllerContext);
23             nccloudContext context = new nccloudContext();
24             DomainManager = new EntityDomainManager<Event>(context, Request);
```

```
25          }
26          //get user email address
27          private string GoogleSID()
28          {
29              var principal = this.User as ClaimsPrincipal;
30              var sid = principal.FindFirst(ClaimTypes.NameIdentifier).Value;
31              return sid;
32          }
33          private async Task<string> GetEmailAddress()
34          {
35              var credentials = await User.GetAppServiceIdentityAsync< ↩
                    GoogleCredentials>(Request);
36              return credentials.UserClaims
37                  .Where(claim => claim.Type.EndsWith("/emailaddress"))
38                  .First<Claim>()
39                  .Value;
40              // Debug.WriteLine(credentials.UserClaims);
41          }
42
43          // GET tables/Event
44          public async Task<IQueryable<Event>> GetAllEvent()
45          {
46              var emailAddr = await GetEmailAddress();
47
48              return Query().Where(i=>i.Location.Custommers.
49              All(item=>item.CustomerEmail == emailAddr));
50          }
51
52          // GET tables/Event/48D68C86-6EA6-4C25-AA33-223FC9A27959
53          public SingleResult<Event> GetEvent(string id)
54          {
55              return Lookup(id);
56          }
57
58          // PATCH tables/Event/48D68C86-6EA6-4C25-AA33-223FC9A27959
59          public Task<Event> PatchEvent(string id, Delta<Event> patch)
60          {
61               return UpdateAsync(id, patch);
62          }
63
64          // POST tables/Event
65          public async Task<IHttpActionResult> PostEvent(Event item)
66          {
67              Event current = await InsertAsync(item);
68              return CreatedAtRoute("Tables", new { id = current.Id }, current);
69          }
70
```

```
        // DELETE tables/Event/48D68C86-6EA6-4C25-AA33-223FC9A27959
        public Task DeleteEvent(string id)
        {
             return DeleteAsync(id);
        }
    }
}
```

Listing D.8: Events Controller

- The third controller is the Location Controller. This controller retrieves the location details where the user has a relative reside in. It is also based on the user's email. Full implementation details are shown in Listing D.9.

```
using System.Linq;
using System.Threading.Tasks;
using System.Web.Http;
using System.Web.Http.Controllers;
using System.Web.Http.OData;
using Microsoft.Azure.Mobile.Server;
using nccloudService.DataObjects;
using nccloudService.Models;
using System.Security.Principal;
using Microsoft.Azure.Mobile.Server.Authentication;
using System.Diagnostics;
using System.Net;
using System.Security.Claims;
namespace nccloudService.Controllers
{
    [Authorize]
    public class LocationController : TableController<Location>
    {
        protected override void Initialize(HttpControllerContext ↩
             controllerContext)
        {
            base.Initialize(controllerContext);
            nccloudContext context = new nccloudContext();
            DomainManager = new EntityDomainManager<Location>(context, Request) ↩
                ;
        }
        //get user email address
        private string GoogleSID()
        {
            var principal = this.User as ClaimsPrincipal;
            var sid = principal.FindFirst(ClaimTypes.NameIdentifier).Value;
            return sid;
        }
        private async Task<string> GetEmailAddress()
        {
```

```
34              var credentials = await User.GetAppServiceIdentityAsync< ↩
                    GoogleCredentials>(Request);
35              return credentials.UserClaims
36                  .Where(claim => claim.Type.EndsWith("/emailaddress"))
37                  .First<Claim>()
38                  .Value;
39              // Debug.WriteLine(credentials.UserClaims);
40          }
41          // GET tables/Location
42          public async Task<IQueryable<Location>> GetAllLocation()
43          {
44              var emailAddr = await GetEmailAddress();
45              return Query().Where(i=>i.Custommers.All(item=>item.CustomerEmail== ↩
                    emailAddr));
46          }
47
48          // GET tables/Location/48D68C86-6EA6-4C25-AA33-223FC9A27959
49          public SingleResult<Location> GetLocation(string id)
50          {
51              return Lookup(id);
52          }
53
54          // PATCH tables/Location/48D68C86-6EA6-4C25-AA33-223FC9A27959
55          public Task<Location> PatchLocation(string id, Delta<Location> patch)
56          {
57               return UpdateAsync(id, patch);
58          }
59
60          // POST tables/Location
61          public async Task<IHttpActionResult> PostLocation(Location item)
62          {
63              Location current = await InsertAsync(item);
64              return CreatedAtRoute("Tables", new { id = current.Id }, current);
65          }
66
67          // DELETE tables/Location/48D68C86-6EA6-4C25-AA33-223FC9A27959
68          public Task DeleteLocation(string id)
69          {
70               return DeleteAsync(id);
71          }
72      }
73  }
```

Listing D.9: Location Controller

- The fourth controller is the Customer Controller. This controller is required only for the desktop implementation version in nursing homes where customers can get added to the database. At this stage this controller in not implemented.

Appendix E

Startup class - nCare backend solution

In the *Startup* class of the solution available below (Listing E.1), the *SetInitilazer* method creates an instance of the *nccloudContext* class which is responsible with the creating, deleting or checking of an existing database.

On publishing the solution to the Azure, database is populated with dummy data for testing propose by the *Seed* method from the *Start-up class*. This method is the only possible way to add data into the database at this stage, when the desktop version is not implemented yet. SQL Management Studio may not work to populated the database due to time stamp and version control constraints from the model classes which gets generated automatically.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Configuration;
4  using System.Data.Entity;
5  using System.Web.Http;
6  using Microsoft.Azure.Mobile.Server;
7  using Microsoft.Azure.Mobile.Server.Authentication;
8  using Microsoft.Azure.Mobile.Server.Config;
9  using nccloudService.DataObjects;
10 using nccloudService.Models;
11 using Owin;
12 using System.Collections.ObjectModel;
13 using System.Linq;
14
15 namespace nccloudService
16 {
17     public partial class Startup
18     {
19         public static void ConfigureMobileApp(IAppBuilder app)
20         {
```

```
21          HttpConfiguration config = new HttpConfiguration();
22          config.EnableSystemDiagnosticsTracing();
23          new MobileAppConfiguration()
24          .UseDefaultConfiguration()
25          .ApplyTo(config);
26          //Use Entity Framework Code First to create database tables based on your ↩
                DbContext
27          Database.SetInitializer(new nccloudInitializer());
28
29          nccloudContext db = new nccloudContext();
30          db.Database.Initialize(true);
31          MobileAppSettingsDictionary settings =
32              config.GetMobileAppSettingsProvider().GetMobileAppSettings();
33
34          if (string.IsNullOrEmpty(settings.HostName))
35          {
36              app.UseAppServiceAuthentication(new AppServiceAuthenticationOptions
37              {
38                  SigningKey = ConfigurationManager.AppSettings["SigningKey"],
39                  ValidAudiences = new[] { ConfigurationManager.AppSettings[" ↩
                        ValidAudience"] },
40                  ValidIssuers = new[] { ConfigurationManager.AppSettings[" ↩
                        ValidIssuer"] },
41                  TokenHandler = config.GetAppServiceTokenHandler()
42              });
43          }
44          app.UseWebApi(config);
45      }
46  }
47
48
49
50  public class nccloudInitializer : CreateDatabaseIfNotExists<nccloudContext>
51  // public class nccloudInitializer : DropCreateDatabaseAlways<nccloudContext>
52  {
53      protected override void Seed(nccloudContext context)
54      {
55          List<Location> locations = new List<Location>
56              {
57                  new Location {Id = Guid.NewGuid().ToString(),LocationName="Tara  ↩
                        Winthrop", Address="Tara Winthrop Private Clinic, Nevinstown  ↩
                        Lane, Swords, Co. Dublin, Ireland" },
58                  new Location {Id = Guid.NewGuid().ToString(),LocationName=" ↩
                        Balbriggan Private", Address="1 Main Street, Balbriggan, Co  ↩
                        Dublin, Ireland"},
```

```
59                  new Location {Id = Guid.NewGuid().ToString(),LocationName="Swords
                        Clinic", Address="Swords, Main Street Swords, Swords, Ireland"
                         }
60              };
61
62          foreach (Location location in locations)
63          {
64              context.Set<Location>().Add(location);
65          }
66          List<Customer> customers = new List<Customer>
67              {
68                  new Customer {Id = Guid.NewGuid().ToString(),CustomerEmail="
                        ionelpal@gmail.com", CustomerName="Ionel Pal", Location=
                        locations[0] },
69                   new Customer {Id = Guid.NewGuid().ToString(),CustomerEmail="
                         elenarpal@gmail.com", CustomerName="Elena Pal", Location=
                        locations[1] },
70                    new Customer {Id = Guid.NewGuid().ToString(),CustomerEmail="
                         xxxire@gmail.com", CustomerName="Ionel Smith", Location=
                        locations[2]}
71              };
72          foreach (Customer customer in customers)
73          {
74              context.Set<Customer>().Add(customer);
75          }
76
77          List<Patient> patients = new List<Patient>
78              {
79                  new Patient {Id = Guid.NewGuid().ToString(), PatientName="John
                        Smith", Location=locations[0], Customers = customers[0] },
80                  new Patient {Id = Guid.NewGuid().ToString(), PatientName="Ionel
                        Pal", Location=locations[1], Customers = customers[1]},
81                   new Patient {Id = Guid.NewGuid().ToString(), PatientName="Anthony
                         Pal", Location=locations[2], Customers = customers[2]}
82              };
83          foreach (Patient patient in patients)
84          {
85              context.Set<Patient>().Add(patient);
86          }
87
88          List<Message> messages = new List<Message>
89              {
90                  new Message {Id = Guid.NewGuid().ToString(), Title="Good News",
                        Details="Just ley you kno that John had a great day today",
                        WrittenBy="Elena", CustomerEmail="xxxirex@gmail.com", Patient=
                        patients[0] },
```

```
91                  new Message {Id = Guid.NewGuid().ToString(), Title="Updates",
                        Details="All the same tody, no new improvements.", WrittenBy="
                        Elena", CustomerEmail="xxxirex@gmail.com", Patient=patients[0]
                         },
92                  new Message {Id = Guid.NewGuid().ToString(), Title="Please Contact
                         us ASP", Details="John needs to be sent to the hospital ASP,
                        we need your concent please. The doctor recomends a surgery.",
                         WrittenBy="Elena", CustomerEmail="xxxirex@gmail.com", Patient
                        =patients[0] },
93                  new Message {Id = Guid.NewGuid().ToString(), Title="New updates",
                        Details="Thanks for yoy quick reply. So far he is stable and
                        resting", WrittenBy="Elena", CustomerEmail="xxxirex@gmail.com"
                        , Patient=patients[0] },
94                   new Message {Id = Guid.NewGuid().ToString(), Title="Good News",
                         Details="Just ley you kno that John had a great day today",
                         WrittenBy="Elena", CustomerEmail="xxxirex@gmail.com", Patient
                         =patients[0] },
95                  new Message {Id = Guid.NewGuid().ToString(), Title="Updates",
                        Details="All the same tody, no new improvements.", WrittenBy="
                        Elena", CustomerEmail="xxxirex@gmail.com", Patient=patients[0]
                         },
96                  new Message {Id = Guid.NewGuid().ToString(), Title="Please Contact
                         us ASP", Details="John need to be sent to the hospital ASP,
                        we need your concent please. The docto recomends a surgery",
                        WrittenBy="Elena", CustomerEmail="xxxirex@gmail.com", Patient=
                        patients[0] },
97                  new Message {Id = Guid.NewGuid().ToString(), Title="New updates",
                        Details="Thanks for yoy quick reply. So far he is stable and
                        resting", WrittenBy="Elena", CustomerEmail="xxxirex@gmail.com"
                        , Patient=patients[0] },
98                   new Message {Id = Guid.NewGuid().ToString(), Title="New updates",
                         Details="Hi Ionel, Your father had a good day tody. He is
                        making massive progress.", WrittenBy="Adriana",CustomerEmail=
                        "ionelpal@gmail.com", Patient=patients[1] },
99                   new Message {Id = Guid.NewGuid().ToString(), Title="Please come
                        by", Details="Hi, please come by to talk about the new
                        recovery plan.", WrittenBy="Adriana",CustomerEmail="
                        ionelpal@gmail.com", Patient=patients[1] },
100                 new Message {Id = Guid.NewGuid().ToString(), Title="Urgent",
                        Details="Hi Ionel, Your father need urgent his new glasses ",
                        WrittenBy="Adriana",CustomerEmail="ionelpal@gmail.com",
                        Patient=patients[1] },
101
102                   new Message {Id = Guid.NewGuid().ToString(), Title="Urgent",
                          Details="Hi Anthony, Your father need urgent his new glasses
                           ", WrittenBy="Adriana",CustomerEmail="anthonypal37@gmail.
                          com", Patient=patients[2] },
```

```
103                     new Message {Id = Guid.NewGuid().ToString(), Title="Updates",
                            Details="Hi Anthony, Today he took his tables without any
                            issues. ", WrittenBy="Elena",CustomerEmail="
                            anthonypal37@gmail.com", Patient=patients[2] },
104                       new Message {Id = Guid.NewGuid().ToString(), Title="Please
                              call", Details="Hi Anthony, Your father need urgent his
                              new glasses ", WrittenBy="Silvia",CustomerEmail="
                              anthonypal37@gmail.com", Patient=patients[2] },
105                         new Message {Id = Guid.NewGuid().ToString(), Title="Updates
                                ", Details="Hi Anthony, Ionel is fealing a bit bettrer
                                today!", WrittenBy="Elena",CustomerEmail="
                                anthonypal37@gmail.com", Patient=patients[2] },
106                           new Message {Id = Guid.NewGuid().ToString(), Title="New
                                  updates", Details="Hi Anthony, I just like to let you
                                   know that he had improved a lot! ", WrittenBy="
                                  Adriana",CustomerEmail="anthonypal37@gmail.com",
                                  Patient=patients[2] },
107                             new Message {Id = Guid.NewGuid().ToString(), Title="
                                    Urgent", Details="Hi Anthony, Your father need
                                    urgent to go hospital, please call or come over
                                    when ready! ", WrittenBy="Maria",CustomerEmail="
                                    anthonypal37@gmail.com", Patient=patients[2] },
108
109                   new Message {Id = Guid.NewGuid().ToString(), Title="Please call
                          to discuss", Details="We have a situation here!", WrittenBy=
                          "Marie", CustomerEmail="elenarpal@gmail.com", Patient=
                          patients[2] }
110               };
111
112           foreach (Message message in messages)
113           {
114               context.Set<Message>().Add(message);
115           }
116
117
118           List<Event> events = new List<Event>
119               {
120                   new Event {Id = Guid.NewGuid().ToString(),EventTitle="Christmas
                          Party", EventDetails="Hi all, Tara Winthrop has setup the
                          Christmas party onn the 16th of December, All welcome",
                          EventDate= new DateTime(2016,11,10,10,0,0), Location =
                          locations[0] },
121                   new Event {Id = Guid.NewGuid().ToString(),EventTitle="Charity Dayy
                          ", EventDetails="Balbriggan Private Clinic has a Charity event
                           on the 24th of November @ 16.00, All welcome!", EventDate=
                          new DateTime(2016,12,11,10,0,0) ,Location = locations[1] } ,
```

```
122                 new Event {Id = Guid.NewGuid().ToString(),EventTitle="Halloween
                        Party", EventDetails="Swords Clinic has a Halloween Party on
                        the 31st of October @20.oo hours, All welcome!", EventDate=
                        new DateTime(2016,09,29,10,0,0), Location = locations[2] },
123                 new Event {Id = Guid.NewGuid().ToString(),EventTitle="Charity
                        Party", EventDetails="Tara Winthrop has a Charity event on
                        Sunday the 24 of September @16.00 hours, All welcome!",
                        EventDate= new DateTime(2016,11,10,10,0,0), Location =
                        locations[0] },
124                 new Event {Id = Guid.NewGuid().ToString(),EventTitle="Charity Dayy
                        ", EventDetails="Balbriggan Private Clinic has a Charity event
                         on the 10th of December @ 20.00, All welcome!", EventDate=
                        new DateTime(2016,10,10,10,0,0) ,Location = locations[1] } ,
125                 new Event {Id = Guid.NewGuid().ToString(),EventTitle="Halloween
                        Party", EventDetails="Halloween Party will be held in the
                        Thrinity Capital hotel on the 31 st @17.00hours.", EventDate=
                        new DateTime(2016,11,29,10,0,0), Location = locations[2] }
126             };
127
128             foreach (Event ev in events)
129             {
130                 context.Set<Event>().Add(ev);
131             }
132
133
134
135             base.Seed(context);
136         }
137     }
138 }
```

Listing E.1: Startup class representation

Appendix F

Cloud Resources

The initial step is to create an account with Microsoft Azure available at the following link (https://azure.microsoft.com/en-us/).

In the web portal create the following resources:

- Create a Resource Group in North Europe, called nCare Resource Group.
- Create a new SQL database called nCaredb in the same Resource Group.
- Add a new Mobile App project called nCare in the nCare Resource Group, and connect it with the newly created database nCaredb.
- Add Authentication provider.

The authentication with the Google is created in the following steps:

- Create a developer account at the following link (https://console.developers.google.com).
- In the Google Developer Console portal create a project called *nCare*.
- Enable the *Google+ API* to gain access to user's profile.
- In the credentials, select *OAth Client ID* and the *Web application*.
- Fill in the details with the nCare's URL from the Azure and save.

Figure F.1 shows how to add the *URL* from the Azure into the Google Developer Console portal.

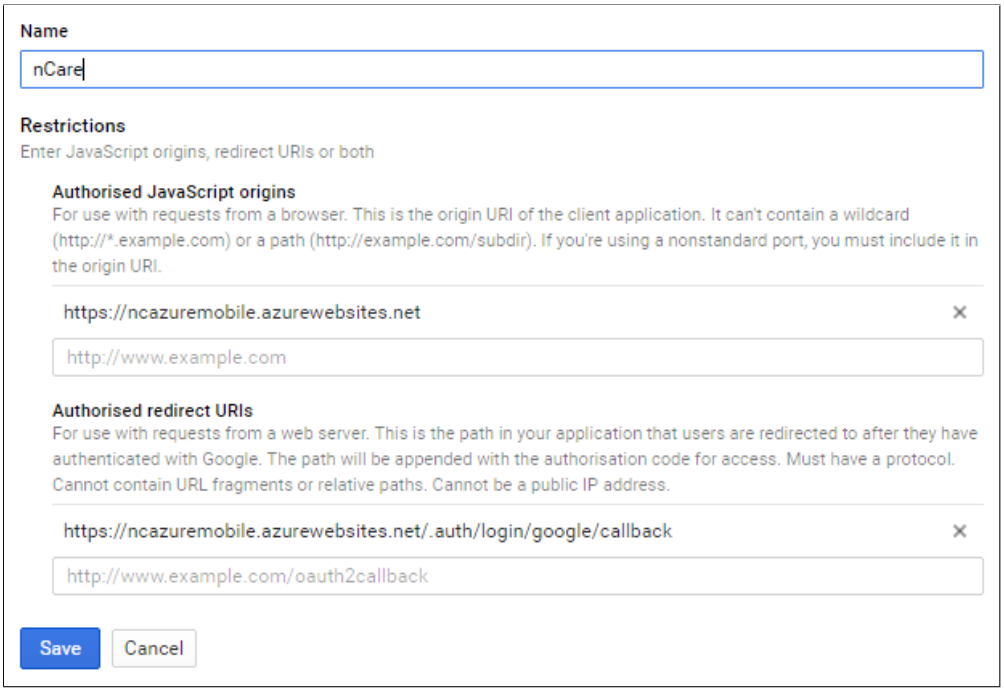


Figure F.1: Google Developer Console redirect URL to Azure

After saving the project details, Google will issue a Client Id and a Client Secret Key. These details needs to be added in the Azure portal and Google needs to be enabled as the authenticator provider.

Appendix G

Xamarin Forms implementation

G.1 Setting-up the development environment

nCare Xamarin implementation was initially started in Visual Studio and alternated with Xamarin Studio in Mac computer for specific IOS application testing. The first step is to install Cross Platform Mobile Development in Visual Studio which will add Xamarin, Apace Cordova and Android SDK as shown in Figure G.1.

Figure G.1: Mobile Development tools instalation in Visual Studio

Before create initial project it is necessary to add into the Visual Studio the Xamarin subscription

and connect remotely to an IOS Macintosh computer, in order to be able to compile the IOS project. Macintosh computer requires to have installed Xcode and an IOS developer subscription. Visual Studio requires to be connected to a Mac computer before build any IOS application. The best way is to have the Mac computer on the same network. This can be setup from the Visual Studio Tools option, navigate to IOS /Xamarin Mac Agent and add the Mac computer IP address.

For the Android project implementation, Visual Studio provides an emulator, this resource sometimes can be very slow. For this project, Xamarin Android Simulator was the primary emulator used for debugging and testing the application in an early stage before testing with real devices.

Android environment requires always to be updated after the initial installation. This can be organized by starting the Android SDK Manager from Visual Studio menu and update Android SDK to include Android's latest versions (API 23 and API 24) and Google USB Driver required for testing the application on real devices. At present, Adroit SDK can to be updated only manually.

G.2 Xamarin Forms solution implementation

G.2.1 Xamarin Forms initial project architecture

In the Visual Studio, click file and select Cross-Platform Blank App (Xamarin.Forms Portable) and complete the wizard. This will create a solution with three projects, a shared class library project named *nCare.Portable*, an Android project *nCare.Droid* and a IOS project named *nCare.IOS* as shown in Figure G.2.

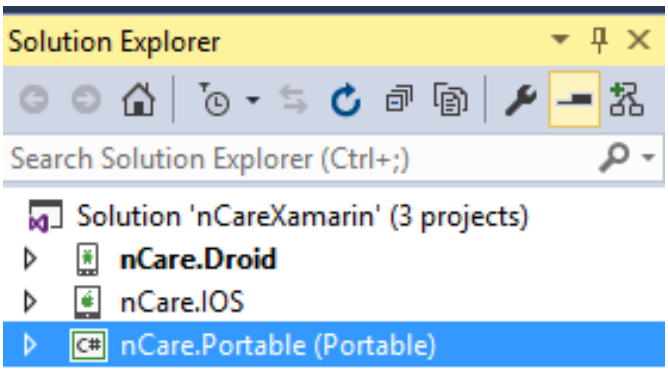


Figure G.2: Xamarin Forms initial project architecture

nCare.Portable is the main working project in the solution, it contains all code that can be shared with others two projects, *nCare.Droid* and *nCare.IOS* and cannot include platform specific code. The root page of the application is the *App.cs* class, located in the *nCare.Portable* project. Platform specific code can be added only in *MainActivity.cs* class, located in the *nCare.Droid* project for the Android implementation, and in the *AppDelegate.cs* class, located in the *nCare.IOS* project, for the IOS implementation.

G.2.2 Setup Domain Model

In portable class library project (*nCare.Portable*) create a folder called Models, this folder will contain the local representation of all the business entity classes on the front-end, for storing and retrieving the data from the Azure, where the backend solution is deployed. The entity data in the backend solution are in a set of key-value pairs format. To map the entity model classes from the front-end with to the backend, the best option is to use the *JsonProperty* annotation.

Relations between entities on the front-end are not supported and they need to be ignored with *JsonIgnore* annotation or not included as in this solution. For offline support, a property to keep track of the data versions is added.

All entity classes have the properties as specified in the UML diagram available in Appendix C. The entity models representation is shown in Listing G.1, Listing G.2 and Listing G.3.

- Location

```
1  public class Location
2      {   string id;
3          [JsonProperty(PropertyName = "id")]
4          public string Id
5          { get { return id; } set { id = value; } }
6          [JsonProperty(PropertyName = "userId")]
7          public string UserID { get; set; }
8          [JsonProperty(PropertyName = "locationname")]
9          public string LocationName { get; set; }
10         [JsonProperty(PropertyName = "address")]
11         public string Address { get; set; }
12
13         [Version]
14        public string Version { get; set; }
15     }
```

Listing G.1: Xamarin Forms Location Model representation

- Message

```
public class Message
  {
    string id;
    [JsonProperty(PropertyName = "id")]
    public string Id
    { get { return id; } set { id = value; }}
    [JsonProperty(PropertyName = "userId")]
    public string UserID { get; set; }
    [JsonProperty(PropertyName = "title")]
    public string Title { get; set; }
    [JsonProperty(PropertyName = "details")]
    public string Details { get; set; }
    [JsonProperty(PropertyName = "writtenby")]
    public string WrittenBy { get; set; }
    [JsonProperty(PropertyName = "updatedat")]
    public string UpdatedAt { get; set; }

    [Version]
    public string Version { get; set; }
  }
```

Listing G.2: Xamarin Forms Message Model representation

- Event

```
public class Event
    {
        string id;
        [JsonProperty(PropertyName = "id")]
        public string Id
        { get { return id; } set { id = value; }}
        [JsonProperty(PropertyName = "userId")]
        public string UserID { get; set; }
        [JsonProperty(PropertyName = "updatedat")]
        public string UpdatedAt { get; set; }

        [Version]
        public string Version { get; set; }
        [JsonProperty(PropertyName = "eventtitle")]
        public string EventTitle { get; set; }
        [JsonProperty(PropertyName = "eventdetails")]
        public string EventDetails { get; set; }
        [JsonProperty(PropertyName = "eventdate")]
        public DateTime EventDate { get; set; }
    }
```

Listing G.3: Xamarin Forms Event Model representation

G.2.3 Web Server Data Access

Communication with the backend solution on Azure requires few steps implementations. The initial step is to add Nuget packages support between front-end and backend for connection, authentication and storage. This is implemented by adding the *Microsoft.Azure.Mobile.Client* and the *Microsoft.Azure.Mobil.Client.SQLiteStorage* Nuget package in all projects. These packages support offline connection and authentication between the front-end and backend on Azure Mobile Apps where the *nCare* backend solution and database are hosted. Figure G.3 shows the installation of the Azure Nuget packages.

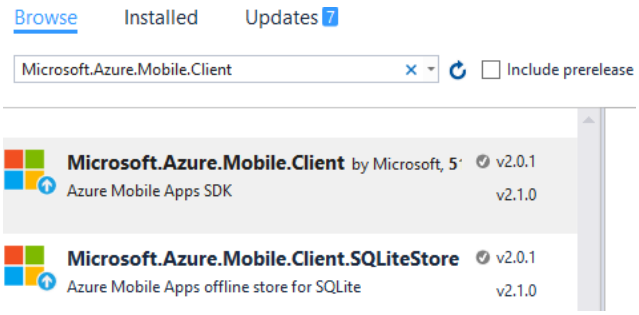


Figure G.3: Azure Nuget Packages

The *Mobile Client* nuget package must first to be initialized in each project by adding the method *Microsoft.WindowsAzure.MobileServices.CurrentPlatform.Init* before the *LoadApplication* method in the *MainActivity* class for Android project and in the *AppDelegates* class for the IOS. This method ensures that communication with the nCare web server are always executed before the application starts.

The second step is to add a class called *ncareManager* in *nCare.Portable* project, this class is a communication layer between the user's mobile client and the *nCare Web Server* hosted on Azure Mobile Apps. It implements methods to access and retrieve data from the database tables hosted in the Azure. The first method implemented in the *ncareManager* class is a static method called *DefaultManager* for accessing the class from outside with get and set properties as shown in Listing G.4.

```
1 public partial class ncareManager {
2  static ncareManager defaultInstance = new ncareManager();
3  public static ncareManager DefaultManager {
4   get { return defaultInstance;}
5   private set { defaultInstance = value;}
6  }}
```

Listing G.4: nCareManager accessor implementation

The third step is to enable the Mobile App SDK and to create a *MobileServiceClient* object called *client* (Listing G.6) that is used to access the mobile service hosted in the Azure.

This client enables the create, read, updated and delete (CRUD) operations between mobile and web server. The client requires to pass the URL of the nCare Web Server from Azure Mobile App, which

is represented by a static variable as showing in Listing G.5.

```
1 public static class Constants
2   {
3   public static string ApplicationURL = @"https://ncazuremobile.azurewebsites.net";
4   }
```

Listing G.5: Azure Mobile APP Web Server URL

The fourth step is to create tables reference for each existing entity model defined in Chapter 6.3.3, by calling *GetTable* method on an instance of the *MobileServiceClient* and instantiate a web-server connection.

```
1
2 public partial class ncareManager
3   {
4     static ncareManager defaultInstance = new ncareManager();
5     MobileServiceClient client;
6
7     IMobileServiceTable<Message> messageTable;
8     IMobileServiceTable<Event> eventTable;
9     IMobileServiceTable<Location> locationTable;
10
11    private ncareManager()
12    {
13      this.client = new MobileServiceClient(Constants.ApplicationURL);
14      this.messageTable = client.GetTable<Message>();
15      this.eventTable = client.GetTable<Event>();
16      this.locationTable = client.GetTable<Location>();
17    }
18       ..............
```

Listing G.6: Web-server connection implementation

The last step is to implements tasks to retrieve each table data for each entity. The first task implemented is the *GetMessagesAync*, this task will get all messages specific to the signed in user.

The implementation of the *"ObdervableCollection"* which notify the user when database has an update. Due to possible network issue and to improve mobile client responsiveness, the task must implement the asynchronous pattern.

The first task implemented is called *GetMessagesAsync*, it is a task to retrieve all the messages from the database as showing in Listing G.7.

```
public async Task < ObservableCollection < Message >> GetMessagesAsync() {
 try {

  IEnumerable < Message > messages = await messageTable
   .ToEnumerableAsync();

  return new ObservableCollection < Message > (messages);
 } catch (MobileServiceInvalidOperationException msioe) {
  Debug.WriteLine(@ "Invalid sync operation: {0}", msioe.Message);
 } catch (Exception e) {
  Debug.WriteLine(@ "Sync error: {0}", e.Message);
 }
 return null;
}
```

Listing G.7: nCareManager Messages task

The second task implemented is called *GetEventsAsync* (Listing G.8) to get all the events specific to the nursing home, where a user has signed up for the service. This is implemented similar to the *GetMessagesAsync* task.

```
public async Task < ObservableCollection < Event >> GetEventsAsync() {
 try {

  IEnumerable < Event > events = await eventTable
   .ToEnumerableAsync();

  return new ObservableCollection < Event > (events);
 } catch (MobileServiceInvalidOperationException msioe) {
  Debug.WriteLine(@ "Invalid sync operation: {0}", msioe.Message);
 } catch (Exception e) {
  Debug.WriteLine(@ "Sync error: {0}", e.Message);
 }
 return null;
}
```

Listing G.8: nCareManager Events task

And lastly, the implementation of the *GetLocationsAsync* (Listing 6.16) task, required to retrieve the nursing home location details from the server.

```
1  public async Task < ObservableCollection < Location >> GetLocationsAsync() {
2   try {
3
4    IEnumerable < Location > locations = await locationTable
5     .ToEnumerableAsync();
6
7    return new ObservableCollection < Location > (locations);
8   } catch (MobileServiceInvalidOperationException msioe) {
9    Debug.WriteLine(@ "Invalid sync operation: {0}", msioe.Message);
10  } catch (Exception e) {
11   Debug.WriteLine(@ "Sync error: {0}", e.Message);
12  }
13  return null;
14 }
```

Listing G.9: nCareManager Location task

G.2.4 User Authentication

User authentication is implemented with the *Microsoft.WindowsAzure.Mobile SDK*, which supports authentication with Google, Twitter, Microsoft and Active Directory Authentication.

The authentication has platform specific implementation and the code cannot be shared, exceptions makes the interfaces. To guarantee the same implementation in each project, an interface called *IAuthenticate* is created in the *nCare.Backend* project, with methods for the authentication and logout. Listing G.10 shows the *IAuthenticate* interface implementation

```
1  public interface IAuthenticate
2     {
3         Task<bool> AuthenticateAsync();
4
5     Task<bool> LogoutAsync();
6     };
```

Listing G.10: IAuthenticate interface

To ensure that the user is authenticated before accessing the application, in the *App.cs* class from *nCare.Shared* project, a static member is added and initialized it before the root of the application (Listing G.11).

```
public class App : Application
 {
      public static IAuthenticate Authenticator { get; private set; }
   public static void Init(IAuthenticate authenticator)
   {
     Authenticator = authenticator;
   }
public App ()
   { // The root page of the application
        MainPage = new NavigationPage(new LogingPage());
   }
```

Listing G.11: Initilise IAuthenticator before the root of the project

The nest step, is to add platform specific implementation on Android and IOS project as following:

- The Android platform specific implementation of the *IAuthenticate* interface is created in *MainActivity* class form *nCare.Droid*. For the development and testing Google is the only authenticator implemented. Listing G.12 shows the implementation of the *IAuthenticate* interface in Android project.

```
public async Task < bool > AuthenticateAsync() {
 bool success = false;
 try {
  if (user == null) {
   // The authentication provider could also be Facebook, Twitter, or Microsof
   user = await ncareManager.DefaultManager.CurrentClient.LoginAsync(this,
       MobileServiceAuthenticationProvider.Google);

   if (user != null) {
    //  CreateAndShowDialog("You are now logged in Successfully", "Logged in!")
        ;
   }
  }
  success = true;
 } catch (Exception ex) {
  CreateAndShowDialog(ex.Message, "Authentication failed");
 }
 return success;
}
```

Listing G.12: IAuthenticator Interface Android

- The IOS implementation of the IAuthenticate interface is created in the *AppDelegate* class from *nCare.IOS* project. Listing G.13 shows the implementation of the *IAuthenticate* interface in IOS project.

```
1
2       public async Task < bool > AuthenticateAsync() {
3   bool success = false;
4   try {
5    if (user == null) {
6     // The authentication provider could also be Facebook, Twitter, or  ↩
          Microsoft
7     user = await ncareManager.DefaultManager.CurrentClient.LoginAsync( ↩
          UIApplication.SharedApplication.KeyWindow.RootViewController,  ↩
          MobileServiceAuthenticationProvider.Google);
8    }
9    success = true;
10  } catch (Exception ex) {
11   var authAlert = new UIAlertView("Authentication failed", ex.Message, null,  ↩
         "OK", null);
12   authAlert.Show();
13  }
14  return success;
15 }
```

Listing G.13: IAuthenticator Interface IOS

G.2.5 User Interface

Xamarin Forms user interface is fully implemented with *XAML* files. This files are mostly used to define a visual content for the page, which is always associated with a C# code behind file for markup code support. Together the two files are tied together in a class with links between. All XAML files include a default *namespace* which has a component in the code behind file constructor called *InitializeComponent*, always responsible to initialize and create the View Page. Listing G.14 is showing the default *namespace* for any XAML file.

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
3             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4             x:Class="XamarinPage"
5             ...
6 </ContentPage>
```

Listing G.14: Xamarin Forms default page architecture

User interface is defined of four XAML main pages with tabbed navigation between, a login page and two detail pages associated with two master detail pages. These pages are defined Chapter 5 and are according to the Use Case Diagrams from Chapter 4.5. The initial step is to add a folder called Views to hold all the view pages for this application. A XAML page can be created in Visual Studio from the Start menu, by adding a Xamarin Forms XAML content page. User Interface are implemented the next subsections.

G.2.6 Login page

Login page is the landing page of the application where user is required to sign-in. The page includes a label and a button wrapped in a Grid Layout view. The button has a special XAML attribute *x:Name="LoginButton"* which is tied up with code behind file in the constructor. Listing G.15 shows the Login page XAML implementation.

```
1  <StackLayout VerticalOptions="Center" Padding="40">
2    <Grid>
3      <Grid.RowDefinitions>
4        <RowDefinition Height="2*" />
5        <RowDefinition Height="3*" />
6        <RowDefinition Height="*" />
7        <RowDefinition Height="200" />
8      </Grid.RowDefinitions>
9      <Grid.ColumnDefinitions>
10       <ColumnDefinition Width="Auto" />
11     </Grid.ColumnDefinitions>
12     <Label Text="Welcome to nCare!" TextColor="#0097A7" FontSize="Large"  ↩
           FontAttributes="Italic" HorizontalOptions="Center" VerticalOptions="Center ↩
           " Grid.Row="0" Grid.Column="0" />
13     <Label Text="Please obtain your Login details from your related Nursing Home!" ↩
            FontAttributes="Italic" FontSize="Medium" TextColor="Gray"  ↩
           HorizontalOptions="Center" VerticalOptions="Center" Grid.Row="1" Grid. ↩
           Column="0" Margin="10,0,20,0" />
14     <Button x:Name="LoginButton" Text="Login" TextColor="White" BackgroundColor=" ↩
           #0097A7" FontSize="Medium" Grid.Row="2" Grid.Column="0" />
15     <Label x:Name="messageLabel" FontSize="Medium" Grid.Row="3" Grid.Column="0" />
16   </Grid>
17 </StackLayout>
```

Listing G.15: Login page XAML file

The Login page C# includes a method called LoginButton_Clicked that check if user is authenticated, then redirects the user accordingly, to the main page or to the Google authentication page. This method is executed on LoginButton event clicked. Listing G.16 shows the Login page code behind file implementation.

```
1  public partial class LogingPage: ContentPage {
2   bool authenticated = false;
3   public LogingPage() {
4    InitializeComponent();
5    LoginButton.Clicked += LoginButton_Clicked;
6   }
7   async void LoginButton_Clicked(object o, EventArgs e) {
8    try {
9     if (App.Authenticator != null) {
10     authenticated = await App.Authenticator.AuthenticateAsync();
11    }
12    if (authenticated == true) {
13     await Navigation.PushAsync(new MainPage());
14    }
15   } catch (InvalidOperationException ex) {
16    if (ex.Message.Contains("Authentication was cancelled")) {
17     messageLabel.Text = "Authentication cancelled by the user";
18    }
19   } catch (Exception) {
20    messageLabel.Text = "Authentication failed";
21   }
22  }
23 }
```

Listing G.16: Login page C# file

G.2.7 Main Page

This a tabbed navigation page and is fully implemented in XAML. The scope of this page is to create the navigation of the application.

The initial step is to declare a property in the header called *local* which is required to create the navigation links between view pages, this is implemented by adding a child page object in the *TabbedPage* element. Child pages include icons images. This images are only supported in IOS and fully ignored by Android. Listing G.17 shows the Main page XAML implementation.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
3      xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4      xmlns:local="clr-namespace:nccloud.Views; assembley=nccloud"
5      x:Class="nccloud.Views.MainPage"
6      Title="nCare"
7      NavigationPage.HasBackButton="False" >
8    <TabbedPage.Children >
9      <local:MessageList Icon="email" />
10     <local:Events Icon="calendar"/>
11     <local:Directions Icon="map"/>
12     <local:LogOutPage Icon="logout"/>
13   </TabbedPage.Children>
14 </TabbedPage>
```

Listing G.17: MainPage XML file

G.2.8 Message page

The Message page is a *Xamarin Master Detail Page* type, which is a page that manages two related XAML type pages, one is the Master page which displays a list of the messages and the other is a Detail page which displays details of a particular message. Master page implements *Data-binding* and *Grid ListView* for dynamic display the data.

Data-binding is used to establish a loose relationship between a model and a ListView, it connects properties of two objects and requires three pieces of information: a source, a path and a target. The source object provides the data and the target object must be a bindable property and will display the data from the source object. The path is usually represented by a label.

In the code behind C# file instantiate a new *ObservableCollection* of Messages data type called messages which is the source object for the data-binding in the XAML page. The *ObservableCollection* is a collection class that provides notification to the XAML page when a new message is added.

When the Message page appears on the user interface, a method called *OnAppering* will automatically be executed and will create asynchronously a message list with the data retrieved from the database by the *GetMessagesAsync* task located in the *nCareManager* class. The message list, then gets assigned to the binding target on the *ItemSource* element.

The navigation to the Detail page is implemented by the *OnItemTapped* and *OnItemSelected* methods which provides support for the Detail page, they are executed when a user is tapping on a specific message, the message gets selected and a new page is pushed on the stack to provide details specific to that message. The methods are specific implemented with tapping event from Xamarin Forms API.

User has the possibility to refresh the page, this event is implemented by *OnRefresh* method, which is similar implemented with *OnAppearing* method. This method is executed on *Refreshing* button clicked event from the markup page. Listing G.18 shows the Message page C# code behind file implementation.

```
1  public partial class MessageList : ContentPage
2    {  IList<Message> messages = new ObservableCollection<Message>();
3       MessageManager manager = MessageManager.DefaultManager;
4
5       public MessageList()
6       { InitializeComponent(); }
7      protected async override void OnAppearing()
8       {   base.OnAppearing();
9          this.IsBusy = true;
10          try
11          { messages = await manager.GetMessagesAsync();
12              mesgList.ItemsSource = messages;
13          }
14          finally { this.IsBusy = false; }
15      }
16   async void OnRefreshing(object sender, EventArgs e)
17      {
18          this.IsBusy = true;
19          try
20          {   messages = await manager.GetMessagesAsync();
21             mesgList.ItemsSource = messages;
22          }
23          finally { this.IsBusy = false; }
24      }
25      async void OnItemTapped(object sender, ItemTappedEventArgs e)
26      {  Message tappedMessage = (Message)e.Item;
27          await this.Navigation.PushAsync(new Details(tappedMessage));
28      }
29      void OnItemSelected(object sender, SelectedItemChangedEventArgs e)
30      {  var message = e.SelectedItem as Message;
31          message = (Message)e.SelectedItem;
32      }
33    }
```

Listing G.18: Message page C# file implementation

The Message page XAML file includes a button for page refresh and a standard Grid ListView template implementation, with DataTemplte and TextCell which defines the binding path for the message object. Listing G.19 shows the Message page XAML file.

```
<StackLayout Orientation="Vertical" VerticalOptions="FillAndExpand" Padding="10">
    <Grid>
      <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="*" />
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
      </Grid.ColumnDefinitions>
      <Label Text="Latest Messages!" FontSize="Medium" TextColor="Olive"  ↩
          HorizontalOptions="StartAndExpand" VerticalOptions="Center" Grid.Row="0"  ↩
          Grid.Column="0" />
      <Button Text="Refresh" FontSize="Small" TextColor="Red" HorizontalOptions="End ↩
          " Clicked="OnRefreshing" VerticalOptions="Center" Grid.Row="0" Grid.Column ↩
          ="1" />
    </Grid>
    <ListView x:Name="mesgList" SeparatorVisibility="None" IsPullToRefreshEnabled=" ↩
        False" Refreshing="OnRefreshing" ItemTapped="OnItemTapped" ItemSelected=" ↩
        OnItemSelected">
      <ListView.ItemTemplate>
        <DataTemplate>
          <TextCell Text="{Binding Title}" Detail="{Binding UpdatedAt }">
          </TextCell>
        </DataTemplate>
      </ListView.ItemTemplate>
    </ListView>
  </StackLayout>
```

Listing G.19: Message page XAML file

G.2.9 Message Detail page

This page is pushed on the navigation stack when a particular message from the Message page is tapped by a user. The selected message object is passed over by the Message page C# file *OnItemTapped* method. The code behind file binding this object with the markup in the constructor and provided a method for returning to the Master Page. Listing G.20 shows the Message Detail page code behind file implementation.

```
public partial class Details : ContentPage
  {  Message tappedMessage;
    public Details(Message tappedMessage)
    {
      this.tappedMessage = tappedMessage;
      InitializeComponent();
      details.BindingContext = tappedMessage;
      writtenBy.BindingContext = tappedMessage;
    }
        async void OnDismissButtonClicked(object sender, EventArgs args)
        { await Navigation.PopAsync();
        }
  }
```

Listing G.20: Message Detail page C# file impelemtation

Markup page includes a Grid View with three labels that are defining the binding path for the message object. Listing G.21 shows the Message Detail page XAML implementation.

```
<StackLayout Orientation="Vertical" VerticalOptions="FillAndExpand" Padding="10">
    <Grid><Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="*" />
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
      </Grid.ColumnDefinitions>
      <Label Text="Details" FontSize="Medium" TextColor="Olive" FontAttributes=" ↩
          Italic" HorizontalOptions="StartAndExpand" VerticalOptions="Center" Grid. ↩
          Row="0" Grid.Column="0" />
      <Button x:Name="dismissButton" Text="Return" FontSize="Small" TextColor="Red"  ↩
          Clicked="OnDismissButtonClicked" HorizontalOptions="End" VerticalOptions=" ↩
          Center" Grid.Row="0" Grid.Column="1" />
    </Grid>
    <Label x:Name="details" Text="{Binding Details}" FontSize="Medium" />
    <Label Text="Updates by," FontAttributes="Italic" TextColor="Olive" FontSize=" ↩
        Medium" />
    <Label x:Name="writtenBy" Text="{Binding WrittenBy}" TextColor="Olive" FontSize= ↩
        "Medium" />
  </StackLayout>
```

Listing G.21: Message Detail page markup file impelemtation

G.2.10 Event page

Event page is similar with the Message page, it has a Master Detail Page and a Detail Page. The Event page C# file instantiates a new ObservableCollection of Events data type called events which is the source object for the markup binging. ObservableColloection is a collection class that provides notifications when an event is added or removed.

In the C# file the first method implemented is called *OnAppearing*. This method is automatically executed when this page appearing on user interface. It calls the nCareManger class to retrieve the latest events and binds them with the markup page on ItemSource element. The second method implemented is a method to refresh the data on the page and has similar implementation with the first method as shown in Listing G.22.

```
1  public partial class Events : ContentPage
2   { IList<Event> events = new ObservableCollection<Event>();
3     ncareManager manager = ncareManager.DefaultManager;
4
5         public Events()
6     {
7       InitializeComponent();
8     }
9     async void OnRefreshing(object o, EventArgs e)
10    {
11      this.IsBusy = true;
12      try
13      {
14        events = await manager.GetEventsAsync();
15        eventList.ItemsSource = events;
16      }
17      finally { this.IsBusy = false; }
18    }
19    protected async override void OnAppearing()
20    {
21      base.OnAppearing();
22      this.IsBusy = true;
23      try
24      {
25         events = await manager.GetEventsAsync();
26        eventList.ItemsSource = events;
27      }
28      finally { this.IsBusy = false; }
29    }
30           .......
```

Listing G.22: Event page C# file implemetation

The last two methods *OnItemTapped* and *OnItemSelected* provides support for Event Detail page, they are executed when a user is tapping on a specific event, the event gets selected and a new page is popping on the stack and provides details specific to that event. The methods are specific implemented

to tapping event from Xamarin Forms API. Implementation details of these methods are shown in Listing G.23.

```
async void OnItemTapped(object sender, ItemTappedEventArgs e)
{
  Event tappedEvent = (Event)e.Item;
  await this.Navigation.PushAsync(new EventDetails(tappedEvent));
}
void OnItemSelected(object sender, SelectedItemChangedEventArgs e)
{
  var ev = e.SelectedItem as Event;
  ev = (Event)e.SelectedItem;
}
```

Listing G.23: Event page C# file navigation methods implementation

Event page markup is including a Grid ListView template with DataTemplate and TextCell that defines the binding path for the event object. The page has similar implementation with the Message page and is tied up with code behind file for binding as shown in Listing G.24.

```
<StackLayout Orientation="Vertical" VerticalOptions="FillAndExpand" Padding="10">
    <Grid><Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="*" />
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
      </Grid.ColumnDefinitions>
      <Label Text="Latest Events!" FontSize="Medium" TextColor="Olive"  ↩
          HorizontalOptions="StartAndExpand" VerticalOptions="Center" Grid.Row="0"  ↩
          Grid.Column="0" />
      <Button Text="Refresh" FontSize="Small" TextColor="Red" HorizontalOptions="End ↩
          " Clicked="OnRefreshing" VerticalOptions="Center" Grid.Row="0" Grid.Column ↩
          ="1" />
    </Grid>
    <ListView x:Name="eventList" SeparatorVisibility="None" IsPullToRefreshEnabled=" ↩
        true" Refreshing="OnRefreshing" ItemTapped="OnItemTapped" ItemSelected=" ↩
        OnItemSelected">
      <ListView.ItemTemplate>
        <DataTemplate>
          <TextCell Text="{Binding EventTitle}" Detail="{Binding EventDate}" />
        </DataTemplate>
      </ListView.ItemTemplate>
    </ListView>
  </StackLayout>
```

Listing G.24: Event page markup implementation

G.2.11 Event Detail page

This page displays details of a tapped event from the Event page. In the constructor of C# Event Detail page, the selected event object from the Detail page is binded on BindingContext with the markup. A method called *OnButtonCliecked* to pop out this page from the navigation stack when user press return button from the markup page is also implemented here. Listing G.25 shows the details implementation.

```
1  public partial class EventDetails : ContentPage
2     {   Event tappedEvent;
3         public EventDetails(Event tappedEvent)
4         {
5             this.tappedEvent = tappedEvent;
6             InitializeComponent();
7             eventdetails.BindingContext = tappedEvent;
8         }
9         async void OnDismissButtonClicked(object sender, EventArgs args)
10        {  await Navigation.PopAsync(); }
11    }
```

Listing G.25: Event Detail page C# implemetation

In the markup page includes a Grid ListView with a Button for page return and a label which is the target element for the binding path as shown in Listing G.26.

```
1  <StackLayout Orientation="Vertical" VerticalOptions="FillAndExpand" Padding="10">
2     <Grid>
3       <Grid.RowDefinitions>
4         <RowDefinition Height="*" />
5         <RowDefinition Height="*" />
6       </Grid.RowDefinitions>
7       <Grid.ColumnDefinitions>
8         <ColumnDefinition Width="*" />
9         <ColumnDefinition Width="*" />
10      </Grid.ColumnDefinitions>
11      <Label Text="Event Details" FontAttributes="Italic" FontSize="Medium"  ↩
            TextColor="Olive" HorizontalOptions="StartAndExpand" VerticalOptions=" ↩
            Center" Grid.Row="0" Grid.Column="0" />
12      <Button x:Name="dismissButton" Text="Return" FontSize="Small" TextColor="Red"  ↩
            Clicked="OnDismissButtonClicked" HorizontalOptions="End" VerticalOptions=" ↩
            Center" Grid.Row="0" Grid.Column="1" />
13    </Grid>
14    <Label x:Name="eventdetails" Text="{Binding EventDetails}" FontSize="Medium" />
15  </StackLayout>
```

Listing G.26: Event Detail page markup implemntation

G.2.12 Direction Page

The Direction page is a page where a user can get direction to the nursing home where he/ she signed-up for the service. The page has the option to start navigation to the location with on only a tapped button event. The application accesses the device native map app and populates automatically the address for navigation.

The initial step is to install the Xamarin.Forms.Maps Nuget package in all projects. In the C# file code behind file, an ObservableCollection called location is instantiated, this will notify if any changes in the database and a call to the nCareManger class to retrieve the nursing home address of the signed in user. The two methods implemented here are *OnAppearing* and *OnNavigateButtonClicked*. The first one will be executed automatically when the page appears on user interface and it binds with the markup file the nursing home address and name proprieties on BindingContext. Listing G.27 shows the code behind file implementation details.

```
1  public partial class Directions: ContentPage {
2    IList < Location > location = new ObservableCollection < Location > ();
3    ncareManager manager = ncareManager.DefaultManager;
4
5    public Directions() {
6     InitializeComponent();
7    }
8    protected async override void OnAppearing() {
9     base.OnAppearing();
10    this.IsBusy = true;
11    try {
12     location = await manager.GetLocationsAsync();
13     inputEntry.BindingContext = location.ElementAt(0).Address;
14     inputTitle.BindingContext = location.ElementAt(0).LocationName;
15    } finally {
16     this.IsBusy = false;
17    }}}
```

Listing G.27: Location page C# file implementation

The second methods are executing when user tap the navigation button from the markup file. This method is grabbing the text input from the markup which was binded by the first method and starts device's default map app and app automatically the navigation address. This is implementation of the Xamarin.Forms.Maps nuget package specific to each platform with a switch statement as shown in Listing G.28.

```
void OnNavigateButtonClicked(object sender, EventArgs e) {
  if (!string.IsNullOrWhiteSpace(inputEntry.Text)) {
   var add = inputEntry.Text;
   switch (Device.OS) {
    case TargetPlatform.iOS:
     Device.OpenUri(
      new Uri(string.Format("http://maps.apple.com/?q={0}",
       WebUtility.UrlEncode(add))));
     break;
    case TargetPlatform.Android:
     Device.OpenUri(
      new Uri(string.Format("geo:0,0?q={0}",
       WebUtility.UrlEncode(add))));
     break;
   }
```

Listing G.28: Location page Navigation method implemetation

The markup includes a Grid View with a navigation button and two labels that are defining the binding path for the location object. Each label binds an object propriety value, e.g. the *inputTitle* for the nursing home name and *inputEntry* for the address as shown in Listing G.29.

```
<StackLayout Orientation="Vertical" VerticalOptions="FillAndExpand" Padding="40">
  <Grid>
   <Grid.RowDefinitions>
    <RowDefinition Height="*" />
    <RowDefinition Height="*" />
  <RowDefinition Height="*" />
    <RowDefinition Height="2*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="Auto" />
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <Label x:Name="inputTitle" Text="{Binding }" FontAttributes="Italic" FontSize="
        Medium" TextColor="Olive" HorizontalOptions="Center" VerticalOptions="Center
        " Grid.Row="0" Grid.Column="0" />
    <Label Text="Address" TextColor="Olive" HorizontalOptions="StartAndExpand"
        VerticalOptions="Center" Grid.Row="1" Grid.Column="0" FontSize="Medium" />
      <Label x:Name="inputEntry" Text="{Binding }" FontSize="Medium"
          HorizontalOptions="StartAndExpand" VerticalOptions="Center" Grid.Row="2"
          Grid.Column="0" />
    <Button Text="Navigate" Clicked="OnNavigateButtonClicked" TextColor="Red"
        FontSize="Small" VerticalOptions="Center" Grid.Row="3" Grid.Column="0" />
    </Grid>
    </StackLayout>
```

Listing G.29: Location page markup implementation

G.2.13 Logout Page

Logout page markup has a simple implementation, a single button called Logout, which has a *Clicked* event which is tied up with *OnLogoutButtonClicked* method from code behind file as shown in Listing G.30.

```
1 <StackLayout VerticalOptions="Center" Padding="40">
2     <Button x:Name="LogOut" Text="Logout" BackgroundColor="Red" TextColor="White"  ↩
          Clicked="OnLogoutButtonClicked" FontSize="Medium" />
3   </StackLayout>
```

Listing G.30: Logout page markup implementation

When a user taps the Logout button, the markup file has attached an event which in the code behind C# file the *OnLogoutButtonClicked* method gets executed and logs out the user and redirects to the Login page. This method calls the Authenticator class which is part Microsoft.Azure.Mobile.Client SDK. This class implements platform specific method called *LogoutAsync* as shown in Listing G.31.

```
1  public partial class LogOutPage: ContentPage {
2   bool CanPushPage = true;
3   public LogOutPage() {
4    InitializeComponent();
5    LogOut.Clicked += OnLogoutButtonClicked;
6   }
7  async void OnLogoutButtonClicked(object sender, EventArgs e) {
8    bool loggedOut = false;
9    if (App.Authenticator != null) {
10    loggedOut = await App.Authenticator.LogoutAsync();
11   }
12   if (loggedOut && CanPushPage) {
13    CanPushPage = false;
14    await Navigation.PushAsync(new LogingPage());
15    CanPushPage = true;
16   }
17   }
18   }
```

Listing G.31: Logout page C# file implementation

For Android the implementation is in Main Activity class from nCare.Droid project as shown in Listing G.32.

```
public async Task < bool > LogoutAsync() {
 bool success = false;
 try {
  if (user != null) {
   CookieManager.Instance.RemoveAllCookie();
   await ncareManager.DefaultManager.CurrentClient.LogoutAsync();
  }
  user = null;
  success = true;
 } catch (Exception ex) {
  CreateAndShowDialog(ex.Message, "Logout failed");
 }
 return success;
}
```

Listing G.32: Logout platform specific Android implementation

And for IOS is in AppDelegate class from nCare.IOS project as shown in Listing G.33.

```
public async Task < bool > LogoutAsync() {
 bool success = false;
 try {
  if (user != null) {
   foreach(var cookie in NSHttpCookieStorage.SharedStorage.Cookies) {
    NSHttpCookieStorage.SharedStorage.DeleteCookie(cookie);
   }
   await ncareManager.DefaultManager.CurrentClient.LogoutAsync();
  }
  user = null;
  success = true;
 } catch (Exception ex) {
  var logoutAlert = new UIAlertView("Logout failed", ex.Message,
   null, "OK", null);
  logoutAlert.Show();
 }
 return success;
}
```

Listing G.33: Logout platform specific IOS implementation

Appendix H

Apache Cordova Implementation

H.1 Setting-up the development environment

The Apache Cordova Development IDE is Visual Studio and it is already installed in Chapter 6.3.1. Application can be compiled with Visual Studio Ripple browser at the start of the development, whoever for advance debugging and testing Visual Studio requires a remote connection with a Mac computer in order to compile an IOS application with a Mac Simulators or with real devices.

In order to establish a remote connection between Visual Studio and a Mac computer, a numbers of steps are required to be completed and are the following:

- Install Xcode from App Store.
- Install Node.js on the Mac computer.
- Install Remotebuil server in Mac Terminal (Taco, 2016).
- Start Remotebuild with the command *"remotebuild certificates generate"* and generate a pin number as shown in Listing 6.46.

```
1  iMac:~ipal$ remotebuild certificate generate
2  Enable remote iOS processing: True
3  Host: iMac.local
4  Port: 3000
5  Secure mode: True
6  Security PIN: 926299
7
8  Alternately to use the taco-cli run 'taco remote add <PLATFORM>' to configure ↩
       the taco CLI to use this build server for the chosen platform, specifying
9  Host: iMac.local
10 Port: 3000
11 PIN: 926299
12 The security PIN is for one-time use and expires in 10 minutes. To generate ↩
       additional PINs, use the following command:
13 remotebuild certificates generate
```

Listing H.1: IOS Remotebuild pin generate

- In Mac terminal restart the Remotebuild server with the *"remotebuil"* command. Now the server is listening in port 3000.
- Go to Visual Studio go to Tools / Options / Tools for Apache Cordova and add the Mac computer's profile name and the pin number from the Mac Remote build session. By default, the communication port is 3000.

H.1.1 nCare Apache Cordova initial project

In Visual Studio go the File and select a new JavaScript Blank App (Apache Cordova) project called nCare.Cordova and compile with Ripple or with Remotebuild from Mac. This will install all default JavaScript libraries.

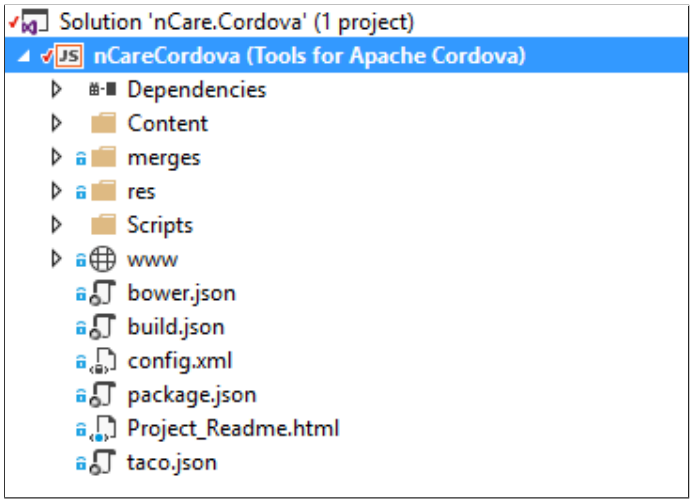


Figure H.1: nCareCordova initial project architecture

The working directory of the project is "www" folder. This folder contains three sub-folders for CSS, images and scripts and a single HTML file called index. This is the only HTML file for application's markup, and index.js from Scrips sub-folder is the working JavaScript file in this implementation.

The responsiveness of the application is developed with jQuery Mobile. This is a touch-optimized mobile framework dedicated to build Apache Cordova/ PhoneGap applications. jQuery mobile requires first to have installed jQuery. jQuery and jQuery Mobile can be added in the project as a Nuget package, which requires after installation to be moved into the "www" directory. By default, Visual Studio provides a wrong location of the installation. Next step is to add jQuery and jQuery Mobile framework reference in the index.html as shown in Listing 6.47.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta name="format-detection" content="telephone=no">
```

```
5         <meta name="msapplication-tap-highlight" content="no">
6         <meta name="viewport" content="user-scalable=no, initial-scale=1, maximum- ↩
              scale=1, minimum-scale=1, width=device-width">
7         <link href="css/jquery.mobile-1.4.5.min.css" rel="stylesheet" />
8         <link href="css/jquery.mobile.theme-1.4.5.min.css" rel="stylesheet" />
9         <link href="css/myCSS.css" rel="stylesheet" />
10        <title>nCARECordova</title>
11    </head>
12 <body>
13
14    <script type="text/javascript" src="cordova.js"></script>
15    <script src="scripts/jquery-2.2.4.min.js"></script>
16    <script src="scripts/jquery.mobile-1.4.5.min.js"></script>
17    <script type="text/javascript" src="scripts/index.js"></script>
18 </body>
19 </html>
```

Listing H.2: jQuery and jQuery Mobile reference in index.html file

H.2 nCare Apache Cordova UI implementation

The first step is to implement the markup for the UI and then to add javaScript support for for each application page. UI is implemented with jQuery Mobile framework. In this framework a user interface page is defined by a *div* with a *data-role="page"* attribute and *class="ui-page"* for CSS. Application's pages are defined in Chapter 5 and are according to the Use Case Diagrams from Chapter 4.5 and are the following pages:

- Login
- Message
- Message Detail
- Event
- Event Detail
- Direction
- Logout

The initially implementation, is to add all the above pages in the index.html file and implement page navigations. Navigation is implemented in each page header, by a jQuery div's attribute *data-role="navbar"* and with a list of hyperlinks. Listing 6.46 shows navigation implemented in the Message page. Similar implementation is done for each other pages with the only difference in the CSS class which has the *ui-active-btn* added specific to each active page. Full implementation details are available in Appendix A.

```
1 <---Message page --->
2 <div data-role="page" id="messagespage" class="ui-page">
3    <div data-role="header">
4       <h3>nCare</h3>
5       <div data-role="navbar">
6          <ul>
```

```
<li><a href="#messagespage" class="ui-btn-active ui-state-persist">Messages</a></li>
        <li><a href="#eventspage">Events</a></li>
          <li><a href="#directionspage">Directions</a></li>
        <li><a href="#logoutpage">Logout</a></li>
        </ul>
      </div>
   </div>
</div>
```

Listing H.3: Message page navigation

H.2.1 Login page

In the Login page body add a button and an "ul" tag for errors display with ids that can be identified from javaScript file as shown in Listing 6.49.

```
<div data-role="page" id="loginpage" class="ui-page">
     <!--header-->\\
      <div>...</div> <!--body content -->
      <div data-role="content">
           <h2> Welcome to nCare!</h2>
           <p>Please optain your Loging details from your related Nursing Home!</p>
       </div><div>
        <button class="bt" id="loginbtn">Login</button>
           <a href="#messagespage" style="display:none" id="log"></a>
       <ul id='errorlog'></ul>
       </div>
   </div>
```

Listing H.4: Login page markup implementation

H.2.2 Message page

The Message page includes a button for refresh page, a loading, "ul" tag for error display and a jQuery *listview* element to display the messages in a list. All elemnets include ids that can be identified from javaScript file as shown in Listing 6.50.

```
<div data-role="page" id="messagespage" class="ui-page">
     < !--header-->
      <div>...</div>
      <!--page content -->
      <div> <fieldset class="ui-grid-a"> <div class="ui-block-a">
                  <h3 data-inline="true">Latest Messages</h3>
              </div> <div class="ui-block-b ">
  <a href="\#" data-role="button" data-inline="true" class="floatright fresh"
    id="refresh">Refresh</a>
```

```
10              </div> </fieldset> </div> <div data-role="content">
11          <div id="loading" class="loading">
12            <img src="images/loading.gif" alt="loaging" width="35px;" /></div>
13          <ul data-role="listview" id="mesg_out"></ul>
14        <p id='errorlog'></p> </div> </div>
```

Listing H.5: Message page markup implementation

H.2.3 Message Detail page

This page gets displayed only when a user tops a message from the list from the Message page. It includes a link back to the Message page and jQuery *listview* element to display the content details. Listening 6.51 shows the implementation including all elements ids for javaScript identifications.

```
1  <div data-role="page" id="mesgdetails" class="ui-page">
2          <div data-role="content">
3        <fieldset class="ui-grid-a">
4            <div class="ui-block-a"><h3>Details</h3></div>
5                <div class="ui-block-b">
6      <a href="#messagespage" data-role="button" data-inline="true"
7      class="floatright fresh">Return</a>
8                </div>
9       </fieldset>
10           <div data-role="content">
11               <ul data-role="listview" id="details-data" data-theme="a"></ul>
12           </div></div></div>
```

Listing H.6: Message Details page markup implementation

H.2.4 Event page

Event page has a similar implementation with the Message page. A button to refresh the page, a loading image div, a jQuery list and "p" tag for errors display as shown in Listing 6.51.

```
1  <div data-role="page" id="eventspage" class="ui-page">
2          < !--header-->
3       <div>...</div>
4       <!--page content -->
5        <div><fieldset class="ui-grid-a">
6                <div class="ui-block-a">
7                    <h3 data-inline="true">Latest Events!</h3> </div>
8                <div class="ui-block-b">
9          <a href="#" data-role="button" id="ev_refresh"
10       data-inline="true class="floatright fresh">Refresh</a>
11               </div></fieldset> </div>
12       <div data-role="content">
```

```
13          <div id="eventloading" class="loading">
14                <img src="images/loading.gif" alt="loaging" width="35px;" /></div>
15          <ul data-role="listview" id="events_out"></ul>
16          <p id='errorlog'></p>
17          </div> </div>
```

Listing H.7: Event page markup implementation

H.2.5 Event Detail page

The Event Detail page includes a jQuery *listview* element to display the event details and a return button to the Event page as shown in Listing 6.52.

```
1  <div data-role="page" id="eventdetails" class="ui-page">
2        <div data-theme="a" data-role="header">
3            <h3>Details</h3></div>
4        <div data-role="content">
5            <fieldset class="ui-grid-a">
6                <div class="ui-block-a">
7                    <h3>Details</h3></div>
8                <div class="ui-block-b">
9            <a href="#eventspage" data-role="button"
10           class="floatright fresh" data-theme="a">Return</a>
11           </div> </fieldset><div data-role="content">
12       <ul data-role="listview" id="evnt-detail" data-theme="a"></ul>
13           </div></div> </div>
```

Listing H.8: Event Details page markup implementation

H.2.6 Direction page

The Direction page markup includes a loading page image div followed-up with a div for nursing home details display i.e nursing home name and address, a button for navigation and and a "p" tag for errors display as shown in Listing 6.53.

```
1  <div data-role="page" id="directionspage" class="ui-page">
2          < !--header-->
3       <div>...</div>
4       <!--page content -->
5        <div data-role="content">
6            <div id="loc\_loading" class="loading ui-page ui-page">
7                <img src="images/loading.gif" alt="loaging" width="35px;" />
8            </div>
9             <div id="map"></div>
10           <div>
11    <a href="#" data-role="button" class="btdir" id="lnav" alt="">Navigate</a>
```

```
12          </div>
13           <p id='errorlog'></p>
14       </div>
15   </div>
```

Listing H.9: Location page markup implementation

H.2.7 Logout page

This page includes button to logout and navigate back to the Login page. The button has an id that can be identified from javaScript file as shown in Listing 6.54.

```
1  <div data-role="page" id="logoutpage">
2          < !--header-->
3       <div>...</div>
4       <!--page content -->
5        <div>
6    <a href="#loginpage" class="lout" data-role="button" id="logoutbtn">Logout</a>
7           <ul id='errorlog'></ul>
8       </div> </div>
```

Listing H.10: Logout page markup implementation

H.2.8 Authentication and Web Server connection

For the developing and testing, authentication will be implemented with Google only. The first step is to add in index.html file a meta tag two URLs, i.e. "data: gap: https://accounts.google.com" for Google authentications and "https://ncazuremobile.azurewebsites.net" for Azure Webeg Server as shown in Listing 6.48.

```
1       <head>
2        .......
3        <meta http-equiv="Content-Security-Policy" content="default-src 'self' data: ↩
             gap: https://accounts.google.com https://ncazuremobile.azurewebsites.net ↩
             ; style-src 'self'; media-src *">
4       ......
5    </head>
```

Listing H.11: Apace Cordova index.html reference to Google and Azure

The second step is to to establish a connection with the web server (nCareBackend) hosted on Azure Mobile App, authenticate the user with Google and retrieve the tables from database. This can be implemented in few sub-steps.

In index.js file add an anonymous function with strict javaScript implementation, which is a self-execution function at application start-up. Inside the function add a javaScript event listener with

onDeviceReady function which will be executed on any user's interface action. Listing 6.49 show the anonymous function implementation.

```
1     (function(){
2      "use strict";
3    document.addEventListener('deviceready', onDeviceReady, false);
4    function onDeviceReady(){ }
5
6     })();
```

Listing H.12: Apace Cordova index.js initial configuration

Create a function to create a connection to the web server on Azure Mobile App and connect to the database tables after user's successfully authentication, as shown in Listing 6.50.

```
1  (function() {
2     var client, messageTable eventTable,locationTable;
3   function onDeviceReady(){ }
4
5      function login()
6       {
7        client = new WindowsAzure.MobileServiceClient('https://ncazuremobile. ↩
             azurewebsites.net');
8        client.login('google').then(function ref(success) {
9            messageTable = client.getTable('message');
10           eventTable = client.getTable('event');
11           locationTable = client.getTable('location');
12       }, handleError);
13       }})();
```

Listing H.13: Apace Cordova Login function implementation

Implement the exception handles function, this function handles any errors that may arise in the application as shown in Listing 6.51.

```
1     function handleError(error) {
2        var text = error + (error.request ? ' - ' + error.request.status : '');
3        console.error(text);
4        $('#errorlog').append($('<li>').text(text));
5    }
```

Listing H.14: Error handle function implementation

Add a div to display the errors information in each html page as shown in Listing 6.52.

```
1   <div><ul id='errorlog'></ul><div>
```

Listing H.15: Display Errors implementation in UI

javaScript implementation for the User Login page

Add a javaScript function to trigger the button event from the Login page. This is implemented with jQuery and is located inside *onDeviceReady* function as shown in Listing 6.63. This function is the start-up process for login the user and retrieve and display the data according to his profile.

```
1  function onDeviceReady() {
2        $("\#loginbtn").click(function () {
3           login();
4        });}
```

Listing H.16: Login button event function implementation

H.3 Read and display tables data in user interface

Add functions to read the tables as shown in Listing 6.62. This will ensure that tables data is displayed only if user has authenticated successfully.

```
1      function login() {
2        client = new WindowsAzure.MobileServiceClient('https://ncazuremobile. ↩
             azurewebsites.net');
3        client.login('google').then(function ref (success) {
4             messageTable = client.getTable('message');
5            eventTable = client.getTable('event');
6            locationTable = client.getTable('location');
7           messgTable();
8           eventsTable();
9           locTable();
10          GetMainPage();
11       }, handleError); }
```

Listing H.17: Add javaScript functions to read database tables when user has logged in successfully

Implement all javaScript functions which were added in Listing 6.62 to read the tables data as shown in Listing 6.63.

```
1   //Read Message table data object
2   function messgTable() {
3   updateSummaryMessage();
4   messageTable .read() .then(messageList, handleError);
5  }
6  //Read Event table data object
7  function eventsTable() {
8   updateSummaryEventMessage();
9   eventTable.read().then(eventList, handleError);
10 }
```

```
11 //Read Direction table data object
12 function locTable() {
13   updateSummaryLocMessage();
14   locationTable .read().then(locationList, handleError)
15 }
```

Listing H.18: Read tables data functions implementation

H.3.1 Parse and display tables data in the HTML pages

H.3.2 Display data in Message page

Each message from the message list is associated with full message details which is implemented in the Message Detail page and gets triggered on the user's tap event. The first step is to add support for the implementation of the Message Detail page by adding a reference in each message when iterating the message object for parsing. This is represented by the jQuery *"data-id"* attribute. In this way, the Message Detail page knows what message object to parse and display the detail value.

The second step in displaying the data process in the Message page, is to create an empty string and then add a function to iterate trough the object and add the object values in the string including the the *"data-id"* attribute which is implemented as a hyperlink, followed by displaying the string in the Message page and hide the loading page icon. Full implementation is shown in listing 6.64.

```
1   //object variable required in the Message Detail page
2  var detailInfo = { id: null, result: null}
3     //loading page icon
4  function updateSummaryMessage() {
5         $('#loading').show();}
6     //parsing the Message object
7  function messageList(result){
8         detailInfo.result = result;
9         var out = '';
10        $.each(result, function(i, row){
11            out += ('<li><a href="" data-id ="' + result[i].id + '">' + result[i]. ↩
                  title + '</a>');
12           out += ('<p>' + result[i].updatedAt.toString().replace(/GMT.*/g, "") + '</ ↩
                 p></li>');
13          $("#mesg_out").listview("refresh");});
14        //display data in the Message page html
15        document.getElementById('mesg_out').innerHTML = out;
16          //hide the loading page icon
17        $('#loading').hide();
18        }
```

Listing H.19: Display Message table data in the Message page

H.3.3 Display data in Message Details page

The first step is to add a function which gets triggered on message a tap event from the Message page. It reads the tapped message object and updates the *"detailInfo.id"* object declared in Listing 6.64 with the object. Using jQuery, this function is changing the Message page to the Message Detail page, which is represented in the html by the *"mesgdetails"* id.
The second step is to add a function with an event listener for the page change event and parse the object message and display the details in the Message Details page html. This is implemented by jQuery *"pagebeforeshow"* event function, which takes in two arguments, the event object and the data object. Listing 6.65 shows the full function implementation.

```
1  //this function is trigger on message tap event
2  $(document).on('vclick', '#mesg_out li a', function () {
3        detailInfo.id = $(this).attr('data-id');
4        $.mobile.changePage("#mesgdetails", { transition: "slide", changeHash: false
             }); });
5    //Listerns on page change and parse the object
6     $(document).on('pagebeforeshow', '#mesgdetails', function () {
7        var outdetails = '';
8        $('#details-data').empty();
9       $.each(detailInfo.result, function (i, row) {
10           if (row.id === detailInfo.id) {
11           outdetails += '<p>' + row.details + '</p>';
12           outdetails += '<h3>' + row.writtenBy + '</h3>'; }});
13      //display data in Message Detail page
14        document.getElementById('details-data').innerHTML = outdetails; });
```

Listing H.20: Display Messages detail data in the Message page

H.3.4 Display data in Event page

This process has similar implementation as in displaying the data in the Message page. Starting by adding support for the Event detail page, followed-up by parsing the object and display the list of messages in the Message page and hiding the loading page icon. Full implementation is showing in Listing 6.66.

```
1  //loading page icon
2  function updateSummaryEventMessage() {
3        $('#eventloading').show(); }
4    //object variable required in the Event Detail page
5   var eventInfo = { id: null, res: null}
6    //parsing the Event object
7     function eventList(evn) {
8        eventInfo.res = evn; var ev_out = '';
9        $.each(evn, function (i, row) {
10           ev_out += ('<li><a href="" dt-id="' + evn[i].id + '">' + evn[i].
                 eventTitle + '</a>');
```

```
11          ev_out += ('<p>' + evn[i].updatedAt.toString().replace(/GMT.*/g, "") + ' ↩
                </p></li>');
12          $("#events_out").listview().listview("refresh");});
13          //display data in the Event page html
14      document.getElementById('events_out').innerHTML = ev_out;
15       //hide the loading page icon
16      $('#eventloading').hide();
```

Listing H.21: Display Event table data in the Event page

H.3.5 Display data in Event Detail page

This process has similar implementation as in displaying the data in the Message Details page. Staring by adding a function to get trigger on the message tap event and then add other function to parse and display the event details in the Event Details page. Full implementation is showing in Listing 6.67.

```
1  //this function is trigger on a event tap
2     $(document).on('vclick', '#events_out li a', function () {
3         eventInfo.id = $(this).attr('dt-id');
4         $.mobile.changePage("#eventdetails", { transition: "slide", changeHash: false ↩
              }); });
5      //Listerns on page change and parse the object
6     $(document).on('pagebeforeshow', '#eventdetails', function () {
7        var outdet = '';
8         $.each(eventInfo.res, function (index, value) {
9             if (value.id === eventInfo.id) {
10          outdet += '<p>' + value.eventDetails + '</p>'; }
11       });
12        //display data in the Event Detail page
13        document.getElementById('evnt-detail').innerHTML = outdet;
14    });
```

Listing H.22: Display Event details in the Event Detail page

H.3.6 Direction page map implementation

The first step to install a Map plugin in the project. There are many plugins available, however Lee Crossley (2016) implemented a pluging with MIT licence and easy implementations. To add this plugin, open the config.xml file form the project solution, go to Plugins navigation tab then Custom Git and add in the Github URL.

The second step is to create a function called *locationList* to retrieve the nursing home name and address from the location table and display them in the Direction page HTML. Followed by other function called *getDirection* which gets trigger on the button tap event from the Direction page.

This function reads the name and the address displayed earlier in the HTML and calls the map plugin by passing these details for navigation. Full implementation is shown in Listing 6.68.

```
1  //display nursing home name and address in html
2     function locationList(lc) {
3         $.each(lc, function (i, v) {
4               l_out += '<h3>' + v.locationName + '</h3>' + '<p id="laddress">' + v. ↩
                    address + '</p>'; });
5            document.getElementById('map').innerHTML = l_out;
6            $('#loc_loading').hide(); }
7       //implement map plugin
8          function getDirection() {
9         var textaddress = document.getElementById('laddress').innerHTML;
10         directions.navigateToAddress(textaddress); }
```

Listing H.23: Display location table data in the Direction page and implement Map navigation

H.3.7 Logout implementation

This is implemented by adding jQuery event listener in the *onDeveiceReady* for the button click event from the Direction page HTML. This function which will trigger the logout function and clear the page cookies as shown in Listing 6.69.

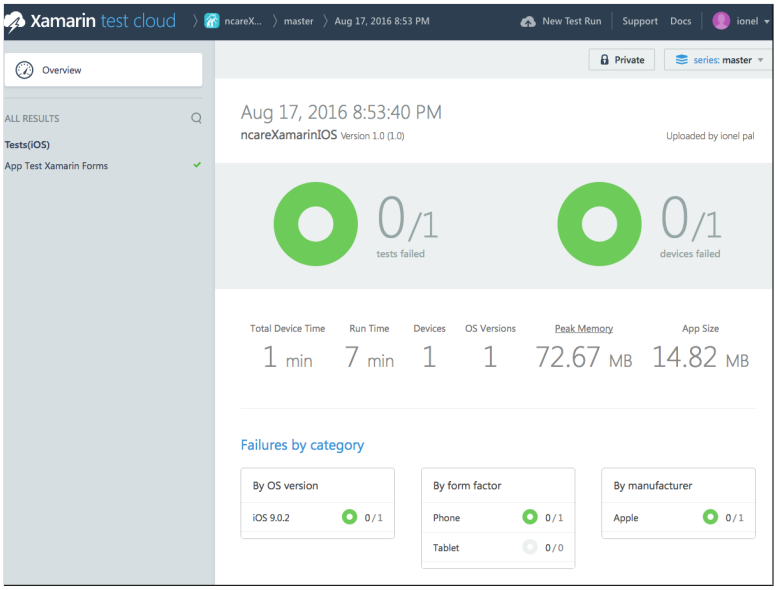
```
1  //logout function trigger
2  function onDeviceReady() {
3  ............
4 $("#logoutbtn").click(function () {
5           logOut()
6    });
7 //clear page cookies
8   function logOut() {
9        window.cookies.clear(function () {
10           console.log('Cookies cleared!');
11       });}
```

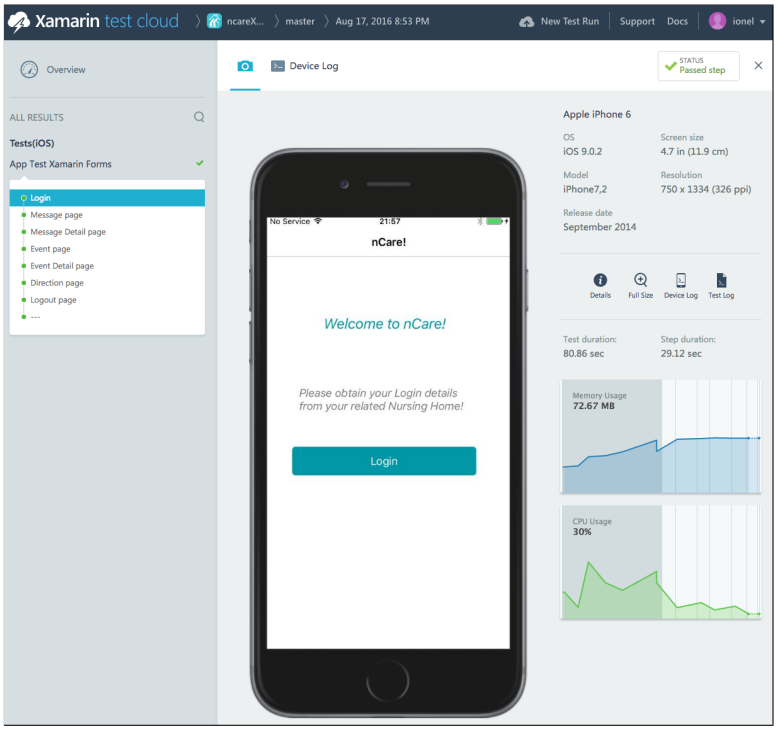
Listing H.24: Logout function implementation

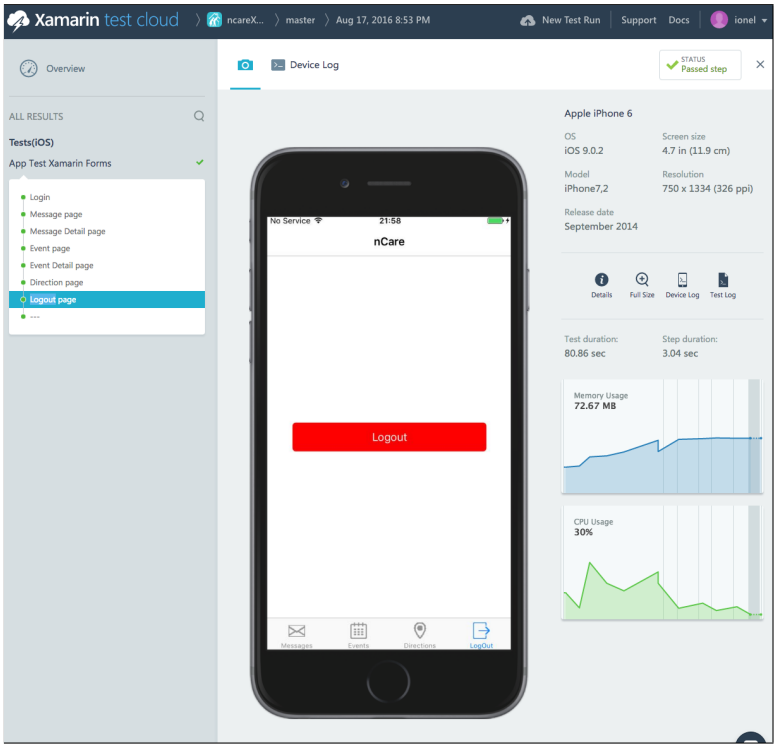
Appendix I

Test Results

I.1 Xamarin Test Cloud result example







I.2 Test results summary by mobile device

Samsung Galaxy S4 (Android 5.0.1)

Xamarin

	Overview		App Loading			Page Navigation		
	Peak Memory (MB)	App Size (MB)	Step Duration (sec)	Memory usage (MB)	CPU (%)	Step Duration (sec)	Memory Usage (MB)	CPU (%)
	232.49	20.21	24.37	232.49	110	5.89	184.63	21
	225.81	20.21	26.5	225.81	100	5.88	187.25	27
	215.24	20.21	22.87	215.24	110	5.71	181.99	27
	216.11	20.21	24.97	216.74	96	6	181.87	22
	219.64	20.21	25.06	219.64	100	5.79	179.67	32
	240.29	20.21	23.22	240.29	110	5.88	182.03	30
	230.26	20.21	24.58	230.26	91	5.2	182.68	29
Mean=	225.69	20.21	24.51	225.78	102.43	5.76	182.87	26.86
Stand Dev=	8.60	0.00	1.12	8.50	7.13	0.24	2.24	3.76

Samsung Galaxy S4 (Android 5.0.1)

Cordova

	Overview		App Loading			Page Navigation		
	Peak Memory (MB)	App Size (MB)	Step Duration (sec)	Memory usage (MB)	CPU (%)	Step Duration (sec)	Memory Usage (MB)	CPU (%)
	238.57	3.93	20.96	238.57	64	6.38	194.55	57
	201.84	3.93	20.44	201.84	66	5.89	200.88	38
	196.61	3.93	20.13	198.61	62	6.47	192.29	22
	201.55	3.93	19.92	201.55	65	6.13	201.27	38
	226.16	3.93	18.42	226.16	54	6.12	202.13	53
	240.64	3.93	21.87	240.64	84	7.03	200.15	75
	249.98	3.93	20.01	249.98	59	5.55	201.92	59
Mean=	222.19	3.93	20.25	222.48	64.86	6.22	199.03	48.86
Stand Dev=	20.32	0.00	0.98	19.97	8.69	0.43	3.65	16.15

Average Results

	Overview		App Loading			Page Navigation		
	Peak Memory (MB)	App Size (MB)	Step Duration (sec)	Memory usage (MB)	CPU (%)	Step Duration (sec)	Memory Usage (MB)	CPU (%)
Xamarin	225.7	20.2	24.5	225.8	102	5.8	182.9	27
Cordova	222.2	3.9	20.3	222.5	65	6.2	199.0	49
Xamarin Stand Dev	8.6	0.0	1.1	8.5	7.1	0.2	2.2	3.8
Cordova Stand Dev	20.3	0.0	1.0	20.0	8.7	0.4	3.6	16.2

Samsung Galaxy S6 (Android 5.1.1)

Xamarin								
	Sumary		App Loading			Page Navigation		
	Peak Memory (MB)	App Size (MB)	Step Duration (sec)	Memory usage (MB)	CPU (%)	Step Duration (sec)	Memory Usage (MB)	CPU (%)
	281.13	20.21	24.39	281.13	100	9.29	247.25	27
	277.27	20.21	21.95	277.27	110	9.42	244.54	25
	280.39	20.21	22.77	280.39	85	9.86	247.47	29
	277.89	20.11	28.46	277.89	86	10.77	248.48	26
	282.59	20.21	24.05	282.59	83	10.56	245.65	50
	267.23	20.11	24.08	267.23	100	9.54	246.32	29
	295.13	20.11	28.45	295.13	83	11.06	245.65	5.2
Mean=	280.23	20.17	24.88	280.23	92.43	10.07	246.48	27.31
Stand Dev=	7.67	0.05	2.40	7.67	9.98	0.66	1.24	12.05

Samsung Galaxy S6 (Android 5.1.1)

Cordova								
	Overview		App Loading			Page Navigation		
	Peak Memory (MB)	App Size (MB)	Step Duration (sec)	Memory usage (MB)	CPU (%)	Step Duration (sec)	Memory Usage (MB)	CPU (%)
	331.18	3.93	20.79	331.18	97	10.38	269.73	17
	340.5	3.93	21.19	340.5	110	10.53	280.13	3.9
	344.99	3.93	28.07	344.99	52	10.45	281.41	23
	326.71	3.93	24.73	326.71	58	10.61	277.42	58
	355.64	3.93	19.87	355.64	39	9.64	272.06	39
	330.33	3.93	21.33	330.33	46	10.62	280.24	20
	355.19	3.93	19.88	355.19	56	9.6	274.62	19
Mean=	340.65	3.93	22.27	340.65	65.43	10.26	276.52	25.70
Stand Dev	10.99	0.00	2.81	10.99	25.02	0.41	4.16	16.28

Average Results								
	Overview		App Loading			Page Navigation		
	Peak Memory (MB)	App Size (MB)	Step Duration (sec)	Memory usage (MB)	CPU (%)	Step Duration (sec)	Memory Usage (MB)	CPU (%)
Xamarin	280.2	20.2	24.9	280.2	92	10.1	246.5	27
Cordova	340.6	3.9	22.3	340.6	65	10.3	276.5	26
Xamarin Stand Dev	7.7	0.0	2.4	7.7	10.0	0.7	1.2	12.1
Cordova Stand Dev	11.0	0.0	2.8	11.0	25.0	0.4	4.2	16.3

IPhone 5S (IOS 9.2)

Xamarin

	Overview		App Loading			Page Navigation		
	Peak Memory (MB)	App Size (MB)	Step Duration (sec)	Memory usage (MB)	CPU (%)	Step Duration (sec)	Memory Usage (MB)	CPU (%)
	73.87	14.82	27.2	73.87	30	3.58	73.87	5.8
	72.81	14.82	27.57	72.81	32	3.53	72.49	4.6
	73.35	14.82	31.16	73.35	29	3.1	72.95	6.4
	76.12	14.82	32.19	76.12	23	3.5	75.73	4.5
	74.62	14.82	30.47	74.62	25	2.89	74.28	4
	76.49	14.82	32.43	76.49	28	2.27	76.39	5.1
	76.3	14.82	32.78	76.3	23	2.8	75.9	3.4
Mean=	74.79	14.82	30.54	74.79	27.14	3.10	74.52	4.83
Stand Dev=	1.40	0.00	2.13	1.40	3.27	0.45	1.41	0.95

IPhone 5S (IOS 9.2)

Cordova

	Overview		App Loading			Page Navigation		
	Peak Memory (MB)	App Size (MB)	Step Duration (sec)	Memory usage (MB)	CPU (%)	Step Duration (sec)	Memory Usage (MB)	CPU (%)
	64.13	4.52	34.52	64.13	30	2.75	63.69	11
	63.64	4.52	31.54	63.64	29	3.66	63.64	9.3
	63.94	4.52	30.74	63.94	27	3.39	63.61	11
	70.47	4.52	31.65	70.47	27	2.95	70.47	9.3
	64.57	4.52	38.96	64.57	27	3.05	64.57	10
	68.52	4.52	27.39	68.52	27	2.85	68.52	11
	67.92	4.52	35.76	67.92	28	3.01	67.45	7.8
Mean=	66.17	4.52	32.94	66.17	27.86	3.09	65.99	9.91
Stand Dev=	2.54	0.00	3.50	2.54	1.12	0.30	2.59	1.12

Average Results

	Overview		App Loading			Page Navigation		
	Peak Memory (MB)	App Size (MB)	Step Duration (sec)	Memory usage (MB)	CPU (%)	Step Duration (sec)	Memory Usage (MB)	CPU (%)
Xamarin	74.8	14.8	30.5	74.8	27	3.1	74.5	5
Cordova	66.2	4.5	32.9	66.2	28	3.1	66.0	10
Xamarin Stand	1.4	0.0	2.1	1.4	3.3	0.4	1.4	1.0
Cordova Stand	2.5	0.0	3.5	2.5	1.1	0.3	2.6	1.1

IPhone 6 Plus(IOS 9.3.4)

Xamarin								
	Overview		App Loading			Page Navigation		
	Peak Memory (MB)	App Size (MB)	Step Duration (sec)	Memory usage (MB)	CPU (%)	Step Duration (sec)	Memory Usage (MB)	CPU (%)
	106.16	14.82	26.19	106.16	18	2.83	104.69	5
	105.61	14.82	26.31	105.61	18	2.72	104.28	2.2
	105.36	14.82	26.71	105.36	17	2.84	104.09	2.4
	106.2	14.82	27.35	106.2	16	2.96	104.94	2.4
	100.63	14.82	26.18	100.63	18	3.12	100.09	4.6
	104.03	14.82	26.84	104.03	16	3.06	103.5	2.7
	105.36	14.82	26.09	105.36	18	2.76	104.02	1.8
Mean=	104.76	14.82	26.52	104.76	17.29	2.90	103.66	3.01
Stand Dev=	1.82	0.00	0.43	1.82	0.88	0.14	1.52	1.16

IPhone 6 Plus(IOS 9.3.4)

Cordova								
	Overview		App Loading			Page Navigation		
	Peak Memory (MB)	App Size (MB)	Step Duration (sec)	Memory usage (MB)	CPU (%)	Step Duration (sec)	Memory Usage (MB)	CPU (%)
	97.75	4.52	27.71	97.75	18	3.17	97.73	9.9
	96.91	4.52	27.92	96.91	18	3.78	96.8	5.6
	97.27	4.52	27.3	97.27	17	2.67	96.56	9.1
	101.16	4.52	32.14	101.16	14	3.64	100.78	7.1
	97.56	4.52	26.8	97.56	15	2.74	96.89	8.8
	97.95	4.52	26.42	97.95	16	2.76	97.5	7.2
	97.19	4.52	25.9	97.19	18	2.68	96.44	8.8
Mean=	97.97	4.52	27.74	97.97	16.57	3.06	97.53	8.07
Stand Dev=	1.34	0.00	1.91	1.34	1.50	0.44	1.40	1.38

Average Results								
	Overview		App Loading			Page Navigation		
	Peak Memory (MB)	App Size (MB)	Step Duration (sec)	Memory usage (MB)	CPU (%)	Step Duration (sec)	Memory Usage (MB)	CPU (%)
Xamarin	104.8	14.8	26.5	104.8	17	2.9	103.7	3
Cordova	98.0	4.5	27.7	98.0	17	3.1	97.5	8
Xamarin Stand Dev	1.8	0.0	0.4	1.8	0.9	0.1	1.5	1.2
Cordova Stand Dev	1.3	0.0	1.9	1.3	1.5	0.4	1.4	1.4