

Project Title: An Investigation of the development of Hybrid Applications for the iPhone Using
Non-Proprietary Development Environments

Ironman

Bsc (Hons) Computing - Web Systems Development

Matric No: S09xxxxx

Honours Project Final Report

Project Supervisor: Richard Foley

“Except where explicitly stated all work in this document is my own”

Signed: _____ Date: _____

Abstract

Due to the relatively recent announcement by Apple to allow application submissions to the App Store that were created using third party development frameworks there have been many new frameworks created to allow users to create iOS applications using standard HTML, CSS and Javascript instead of using Objective-C. This project aimed to investigate if the use of these third party frameworks, specifically QuickConnectiPhone as it is one of the more popular options, to determine if their use can reduce the complexity and length of iOS applications without compromising the applications quality.

Preliminary research into the most commonly implemented functionality of iOS applications was undertaken to determine the best aspects of a typical iOS App to test during the project to illustrate the attributes of the “standard” application. The Apple App Store along with the developer website of the third party frameworks proved to be a helpful recourse when investigating the common App traits.

Two separate Apps were created for the project that shared the same common functionality but were developed in entirely different ways. The first control App was created using objective-C to test how a normally created App performs, the second was created with QuickConnectiPhone to determine the differences between the two approaches in terms of complexity, simplicity and development time.

The results from the development log created during both App developments, the analysis of each App against the Apple submissions guidelines and the output from the instruments benchmark tests of each App positively indicated that the QuickConnect App was functionally the same as the objective-C App while significantly cutting development time and complexity, supporting the projects initial hypothesis that it is beneficial to pursue the use of third party frameworks in iOS development.

Acknowledgements

I would like to take this opportunity to thank my project supervisor Dr Richard Foley for all his help and advice during the course of the project. I would also like to show my appreciation for the feedback I received that guided me towards a higher quality end product than would have otherwise created.

Without his guidance I would have been unable to create such a rigorously developed project to the level at which I did.

Contents

1. Introduction	7
1.1 Background	7
1.2 Project Overview	10
1.2.1 Project Methods	10
1.2.2 Objectives	10
1.2.3 Hypothesis	12
1.3 Report Structure	13
1.3.1 Literature Review	13
1.3.2 Project Methods	13
1.3.3 Design and implementation	13
1.3.4 Results Evaluation	13
1.3.5 Conclusions	13
2. Literature Review	14
2.1 The Benefits and drawbacks of the QCi SDE in comparison to Objective-C	14
2.1.1 The functional differences between QCi and Obj-C	14
2.1.2 The Benefits of using QCi and other environments	15
2.1.3 The Drawbacks of using QCi and other Third Party SDE's	16
2.1.4 The Benefits of using Objective-C	16
2.1.5 The Drawbacks of using Objective-C	16
2.2 Existing Third Party App characteristics	17
2.3 Appropriate development and processor metrics	17
2.4 iOS Security Vulnerabilities	18
2.5 Literature Conclusion	19
3. Project Methods	20
3.1 Research Method	20
3.2 Problem Analysis	20
3.2.1 Existing TKD App functionality analysis	20
3.2.2 Apple iOS Human Computer Interface Guidelines	21
3.2.3 Apple iOS Application Programming Guidelines	22
3.2.4 Apple App Store review Guidelines	22
3.2.5 App Functional and Non Functional Requirements	23
4. Design and Implementation	25
4.1 General design Specification for Project Apps	25
4.1.1 Initial Screen designs	25
4.1.2 Review of Both Apps User interface Needs	26
4.1.2.1 Objective-C	26
4.1.2.2 QuickConnectiPhone	26
4.1.3 Finalised Screen designs	26
4.1.3.1 Objective-C	26

4.1.3.2 QuickConnectiPhone	27
4.2 Objective-C App Development	27
4.2.1 Incremental development phases	28
4.2.2 Testing Plan	29
4.3 QuickConnectiPhone App Development	30
4.3.1 Incremental development phases	30
4.3.2 Testing Plan	31
5. Results Evaluation	32
5.1 Ease of Development	32
5.1.1 Objective-C App development log	32
5.1.1.1 UI Design	32
5.1.1.2 Development	32
5.1.1.3 Conclusions	33
5.1.2 QuickConnectiPhone App development log	33
5.1.2.1 Development	33
5.1.2.2 Conclusions	34
5.2 Adherence to Guidelines	34
5.2.1 Objective-C App	34
5.2.2 QuickConnectiPhone App	35
5.3 Instruments Benchmark tests	36
5.3.1 Objective-C App	37
5.3.2 QuickConnectiPhone App	38
5.3.3 Conclusions	39
6.0 Final Conclusions	40
6.1 Project Recap	40
6.2 Final Conclusions Discussion	40
6.3 Project Limitations and Future Work	41
7.0 References	43
8. Appendices	46
Appendix A - Apple Developer Guidelines Document	46
Appendix B - UI paper screen page designs	48
Appendix C - Interface Builder Element library	49
Appendix D - Final User Interface Screens	50
Appendix E - Objective-C App code printout	52
Appendix F - QuickConnectiPhone App code printout	64
Appendix G - Testing Plans	67
Appendix H - App Development Logs	73
Appendix I - App Instruments Benchmark Test Data	78

Table of Figures

Figure 1 - The distribution of Firmware versions among Android handsets.	7
Figure 2 - iOS MVC application architecture	14
Figure 3 - Application Lifecycle	15
Figure 4 - Main page, home page, map page, patterns page and video playback control paper screen designs	25
Figure 5 - Final Obj-C screen Designs	26
Figure 6 - Final QCi screen designs	27
Figure 7 - Creating the table view elements in Obj-C	28
Figure 8 - Media playback controls in obj-C	29
Figure 9 - obj-C main page test excerpt	29
Figure 10 - index.html page from QCi App	30
Figure 11 - Main javascript page for client functionality	31
Figure 12 - location services permissions	35
Figure 13 - obj-c activity monitor	37
Figure 14 - memory allocation	37
Figure 15 - CPU monitor	38
Figure 16 - Memory Leaks Test	38
Figure 17 - QCi CPU monitor	39
Figure 18- QCi Memory Leaks	39

1. Introduction

The introduction of the Report will lay the groundwork for the following sections, outlining the background of the report, the overview of the methods being employed, the objectives of both the secondary and primary research methods and the initial hypothesis of the project.

1.1 Background

In recent years mobile computing has become increasingly popular among the general public, a view that is supported by Schroeder (2010) who notes this trend especially among teenagers and students who make use of popular social networking sites, such as Facebook or Twitter. Being able to access the internet anywhere and anytime to get in touch with friends, access files and data, play games or watch videos on the move is the core aim of mobile computing. Principally among these mobile devices are the ever growing selection of smartphones which have been on the increase in sales for the past few years according to International Data Corporation (IDC 2010). Smartphones would seem to have revolutionized the way people use their mobiles and changed the way we consume content on the move. As supported by Pash (2010), modern smartphones have much of the functionality of a desktop PC and in many cases are even more “personal computers” than their desktop counterparts, users can play games, read and write e-mails, access websites, stream audio and video from various sources and even create and edit office documents such as word files, spreadsheets and slideshows which can then be sent to PCs to be printed.

The four giants in the smartphone industry are Apple’s iOS, Google’s Android, Nokia’s Symbian and RIM’s Blackberry OS as listed on the IDC (2010) latest quarterly report for profit and unit sales. Globally Nokia maintains the majority of the market but Apple and Google are quickly creating fierce competition and closing the gap on each other as Google catches up to the more powerful Apple. Google and Apple provide a vivid contrast between each other in terms of their approach to software development and marketing strategies, Google aims for open source ideals which allows for a massive array of developers to create apps for their OS, although with open source comes fragmentation and each individual handset bundled with android runs a different version of the OS with its own custom skin depending on the manufacturer of the phone which means that many Apps are not universal among Android devices. This fragmentation of the Android operating system among handsets can be seen in Figure 1 (Android developers 2011). It can be seen that the majority of Android handsets use varying degrees of outdated firmwares as the manufacturers of the hardware take time to perform modifications to the “vanilla” code during which Google will often release new versions before older handsets have even received the old version. Samsung uses its own Touch Wiz UI skin and HTC uses its own HTC Sense, each of which attempts to optimize the user interface and overall experience of using Android but takes varying lengths of time to completely accommodate the new API’s available in each new release.

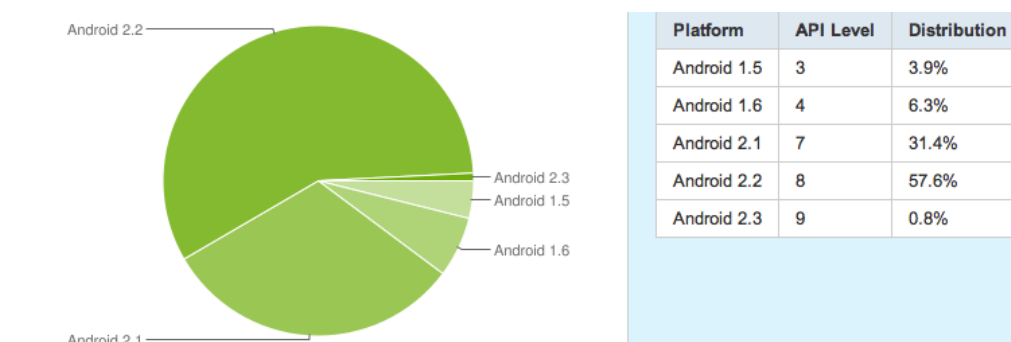


Figure 1 - The distribution of Firmware versions among Android handsets.

Whereas in contrast Apple is very closed in, only offering a single smartphone handset, although also offering the same OS on their iPod Touch and iPad devices, which is built by themselves and dictating what content is allowed on it and how it is created as described in the App store guidelines document (Apple inc 2011). This unity provides a very solid user experience for the average consumer and despite being closed off to outside markets, still has far more registered apps compared to the Google marketplace, standing at over 250,000 registered Apps according to Apple themselves versus Googles 100,000 according to a report by PCWorld (2010).

The iPhone OS, now iOS, a rebranding due to the increase in Apples devices running the OS such as the iPod Touch and iPad was first created in 2007 along with the first iPhone and with it came the App store where third party developers could create software for iOS devices, or iDevices for customers to download onto their handsets. In the past three years the Apple iPhone has become an icon of the mobile device world and have sold over 41 million handsets worldwide from June 2007 to December 2010 (Number Of 2010), Etherington (2010) shows that it is currently dominating the market and setting trends in the mobile device field. In the first quarter of 2010 the iPhone supplied over 40% of Apples revenue, selling 8.75 million units and growing Apples market share of mobile subscribers up to 25.1% according to numbers cited from Arthur (2010). The continuing success of the handset is largely due to its meticulously thought out and well designed user interface and a strict set of development guidelines which every App developer must adhere to as indicated in the Apple inc (2011) iPhone Human Interface Guidelines in order to have their App on the store at all. There are over one hundred points that each App must not violate in order to be accepted onto the iPhone, guidelines such as: "Apps that alter the functions of standard switches, such as the Volume Up/Down and Ring/Silent switches, will be rejected" (Apple inc 2011). The success of the iPhone is largely attributed to this meticulous screening process of all content that could ever be loaded onto the devices to ensure one hundred percent compatibility and prevent any fraudulent or harmful material from infecting the device.

iOS Apps can cover all manner of content from turn by turn navigation aids from manufacturers the TomTom GPS to word processing environments and powerful portable games from companies such as Sega and Activision, all of which can be viewed on the Apple App store (Apple inc 2011) directly. This diversity can offer up many opportunities to a host of potential developers who may wish to create a new mobile version of their software for iOS or even create a standalone App that does something entirely new. What has somewhat hindered the number of iOS developers other than the strict guidelines of the submission process is the requirement for all Apps to be written in Apple's own proprietary language Objective-C, which isn't a problem for many developers who choose to develop for iOS, but could possibly stop other developers from migrating to the platform, especially if they are already embedded in another OS development.

Now that it is no longer necessary to use objective-C in iOS development as of September 2010 (Shankland 2010), developers can now use any other suitable language to code in as reported by Krill (2010) and still have the App accepted by Apple. This has even led to Adobe, who have recently been in conflict with Apple over the use of its Flash media format have announced they will resume support for App development for the iPhone as reported by Shankland (2010), now that third party development environments have been allowed, Adobe's CS5 creative suite can be used to create iOS Apps such as their own Photoshop Express App which is already present on the App store at the moment offering users the ability to edit their photos on the device.

There are several third party software development environments available to potential iOS developers, QuickConnect iPhone (Barney 2009), Phone Gap (Stark 2010) and Adobe CS5 (Shankland 2010) are only a few of the main options available. QuickConnectiPhone which has the ability to manipulate the most of Apple's development API's allows the developer to write code for iOS apps using HTML with CSS and JavaScript within the iPhone development environment where it then is translated by the QCi framework into objective-C that the iPhone can understand (Barney 2009). Phone Gap which is cross compatible with the most platforms works in a similar

way but is limited to the type of applications it can be used for, PhoneGap applications must access web services from a pre-existing site to fetch data (Stark 2010). This is a useful piece of functionality for developers who are wishing to develop a dedicated App for mobile versions of websites, especially across several mobile platforms at once as using PhoneGap should cut development time significantly.

The QuickConnect family of frameworks has existed long before Apples recent decision to open up the Apps store to third party SDE's (Teton Technical 2008), as it is part of a much larger family of environments that allow easy development of iOS apps, Android apps, Blackberry apps, Mac and even Linux applications as listed by Barney (2009). The SDE was designed to allow developers to create hybrid applications across a variety of platforms, thus cutting development times for cross platform applications significantly as all version of the application use the same code and the same framework within the SDE which is then translated into the target platforms native language to be compiled (Barney 2009). It can be stated that QuickConnectiPhone is clearly a very robust development environment and its use can make the development of cross platform software much simpler and faster to achieve.

Apple has long played a major role in the smartphone industry, beating out long standing competitors like RIM and Nokia (IDC 2010) in the area of profits and market share. Although not everybody chooses to write apps for iOS for different reasons, chief among these is the stringent list of App Store guidelines that must be followed (Apple inc. 2010) and the widely held belief that a developer must use Apple's Objective-C (Barney 2009). Despite the shortcomings of the platform it is still the most successful of the mobile platforms based on unit sales and profits from the IDC (2010) alone. The QCi framework has shown that it can make the process of iOS App development much simpler to potential developers and can be made readily available to those developers who wish to use the framework. Although as it adds an additional layer of working between the user writing the code and the API being implemented (Barney 2009) it is worth further investigation into its benefits and potential shortcomings.

1.2 Project Overview

The project overview section is intended to identify and justify the projects primary research method while identifying a question that the project will attempt to answer through the investigations development and testing. It will then outline and give details of the project objectives and state the proposed hypothesis of the project.

1.2.1 Project Method

The focus of the project is to investigate how much the use of relatively newly available non proprietary development framework QuickConnectiPhone can speed up iOS App development and increase their efficiency when implemented. The following research question's purpose is to illustrate the projects aims.

Does the use of the QuickConnectiPhone software development framework provide a significantly measurable increase in the speed and ease of which iOS Apps can be created by developers opposed to the use of Apple's traditional Objective-C approach?

1.2.2 Objectives

The core aim of this project is to ascertain how beneficial it is to use the QuickConnectiPhone software development environment to develop iOS Apps versus Apple's traditional method of learning objective-C. This will be attained by monitoring and reporting on the difference in the various stages of App development process and the efficiency of the created Apps with the use of processor metrics to monitor efficiency and resource usage, along with a log of each development, noting any issues that arise in each increment in the Apps creation.

The objectives of Main Literature Review are to:

- *Investigate any benefits of the QuickConnectiPhone SDE in comparison to to Apple's traditional method of using Objective-C.*

For QCi to be a viable alternative to Objective-C it must have some notable advantages that can be found in literature examples to support any claims that are made. Such examples of these benefits can be found in the works of Barney (2009) and Stark (2010), among others.

- *Investigate existing Apps created using QuickConnectiPhone.*

It is important to determine the impact the QCi has already had on the App community by tracking down examples of pre-existing apps on the App Store that used QCi or other third party SDE's to create them. These examples will help to understand any constraints within the QCi framework.

- *Investigate the available applications that can be used to measure development and processor metrics in an iOS development*

The development process of the project must be documented and maintained to a satisfactory high standard by use of a pre existing system. According to Apple their own SDK contains an extensive array of measurement tools for analysing developers Apps during development to ensure all submitted Apps are efficient when in use. This should be looked into to see what each tool does and how it can be used to help measure the effectiveness of using QCi although alternatives should not be overlooked.

- *Investigate the state of security on the iOS platform to determine if the QuickConnectiPhone is more or less secure than the Objective-C method*

As the QCi development framework only makes use of the UIWebView element of the iOS SDK it should be investigated to ascertain if this is positive or negative security issue for developers to consider when making the choice to use these methods.

The projects primary method objectives to be completed are :

- *Review the categories available in Apple's App Store and investigate a viable example to emulate with the project App.*

There are many categories in the App store that must be examined, such as education, reference or entertainment. It is important to review popular Apps in each of these categories and locate those which make use of the majority of Apple's API's in a productive way. The chosen Apps can then be used as a guide and reference point when creating the test App for the project and so must be investigated thoroughly.

- *Design Specification for Project App*

Based on the findings from researching existing App Store Apps, a suitable design must be completed before any development begins, that satisfies the needs of the project.

- *Learn how to create the App using Objective-C with Apples SDK*

Use appropriate literature and internet resources such as Sams teach yourself iPhone development in 24 hours to learn how to implement each API available to iOS developers. From a lengthy search and assessment of customer reviews this example book appears to be of high quality and worth using for the project.

- *Create the iOS application in traditional objective-C and log development progress.*

Use the new skills learned from the chosen literature materials to create an App based on the examples found in the literature review that use a large array of API's. During the creation of each API a log will be kept of any issues that arise while coding and noted for use in the final report.

- *Run various processor metrics on the app while running to examine the load on the machine.*

Apples own iOS SDK program Instruments contains various processor metrics that measure all aspects of an App to ensure high levels of performance from developers creations.

- *Learn how to create the App using QuickConnectiPhone SDE*

Use Developing Hybrid Applications for the iPhone using HTML and JavaScript (Barney 2009) to learn how to create an iPhone app using this framework in place of using Objective-C.

- *Create the iOS application using the QuickConnectiPhone SDE and log development progress.*

Use the new skill learned from Developing Hybrid Applications using HTML and JavaScript to create the same App that was created using Objective-C. During the creation of each API a log will be kept of any issues that arise while coding and noted for use in the final report.

- *Run various processor metrics on the app while running to examine the load on the machine.*

Apples own iOS SDK program Instruments contains various processor metrics that measure all aspects of an App to ensure high levels of performance from developers creations.

- *Create a development report that illustrates QCi's benefits against Objective-C*

Create a development report that illustrates, using clear figures, all the benefits and any drawbacks of using the QuickConnectiPhone SDE in iOS development versus the use of objective-C using the data collected during development and testing phases.

1.2.3 Hypothesis

The hypothesis for the project is that the use of the QuickConnectiPhone development environment will significantly speed up the development of iOS Apps, reduce the difficulty of their creation and increase or at least maintain processing efficiency when operating.

This view is supported by several authors and developers who support the use of third party iOS SDE's, (Barney 2009) the writer of Developing hybrid Applications using HTML and JavaScript, who is the founder of the QuickConnectFamily Framework, and (Stark 2010) author of Building iPhone Apps with HTML CSS and JavaScript.

1.3 Report Structure

This section outlines the contents of the following sections of the interim report, identifying the main area of each and providing a brief detailed description of the sections aims and content.

1.3.1 Literature Review

The Literature review is contained within chapter two of this report and investigates the areas of interest outlined in the literature review objectives located in section one of the report. It begins with an investigation into the functional differences between the methods of objective-C and QuickConnect before moving onto examine their individual benefits and drawbacks. Apps that are already published on the App store that used QuickConnectiPhone during development are then examined for their relevant capabilities. It will review the many categories of the App store and outline those suitable Apps which display the use of a wide array of Apple API's, identifying their characteristics in order to create a basis for the project app. Finally it will investigate the capabilities of Apples SDK program called instruments which is designed to run various processor metrics on developers Apps, the exact abilities of the program will be investigated as will any viable alternative programs.

1.3.2 Methods

The projects methods are located in section in chapter three of the report. The develop and test method of this project will be specified and justified, by allocating practical activities and resources of the development process and provide clear reasoning for each choice. Finally the functional and non functional requirements of the Apps will be investigated and specified

1.3.3 Design and implementation

The design and implementation in section in chapter four of the report documents the process of the user interface of the two test Apps and the process of the Apps functionality development. It documents the initial user interface screen designs of the Apps before investigating Apple's HCI guidelines document and subsequently finalises the revised designs based on the guidelines. It then moves on to details the incremental development process of the two test Apps.

1.3.4 Results Evaluation

The result evaluation section found in chapter four of the report details the results of the two test Apps development logs, outlining any issues with the development of the Apps. The analysis of the Apps then follows in a comparison with Apple's HCI, App development and Submission guidelines documents to ensure the level of quality Apple expects of App Store submissions. Finally the Instruments benchmark tests of the two Apps are reviewed to locate any notable difference in the way in which each App runs on the device.

1.3.5 Final Conclusions

The final conclusions in chapter 6 recaps on the event of the report before moving on to discuss the limitations of the project and how future work could be done to further the investigation into the subject material. The section then finishes off by detailing the findings of the project and how they relate to the initial hypothesis and how well the original project question is answered by the findings.

2. Literature Review

The literature review is intended to investigate the areas described in the secondary research methods section from chapter one of the report. It will investigate the benefits of using QuickConnect and of using Objective-C in App development and the functional differences between the two approaches. It will then go on to examine the characteristics of existing applications created using both methods. The fourth area of investigation will be into the various processor metrics and benchmarking applications that can be used to assess an applications efficiency. Finally it will investigate the known security issues associated with iOS and which method of App development is more susceptible to these issues.

2.1 The benefits of the QCi SDE in comparison to Objective-C

Both Barney (2009) and Stark (2010) claim that making use of the ability to create cross platform capable iPhone applications using HTML, CSS and JavaScript have several advantages in comparison to using objective-C. This section looks into the functional differences between the two separate approaches and investigates their individual benefits and drawbacks.

2.1.1 The functional differences between QCi and Objective-C

Functionally the methods employed by the QuickConnect framework and the Objective-C approach are very different. As seen in figure 2 Apple inc (2010) [D], the iOS application architecture is based around the Model View Controller in that the view is the user interface output, the model is the back end functionality and the controller handles and passes user input to the model to be processed which then updates the view as needed. This architecture is reflected in the modular way in which traditional way iOS App are constructed (Ray 2010) by having the App interface created entirely separately from the rest of the application within the interface builder environment where each view is constructed and simply linked through outlet and action functions that are referenced in the Xcode project to allow interaction between the interface and the functions of the code. In contrast an application created by QuickConnect is essentially a web application contained within an objective-C wrapper to allow it to run natively on an iDevice (Barney 2009) and as such is constructed in the very same way any other web application is created. The user interface is created using HTML tags and accompanying CSS file to style the page and provide the output, while the JavaScript code is used to execute functions that send and receive data and manipulate the web page DOM structure based on the results of the function.

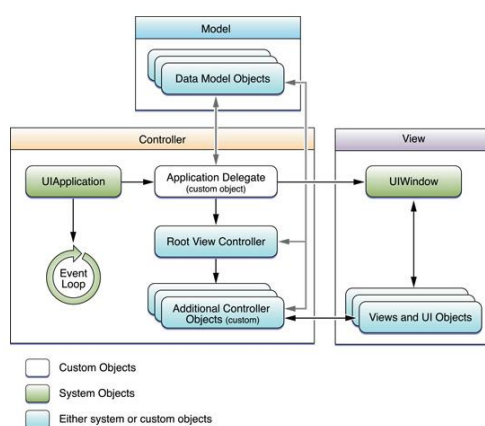


Figure 2 - iOS MVC application architecture

Additionally the lifecycle of an Objective-C app differs from a JavaScript app in the way that event are handled by the applications. The example in figure 3 Apple inc (2010) [D] illustrates the lifecycle of an event in a traditional iOS application built around Objective-C as described by Ray (2010). When a user taps an application icon on the iDevice home screen the UIKit loads the main function method that initiates the UIApplication method that then goes on to call the user coded functions present in the "didFinishLoadingWithOptions" section of the rootController page in the

code. This function is effectively the same as the on-load element within a web application that sets up the application each time it is activated. Once the function begins the event handler loops the event until all the required tasks are complete and the function removes itself from the current view and is moved to the background should it need to be called again.

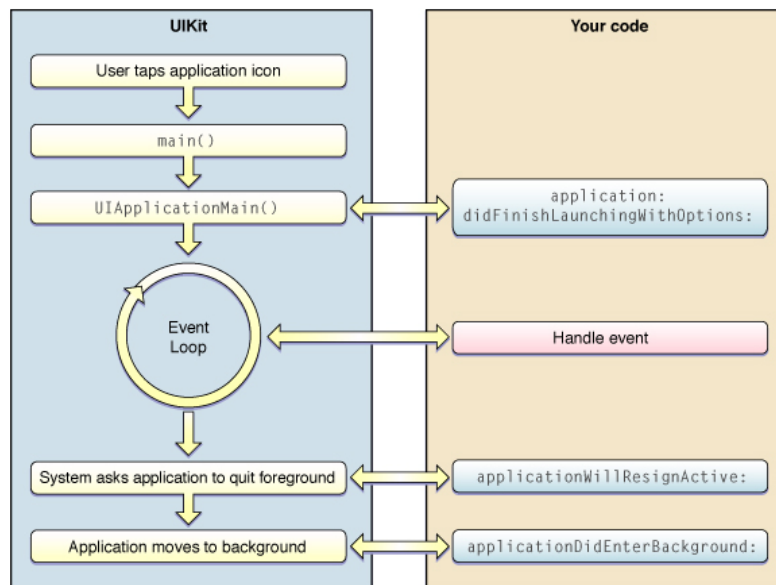


Figure 3 - Application Lifecycle

2.1.2 The benefits of using QCi and other third party environments

Although too many it is not simply the need to learn a new language that hinders a developers desire to code for iOS, in many cases it is Apple's extensive, strict and extremely thorough set of submission guidelines which must be followed to the letter if an App is to be accepted into the store, a process that can take anywhere from a week to a month in some cases. It is possible for anyone to view the Developers guideline document to App submissions (Apple inc, 2011)[E] from Apple's developer website should they consider undertaking an iOS development as their choice of platform and even the use of third party frameworks will not allow developers to avoid adhering to these guidelines during their App submissions. It can be seen from the instructional books by (Barney, 2010) and (Stark, 2010) that one of the main benefits of making use of the QuickConnectiPhone framework is the ability for web developers to make use of pre existing development experience in programming with common web technologies such as HTML, JavaScript and CSS by removing the need to learn Apple's Objective-C coding language used on all iOS devices when creating Apps aimed at Apple's App Store. This is the main advantage advertised to developers by these third party development frameworks and draws in many users.

Another main advantage of using QuickConnectiPhone is the extensive cross platform compatibility prospects for App developers where little change is often required to the App source code (Barney 2009, Teton Technical 2011) once it has been implemented for the first time in each instance and must only be transferred into the alternative platforms framework before making adjustments to the code before a working version can be produced. This is achieved by the various compatible frameworks developed by QuickConnect, such as QuickConnectAndroid, QuickConnectBlackberry and so on. The QuickConnect Family of Frameworks supports use of cross platform code on iOS, Android, Blackberry, Linux and Mac (Barney, 2009) all to a certain extent. Although it should be noted that the Android, Blackberry, linux and Mac Frameworks are still in open development as the community is open source and relies on the developers to continue its evolution by implementing additional functionality into the framework. Whereas the

QuickConnectiPhone framework is at the forefront of the development community and receives the most attention from developers, as a result not all of the other frameworks can match the functionality of QCi when developing Apps.

In addition to the already mentioned third party SDE's QuickConnect and PhoneGap by barney(2009) and Stark(2010) there are a wide variety of other SDE's available to developers (Chris 2009, Krill 2010). Other applications such as Appcelerator titanium, Hhomobile, Novell, iWebKit and JQTouch all feature similar abilities to the QuickConnect and PhoneGap platforms, although most are not as sophisticated or extensive as QuickConnect and PhoneGap which are two of the most commonly used platforms for third party development (Barney 2009).

The real world benefits of making use of developing third party, cross platform capable applications for the iPhone and other platforms has been demonstrated successfully by a London based development company called Grapple Mobile (Wauters, 2011) who have made a successful business developing mobile applications in this way, earning millions of pounds in development contracts with Sky, Microsoft and Premier Inn over the past year and a half since the companies creation in January 2010, shortly after Apple's announcement to allow third party framework development.

2.1.3 The Drawbacks of using QCi and other Third Party SDE's

As indicated in the previous section there are many advantages to pursuing the development of third party application environments. Although that is not to say it doesn't also have its drawbacks, these frameworks may decrease development time and complexity but at the cost of core functionality on the iOS platform (Barney 2009, Stark 2010). These frameworks cannot fully utilize all of the available API's Apple allows native developers to exploit, in the case of QuickConnect they are camera access and image geo-location. PhoneGap does not have access to slightly more of Apple's API's such as Maps, phone integration, system sounds, Bluetooth, offline file storage or camera access.

The other less well known frameworks on offer to developers feature even less API access than that which was demonstrated by QuickConnect and PhoneGap, Wauters (2011) comments on the abilities of a selection of the less well known frameworks which provide as much access to the native API's of the iOS platform as a standard web App that can be accessed through the Safari browser.

2.1.4 The Benefits of using Objective-C in iOS SDK

The main benefit of using Objective-C in iOS development is the full access to all of the API's available to developers and the use of the custom Interface Builder Application (Ray, 2010) that allows for the creation of highly interactive and aesthetically pleasing UI's for iOS apps for relatively little effort. Many long standing Object Oriented purist developers feel that Obj-C is in many ways one of the best OO languages to develop with (Roehrl, Schmiedl 2002) as it allows objects to send messages to one another even if the format is not supported by one of the objects so it can be passed on to another object that does understand it.

2.1.5 The Drawbacks of using Objective-C in iOS SDK

The main drawback to using Objective-C to code iOS Application is considered by some to be the lack of cross platform compatibility (Stark 2010) or the difficulty many associate with learning the object oriented programming style. Many web developers will not code application in Obj-C because they know they can achieve much of the same functionality by making use of web technologies instead. According to Rutman(1997) many develops do not like using Obj-C as they see it as taking the worst elements of C and blending them with the worst elements of Smalltalk. He goes on to talk about the features which Obj-C is missing that C++ incorporates and that many developers do not like the idea of loosing those features. An example of a missing feature from

Obj-C is the use of overloading, where objective-C uses a workaround for method overloading but not operator overloading, a feature some developers would not like to lose when developing.

2.2 Existing Application Characteristics

This section investigates examples of applications that already exist and have been developed using third party frameworks and were successfully published onto the Apple App Store, this is being done because it is important for the project to understand the practical implementations of the technologies being investigated for this project to gain a better understanding of their capabilities.

As seen from submissions from QCi developers on the community page and from the App listing directories of PhoneGap it can be seen that the most common types of Apps chosen to be implemented using QuickConnectiPhone and other third party development frameworks are mostly reference, educational or organizational Apps(Barney, 2009) such as the publicly available OIS iMobile(2011) produced by Orders in seconds, the company created the App to be intended for use by self employed business men for sales force automation services, managing business calls and merchandising activities. The OIS iMobile App makes use of most of the API's available to iOS developers such as Google Maps integration to locate warehouses and offices inside the App, calculator functions to help with ordering parts and materials, address book synchronization for keeping all customers and parts suppliers contact details grouped together to easily locate important contacts and finally it allows for quick access to the phone and email functions on the iPhone for quick communication. This App is a model example of the abilities of QCi and what the project aims to implement as it taxes the iOS platform enough that product metrics during the testing phase of development should provide comparable results when both development methods are implemented to test the project hypothesis.

Unlike the QuickConnect Framework which is a development driven community other popular third party frameworks such as PhoneGap which is owned by Nitobi Web & Mobile Applications who actively record all Apps designed using their framework and so have an extensive listing of all published Apps on their website for all platforms including the iPhone, android and Symbian. Whereas QCi does not require developers to submit their Apps that have been published onto the Apple App store(Barney, 2009), instead developers will often elaborate on their publications on the QuickConnect community page as they are released during discussions on different functionality aspects of QCi. As a result there are many examples of high quality Apps to examine that are published on the Apple App store that have been created using various third party frameworks. Due to the necessity to submit the names of all published Apps using the PhoneGap framework their website contains a wide variety notable Apps which are relevant to the project aim and warrant recognition. PhoneGap was not chosen as the development framework for the project due to its inability to manipulate all of the native iOS API's that QCi has access to, some of which were used in the OIS iMobile App such as address book and phone integration.

2.3 Appropriate development and processor metrics for iOS

Instruments is an example of an application that can be used in the project to run analysis on each test App to identify any differences in the efficiency of the developed App side by side to ascertain if one development approach is superior to the other. Instruments was designed by Apple to be the primary testing suite used by iOS developers (Apple inc, 2011)[D] when creating Apps for the iPhone/ iPod Touch, iPad and all of the Mac desktop applications. It can be used for testing each aspect of the functionality of Apps such as memory leaks, power consumption and resource management in iOS developments and has several default templates that were created to measure different aspects of the testing app. Or alternatively the developer can choose to create a custom template that runs several different testing scenarios simultaneously which will be the setup chosen for the test App to create a timeline of data from each perspective that is relevant to the test.

The list of preset testing templates in Instruments is extensive, covering the majority of types of product metrics (Apple inc, 2011)[D] a developer could wish to use during the testing of their App development. The Allocations template can measure heap memory usage, the Leaks template measures general memory usage,

the Activity Monitor template monitors overall system activity and the System Usage template records I/O system activity. These testing templates attempt to examine all areas of an applications functionality and look for the targeted defects within the test parameters to assist developers in the testing phase of the development process, although within the context of their project they would be used for comparability reasons between the two test Apps.

Other options for testing web based application are the acid tests created by the Web Standards project (Hickson, 2011) that run several test scenarios on the target application to test for web standards compliance and the effectiveness of the application being tested.

Although other product metrics applications exist for testing Apps the iOS SDK native Instruments application is best suited to the testing of iOS Apps as the development platform was designed for use with Instruments and fully taxes the iOS and Mac OSX systems and display clear and useful information from the tests.

2.4 iOS Security Vulnerabilities

The Concern of device security was an issue for the project as It was made apparent to the author that mobile devices can suffer from fairly detrimental security issues at times. For this reason an investigation into iOS security vulnerabilities was undertaken for the project to determine is an application created using QuickConnect would be more or less secure than the Objective-C counterpart App in the project.

Apple designs and builds both the hardware and software for all of its devices across the entire Apple range of products. Interoperability across devices is extremely high allowing all Apple products to synchronise seamlessly between one another without any action on the part of the user, a feature that prominently fits into everyday use of mobile devices such as smart-phones and media tablets.

The iOS platform shares a core kernel with Mac OSX and like Mac OSX (Apple, 2011 [B]) is not affected by the majority of viruses that are transmitted via the Internet (Landesman, 2011) due to robust software and hardware architectures, contrasting with the vulnerabilities associated with Windows machines. Vulnerabilities associated with iOS are unique to itself and to some extent the OSX platform. Security analysts who have observed the platform since inception have encountered the process known to developers as jailbreaking. Users exploit insecurities in the operating system allowing device root access (Pandya 2008), facilitating installation of third party applications without authorisation from Apple, increasing likelihood of Stack Buffer overflow attacks. Recently noted vulnerabilities of the iOS include a PDF fault that was introduced with the release of OS 4.0 (Miller, 2010) allowing implementation of malicious code onto iDevices through a fault in the way Apple operating system parsed PDF files within the mobile safari browser. As the test Application using QuickConnect is entirely within an artificial browser wrapped in the objective-C application (Barney, 2009) it will be subject to all the known and unknown security vulnerabilities associated with the use of the mobile safari platform on the browser.

The main security feature of the iPhone is the locked down OS prohibiting third party software installation. Due to the locked down architecture of iOS an unmodified iPhone can only be attacked via Mobile Safari browser via a connection to a malicious webpage (Miller et al. 2007) executing JavaScript code, forcing the phone to make a connection to a controlled server and transmit personal data such as SMS messages and contact details. More recently observed is the PDF fault introduced with the release of OS 4.0 which allowed malicious code implementation in iDevices through a fault in the parsing of PDF files. As the majority of the attacks an iOS device is vulnerable to are javascript attacks, a technology that the QCi App relies on to provide rich user functionality, and one that could endanger the application should it be used without precaution.

The first iPhone Jailbreak program came into being shortly after the first iPhone gained released in 2007 The entire iPhone file system could be accessed by users, allowing alteration of system settings or installing third party applications. A direct drawback to tethered jailbreaking was synchronising the device with iTunes was no longer available without a factory restore. Once jailbroken an iPhone is installed with a program called Cydia (Chronic Dev Team 2011) which is the jailbreak communities App store and give users the ability to

alter the iPhones operating system in ways that can greatly increase the usability of the device (White, 2010). Jailbreaking is a process more and more developers are turning to these days to avoid Apple's strict submissions guidelines. Even Toyota, A massive Car company created an application for the Cydia App store (Hardigree, 2011) although it was quickly pulled from the market by request from Apple after the story made it to the news.

The benefits of jailbreaking include a multitude of system tweaks and alteration such as adding new menus onto the springboard or even allowing the use of free Wi-Fi hotspot tethering using MyWi that a user can purchase from the store and avoid paying their carrier large tethering bills. There are drawbacks to jailbreaking an iPhone primarily that of the compromised security system needed in order to facilitate third party Apps means that more vulnerabilities exist within jailbroken devices. This is a direct concern to the project as the QCi is entirely written in web technologies and is javascript based. It is entirely a question of trust of the maker of the App and common sense of the user when installing Apps onto the phone. With root access, a malicious App could access any information on the phone, transmitting stored information recording input from the user such as passwords and contact details.

2.5 Literature Conclusion

Through the course of the literature review there have been many examples of why a developer would choose to make use of a third party framework to build their application. They could simply not wish to learn objective-C because they either do not have the time and are already proficient in web programming or they simply do not like the styling of the language, or they may want to allow cross platform compatibility within their application as it was proven many do by the success of the Grapple Mobile organisation.

The review also noted the disadvantages to making use of third party frameworks to contrast the advantages that developers are attracted by. As third party application are not created in the same manner as obj-C applications they differ in the method of construction being implemented and thus much of the Apple guideline documents available to developers do not apply when using third party tools. It was also indicated that third party applications are more vulnerable to the iOS platforms inherent security vulnerabilities as they are all run within the UIWebView element of an application that is effectively just the browser itself, the weakest point of security on the device..

Despite the apparent security issues associated with the use of third party frameworks the benefits outweigh the dangers in terms of development as in many cases the device would have to be directly attacked in order to be in any danger so it is concluded from the literature review that it worth pursuing the use of third party frameworks in iOS development as it affords developers greater flexibility when creating their applications while possibly reducing development time and effort.

1. Methods

This chapter of the report will cover the research methods that will be used to identify the projects functionality needs, in order for the Apps to be created the functionality they should contain will be analysed, Apple's HCI guidelines for App development must be adhered to and the final functional requirements will be defined. The develop and test method being used for this project has been chosen to fully illustrate the project problem by practically comparing the two technologies with a series of development and product metrics testing for variation in efficiency and ease of development.

3.1 Research Methods

The projects aim is to investigate the benefits of using the QuickConnectiPhone framework in the development of iPhone Apps when compared to using Objective-C to do so. In order to evaluate this in a meaningful way a develop and test method is being used in this project where a specified App will be created using two different development approaches and evaluated on the basis of system performance and the ease of use of the development environment. The project will pay close attention to the App store guidelines for submission of Apps to the App store (Apple 2010) as by doing so the author hopes to maintain realism when developing the App.

In a develop and Test type project a concept is evaluated by the creation of a fully testable working prototype of the initial concept which can then be subjected to the necessary tests to evaluate the projects initial hypothesis. These methods are a common method type when dealing with any kind of practical objective as it is necessary to have a physical outcome to properly evaluate rather than analysing theoretical experiment data. The methodology can be found to be used by many researchers (Griswold et al. 2004, Naismith et al. 2005, Ganchev et al. 2006) who base their hypothesis on the evaluation of prototype developments.

The development of the Apps will take place at the authors home or in the Computing and engineering department of the Glasgow Caledonian University, as any App development must take place on an Apple Mac computer as the iOS SDK is not available for the Windows platform (Apple 2010), though it is possible to implement a remote Desktop client to access the authors home Mac setup from a Windows machine should it be necessary to do so using such tools as LogMein which allows remote access to the users home computer via Windows, Mac and even iOS devices (LogMein inc 2011).

3.2 Problem analysis

The Problem Analysis section deals with the analysis of the proposed Apps necessary functionality, investigate Apple's HCI development guidelines to ensure the project App does not violate any of the terms and finally clearly state the finalised requirements for the project App.

3.2.1 Existing Taekwondo App Functionality Analysis

The Apple App store contains many categories of Apps available to download onto a users iDevices and consist of: books, Business, Education, Entertainment, Finance, Games, Health and Fitness, Lifestyle, Medical, Music, Navigation, News, Photography, Productivity, Reference, Social Networking Sports, Travel, Utilities and Weather. Each category of Apps contains thousands of examples for users to explore with each offering fairly unique examples, with the exception of duplicated Apps, to provide a wide range of functionality to the user.

Based on the information collected from Apps from each category of Apple's App store the most viable application type to develop and use for the project test App will be a menu based reference material App primarily featuring text, images, embedded video and map integration, such an App is

suitable as it requires limited input from the user and is instead aimed at delivering text and video content to the user to put a measurable strain on the device. The best method to follow to ensure a quick App development is for the project to choose a field of interest that the author has a sufficient knowledge base and practical experience with that will not require intensive graphics or large amounts of research to provide rich content for the App. The most viable option would be to create a taekwondo reference app as the author has years of experience with the subject and extensive literature on the material.

The goal of the project App will be to improve upon the small selection of App Store published Taekwondo Apps that were investigated during the review App Store as customer feedback for the existing Taekwondo reference Apps is largely negative due to lack of functionality and poor usability design. Based on the findings from the review of existing Taekwondo Apps in the reference category and customer feedback on each App the project App should feature descriptions of each grades theoretical and practical knowledge requirements as its main content for users. It should also stream YouTube videos or play locally stored videos of demonstrating techniques that can be easily emulated correctly by the students watching to elaborate on the descriptions given. Additionally it should have Google maps integration showing listed Taekwondo schools and contact information for instructors in relation to the user location and a news section listing events such as upcoming competitions or gradings and seminars. These features and functionality are a compilation of the best features from the pre existing Taekwondo Apps as well as others not found in the published Apps.

3.2.2 Apple iOS Human Computer Interface Guidelines

The iOS HCI Guidelines document (Apple, 2011)[C] is in place to give developers a clear indication of what is and is not acceptable design practice within submitted Apps. The HCI guidelines document covers every aspect of App design that can be applied to a development project and as such is an extremely large document. Those that are applicable to the test Apps will be reviewed and adhered to as best as possible throughout the design of the Apps.

The platform characteristics of an iPhone App is a paramount concern when designing an App as the screen is both the output and primary input method and screen real-estate is scarce when dealing with such a small screen. As much information as possible has to be displayed to the user without making the screen seem too cluttered in the process, which is one of the first hurdles to overcome during App design.

Apple's Human Interface Principles take care to encourage consistency with App developers. It encourages developers to maintain aesthetic integrity when designing application interaction controls, ensuring that an icon always means the same thing every time it appears, this also coincides with the encouragement of consistency during design. It is also encouraged to allow users to directly manipulate on screen objects like flicking and pinching rather than press separate buttons for moving around the screen as this encourages a greater sense of interaction with the user. The feedback a user receives when making a request on screen is stressed very much as users expect immediate response when interacting with an App and so the appropriate responses such as highlighting a button, showing a processing timer or even displaying a message in circumstances of long waits assures the user that something is happening. The final element that is stressed by the document is to always keep the user in control, should there be adverse consequences to an action warnings should be displayed to the user rather than the App take control for them as this distances the user from the App experience.

A well laid out App design strategy is supplied within the HCI guidelines outlining each phase of development a successful App should go through, a strategy that the project App development closely follows. The first phase indicates a clear method of creating an application definition statement of requirements, first by listing all the features you think the user might like, determining who your users are, filter the list to determine which of the features should be employed and which

should be left out for better usability. The next step is to design the app for the target device, once you know what the app will do and who it is for you have to decide how it will deliver that content to the user, the document deals heavily with the use of iOS UI paradigms and interface elements which are all readily supplied to developers within the iOS SDK and allow developers to create consistent graphically rich applications with little effort.

Once the original design is established the customisation of the UI should be considered as the iOS SDK allows for heavy customisation by using original artwork and interface elements, so long as they do not breach any app submission guidelines. Finally it is recommended that all UI designs are prototyped and iterated before large amounts of resources are spent on a potentially unsuitable product.

3.2.3 Apple iOS Application Programming Guidelines

The iOS Application Programming Guidelines aim to aid developers in the creation of their App functionality after the design is at least partially complete by defining useful descriptions of the iOS SDK and its functions. The document contains sections on understanding the iOS runtime environment, designing the core of your application, supporting common application behaviors, executing code in the background, meeting App Store and system requirements and tuning the performance for the device. For the project test App the section on system requirements and tuning performance for the device is the most applicable as this is the area which directly relates to the project's key objectives to analyse app development and performance.

The entry on tuning for performance and responsiveness covers several topics to aid in creating an efficient and responsive application. The first segment in the section is an advisement not to block the main thread of the application, the main thread being where the application handles all user input, any task that takes a significant amount of time to complete should not block user input and should be sent into the background and be sent asynchronously to allow the user to continue until a response is retrieved, should the main thread be blocked in an app upon startup iOS will quit the App automatically. Memory management is a large part of iOS development as it is a finite resource on a mobile device and as much of it must be preserved as possible when not in use. iOS employs several low level memory warnings to applications that trigger responses to free up as much as possible to continue normal functions, although this functionality has to be included into the app to be implemented. It also advises on how best to avoid memory leaks, making resource files as small as possible, using core data sets for efficiency and to allocate memory wisely by manually releasing objects as soon as you are completed with them and avoid using autorelease as much as possible. It also advises on how to best optimize power consumption through various methods; by avoiding work that requires polling, accessing the disk as little as possible, connecting to external services only when needed and always connect to wifi signals in preference to cellular to use less power.

3.2.4 Apple App Store Review Guidelines

There are many sections of the Apple Developers Guideline document (Apple, 2011)[E], covering topics from functionality to legal requirements. For the purposes of the project only the key sections of this document will be adhered to due to time restrictions as the full guidelines document contains over one hundred articles to follow. The most appropriate sections to be followed by the project App are; Functionality, Location, Media Content, User Interface, Violence and Privacy.

The functionality section of the guidelines is a necessary section to abide by as all Apps provide various aspects of functionality for the user to make use of and so it must be regulated to determine what is and isn't acceptable, the complete copy of the App guidance document can be found in Appendix A. The Functionality section states that Apps will be rejected if they; crash, exhibit bugs, do not perform as advertised, include hidden features, use non-public API's, that download additional code and that browse the web must use the Webkit framework. These are all the key

point in the functionality section of the guidelines and they are all very important points to consider when developing the project App as they give very good guidance on how to create the App and what practices to avoid.

The Location section is applicable as the map services employed within the Apps makes use of the users location when displaying local schools to the user. This section states that an App will be rejected if it; does not notify the user before transmitting or collecting location data, or use location services for autonomous control of industrial vehicles or emergency services.

The Media content section is applicable as the Apps include a video section for users to watch pre recorded examples of Taekwondo patterns being performed by a black belt. It states that Apps will be rejected if they; do not use the media player framework, tries to mimic the iPod application, the bit-rate for streamed audio content must not exceed 5MB over 5 minutes and video must be streamed over HTTP live streaming.

The User Interface section applies as all Apps need a user interface and Apple advocates a unified “feel” to all Apps that are published. It states that apps will be rejected if they: do not comply with all terms and conditions of the Apple HCI guidelines document, look similar to Apple first party Apps, that do not use system provided interface items, alter the function of hardware buttons and if the user interface is too cumbersome or non intuitive.

The violence section applies because the Apps will show videos that can be construed as violent by some and so it is necessary no violations are made with the content in the Apps. It states an App will be rejected if it; depicts realistic images of people or animals being maimed or killed, depict abuse to children or involve realistic description of the use of weapons and encourage the reckless use of said weapons.

Finally the privacy section is applicable as the users privacy is of key concern developers and Apple. It states that apps will be rejected if they attempt to collect or transmit data about a user without obtaining the user's prior permission and providing the user with access to information about how and where the data will be used, that require the user to share private information to achieve functionality or target minors for data collection.

3.2.5 Functional and Non Functional Requirements of App

As previously investigated in the literature review and the analysis of the functionality of pre existing Taekwondo Apps it has been determined the most viable application type to develop and use for the project test App will be a menu based reference material App primarily featuring text, images, embedded video and map integration, such an App is suitable as it requires limited input from the user and is instead aimed at delivering text and video content to the user to put a measurable strain on the device.

The functional Requirements of the Apps will be:

- The product will present a menu based interface.
- The product will feature a link to the United Kingdom Taekwondo Federation homepage.
- The product will feature a map service with geo-location to display nearby TKD schools.
- The product will contain detailed description of Taekwondo theory.
- The product will contain videos of TKD patterns.

The Non Functional Requirements of the Apps will be:

- The product will be hierarchically menu based.
- The product will be a simple menu format.
- The product will be easy to use.
- The product pattern descriptions and videos will be available offline.
- The product map feature will be only available when online.
- The Product home page feature will be only available when online.

4. Design and Implementation

The Design and Implementation chapter documents the steps within the design and implementation stages of the project App development. The process of designing the user interface from starting paper designs to finalised product design is documented and the decisions explained. The Objective-C and QCi App developments are described in detail separately indicating the how each aspect of functionality will be implemented for both Apps.

4.1 Design Specification for Apps

This section details the initial screen designs for the Apps before reviewing the individual development environments for the two methods and finally produces a set of distinct final user interface screens based on the findings.

4.1.1 Initial UI Screen Designs

To begin building a user interface for the Apps based on the functional requirements papers screen designs were made to provide a clearer picture of each screen's layout and functionality. The below screenshots depict these paper designs, the full collection of which can be found in Appendix B.

The main page of the User interface, depicted in page one of figure 4 is based upon the iOS Grouped Table View to display a segmented interactive list of hierarchically organised views for each functional page. The main page shows three options, option 1 takes the user to the main page of the United Kingdom Taekwon-Do Federation website Home Page, illustrated in page two of figure 4, to allow users of the app to access the facilities of the website without leaving the App. Option two in page three of figure 4 accesses the Map view of the application to show the user all nearby taekwondo schools based on a location based Google search that displays the schools location and contact information on the map for the user to interact with. Screen four of figure 4 contains the third and final option of the three from the main page of the App, the patterns page of the App contains information on all the colour belt taekwondo patterns and embedded video elements contained locally to display videos of the patterns being practiced by an instructor for the students use. Screen 5 of figure 4 displays a sketch of the video playback control displayed when a user touches the button to play the pattern video.

These designs were chosen for the initial design of the test App because all the controls and interactive elements used are Apple's own design elements. They can be found in the interface builder application toolbox library that is used by all applications user interfaces when developed within the iOS SDK as shown in Appendix C and so provide a standardised interface that upholds all of Apple's HCI Guidelines.

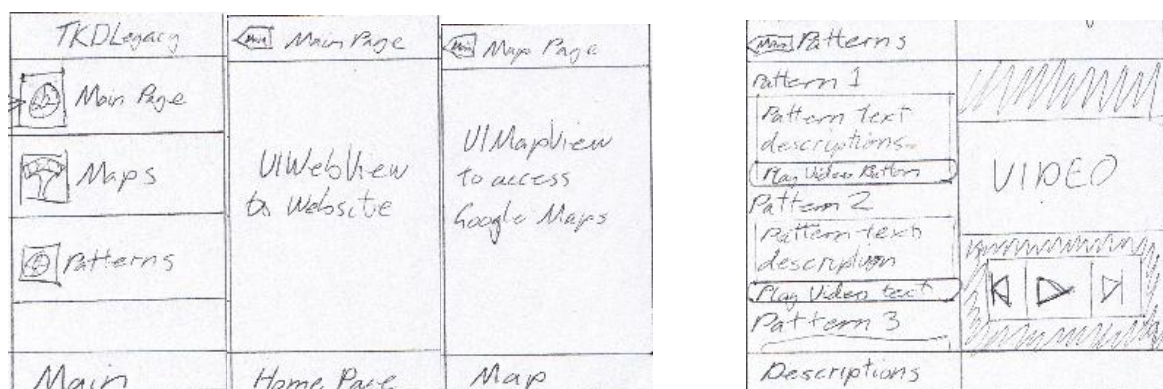


Figure 4 - Main page, home page, map page, patterns page and video playback control paper screen designs

4.1.2 Review of Both App's SDE User Interface Design Abilities

As each of the two test Apps use different methods for creating the user interfaces both cant simply share the interface that was originally designed in the paper screens under the assumption that the apple UI element library would supply the interface controls to both Apps.

4.1.2.1 Objective-C

As the initial screen designs were created with reference from the interface builder element library in Appendix C that will use when creating the user interface for the the objective-C App there will be no aesthetic change to this example.

4.1.2.2 QuickConnectiPhone

The QuickConnectiPhone does not use Apple's interface builder application to create the onscreen elements for users to interact with as it uses the devices ability to emulate a web view win the iOS SDK to create a hybrid web application within the App that is programmed using XHTML, CSS and Javascript. As the design of the QCi application is dictated by CSS and XHTML the design of the App will have to be altered to accommodate standard web controls instead of Apple's custom controls, although it would be possible to emulate the Apple interface it is not a viable option due to time constraints.

4.1.3 Finalised UI Screen Designs

This section contains the screen shots of each Apps finalised user interfaces constructed as prototypes to be tested before the App functionality was created to allow for easily implemented revisions to the design without undoing any work.

4.1.3.1 Objective-C

The final screen designs for the Objective-C App were created in interface builder and the accompanying functionality was created with Xcode to allow each section of the app to perform its task, a selection of the screenshots in figure 5 from Appendix D show the functionality of the App being employed with its final UI look.

The first of the screenshots depicts the final layout for the main page table scroll element that allows access to the App functionality. The second screen outlines the view of the UKTF homepage website where the user can access news and event without leaving the app. The third is a shot of the map view with all local TKD schools on display, should the user click one of the pins contact details will appear to the user for use. Finally the patterns page is last and shows the textual descriptions of the grades with buttons that trigger the playback of each video.

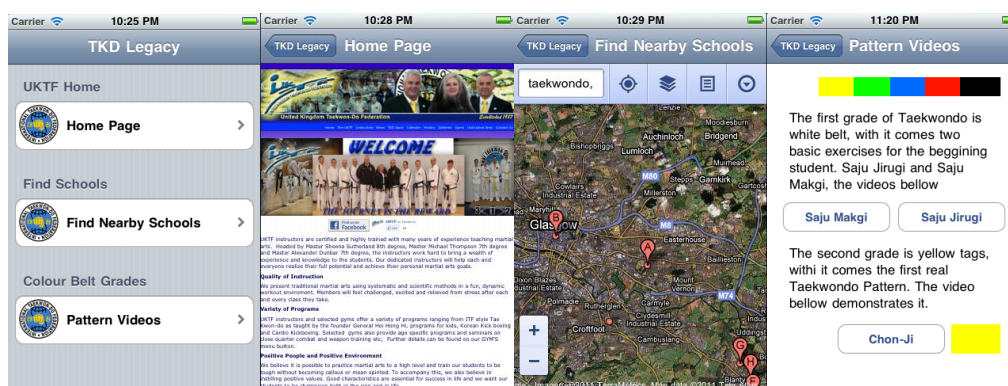


Figure 5 - Final Obj-C screen Designs

4.1.3.2 QuickConnectiPhone

As the QuickConnectiPhone App was created using web application technologies and as such was designed using XHTML and CSS the final design of the app was different than that of the Objective-C App as it didn't have access to the same library of interactive elements.

Bellow are in figure 6 are a selection of screenshots of the QCi App, which can also be found in Appendix D. The first screen shows the layout of the main page used to access the functionality of the App. It is immediately noticeable that the QCi App looks very different from the Obj-C App as the QCi App doesn't have access to the table view control used by the Obj-C App. It instead uses XHTML elements styled using CSS to provide the three sections to choose from. The Home page button and Map view button both provide the exact same function as the Objective-C App other than it being implemented slightly differently. The patterns page is similarly just as noticeably different as the main page as it too is created using XHTML and CSS.

The onscreen video previews are achieved by using the HTML 5 in-line video tag element to display the locally stored videos, although it would work the exact same if the videos were streamed externally apart from requiring internet access. JavaScript functionality was included in the App to provide the ability to swap between views without having to reload the page by having the buttons call a function to manipulate the DOM and change the visibility of certain DIV elements on the page, making it seem like the view was changing while removing the loading time between changes.

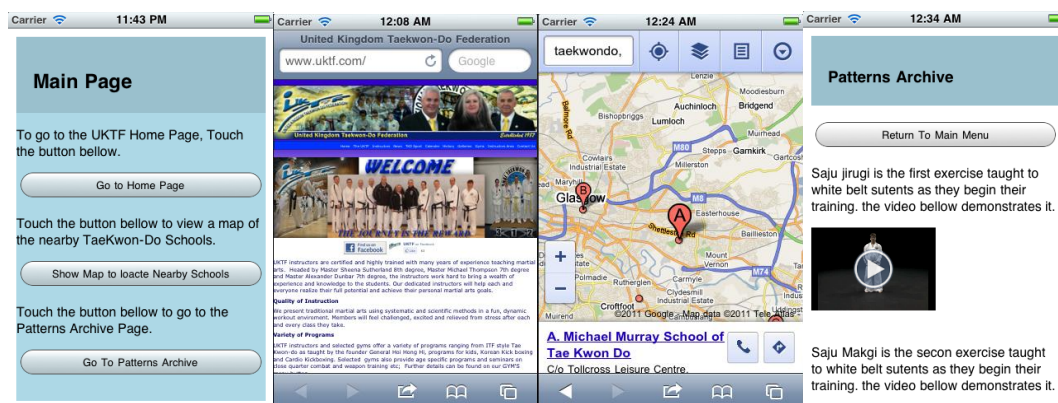


Figure 6 - Final QCi screen designs

4.2 Objective-C App Development

The development of the Objective-C based App will likely take longer to complete than it's QuickConnectiPhone counterpart as images and source material for the App will have to be gathered from literature and internet sources before development can be fully completed and as such will increase the span of the development in comparison to the second App. Each app will be developed incrementally, creating one piece of functionality at a time and logging the development process of that section. This method provides two key advantages to the project, it allows for incremental testing of the application to ensure each new element functions properly and it provides a simplified way of recording each phase of development for comparison. The full code listing for the Objective-C App can be found in Appendix E.

4.2.1 Incremental Development phases

The Objective-C App development will be segmented into three individual parts; firstly the design of the User Interface elements using the iOS SDK's interface builder application. Then Xcode will be used to incrementally create the functionality of the App while running incremental test cases on each segment throughout development. Finally Instruments will be used to test the Application and ultimately discover any differences in the performance of the two Apps. As the development of each App progresses a log of the progress will be kept outlining any difficulties that arise when implementing the functionality which can then be compared to the QuickConnect App to determine if the issue is persistent through both developments or not.

After the user interface design was finalised the first stage of developing the application functionality was to begin by coding the segmented table view on the main page which can be seen in figure 7. This required an array of array elements of dictionary elements to properly format the data being displayed in the view. Each top level array separates the table view into the three separated heading sections, while the internal array is set up to contain the dictionary elements which held all the values needed for when a user makes a selection, the name of the element, the image attachment and the url. These values were then passed onto other functions upon receipt of input from the user to navigate to the new view within the App.

```
-(void) createPatternData {  
  
    NSMutableArray *mainDetails;  
    NSMutableArray *findSchools;  
    NSMutableArray *patterns;  
  
    patternSections=[[NSMutableArray alloc] initWithObjects:@"UKTF Home",@"Find Schools",@"Colour Belt Grades",nil];  
  
    mainDetails=[[NSMutableArray alloc] init];  
  
    findSchools=[[NSMutableArray alloc] init];  
  
    patterns=[[NSMutableArray alloc] init];  
  
    [mainDetails addObject:[NSMutableDictionary alloc] initWithObjectsAndKeys:@"Home  
Page",@"name",@"itf@2x.png",@"picture",@"http://www.uktf.com/index.php",@"url",nil];  
  
    [findSchools addObject:[NSMutableDictionary alloc] initWithObjectsAndKeys:@"Find Nearby  
Schools",@"name",@"itf@2x.png",@"picture",@"http://maps.google.com/maps?f=q&source=s_q&hl=en&geocode=&q=taekwondo,+g331ly&aq=&sl=37.062  
5,-95.677068&sspn=76.273886,80.244141&ie=UTF8&hq=taekwondo,&hnear=Glasgow+G33+1LY,+United+Kingdom&ll=55.856817,-  
4.182358&spn=0.427783,0.626907&z=11",@"url",nil];  
  
    [patterns addObject:[NSMutableDictionary alloc] initWithObjectsAndKeys:@"Pattern Videos",@"name",@"itf@2x.png",@"picture",nil];  
  
    patternData=[[NSMutableArray alloc] initWithObjects:mainDetails,findSchools,patterns,nil];  
  
    [mainDetails release];  
    [findSchools release];  
    [patterns release];  
}
```

Figure 7 - Creating the table view elements in Obj-C

The first element of functionality to be created was the home page view which simply consisted of an internally generated WebUIWebView element to display external url's from within the confines of an App to allow the user to access external information easily without leaving the application and accessing the browser. All that was required of the page was to load the request url from the main page and allow the web service to take over from there.

The second element, the map view was very much as similar to the home page in respect to only having to receive the parameters from the main page and pass on the information to the map view which accesses Google's Map services and does all the work from there.

The third element, patterns, is where much of the work was done in terms of putting strain on the device locally rather than passing off information to an external server. Other than creating a graphical gaming environment video playback is the best way to test how well an application can take the strain. The code for each button has to be created to initiate the media player framework. It

must source the file and the player object then add the player object to the view so it can be seen to the user and then the memory it uses during playback has to be handled either when the video completes or is quit. A simple auto release cant be used here or the application will crash, a separate object has to be created and tied to each instance of the media player objects that handle the memory de-allocation process to remove the object entirely as seen in figure 8.

```

-(IBAction)playSM:(id)sender {
    NSString *movieFile;
    MPMoviePlayerController *moviePlayer;

    movieFile=[[NSBundle mainBundle]pathForResource:@"Saju Makgi" ofType:@"m4v"];

    moviePlayer=[[MPMoviePlayerController alloc] initWithContentURL:[NSURL URLWithString:movieFile]];

    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(playSMFinished:) name:MPMoviePlayerPlaybackDidFinishNotification
    object:moviePlayer];

    [self.view addSubview:moviePlayer.view];
    [moviePlayer setFullscreen:YES animated:YES];
    [moviePlayer play];
}

-(void)playSMFinished:(NSNotification*)theNotification {
    MPMoviePlayerController *moviePlayer=[theNotification object];

    [[NSNotificationCenter defaultCenter] removeObserver:self name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer];

    [moviePlayer.view removeFromSuperview];
    [moviePlayer release];
}

```

Figure 8 - Media playback controls in obj-C

4.2.2 Testing Plan

This section demonstrates the testing that the Objective-C App went through during its development phase to ensure the functionality executed properly.

The tests starts with the main page then move on in order with the home page, map page and finally the patterns page where the majority of the testing takes place. The full test plan for the Objective-C App can be found in Appendix G. The tests were largely successful in that only the occasional bug was discovered and most were due to badly formed code elements in the patterns page describing the composition of the media player elements to play and stop the pattern videos.

As seen in figure 9 the construction of the main page of the Obj-C app causes several problems in the beginning of the development phase. Due to the complicated nature of the code required to create the proper table view user interface element described earlier in figure 7 the result was various code errors that had to be rectified to provide proper functionality to the App

3	Select home page button	View changes to home page view and displays the UKTF homepage	Home page shown incorrectly	Reset the display parameter in interface builder to scale to fit the page
4	select map page button	View changes to the google map view and displays nearby TKD schools	No schools shown	Problem corrected by appending the google map API url with the proper parameters
5	Select pattern page button	View changes to patterns page and displays the descriptions of each grade and the buttons to play the videos	Patterns page shown	
6	Interact with UKTF homepage	The page should function as a normal webpage	Page functions normally	
7	Navigate back to main page	View changes back to main page view	Option to revert not present	Add the proper root value to the table in interface builder so the view knows where its location is in the stack

Figure 9 - obj-C main page test excerpt

The only other real area of difficulty encountered during the testing was the implementation of the videos where several of the complicated media player objects as illustrated earlier in figure 8 would not work and the faults had to be tracked down and corrected.

4.3 Develop QCi App

This section covers the descriptions of the incremental development phases of the QuickConnect app and details how each of the functions were carried out by providing code examples and explaining their functions.

4.3.1 Incremental Development phases

As the QCi framework integrates itself into the iOS SDK the QCi development process is extremely similar to a standard Objective-C development apart from the physical coding and code generation of the UI creation where the QCi framework uses XHTML CSS and JavaScript to manage user input and communications with the web or the application database.

The QCi development is very different from that of the Obj-C as it is effectively a web application created within the confines of an iOS Applications WebUIWebView element and as such is subject to all the rules of web applications relating to their development and abilities. The views of the App are represented as an XHTML document styled with CSS and a JavaScript document provides the client side interactive functionality. The code in figure 10 provides an example of the main user interface code for the QCi App, the remainder of which can be found in Appendix F. The index page of the App contains all the necessary code to create the elements for the application including the HTML 5 “video” tags to play the locally stored media.

As all the user interface and much of the functionality code is contained in one XHTML page each DIV tag separates the different views of the App into their component parts where only one of which is displayed at a time, the default being “view 1” when the page loads

```
<body onload="load();">
<div id='view1' class='view'>
<div id='header'>
<br/>
<h2 id='main'>Main Page</h2>
</div>
<p>To go to the UKTF Home Page, Touch the button below.</p>
<a href="http://www.uktf.com/"><input id='buttons' type='button' value='Go to Home Page' onclick='load web page'/></a>
<br/>
<p>Touch the button below to view a map of the nearby TaeKwon-Do Schools.</p>
<a href="http://maps.google.com/maps?f=q&source=s_q&hl=en&geocode=&q=taekwondo,+g331ly&aq=&sl=37.0625,-95.677068&sspn=76.273886,80.244141&ie=UTF8&hq=taekwondo,&hnear=Glasgow+G33+1LY,+United+Kingdom&ll=55.856817,-4.182358&spn=0.427783,0.626907&z=11"><input id='buttons' type='button' value='Show Map to locate Nearby Schools' onclick='load web page'/></a>
<br/>
<p>Touch the button below to go to the Patterns Archive Page.</p>
<input id='buttons' type='button' value='Go To Patterns Archive' onclick='showView(4)'/>
</div>

<div id='view4' class='view' style='background-color:white;'>
<div id='header'>
<br/>
<h3>Patterns Archive</h3>
</div>
<br/>
<input id='buttons' type='button' value='Return To Main Menu' onclick='showView(1)'/>
<br/>
<p>Saju jirugi is the first exercise taught to white belt sutents as they begin their training. the video bellow demonstrates it.</p>
<video id='Saju_Jirugi' src='../Saju Jirugi.m4v' controls='true' height=100 width=150>Not supported</video>
<br/><br/>
<p>Saju Makgi is the secon exercise taught to white belt sutents as they begin their training. the video bellow demonstrates it.</p>
<video id='Saju_Makgi' src='../Saju Makgi.m4v' controls='true' height=100 width=150>Not supported</video>
<br/><br/>
<p>Once a student passes their first grading and reaches yellow tags they learn their first real pattern Chon Ji, The video bellow demonstrates it.</p>
.....
.....
...
```

```
</div>
</body>
```

Figure 10 - index.html page from QCi App

The JavaScript code in figure 11 provides the functionality for the App to switch between the views represented as div tags in the html document to simulate various pages of the App. The JavaScript functions are used to avoid loading times between pages and instead alters the page DOM by changing the visibility settings of each division. The load function displays view 1 to the user that displays the main view of the App, the showView App changes the appropriate style attributes to changes each views position and visibility.

```
function load()
{
    showView(1);
    view4.style.visibility = 'hidden';
}

var currentView = null;

function showView(viewNumber){
    var nextView = document.getElementById('view'+viewNumber);
    //if there is a view to change to
    if(nextView){
        //if at least one view has been displayed
        if(currentView){
            currentView.style.zIndex = 0;
            currentView.style.opacity = 0;
            currentView.style.visibility = 'hidden';
        }
        nextView.style.zIndex = 10;
        nextView.style.opacity = 1.0
        nextView.style.visibility = 'visible';
        //reset currentView
        currentView = nextView;
    }
}
```

Figure 11 - Main JavaScript page for client functionality

4.3.2 Testing Plan

This section demonstrates the testing that the QuickConnectiPhone App went through during it's development phase to ensure the functionality executed properly, as the functionality of the two apps are identical then the testing plans are very much alike.

The tests starts with the main page then move on in order with the home page, map page and finally the patterns page where the majority of the testing takes place. The full test plan for the QuickConnect App can be found in Appendix G. The tests were largely successful in that only the occasional bug was discovered and most were due to badly formed code elements in the html divs and the showvalue function. This was largely due to the more simplistic nature of the QCi App, the complicated objects that were required to create the table view and media player objects in the objective-C code were not present in the QCi example, thus reducing the complexity of the final App.

5. Results Evaluation

In this section the information gathered from the development logs of each App and test results from Instruments will be compared and evaluated after each has been completed, looking for any indication that the use of one method or the other provides a superior quality app without extending the time of the development or increasing the difficulty of the creation. As there has been no significant reports on there being an issue such as this with using third party development tools it is expected that any effect brought on by the use of QCi will not be of significant threat to the Apps performance if it exists at all.

5.1 Ease of Development

The ease of development section deals with analysing the results from each individual development log to determine any noticeable shortcomings or issues that apparent during either method of development. Both of the complete development logs can be found in Appendix H while they are analysed in detail in this section.

5.1.1 Objective-C Development log

This section contains the Analysis of the Objective-C App development log which is split into two sections as the UI was created separately from the code in interface builder whereas the code was created in Xcode and so there are two sections here.

5.1.1.1 UI Design

The construction of the User interface for the Objective-C App as it had already been mocked up and prototyped before development even began so when everything was put in place to create the UI completely there were very few issues.

The first task to be undertaken for the UI was to build the table view within the application rootViewController object and connect it to the datasource to receive the needed information to populate the cells and attach it to the file owner directory so it could be associated with the rest of the project. This initially caused some problems as it wasn't recognizing the stack but was easily resolved by digging down the layers and properly assigning the name of the root element.

The homepage and mapPage views were dropped into the UI from the element library in interface builder and were configured with little effort. It wasn't until the creation of the patterns page that any other issues arose, when creating the scroller element it was observed that the object wasn't functioning as intended when initial tests were conducted although this was rectified with the addition of an additional onload function within the objective C code of the project to initiate the scrolling effect and set its size parameters.

5.1.1.2 Development

The development of the App was relatively straightforward in terms of construction; there were four main areas to code, the main page, home page, map page and patterns page. Each increment of development was recorded separately as it was created and so the log in Appendix H for the Obj-C App Development phase is sectioned off into the various increments of the App.

The code on the main page to create the objects to contain the necessary information and assign them to the proper elements of the table view and then have those cells be properly formatted required a fairly complex array of objects to perform properly. This issue caused

several problems that inevitably required hours of meticulous code tweaks and alterations until the desired effect was achieved and the hierarchy of the code was properly aligned.

The home page and Map page view, both essentially being nested view elements that passed on urls and parameters to external web services required very little work to function. The home page was created without any incident and ran on the first attempt, while the map page only failed to work when the urls were incorrectly formatted within the code, an error easily fixable by editing the url element passed from the main page.

The patterns page required the largest amount of physical code creation although much was code replication as many of the objects performed the same function with different resources. When creating the original code object to initiate the media player object to play the source file the object failed to load at all as the release mechanism for the object required a second special function to handle the memory management of the video player object which couldn't be handled normally. Once this second handler was created and properly formatted to the source the video played and exited properly. After the first video player object was created the code was replicated and edited to fit the new source files, this process only led to minor coding errors that were easily rectified.

5.1.1.3 Conclusions

Based on the experience of developing with Objective-C the process of creating application user interface and functionality separately with very loose coupling is an extremely efficient and practically useful method of development. The user interface view and the functionality model were entirely separate from each other and only passed messages and updates to one another. This allowed for a very issue free user interface development that could be easily transferable to another App should it be needed to.

The object oriented development process of the Objective-C development made for efficient development and use of resources as long as the developer was properly aware of the resources being implemented at any one time.

5.1.2 QuickConnectiPhone Development log

This section contains the analysis of the QuickConnectiPhone App development log which differs from the Objective-C App log as the UI and the implementation was contained within the same documents and so was run concurrently. As such it was decided to combine the logs into one complete development log for this App.

5.1.1.1 Development

The development of the QuickConnectiPhone App was identical to that of a rich internet application in that it used XHTML with CSS and JavaScript to display information to the user and manipulate the onscreen elements based on user input.

As the QCi App was just like any other web application it followed the same procedure for development. The division elements were created in the html page to create the different sections of functionality, a different DIV tag was created for each view in the App instead of linking to different html pages which could have been used but the view method removed the loading times from the page requests.

The only real issues when creating the html design and layout of the page was the stylesheet, due to the confines of the small iPhone screen objects were proving to be tricky to control in their placement although after trial and error of various style elements the design was completed.

Creating the functionality for the App was done with a mixture of JavaScript and HTML tags, the buttons placed on the main page would load each of the three other views in the App without navigating away from the page by issuing a JavaScript function call to alter the style attributes of the proper divisions to display only the selected view. The issue with the JavaScript function was making it display the new view properly while entirely hiding the other views, a process which again involved trial and error of stylesheet attribute manipulation within the function.

The homePage view and mapPage view functionality was very simple as all it was a link to the external resource which then loaded the page to be manipulated by the browser. As this was a very simple process there were no problems encountered

The patterns page of the QCi App proved to be extremely simple in comparison to the Obj-C App as all that was required to properly play the video files was a standard video tag with a reference to the file location. When the element was touched the video played in fullscreen mode without any problems.

5.1.1.2 Conclusions

In terms of simplicity the QCi App was far easier to create and implement than the Obj-C App was, resulting in a shorter development time in the area of several days. Although the development was simpler the QCi App was essentially just a web application running natively on the device and as such was in part restricted to the functionality of a web application. The App could make use of the iDevices mediaplayer, access GPS information, record and play audio files and embed Google maps, in addition to these it is still a downloadable application from the app store instead of an ordinary web app accessed through a browser. In particular the QCi App far exceeds the Obj-C app in the area of media play back; the QCi App required a single video tag element to play the source file where the Obj-C app required two very large functions to handle the playback, resulting in much more effort to achieve the same functionality

5.2 Adherence to Guidelines

This section looks at the Apple design, development and submission guidelines investigated earlier in the report and analyses how closely the test Apps match up to the requirements laid down by Apple to be classed as an App suitable for the App store.

5.2.1 Objective-C App

As the Objective-C was built using Apple technology it is easier to meet the App requirements as many are met simply by the way Objective-C and Interface builder functions. As previously investigated the The App submissions guidelines document contains six relevant section to the test Apps that they will be compared against. The sections are; functionality, location. Media content, user interface, violence, objectionable content and privacy.

The functionality section describes what will not be acceptable functions within a submitted App, features such as hidden content, spam, duplicates apps that crash or exhibit bugs and apps that are just marketing advertisements. Through the development log and testing plans it has been proven that the App does not crash or exhibit bugs. There are no hidden features within the App, it is not an advertisement, it does not attempt to duplicate an already existing App and it does not spam the user.

The location section applies to the use of geo-location services, stating that an app must indicate to the user when geo-location will be used and not record or transmit the users location data without the user knowledge. The test App only needs to use geo-location within the map Page of the App and as seen in figure 12 the user is prompted by the application to allow the user of location services.



Figure 12 - location services permissions

The media content section refers to the use of video or audio within Apps that must use Apple's media player framework when playing such files, either local or web based, whereas web based file must not be over a specified size and bit rate. The test App does make use of Apple's media player framework when playing the patterns videos and all the files are stored locally.

The user interface states that the submitted App must comply with Apples HCI guidelines, it cant look similar to Apple's first party iOS Apps and interface that are not simple and easy to use for the first time user will be rejected. The Test app only makes use of the interface builder elements in the standard way were intended to be used for, it does not look like Apple's first party apps and the UI is very simple and easy to use in an intuitive way for the first time user.

The violence and objectionable content sections are included in the guidelines summary due to the nature of the test App as a martial arts reference. The sections state that graphic depictions of realistic images of people or animals being maimed or killed, apps that depict violence to children, involve realistic depictions of weapons being misused and apps that present excessively rude or objectionable content will be rejected. The test App does not depict graphic images of violence or death to anyone, it only contains detailed demonstrations of taekwondo patterns by a black belt instructor although it could be construed by some as depicting the use of weapons.

The privacy section states that an app cannot transmit any of the users data without their knowledge, require users to share private information or those that target minors for data collection will be rejected. The test App does not collect any user data or even require any input from the user, the only piece of information it requests from the user is for the locations services in the map page.

5.2.2 QuickConnectiPhone App

As the QuickConnectiPhone uses the iOS SDK to launch the developed App it makes complying to Apple's guidelines very well. The App submissions guidelines document for the QCi App contains the same six relevant section to the test Apps that they will be compared against. The sections are; functionality, location. Media content, user interface, violence, objectionable content and privacy.

The functionality section of the document describes what will not be acceptable functions within a submitted App where features such as hidden content, spam, duplicates, apps that crash or exhibit bugs and apps that are just marketing advertisements. In the same way as with the Objective-C App the QuickConnect App was tested through the development log and testing plans it has been proven

that the App does not crash or exhibit bugs. There are no hidden features within the App, it is not an advertisement, it does not attempt to duplicate an already existing App and it does not spam the user.

The location section of the document applies to the use of geo-location services within an App, stating that an app must indicate to the user when geo-location will be used and not record or transmit the users location data without the user knowledge. The QCi test App, like the Obj-C App only needs to use geo-location within the map Page of the App and a the user is prompted by the application to allow the user of location services before any data is collected.

The media content section refers to the use of video or audio within the submitted App, it states that the App must use Apple's media player framework when playing such files, either local or web based, whereas web based file must not be over a specified size and bit rate. The test QCi App does make use of Apple's media player framework when playing the patterns videos and all the files are stored locally even though it does not use obj-C code to create the interface for the media player object it still makes use of the underlying framework in the same way any web base video does.

The user interface section states that the submitted App must comply with Apples HCI guidelines; it can't look similar to Apple's first party iOS Apps and interface that are not simple and easy to use for the first time user will be rejected. As the QCi test app was created using web application technology instead of Apple's interface builder greater care was taken to make the App comply with the guidelines. Additionally the app does not look like Apple's first party apps and the UI is very simple and easy to use in an intuitive way for the first time user.

The violence and objectionable content sections are included in the guidelines summary due to the nature of the test App as a martial arts reference. These sections state that graphic depictions of realistic images of people or animals being maimed or killed, apps that depict violence to children, involve realistic depictions of weapons being misused and apps that present excessively rude or objectionable content will be rejected. The QCi test App does not depict graphic images of violence or death to anyone; it only contains detailed demonstrations of taekwondo patterns by a black belt instructor although it could be construed by some as depicting the use of weapons.

The privacy section of the document states that an app cannot transmit any of the users data without their knowledge, require the user to share private information or target minors for data collection will be rejected. The test App does not collect any user data or even require any input from the user, the only piece of information it requests from the user is for the locations services in the map page

5.3 Instruments Benchmark Tests

This section covers the evaluation of the instruments application benchmark tests of the two test Apps. As each App is completed Instruments will be used to run an analysis of its performance and efficiency when performing tasks in a real time test of its abilities, menus will be navigated, videos streamed from the internet and interaction with Google maps. Instruments allows the user to define a custom set of tests to apply to a real time assessment of the applications process and displays the results in a line graph against time which can be used in a side by side comparison which will show any differences between the two Apps once testing is complete.

The Instruments test are split into four tests; the cpu monitor, memory leaks monitor, memory allocations monitor and the activity monitor. These tests run the application in real time while monitoring the defined attributes of the test. The issue with this is as the tests are run in real time it becomes problematic to compare two separate tests unless you have a frame of reference to apply to the two. Figure x illustrates the time frame set up for the experiments so that as each time the test is run with a different monitor attached on both test apps the time each action is executed will be approximately the same.

Table 1 - Instruments testing timeline

Time Stamp	Action
5s	Press home page button and navigate page
15s	Press back button
25s	Press Map page button and navigate map
40s	Press back button
45s	Press patterns button
50s	Press play on the first video and allow it to end
85s	Play the second video and allow it to end
110s	Press play on the third video and quit 10 seconds in
140s	End test

5.3.1 Objective-C App

As the Objective-C App is the control App for the project it was tested against the instruments benchmarks before the QuickConnect App to give an idea of what a standard App looks like when it is run through the instrument tests. Although the analysis of the benchmark tests include screens from the instruments application the full copy of the tests can be seen in Appendix I.

The first test to be run was the Activity monitor test to identify what action during the running of the app put the most load on the system, it was found that the Map page cause the most system load due to the loading of each successive image as the map was panned and zoomed. The video player came close to the level of system load as the map but failed to reach the same heights as what the map was demanding of the system as seen in figure 13, although it did last much longer due to the length of the videos being played.

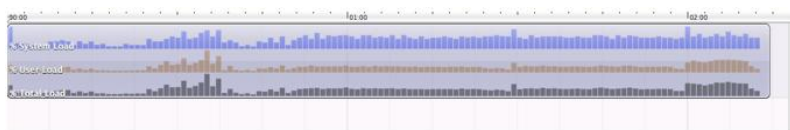


Figure 13 - obj-c activity monitor

The next test was the memory allocations test to identify what action cause the most system memory to be occupied, as seen in figure 14 it was found that the videos section by far required the largest allocation of memory during the playback of the pattern videos stored on file, some files requiring more than others due to some variations in quality.

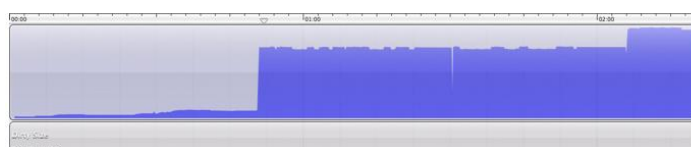


Figure 14 - memory allocation

The CPU monitoring test found that the map application again cause the most system load causing the cpu monitor to spike largely as the map was interacted with as seen in figure 15. Again this is believed to be due to the amount of images being loaded by the application when the map is being panned and zoomed.

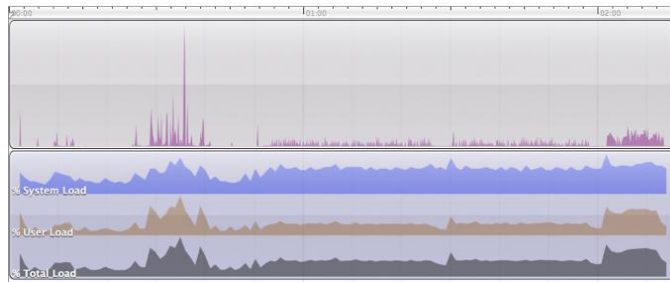


Figure 15 - CPU monitor

The final test was to look for memory leaks within the application when an action is taken and the memory being used by the object is not properly dealt with. Figure 16 shows that the use of the map and the playing of the first video of the patterns page cause sizable chunks of memory to be dropped by the system. It is believed this is due to the special way the application handles the memory of these tasks instead of the manual approach applied to most application actions.

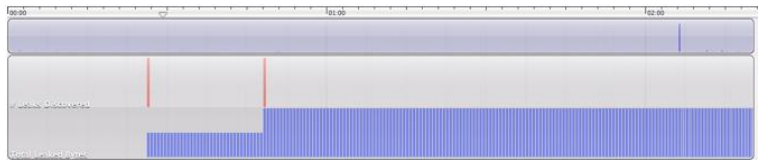


Figure 16 - Memory Leaks Test

5.3.2 QuickConnectiPhone App

The tests for the QuickConnect App were carried out in exactly the same manner as the objective-C tests were done to ensure the comparability of the two sets of data. The full test printouts for the QuickConnect benchmarks can be found after the Objective-C tests in Appendix I.

From the analysis of the Activity monitor and memory allocation tests for the QCi App the results appeared to be identical to the previous tests of the Obj-C App, where in the same way it was found that the map page cause the most system activity and system load, while the video playback by far cause the most memory allocations of the entire App but only while playback.

The tests for the CPU monitoring are where the two Apps begin to differ in their output from instruments. As seen in figure 17 the spike present in the Obj-C app CPU monitoring test is missing in the test for the QCi App, this could possibly be due to the way in which the QCi App runs within the UIWebView of the application framework or it could be a random occurrence.

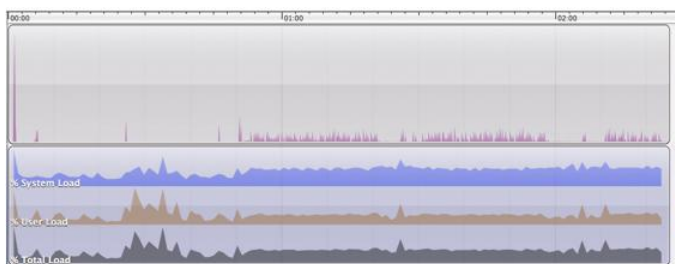


Figure 17 - QCi CPU monitor

It was found that the memory leaks test for the QCi App also differed from the Obj-C App but this time in a greater way than the CPU monitoring tests. As seen in figure 18 the point of the leaks is entirely different from the Obj-C App, this time appearing at the very beginning of the test when the UKTF home page is loaded and again a smaller leak when the first video I played. This differed from the Obj-C App as there were no leaks from the App until the map was accessed which in the QCi App cause no leak at all.

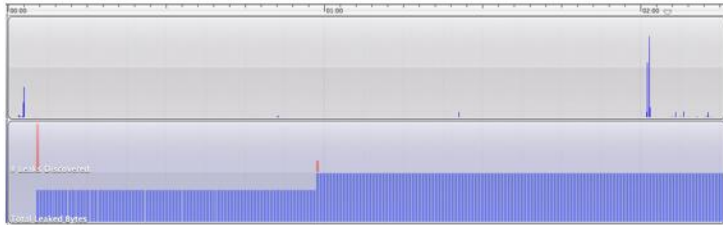


Figure 18 - QCi Memory Leaks

5.3.3 Conclusions

From the analysis of the two sets of testing data from the instruments benchmarks it can be seen that there is little functional difference between the running of the two Apps on this scale. Much of the results were either identical or very similar and on the test where the changes were noticeable in the tests they had little effect on the running of the application itself. Perhaps the memory leaks or allocations for the Apps would have an effect on a much larger application but that testing is beyond the scope of this project.

6. Final Conclusions

This section will contain all the final conclusions obtained from the test project. The project will be recapped summarising its goals and methods before reviewing the results of the testing on the project applications. The discussion will detail the tests that were executing and their findings alongside a commentary of the test results and what they indicate in relation to the initial hypothesis of the project. The limitations of the project will be discussed and any future work that could be undertaken should there be more time available in the future to pursue the subject deeper will be outlined.

6.1 Project Recap

With Apple's introduction of the ability to create native applications using third party tools, developers have been exploiting the opportunity to explore new avenues of development of iOS Apps. This new ability given to developers brings to light new questions about iOS development, chief among which is what is the best method to developing these Apps and is it worth it pursuing the alternative development methods. A literature review of the subject field was undertaken to investigate and understand the use of the most popular third party development frameworks made available to developers such as QuickConnectiPhone, to look into their benefits and drawbacks and the potential security issues they could be vulnerable to.

The aim of the project was to examine and answer the question:

Does the use of the QuickConnectiPhone software development framework provide a significantly measurable increase in the speed and ease of which iOS Apps can be created by developers opposed to the use of Apple's traditional Objective-C approach?

Based upon this the projects purpose was to evaluate the beneficial or impeding impact the use of the QuickConnect framework has on iOS App development. This was accomplished by means of a Develop and Test style project that aimed to create an iOS Application using both the objective-C method and the QuickConnect method, recording the development process of both apps, then performing evaluations on the created Apps to determine any functional differences between the two methods. The application specification was constructed from examination of existing applications to determine the appropriate level of functionality to include for the testing purposes. When the App requirements were specified simple paper screen user interface examples were created and then refined using Apple's own HCI guidelines document for iOS Applications. The two distinct finalised user interface designs were then implemented along with each of their respective applications during the development stage of the project which was logged for later comparison. Once the two applications were created the resulting development logs were compared and analysed to identify specific issues associated with either method of development. The test apps were then evaluated against Apple's HCI, app development and app submissions guidelines documentation to ensure both apps are viable for publication into the App Store to ensure equal quality of the two. The final test for the apps was to run them through the instruments benchmark test utility within the iOS SDK to look for memory leaks, cpu usage, memory usage and activity levels throughout the application during specific tasks. The resultant data gathered from these tests contributed largely to the the authors conclusions relative to the initial hypothesis that; the use of the QuickConnectiPhone development environment will significantly speed up the development of iOS Apps, reduce the difficulty of their creation and increase or at least maintain processing efficiency when operating.

6.2 Final Discussion of Results

The main aim of the project was to identify whether or not the use of third party development frameworks, specifically the QuickConnect family framework makes the process of iOS app development simpler without compromising quality. Several sets of result were presented chapter five of the report detailing the outcome of the three main tests of the project that were carried out on the two Apps. The first set of information examined in the results section was the development log that was created alongside the development of each App noting issues with the development that had to be

overcome to implement each specific piece of functionality. The next was an evaluation of the two test Apps against Apple's HCI, app development and app submissions guidelines documentation to ensure both apps are viable for publication into the App Store. Finally the two apps were subject to an array of instruments benchmark tests.

Application development logs

Based on the experience of developing with Objective-C the process of creating application user interface and functionality separately with very loose coupling is an extremely efficient and practically useful method of development. The user interface view and the functionality model were entirely separate from each other and only passed messages and updates to on another. This allowed for a very issue free user interface development that could be easily transferable to another App should it be needed to. The object oriented development process of the Objective-C development made for efficient development and use of resources as long as the developer was properly aware of the resources being implemented at any one time.

In terms of simplicity the QCi App was far easier to create and implement than the Obj-C App was, resulting in a shorter development time in the area of several days. Although the development was simpler the QCi App was essentially just a web application running natively on the device and as such was in part restricted to the functionality of a web application. The App could make use of the iDevices media player, access GPS information, record and play audio files and embed Google maps, in addition to these it is still a downloadable application from the app store instead of an ordinary web app accessed through a browser. In particular the QCi App far exceeds the Obj-C app in the area of media play back, the QCi App required a single video tag element to play the source file where the Obj-C app required two very large functions to handle the playback, resulting in much more effort to achieve the same functionality

Adherence to Guidelines

Based on the findings from the examination of the Apple guidance documents both applications appear to adequately meet Apple's requirements for applications to be submitted to the App Store. Both applications were compared to the six relevant sections of the Application submissions guidelines to ensure the applications are both suitable for submission. The sections are; functionality, location, media content, user interface, violence, objectionable content and privacy. Both applications met the requirements of the guidelines sections indicating that neither app was more or less eligible for submission to the App Store.

Instruments Benchmark Tests

From the analysis of the two sets of testing data from the instruments benchmarks it can be seen that there is little functional difference between the runnings of the two Apps on this scale. Much of the results were either identical or very similar and on the test where the changes were noticeable in the tests they had little effect on the running of the application itself. Perhaps the memory leaks or allocations for the Apps would have an effect on a much larger application but that testing is beyond the scope of this project. Based on these results it could be concluded that the use of third party applications to develop iOS apps does not affect the quality of the created application in any measurable ways.

6.3 Project Limitations and Future Work

While the results of the projects proved to support the authors initial hypothesis the project itself had a limited scope that imposed limitations on the research and the associated results. With a limitation of eight weeks for development of the two applications and further two weeks for testing of the applications, as indicated by the time scale of the application development the project was on a small

scale and as such any results gleaned from the testing of the applications would only apply to the small scale and not necessarily scale up to larger applications.

While the limitations of the project demonstrated that the results of the test may not necessarily scale up to larger applications they also suggest further work that can be undertaken with the project. The process of the development could be undertaken on a much larger scale to assess how the results may differ. As discovered in the literature review there are many more third party development frameworks that can be used in iOS development. With more time further work could be done to investigate the alternative development frameworks to identify how they compare to the objective-C model of iOS application development.

6.4 Conclusion

This project was undertaken to discover if the use of the third party development framework QuickConnectiPhone is beneficial to developers and is worth pursuing in favor of using the traditional objective-C approach. The two applications created were designed to illustrate the most commonly used functionality of iOS applications and determine which method proved to be the superior in terms of complexity and efficiency. The results from the application development logs, the Apple guideline documents and the Instruments benchmark tests all supported the initial hypothesis that the use of the third party framework would be beneficial to developers, especially those who already possess a background in web technologies as the QCi App was created much faster and with less effort than the objective-C App. Even though the QCi App exhibited these benefits they were achieved by undermining the structural soundness of the obj-C app style, they proved to be a possible security risk to the platform through the heavy reliance on JavaScript technologies and the existing Apple documentation was largely useless when not using obj-C.

Fundamentally the initial project hypothesis was proven to be correct in that the use of the QuickConnectiPhone framework did in fact reduce development time of the application by allowing the developer to use the standard web technologies HTML, CSS and JavaScript in favor of objective-C without harming the quality of the functionality within the created application.

7. References

- Android Developers (2011) Platform Versions.
<http://developer.android.com/resources/dashboard/platform-versions.html> (accessed February 15 2011)
- Apple inc (2010) [A] Apple App Store. <http://www.apple.com/iphone/appstore/> (accessed October 5, 2010)
- Apple inc (2010) [B] Apple iOS Reference Library,
<http://developer.apple.com/library/iOS/navigation/index.html>
- Apple inc (2010) [C] iPhone Human Interface Guidelines.
<http://developer.apple.com/library/iOS/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html> (accessed October 5, 2010)
- Apple inc (2010) [D] iOS Application Programming Guide.
http://developer.apple.com/library/iOS/#documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Introduction/Introduction.html#/apple_ref/doc/uid/TP40007072 (accessed October 5, 2010)
- Apple inc, (2011) [E] App store Guidelines (accessed October 24, 2010)
<http://stadium.weblogsinc.com/engadget/files/app-store-guidelines.pdf>
- Apple inc (2010) [F] Over 250,000 ways to make the iPhone even better. (accessed November 2 2010)
<http://www.apple.com/iphone/apps-for-iphone/>
- Apple inc (2010) [G] Developing for performance. (accessed January 13th 2011)
http://developer.apple.com/library/mac/#documentation/Performance/Conceptual/PerformanceOverview/Introduction/Introduction.html%23/apple_ref/doc/uid/TP40001410-CH202-SW1
- Apple inc (2010) [H] Objective-C.
<http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/> (accessed October 5, 2010)
- Arthur, C (2010) Apple's amazing numbers: where did the iPhone sales come from? (accessed October 24, 2010) <http://www.guardian.co.uk/technology/blog/2010/apr/21/apple-financial-results-analysis>
- Ballard, B (2007) Designing the Mobile User Experience. John Wiley & Sons (accessed October 12, 2010) http://www.amazon.co.uk/Designing-Mobile-Experience-Barbara-Ballard/dp/0470033614/ref=pd_sim_b_1
- Barney, S.B (2009) Developing Hybrid Applications for the iPhone: Using HTML, CSS, and Javascript to build dynamic Apps for the iPhone. Addison Wesley (accessed October 3, 2010)
- Canali, C, Colajanni, M, & Lancellotti, R (2010). Resource Management Strategies for the Mobile Web. Mobile Networks and Applications.(Accessed October 12, 2010)
<http://proquest.umi.com/pqdweb?did=1979003521&sid=2&Fmt=2&clientId=6297&RQT=309&VName=PQD>
- Chris M (2009) 18 Mobile Frameworks and Development tools for Creating iPhone Apps. (accessed January 9, 2011) <http://iphoneized.com/2009/11/18-mobile-frameworks-development-tools-creating-iphone-apps/>
- Chronic Dev Blog 2011, [online]. Available at: <http://chronic-dev.org/blog/> (Accessed March 3rd, 2011)
- Etherington D (2010) iPhone Dominating Worldwide Smartphone Usage Report, GIGAOM (accessed October 29 2010) <http://gigaom.com/apple/iphone-dominating-worldwide-smartphone-usage-report/>

Ganchev, I., Stojanov, S., O'Droma, M. & Meere, D. (2006), *An InfoStation-Based Multi-Agent System for the Provision of Intelligent Mobile Services in a University Campus Area*. (Accessed January 12, 2011)

Golafshani, N. (2003), "Understanding reliability and validity in qualitative research", *The Qualitative Report*,.

Griswold, W., Shanahan, P., Brown, S., Boyer, R., Ratto, M., Shapiro, R. & Truong, T. (2004), "ActiveCampus: Experiments in Community-Oriented Ubiquitous Computing," *Computer*, pp. 73-81".

Hardigree, M. (2011) Even Toyota is 'Apple's Bitch' <http://www.gizmodo.com.au/2011/04/even-toyota-is-apples-bitch/> (accessed April 16th, 2011)

Hickson, I. (2011) The Web Standards Project Acis Tests. (Accessed April 15th, 2011) <http://www.acidtests.org/>

International Data Corporation IDC (2010) Apple Joins Top Five Mobile Phone Vendors as Worldwide Market Grows Nearly 15% in Third Quarter, According to IDC (accessed October 29, 2010) <http://www.idc.com/about/viewpressrelease.jsp?containerId=prUS22550010§ionId=null&elementId=null&pageType=SYNOPSIS>

Jones, M. Marsden, G. (2005) *Mobile Interaction Design*. John Wiley & Sons (Accessed October 12, 2010) <http://www.amazon.co.uk/Mobile-Interaction-Design-Matt-Jones/dp/0470090898>

Krill PK (2010). The Wild west of third-party iPhone development. Infoworld.com. (accessed October 5, 2010) <http://www.infoworld.com/d/developer-world/the-wild-west-third-party-iphone-development-857>

Landesman, M. 2011, Mac Virus FAQs: Do You Really Need Mac Antivirus Software? <http://antivirus.about.com/od/macintoshresource/tp/macvirusfaqs.htm> (Accessed March 3rd, 2011)

LogMein inc (2011) <https://secure.logmein.com/> (Accessed January 14, 2011)

Miller, B.A. & Chatschik, B. 2001, *Bluetooth Revealed*, 2nd edn, Prentice Hall PTR, Upper Saddle River, NJ, USA. Miller, C., Honoroff, J. & Mason, J. 2007, Security Evaluation of Apple's iPhone.[online]. Available at: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.3254&rep=rep1&type=pdf> (Accessed March 2nd, 2011)

Miller, P. (2010), JailbreakMe using PDF exploit to hack your iPhone, so could the baddies; Apple looking into reports -- Engadget [online]. Available at: <http://www.engadget.com/2010/08/03/jailbreakme-using-pdf-exploit-to-hack-your-iphone-so-could-the/> (Accessed March 3rd, 2011)

Naismith, L., Sharples, M. & Ting, J. (2005), "Evaluation of CAERUS: A Context Aware Mobile Guide", *Proceedings of mLearn 2005 - Mobile technology: The future of learning in your hands*, Cape Town, South Africa.; mLearn 2005 (Accessed January 12, 2011)

Number Of (2010) How many iPhones have been sold worldwide? <http://www.numberof.net/number-of-iphones-sold-2/> (accessed February 15, 2011)

Oates, B.J. (2006), *Researching information systems and computing*, Sage Publications Ltd, London.

Orders in seconds in (2011) OIS iMobile <http://itunes.apple.com/us/app/ois-imobile/id411806855#> (accessed January 17, 2011)

Pandya, V.R. 2008, iPhone Security Analysis.[online]. Available at: http://www.cs.sjsu.edu/faculty/stamp/students/pandya_vaibhav.pdf (Accessed March 3rd, 2011)

Pash (2010) Your smartphone is a better PC than your desktop ever was or will be. lifehacker
<http://lifehacker.com/5681573/your-smartphone-is-a-better-pc-than-your-pc-ever-was-or-will-be>

PCWorld (2010) Android Apps hit 100K, Developers not going wild (accessed November 2 2010)
http://www.pcworld.com/article/208743/android_apps_hit_100k_developers_not_going_wild.html

PhoneGap App listings (2011) <http://www.phonegap.com/apps> (accessed January 15, 2011)

QuickConnect Wiki (2011) <http://quickconnect.pbworks.com/w/page/9183363/FrontPage> (Accessed January 15 2011)

QuickConnectFamily Page (2011) <http://www.quickconnectfamily.org/> (Accessed January 14, 2011)

QuickConnectFamily Google Group (2011)
http://groups.google.com/group/quickconnectiPhone/browse_thread/thread/890f841094525cfb (Accessed January 16, 2011)

Ray, J. R (2010) Sams Teach Yourself iPhone Application Development in 24 Hours 2nd edition. Sams. (accessed October 3, 2010)

Roehrl, A. Schmiedl, S. (2002) Objective-C: the more flexible C++. (Accessed April 4th, 2011)
<http://www.linuxjournal.com/article/6009>

Rutman, M (1997) C++ Versus Objective-C (Accessed April 12th, 2011)
<http://www.mactech.com/articles/mactech/Vol.13/13.03/CandObjectiveCCompared/>

Saumya, R (2009) 10 Development environments for the iPhone. (accessed January 10, 2011)
<http://saumyaray.wordpress.com/2009/08/03/10-development-environments-for-iphone/>

Schroeder S (2010) AdMob: Smartphone Usage Up, iPhone and Android on the Rise, mashable (accessed October 2010) <http://mashable.com/2010/03/25/smartphones-iphone-android/>

Shankland SS (2010) Adobe resurrects Flash to iPhone App tool. http://news.cnet.com/8301-30685_3-20016033-264.html (accessed October 5, 2010)

Stark, J. S (2010) Building iPhone Apps with HTML, CSS, and Javascript: Making App Store Apps Without Objective-C of Cocoa. O'Reilly Media. (accessed October 3, 2010)

Teton Technical 2008 QuickConnect iPhone: an iPhone UIWebView hybrid framework (accessed November 2 2010) <http://tetontech.wordpress.com/2008/05/28/quickconnect-iphone-an-iphone-hybrid-framework/>

Wauters, R. (2011) Grapple Mobile Wants You to know they did not Raise \$10 million (accessed April 3rd, 2011) <http://techcrunch.com/2011/01/20/grapple-mobile-wants-you-to-know-they-did-not-raise-10-million/>

White, J. 2010, Cydia – Ten Essential Tweaks And Apps :: App Advice [online]. Available at: <http://appadvice.com/appnn/2010/10/cydia-essential-tweaks-apps/> (Accessed March 3rd, 2011)

Appendix A - Apple Developer Guidelines Document (Sections Employed in Project)

Functionality

- 2.1 Apps that crash will be rejected
- 2.2 Apps that exhibit bugs will be rejected
- 2.3 Apps that do not perform as advertised by the developer will be rejected
- 2.4 Apps that include undocumented or hidden features inconsistent with the description of the app will be rejected
- 2.5 Apps that use non-public APIs will be rejected
- 2.6 Apps that read or write data outside its designated container area will be rejected
- 2.7 Apps that download code in any way or form will be rejected
- 2.8 Apps that install or launch other executable code will be rejected
- 2.9 Apps that are "beta", "demo", "trial", or "test" versions will be rejected
- 2.10 iPhone apps must also run on iPad without modification, at iPhone resolution, and at 2X iPhone 3GS resolution
- 2.11 Apps that duplicate apps already in the App Store may be rejected, particularly if there are many of them
- 2.12 Apps that are not very useful or do not provide any lasting entertainment value may be rejected
- 2.13 Apps that are primarily marketing materials or advertisements will be rejected
- 2.14 Apps that are intended to provide trick or fake functionality that are not clearly marked as such will be rejected
- 2.15 Apps larger than 20MB in size will not download over cellular networks (this is automatically prohibited by the App Store)
- 2.16 Multitasking apps may only use background services for their intended purposes: VoIP, audio playback, location, task completion, local notifications, etc
- 2.17 Apps that browse the web must use the iOS WebKit framework and WebKit Javascript 2.18 Apps that encourage excessive consumption of alcohol or illegal substances, or encourage minors to consume alcohol or smoke cigarettes, will be rejected
- 2.19 Apps that provide incorrect diagnostic or other inaccurate device data will be rejected
- 2.20 Developers "spamming" the App Store with many versions of similar apps will be removed from the iOS Developer Program

Location

- 4.1 Apps that do not notify and obtain user consent before collecting, transmitting, or using location data will be rejected
- 4.2 Apps that use location-based APIs for automatic or autonomous control of vehicles, aircraft, or other devices will be rejected
- 4.3 Apps that use location-based APIs for dispatch, fleet management, or emergency services will be rejected

Media content

- 9.1 Apps that do not use the MediaPlayer framework to access media in the Music Library will be rejected
- 9.2 App user interfaces that mimic any iPod interface will be rejected
- 9.3 Audio streaming content over a cellular network may not use more than 5MB over 5 minutes
- 9.4 Video streaming content over a cellular network longer than 10 minutes must use HTTP Live Streaming and include a baseline 64 kbps audio-only HTTP Live stream

User interface

- 10.1 Apps must comply with all terms and conditions explained in the Apple iPhone Human Interface Guidelines and the Apple iPad Human Interface Guidelines
- 10.2 Apps that look similar to apps bundled on the iPhone, including the App Store, iTunes Store, and iBookstore, will be rejected
- 10.3 Apps that do not use system provided items, such as buttons and icons, correctly and as described in the Apple iPhone Human Interface Guidelines and the Apple iPad Human Interface Guidelines may be rejected
- 10.4 Apps that create alternate desktop/home screen environments or simulate multi-app widget experiences will be rejected
- 10.5 Apps that alter the functions of standard switches, such as the Volume Up/Down and Ring/Silent switches, will be rejected
- 10.6 Apple and our customers place a high value on simple, refined, creative, well thought through interfaces. They take more work but are worth it. Apple sets a high bar. If your user interface is complex or less than very good it may be rejected

Violence

- 15.1** Apps portraying realistic images of people or animals being killed or maimed, shot, stabbed, tortured or injured will be rejected
- 15.2** Apps that depict violence or abuse of children will be rejected
- 15.3** "Enemies" within the context of a game cannot solely target a specific race, culture, a real government or corporation, or any other real entity
- 15.4** Apps involving realistic depictions of weapons in such a way as to encourage illegal or reckless use of such weapons will be rejected
- 15.5** Apps that include games of Russian roulette will be rejected






Objectionable content

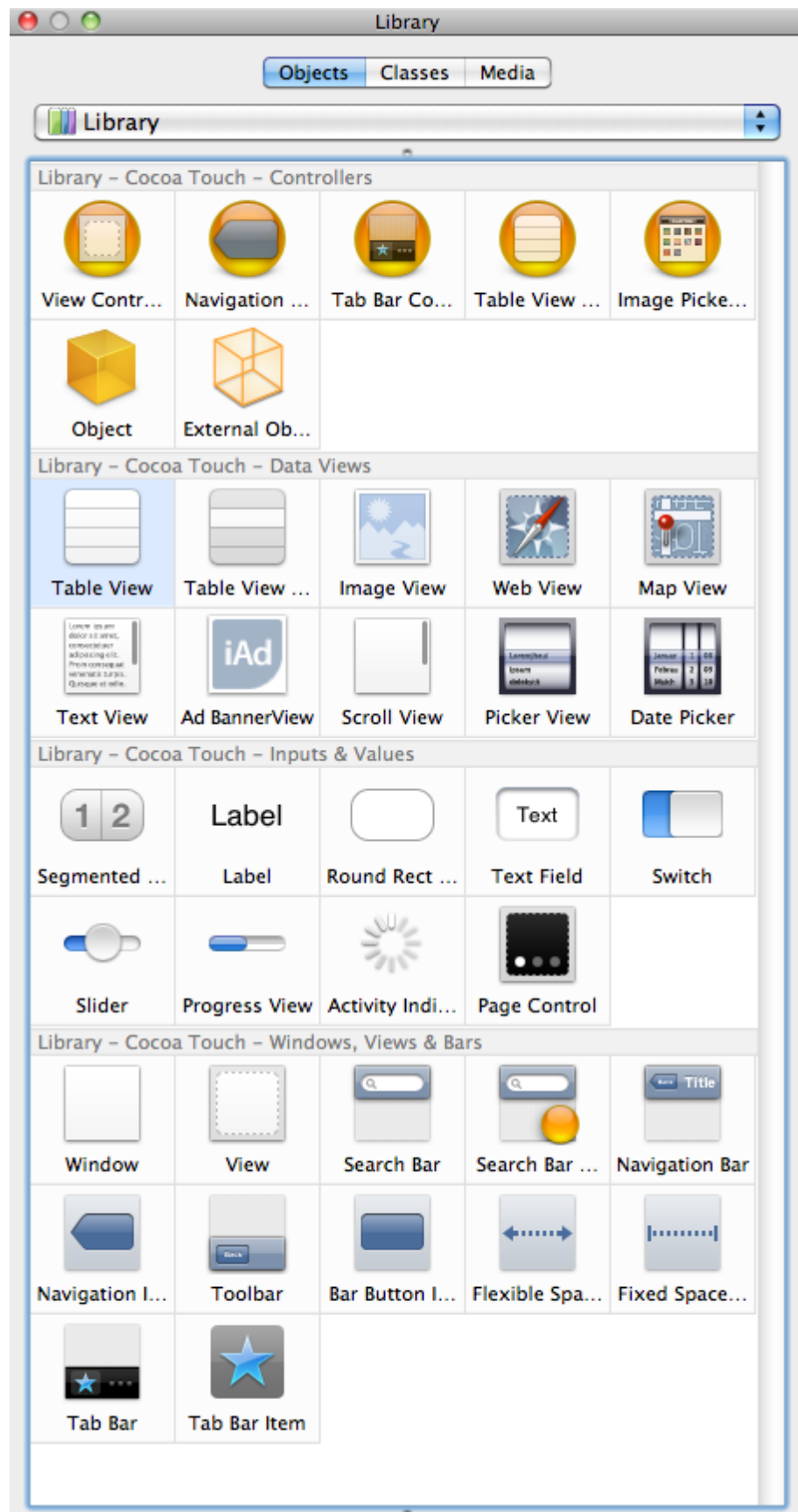
- 16.1** Apps that present excessively objectionable or crude content will be rejected
- 16.2** Apps that are primarily designed to upset or disgust users will be rejected

17. Privacy

- 17.1** Apps cannot transmit data about a user without obtaining the user's prior permission and providing the user with access to information about how and where the data will be used
- 17.2** Apps that require users to share personal information, such as email address and date of birth, in order to function will be rejected
- 17.3** Apps that target minors for data collection will be rejected

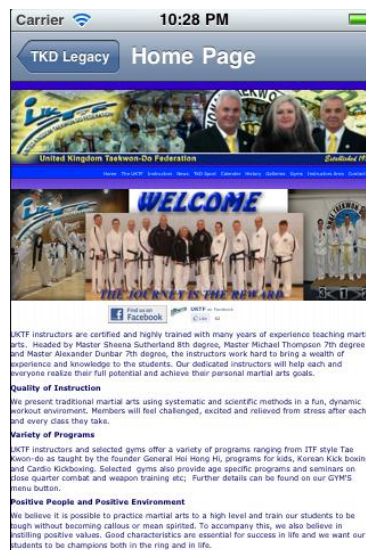
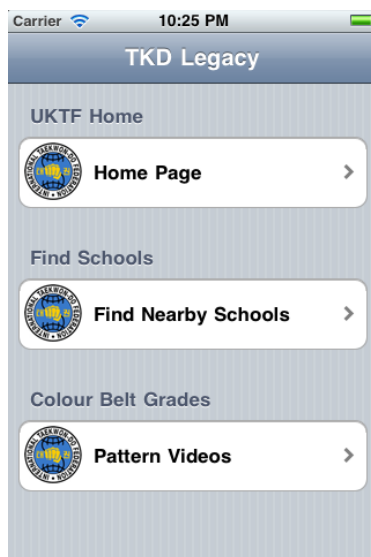
Appendix B - UI paper screen page designs

TKD Legacy	Main Page	Maps Page
Graphic →  Main Page	UIWebView to Website	UIMapView to access Google Maps
 Maps		
 Patterns		
Main	Home Page	Map
 Patterns		
Pattern 1	VIDEO	
Pattern text descriptions		
Play Video Button		
Pattern 2		
Pattern text description		
Play Video text		
Pattern 3		
Descriptions		

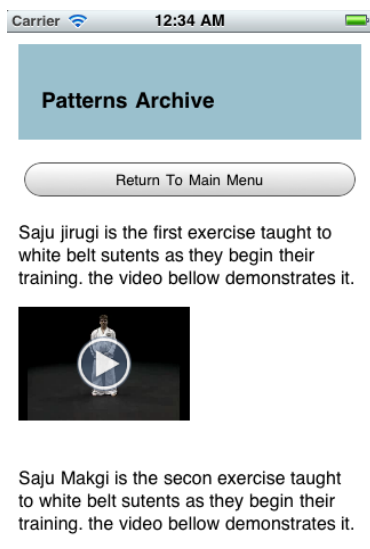
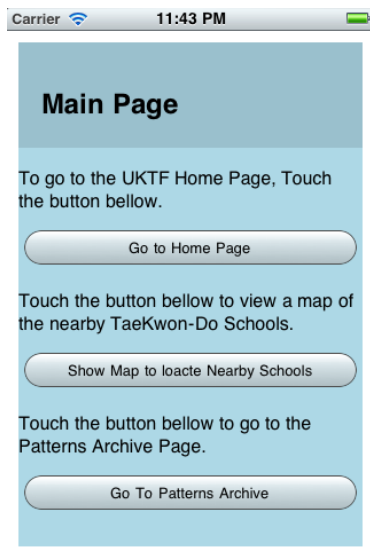


Appendix D - Final User Interface Screens

Objective-C



QuickConnectiPhone



Appendix E - Objective-C App code printout

```
//
// RootViewController.h
// TKDOBC
//
// Created by Ironman on 19/03/2011.
// Copyright 2011 __MyCompanyName__. All rights reserved.
//

#import "patternDetails.h"
#import "mapView.h"
#import "gradeView.h"

@interface RootViewController : UITableViewController {
    NSMutableArray *patternData;
    NSMutableArray *patternSections;
}

-(void) createPatternData;

@end

//
// RootViewController.m
// TKDOBC
//
// Created by Ironman on 19/03/2011.
// Copyright 2011 __MyCompanyName__. All rights reserved.
//

#define sectionCount 2
#define mainSection 0
#define mapSection 1
#define patternSection 2

#import "RootViewController.h"

@implementation RootViewController

#pragma mark -
#pragma mark View lifecycle

- (void)viewDidLoad {
    [self createPatternData];
    [super viewDidLoad];

    // Uncomment the following line to display an Edit button in the navigation bar for this view controller.
    // self.navigationItem.rightBarButtonItem = self.editButtonItem;
}

/*
- (void)viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];
}
*/
/*
- (void)viewDidAppear:(BOOL)animated {
    [super viewDidAppear:animated];
}
*/
/*
- (void)viewWillDisappear:(BOOL)animated {
    [super viewWillDisappear:animated];
}
*/
/*
- (void)viewDidDisappear:(BOOL)animated {
    [super viewDidDisappear:animated];
}
*/
*/
```

```

/*
// Override to allow orientations other than the default portrait orientation.
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation {
    // Return YES for supported orientations.
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
*/

#pragma mark -
#pragma mark Table view data source

// Customize the number of sections in the table view.
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return [patternSections count];
}

// Customize the number of rows in the table view.
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
    return [[patternData objectAtIndex:section] count];
}

// Customize the appearance of table view cells.
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {

    static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier] autorelease];
    }

    // Configure the cell.
    [[cell.textLabel] setText:[patternData objectAtIndex:indexPath.section] objectAtIndex:indexPath.row] forKey:@"name"];
    [[cell.imageView] setImage:[UIImage imageNamed:[patternData objectAtIndex:indexPath.section] objectAtIndex:indexPath.row]
objectForKey:@"picture"]];
    cell.accessoryType=UITableViewCellAccessoryDisclosureIndicator;

    return cell;
}

/*
// Override to support conditional editing of the table view.
- (BOOL)tableView:(UITableView *)tableView canEditRowAtIndexPath:(NSIndexPath *)indexPath {
    // Return NO if you do not want the specified item to be editable.
    return YES;
}
*/

/*
// Override to support editing the table view.
- (void)tableView:(UITableView *)tableView commitEditingStyle:(UITableViewCellEditingStyle)editingStyle forRowAtIndexPath:(NSIndexPath
*)indexPath {

    if (editingStyle == UITableViewCellEditingStyleDelete) {
        // Delete the row from the data source.
        [tableView deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath] withRowAnimation:UITableViewRowAnimationFade];
    }
    else if (editingStyle == UITableViewCellEditingStyleInsert) {
        // Create a new instance of the appropriate class, insert it into the array, and add a new row to the table view.
    }
}
*/

/*
// Override to support rearranging the table view.
- (void)tableView:(UITableView *)tableView moveRowAtIndexPath:(NSIndexPath *)fromIndexPath toIndexPath:(NSIndexPath *)toIndexPath {
}
*/

```

```

/*
// Override to support conditional rearranging of the table view.
- (BOOL)tableView:(UITableView *)tableView canMoveRowAtIndexPath:(NSIndexPath *)indexPath {
    // Return NO if you do not want the item to be re-orderable.
    return YES;
}
*/

- (NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section {
    return [patternSections objectAtIndex:section];
}

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath {

    UIAlertView *noSelect;
    noSelect= [[UIAlertView alloc] initWithTitle:@"Invalid Selection" message:@"The chosen selection does not exist." delegate:nil
cancelButtonTitle:@"OK" otherButtonTitles:nil];

    patternDetails *patternDetails = [[patternDetails alloc] initWithNibName:@"patternDetails" bundle:nil];
    patternDetails.title=[[patternData objectAtIndex:indexPath.section] objectAtIndex:indexPath.row] objectForKey:@"name"];

    mapView *mapview = [[mapView alloc] initWithNibName:@"mapView" bundle:nil];
    mapview.title=[[patternData objectAtIndex:indexPath.section] objectAtIndex:indexPath.row] objectForKey:@"name"];

    gradeView *gradeview = [[gradeView alloc] initWithNibName:@"gradeView" bundle:nil];
    gradeview.title=[[patternData objectAtIndex:indexPath.section] objectAtIndex:indexPath.row] objectForKey:@"name"];

    switch (indexPath.section) {
        case mainSection:
            patternDetails.detailURL=[[NSURL alloc] initWithString:[[[patternData objectAtIndex:indexPath.section]
objectAtIndex:indexPath.row] objectForKey:@"url"]];
            [self.navigationController pushViewController:patternDetails animated:YES];
            break;
        case mapSection:
            patternDetails.detailURL=[[NSURL alloc] initWithString:[[[patternData objectAtIndex:indexPath.section]
objectAtIndex:indexPath.row] objectForKey:@"url"]];
            [self.navigationController pushViewController:patternDetails animated:YES];
            break;
        case patternSection:
            [self.navigationController pushViewController:gradeview animated:YES];
            break;
        default:
            [noSelect show];
            break;
    }

    [noSelect release];
    [patternDetails release];
    [mapview release];
    [gradeview release];
}

#pragma mark -
#pragma mark Memory management

- (void)didReceiveMemoryWarning {
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Relinquish ownership any cached data, images, etc that aren't in use.
}

- (void)viewDidUnload {
    // Relinquish ownership of anything that can be recreated in viewDidLoad or on demand.
    // For example: self.myOutlet = nil;
}

- (void) createPatternData {

    NSMutableArray *mainDetails;
    NSMutableArray *findSchools;
    NSMutableArray *patterns;

```



```

patternSections=[[NSMutableArray alloc] initWithObjects:@"UKTF Home",@"Find Schools",@"Colour Belt Grades",nil];

mainDetails=[[NSMutableArray alloc] init];

findSchools=[[NSMutableArray alloc] init];

patterns=[[NSMutableArray alloc] init];

[mainDetails addObject:[NSMutableDictionary alloc] initWithObjectsAndKeys:@"Home
Page",@"name",@"itf@2x.png",@"picture",@"http://www.uktf.com/index.php",@"url",nil]];

[findSchools addObject:[NSMutableDictionary alloc] initWithObjectsAndKeys:@"Find Nearby
Schools",@"name",@"itf@2x.png",@"picture",@"http://maps.google.com/maps?f=q&source=s_q&hl=en&geocode=&q=taekwondo,+g331ly&aq=&sl=
37.0625,-95.677068&sspn=76.273886,80.244141&ie=UTF8&hq=taekwondo,&hnear=Glasgow+G33+1LY,+United+Kingdom&ll=55.856817,-
4.182358&spn=0.427783,0.626907&z=11",@"url",nil]];

[patterns addObject:[NSMutableDictionary alloc] initWithObjectsAndKeys:@"Pattern Videos",@"name",@"itf@2x.png",@"picture",nil]];

patternData=[[NSMutableArray alloc] initWithObjects:mainDetails,findSchools,patterns,nil];

[mainDetails release];
[findSchools release];
[patterns release];
}

- (void)dealloc {
    [patternData release];
    [patternSections release];
    [super dealloc];
}

@end

//
// patternDetails.h
// TKDOBC
//
// Created by Ironman on 19/03/2011.
// Copyright 2011 __MyCompanyName__. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface patternDetails : UIViewController {
    IBOutlet UIWebView *detailWebView;
    NSURL *detailURL;
}

@property(nonatomic,retain) NSURL *detailURL;
@property(nonatomic,retain) UIWebView *detailWebView;

@end

//
// patternDetails.m
// TKDOBC
//
// Created by Ironman on 19/03/2011.
// Copyright 2011 __MyCompanyName__. All rights reserved.
//

#import "patternDetails.h"

@implementation patternDetails

@synthesize detailWebView;
@synthesize detailURL;

// The designated initializer. Override if you create the controller programmatically and want to perform customization that is not appropriate for
viewDidLoad.
/*

```

```

- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil {
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization.
    }
    return self;
}
*/

// Implement viewDidLoad to do additional setup after loading the view, typically from a nib.
- (void)viewDidLoad {
    [detailWebView loadRequest:[NSURLRequest requestWithURL:detailURL]];
    [super viewDidLoad];
}

/*
// Override to allow orientations other than the default portrait orientation.
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation {
    // Return YES for supported orientations.
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
*/

- (void)didReceiveMemoryWarning {
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc. that aren't in use.
}

- (void)viewDidUnload {
    [super viewDidUnload];
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}

- (void)dealloc {
    [detailURL release];
    [detailWebView release];
    [super dealloc];
}

@end

//
// mapView.h
// TKDOBC
//
// Created by Ironman on 19/03/2011.
// Copyright 2011 __MyCompanyName__. All rights reserved.
//

#import <CoreLocation/CoreLocation.h>
#import <MapKit/MapKit.h>
#import <UIKit/UIKit.h>

@interface mapView : UIViewController {
    IBOutlet MKMapView *map;
    MKPlacemark *zipAnnotation;
}

@property (nonatomic, retain) MKMapView *map;

@end

//
// mapView.m
// TKDOBC
//
// Created by Ironman on 19/03/2011.
// Copyright 2011 __MyCompanyName__. All rights reserved.

```



```
//
#import "mapView.h"

@implementation mapView
@synthesize map;

- (void)centerMap {
    NSString *queryURL;
    NSString *queryResults;
    NSArray *queryData;
    double latitude;
    double longitude;
    MKCoordinateRegion mapRegion;

    queryURL = @"http://maps.google.com/maps/geo?output=csv&q=G331LY";

    queryResults = [[NSString alloc] initWithContentsOfURL:
        [NSURL URLWithString:queryURL]
        encoding: NSUTF8StringEncoding
        error: nil];

    // Autoreleased
    queryData = [queryResults componentsSeparatedByString:@","];

    if([queryData count]==4) {
        latitude=[[queryData objectAtIndex:2] doubleValue];
        longitude=[[queryData objectAtIndex:3] doubleValue];
        // CLLocationCoordinate2D;
        mapRegion.center.latitude=latitude;
        mapRegion.center.longitude=longitude;
        mapRegion.span.latitudeDelta=0.2;
        mapRegion.span.longitudeDelta=0.2;
        [map setRegion:mapRegion animated:YES];

        /*
        if (zipAnnotation!=nil) {
            [map removeAnnotation: zipAnnotation];
        }
        zipAnnotation = [[MKPlacemark alloc] initWithCoordinate:mapRegion.center addressDictionary:fullAddress];
        [map addAnnotation:zipAnnotation];
        [zipAnnotation release];
        */
    }

    [queryURL release];
    [queryResults release];
}

/*
- (MKAnnotationView *)mapView:(MKMapView *)mapView
    viewForAnnotation:(id <MKAnnotation>)annotation {
    MKPinAnnotationView *pinDrop=[[MKPinAnnotationView alloc]
        initWithAnnotation:annotation reuseIdentifier:@"citycenter"];

    pinDrop.animatesDrop=YES;
    pinDrop.canShowCallout=YES;
    pinDrop.pinColor=MKPinAnnotationColorPurple;
    return pinDrop;
}
*/

// The designated initializer. Override if you create the controller programmatically and want to perform customization that is not appropriate for
viewDidLoad.
/*
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil {
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization.
    }
    return self;
}
*/

```

```

*/

/*
// Implement viewDidLoad to do additional setup after loading the view, typically from a nib.
- (void)viewDidLoad {
    [super viewDidLoad];
}
*/

/*
// Override to allow orientations other than the default portrait orientation.
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation {
    // Return YES for supported orientations.
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
*/

- (void)didReceiveMemoryWarning {
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc. that aren't in use.
}

- (void)viewDidUnload {
    [super viewDidUnload];
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}

- (void)dealloc {
    [map release];
    [super dealloc];
}

@end

//
// gradeView.h
// TKDOBC
//
// Created by Ironman on 20/03/2011.
// Copyright 2011 __MyCompanyName__. All rights reserved.
//

#import <UIKit/UIKit.h>
#import <MediaPlayer/MediaPlayer.h>

@interface gradeView : UIViewController {
    IBOutlet UIScrollView *scroller;

}

-(IBAction)playSM:(id)sender;
-(IBAction)playSJ:(id)sender;
-(IBAction)playCJ:(id)sender;
-(IBAction)playDG:(id)sender;
-(IBAction)playDS:(id)sender;
-(IBAction)playWH:(id)sender;
-(IBAction)playYG:(id)sender;
-(IBAction)playJG:(id)sender;
-(IBAction)playTG:(id)sender;
-(IBAction)playWR:(id)sender;

@property (nonatomic,retain) UIScrollView *scroller;

@end

//
// gradeView.m
// TKDOBC
//
// Created by Ironman on 20/03/2011.
// Copyright 2011 __MyCompanyName__. All rights reserved.

```

```
//
#import "gradeView.h"

@implementation gradeView
@synthesize scroller;

-(IBAction)playSM:(id)sender {
    NSString *movieFile;
    MPMoviePlayerController *moviePlayer;

    movieFile=[[NSBundle mainBundle]pathForResource:@"Saju Makgi" ofType:@"m4v"];

    moviePlayer=[[MPMoviePlayerController alloc] initWithContentURL:[NSURL fileURLWithPath:movieFile]];

    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(playSMFinished:)
name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer];

    [self.view addSubview:moviePlayer.view];
    [moviePlayer setFullscreen:YES animated:YES];
    [moviePlayer play];
}

-(void)playSMFinished:(NSNotification*)theNotification {
    MPMoviePlayerController *moviePlayer=[theNotification object];

    [[NSNotificationCenter defaultCenter] removeObserver:self name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer];

    [moviePlayer.view removeFromSuperview];
    [moviePlayer release];
}

-(IBAction)playSJ:(id)sender {
    NSString *movieFile2;
    MPMoviePlayerController *moviePlayer2;

    movieFile2=[[NSBundle mainBundle]pathForResource:@"Saju Jirugi" ofType:@"m4v"];

    moviePlayer2=[[MPMoviePlayerController alloc] initWithContentURL:[NSURL fileURLWithPath:movieFile2]];

    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(playSJFinished:)
name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer2];

    [self.view addSubview:moviePlayer2.view];
    [moviePlayer2 setFullscreen:YES animated:YES];
    [moviePlayer2 play];
}

-(void)playSJFinished:(NSNotification*)theNotification {
    MPMoviePlayerController *moviePlayer2=[theNotification object];

    [[NSNotificationCenter defaultCenter] removeObserver:self name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer2];

    [moviePlayer2.view removeFromSuperview];
    [moviePlayer2 release];
}

-(IBAction)playCJ:(id)sender {
    NSString *movieFile;
    MPMoviePlayerController *moviePlayer;

    movieFile=[[NSBundle mainBundle]pathForResource:@"Chon-Ji" ofType:@"m4v"];

    moviePlayer=[[MPMoviePlayerController alloc] initWithContentURL:[NSURL fileURLWithPath:movieFile]];

    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(playCJFinished:)
name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer];

    [self.view addSubview:moviePlayer.view];
    [moviePlayer setFullscreen:YES animated:YES];
}
```

```

        [moviePlayer play];
    }

    -(void)playCJFinished:(NSNotification*)theNotification {
        MPMoviePlayerController *moviePlayer=[theNotification object];

        [[NSNotificationCenter defaultCenter] removeObserver:self name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer];

        [moviePlayer.view removeFromSuperview];
        [moviePlayer release];
    }

    -(IBAction)playDG:(id)sender {
        NSString *movieFile;
        MPMoviePlayerController *moviePlayer;

        movieFile=[[NSBundle mainBundle]pathForResource:@"Dan-Gun" ofType:@"m4v"];

        moviePlayer=[[MPMoviePlayerController alloc] initWithContentURL:[NSURL fileURLWithPath:movieFile]];

        [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(playDGFinished:)
        name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer];

        [self.view addSubview:moviePlayer.view];
        [moviePlayer setFullscreen:YES animated:YES];
        [moviePlayer play];
    }

    -(void)playDGFinished:(NSNotification*)theNotification {
        MPMoviePlayerController *moviePlayer=[theNotification object];

        [[NSNotificationCenter defaultCenter] removeObserver:self name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer];

        [moviePlayer.view removeFromSuperview];
        [moviePlayer release];
    }

    -(IBAction)playDS:(id)sender {
        NSString *movieFile;
        MPMoviePlayerController *moviePlayer;

        movieFile=[[NSBundle mainBundle]pathForResource:@"Do-San" ofType:@"m4v"];

        moviePlayer=[[MPMoviePlayerController alloc] initWithContentURL:[NSURL fileURLWithPath:movieFile]];

        [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(playDSFinished:)
        name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer];

        [self.view addSubview:moviePlayer.view];
        [moviePlayer setFullscreen:YES animated:YES];
        [moviePlayer play];
    }

    -(void)playDSFinished:(NSNotification*)theNotification {
        MPMoviePlayerController *moviePlayer=[theNotification object];

        [[NSNotificationCenter defaultCenter] removeObserver:self name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer];

        [moviePlayer.view removeFromSuperview];
        [moviePlayer release];
    }

    -(IBAction)playWH:(id)sender {
        NSString *movieFile;
        MPMoviePlayerController *moviePlayer;

        movieFile=[[NSBundle mainBundle]pathForResource:@"Won-Hyo" ofType:@"m4v"];

        moviePlayer=[[MPMoviePlayerController alloc] initWithContentURL:[NSURL fileURLWithPath:movieFile]];

        [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(playWHFinished:)
        name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer];

        [self.view addSubview:moviePlayer.view];
        [moviePlayer setFullscreen:YES animated:YES];
    }

```

```

        [moviePlayer play];
    }

    -(void)playWHFinished:(NSNotification*)theNotification {
        MPMoviePlayerController *moviePlayer=[theNotification object];

        [[NSNotificationCenter defaultCenter] removeObserver:self name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer];

        [moviePlayer.view removeFromSuperview];
        [moviePlayer release];
    }

    -(IBAction)playYG:(id)sender {
        NSString *movieFile;
        MPMoviePlayerController *moviePlayer;

        movieFile=[[NSBundle mainBundle]pathForResource:@"Yul-Gok" ofType:@"m4v"];

        moviePlayer=[[MPMoviePlayerController alloc] initWithContentURL:[NSURL fileURLWithPath:movieFile]];

        [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(playYGFinished:)
        name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer];

        [self.view addSubview:moviePlayer.view];
        [moviePlayer setFullscreen:YES animated:YES];
        [moviePlayer play];
    }

    -(void)playYGFinished:(NSNotification*)theNotification {
        MPMoviePlayerController *moviePlayer=[theNotification object];

        [[NSNotificationCenter defaultCenter] removeObserver:self name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer];

        [moviePlayer.view removeFromSuperview];
        [moviePlayer release];
    }

    -(IBAction)playJG:(id)sender {
        NSString *movieFile;
        MPMoviePlayerController *moviePlayer;

        movieFile=[[NSBundle mainBundle]pathForResource:@"Joong-Gun" ofType:@"m4v"];

        moviePlayer=[[MPMoviePlayerController alloc] initWithContentURL:[NSURL fileURLWithPath:movieFile]];

        [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(playJGFinished:)
        name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer];

        [self.view addSubview:moviePlayer.view];
        [moviePlayer setFullscreen:YES animated:YES];
        [moviePlayer play];
    }

    -(void)playJGFinished:(NSNotification*)theNotification {
        MPMoviePlayerController *moviePlayer=[theNotification object];

        [[NSNotificationCenter defaultCenter] removeObserver:self name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer];

        [moviePlayer.view removeFromSuperview];
        [moviePlayer release];
    }

    -(IBAction)playTG:(id)sender {
        NSString *movieFile;
        MPMoviePlayerController *moviePlayer;

        movieFile=[[NSBundle mainBundle]pathForResource:@"Toi-Gye" ofType:@"m4v"];

        moviePlayer=[[MPMoviePlayerController alloc] initWithContentURL:[NSURL fileURLWithPath:movieFile]];

        [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(playTGFinished:)
        name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer];

```

```

        [self.view addSubview:moviePlayer.view];
        [moviePlayer setFullscreen:YES animated:YES];
        [moviePlayer play];
    }

    -(void)playTGFinished:(NSNotification*)theNotification {
        MPMoviePlayerController *moviePlayer=[theNotification object];

        [[NSNotificationCenter defaultCenter] addObserver:self name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer];

        [moviePlayer.view removeFromSuperview];
        [moviePlayer release];
    }

    -(IBAction)playWR:(id)sender {
        NSString *movieFile;
        MPMoviePlayerController *moviePlayer;

        movieFile=[[NSBundle mainBundle]pathForResource:@"Hwa-Rang" ofType:@"m4v"];

        moviePlayer=[[MPMoviePlayerController alloc] initWithContentURL:[NSURL URLWithString:movieFile]];

        [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(playWRFinished:)
        name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer];

        [self.view addSubview:moviePlayer.view];
        [moviePlayer setFullscreen:YES animated:YES];
        [moviePlayer play];
    }

    -(void)playWRFinished:(NSNotification*)theNotification {
        MPMoviePlayerController *moviePlayer=[theNotification object];

        [[NSNotificationCenter defaultCenter] addObserver:self name:MPMoviePlayerPlaybackDidFinishNotification object:moviePlayer];

        [moviePlayer.view removeFromSuperview];
        [moviePlayer release];
    }
}

// The designated initializer. Override if you create the controller programmatically and want to perform customization that is not appropriate for
viewDidLoad.
/*
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil {
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization.
    }
    return self;
}
*/

// Implement viewDidLoad to do additional setup after loading the view, typically from a nib.
- (void)viewDidLoad {
    [super viewDidLoad];
    scroller.contentSize=CGSizeMake(320.0, 900.0);
    [super viewDidLoad];
}

/*
// Override to allow orientations other than the default portrait orientation.
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation {
    // Return YES for supported orientations.
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
*/

- (void)didReceiveMemoryWarning {
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc. that aren't in use.
}

```

```
- (void)viewDidLoad {  
    [super viewDidLoad];  
    // Release any retained subviews of the main view.  
    // e.g. self.myOutlet = nil;  
}  
  
- (void)dealloc {  
    [scroller release];  
    [super dealloc];  
}  
  
@end
```

Appendix F - QuickConnectiPhone App code printout

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>QCXcodeTemplate</title>
  <base href="mobile/">
  <meta http-equiv="content-type" content="text/html; charset=utf-8">
  <meta name="viewport" content="width=device-width, minimum-scale=1.0, maximum-scale=1.6">
  <meta name="apple-mobile-web-app-capable" content="YES">
  <link rel="stylesheet" href="main.css">
  <script type="text/javascript" src="Parts/parts.js" charset="utf-8"></script>
  <script type="text/javascript" src="../QCJSLib/setupQC.js" charset="utf-8"></script>
  <script type="text/javascript" src="main.js" charset="utf-8"></script>
</head>
<body onload="load();">

  <div id='view1' class='view'>
  <div id='header'>
  <br/>
  <h2 id='main'>Main Page</h2>
  </div>
  <p>To go to the UKTF Home Page, Touch the button bellow.</p>
  <a href="http://www.uktf.com/"><input id='buttons' type='button' value='Go to Home Page' onclick='load web page'/></a>
  <br/>
  <p>Touch the button bellow to view a map of the nearby TaeKwon-Do Schools.</p>
  <a href="http://maps.google.com/maps?f=q&source=s_q&hl=en&geocode=&q=taekwondo,+g331ly&aq=&sll=37.0625,-
95.677068&sspn=76.273886,80.244141&ie=UTF8&hq=taekwondo.&hnear=Glasgow+G33+1LY,+United+Kingdom&ll=55.856817,-
4.182358&spn=0.427783,0.626907&z=11"><input id='buttons' type='button' value='Show Map to loacte Nearby Schools' onclick='load web page'/></a>
  <br/>
  <p>Touch the button bellow to go to the Patterns Archive Page.</p>
  <input id='buttons' type='button' value='Go To Patterns Archive' onclick='showView(4)' />
  </div>

  <div id='view4' class='view' style='background-color:white;'>
  <div id='header'>
  <br/>
  <h3>Patterns Archive</h3>
  </div>
  <br/>
  <input id='buttons' type='button' value='Return To Main Menu' onclick='showView(1)'/>
  <br/>
  <p>Saju jirugi is the first exercise taught to white belt sutents as they begin their training. the video bellow demonstrates it.</p>
  <video id='Saju_Jirugi' src='../Saju Jirugi.m4v' controls='true' height=100 width=150>Not supported</video>
  <br/><br/>
  <p>Saju Makgi is the secon exercise taught to white belt sutents as they begin their training. the video bellow demonstrates it.</p>
  <video id='Saju_Makgi' src='../Saju Makgi.m4v' controls='true' height=100 width=150>Not supported</video>
  <br/><br/>
  <p>Once a student passes their first grading and reaches yellow tags they learn their first real pattern Chon Ji, The video bellow demonstrates it.</p>
  <video id='chon ji' src='../Chon-Ji.m4v' controls='true' height=100 width=150>Not supported</video>
  <br/><br/>
  <p>Once a student passes their second grading and reaches yellow belt they learn their next pattern Dan Gun, The video bellow demonstrates it.</p>
  <video id='dan-Gun' src='../Dan-Gun.m4v' controls='true' height=100 width=150>Not supported</video>
  <br/><br/>
  <p>Once a student passes their Third grading and reaches green tags they learn their next pattern Do San, The video bellow demonstrates it</p>
  <video id='Do-San' src='../Do-San.m4v' controls='true' height=100 width=150>Not supported</video>
  <br/><br/>
  <p>Once a student passes their fourth grading and reaches green belt they learn their next pattern Won Hyo, The video bellow demonstrates it.</p>
  <video id='Won-Hyo' src='../Won-Hyo.m4v' controls='true' height=100 width=150>Not supported</video>
  <br/><br/>
  <p>Once a student passes their fifth grading and reaches blue tags they learn their next pattern Yul Gok, The video bellow demonstrates it</p>
  <video id='Yul-Gok' src='../Yul-Gok.m4v' controls='true' height=100 width=150>Not supported</video>
  <br/><br/>
  <p>Once a student passes their sixth grading and reaches blue belt they learn their next pattern Joong Gun, The video bellow demonstrates it</p>
  <video id='Joong-Gun' src='../Joong-Gun.m4v' controls='true' height=100 width=150>Not supported</video>
  <br/><br/>
  <p>Once a student passes their seventh grading and reaches red tags they learn their next pattern Toi Gye, The video bellow demonstrates it</p>
  <video id='Toi-Gye' src='../Toi-Gye.m4v' controls='true' height=100 width=150>Not supported</video>

  </div>

</body>
</html>
```



```

body {
    margin: 0px;
    min-height: 356px;
    font-family: Helvetica;
    background-repeat: repeat;
    width: 320px;
}

#main{
    font: compact;
}

.view{
    position:absolute;
    top: 10px;
    left: 10px;
    right: 10px;
    bottom: 10px;
    background-color:lightBlue;
    z-index:0;
    opacity:0;
    padding-bottom: 0px;
    -webkit-transition: opacity 1s linear;
}

#header {
    background-color:#9AC0CD;
    padding-left: 20px;
    padding-bottom: 5px;
}

#buttons {
    margin-left:5px;
    margin-bottom: 5px;
    font-size:80%;
    width: 290px;
    height: 30px;
}

//
// Function: load()
// Called by HTML body element's onload event when the web application is ready to start
//
function load()
{
    showView(1);
    view4.style.visibility = 'hidden';
}

var currentView = null;

function showView(viewNumber){
    var nextView = document.getElementById('view'+viewNumber);
    //if there is a view to change to
    if(nextView){
        //if at least one view has been displayed
        if(currentView){
            currentView.style.zIndex = 0;
            currentView.style.opacity = 0;
            currentView.style.visibility = 'hidden';
        }
        nextView.style.zIndex = 10;
        nextView.style.opacity = 1.0
        nextView.style.visibility = 'visible';
        //reset currentView
        currentView = nextView;
    }
}

```

Appendix G - Testing Plans

Objective-C app testing plan

Test ID	Description	Expected Result	Actual Result	Resolution
1	Scroll view up	View will scroll to reveal any content above not shown	View scrolls	
2	Scroll view down	View will scroll to reveal any content below not shown	view scrolls	
3	Select home page button	View changes to home page view and displays the UKTF homepage	Home page shown incorrectly	Reset the display parameter in interface builder to scale to fit the page
4	select map page button	View changes to the google map view and displays nearby TKD schools	No schools shown	Problem corrected by appending the google map API url with the proper parameters
5	Select pattern page button	View changes to patterns page and displays the descriptions of each grade and the buttons to play the videos	Patterns page shown	
6	Interact with UKTF homepage	The page should function as a normal webpage	Page functions normally	
7	Navigate back to main page	View changes back to main page view	Option to revert not present	Add the proper root value to the table in interface builder so the view knows where its location is in the stack
8	Interact with map view	Map moves around. Contact details for schools appears when selected	Map moves and details appear	
9	navigate back to main page	View changes back to main page view	View reverts	
10	Scroll text descriptions of grades	The text for each grade should scroll to reveal the extra content should it be too long for the box	Descriptions scroll	
11	Press saju jirugi play button	Saju Jirugi video should play in full screen mode	Saju jirugi video plays	

Test ID	Description	Expected Result	Actual Result	Resolution
12	Allow Saju Jirugi video to end	Video should close and return to patterns view	Video closes successfully	
13	Quit Saju Jirugi video	video should close and return to patterns view	Application crashes	Movieplayer object not properly released from view, works once changed.
14	Press Saju Makgi play button	Saju Makgi video should play in full screen mode	Saju Makgi video plays	
15	Allow Saju Makgi video to end	Video should close and return to patterns view	Video closes successfully	
16	Quit Saju Makgi video	video should close and return to patterns view	Video closes successfully	
17	Press Chon Ji play button	Chon Ji video should play in full screen mode	Chon Ji video plays	
18	Allow Chon Ji video to end	Video should close and return to patterns view	Video closes successfully	
19	Quit Chon Ji video	video should close and return to patterns view	video closes successfully	
20	Press Dan Gun play button	Dan Gun video should play in full screen mode	Application crashes	Movie file resource url incorrect, requires changing.
21	Allow Dan Gun video to end	Video should close and return to patterns view	Video closes successfully	
22	Quit Dan Gun video	video should close and return to patterns view	Video closes successfully	
23	Press Do San play button	Do San video should play in full screen mode	Do San video plays	
24	Allow Do San video to end	Video should close and return to patterns view	Video closes successfully	
25	Quit Do San video	video should close and return to patterns view	video closes successfully	
26	Press Won Hyo play button	Won Hyo video should play in full screen mode	Won Hyo video plays	
27	Allow Won Hyo video to end	Video should close and return to patterns view	Video closes successfully	

Test ID	Description	Expected Result	Actual Result	Resolution
28	Quit Won Hyo video	video should close and return to patterns view	video closes successfully	
29	Press Yul Gok play button	Yul Gok video should play in full screen mode	Yul Gok video plays	
30	Allow Yul Gok video to end	Video should close and return to patterns view	Application crashes	Movie player file path incorrect
31	Quit Yul Gok video	video should close and return to patterns view	video closes successfully	
32	Press Joong Gun play button	Joong Gun video should play in full screen mode	Joong Gun video plays	
33	Allow Joong Gun video to end	Video should close and return to patterns view	Video closes successfully	
34	Quit Joong Gun video	video should close and return to patterns view	video closes successfully	
35	Press Toi Gye play button	Toi Gye video should play in full screen mode	Toi Gye video plays	
36	Allow Toi Gye video to end	Video should close and return to patterns view	Application crashes	Movie player file path incorrect
37	Quit Toi Gye video	video should close and return to patterns view	video closes successfully	
38	Press Hwa Rang play button	Hwa Rang video should play in full screen mode	Hwa Rang video plays	
39	Allow Hwa Rang video to end	Video should close and return to patterns view	Video closes successfully	
40	Quit Hwa Rang video	video should close and return to patterns view	video closes successfully	

QuickConnectiPhone app Test Plan

Test ID	Description	Expected Result	Actual Result	Resolution
1	Scroll view up	View will scroll to reveal any content above not shown	View scrolls	
2	Scroll view down	View will scroll to reveal any content below not shown	view scrolls	
3	Select home page button	View changes to home page view and displays the UKTF homepage	Home page shown correctly	
4	select map page button	View changes to the google map view and displays nearby TKD schools	Incorrect location	Incorrect location parameter in google url
5	Select pattern page button	View changes to patterns page and displays the descriptions of each grade and the buttons to play the videos	Patterns page shown	
6	Interact with UKTF homepage	The page should function as a normal webpage	Page functions normally	
7	Navigate back to main page	View changes back to main page view	View reverts	
8	Interact with map view	Map moves around. Contact details for schools appears when selected	Map moves and details appear	
9	navigate back to main page	View changes back to main page view	View reverts	
10	Scroll text descriptions of grades	The text for each grade should scroll to reveal the extra content should it be too long for the box	Descriptions scroll	
11	Press saju jirugi play button	Saju Jirugi video should play in full screen mode	Video does not play	Video tag url incorrect
12	Allow Saju Jirugi video to end	Video should close and return to patterns view	Video closes successfully	

Test ID	Description	Expected Result	Actual Result	Resolution
13	Quit Saju Jirugi video	video should close and return to patterns view	Video closes successfully	
14	Press Saju Makgi play button	Saju Makgi video should play in full screen mode	Saju Makgi video plays	
15	Allow Saju Makgi video to end	Video should close and return to patterns view	Video closes successfully	
16	Quit Saju Makgi video	video should close and return to patterns view	Video closes successfully	
17	Press Chon Ji play button	Chon Ji video should play in full screen mode	Video does not play	Improperly closed video tag
18	Allow Chon Ji video to end	Video should close and return to patterns view	Video closes successfully	
19	Quit Chon Ji video	video should close and return to patterns view	video closes successfully	
20	Press Dan Gun play button	Dan Gun video should play in full screen mode	Dan Gun video plays	
21	Allow Dan Gun video to end	Video should close and return to patterns view	Video closes successfully	
22	Quit Dan Gun video	video should close and return to patterns view	Video closes successfully	
23	Press Do San play button	Do San video should play in full screen mode	Do San video plays	
24	Allow Do San video to end	Video should close and return to patterns view	Video closes successfully	
25	Quit Do San video	video should close and return to patterns view	video closes successfully	
26	Press Won Hyo play button	Won Hyo video should play in full screen mode	Won Hyo video plays	
27	Allow Won Hyo video to end	Video should close and return to patterns view	Video closes successfully	

Test ID	Description	Expected Result	Actual Result	Resolution
28	Quit Won Hyo video	video should close and return to patterns view	video closes successfully	
29	Press Yul Gok play button	Yul Gok video should play in full screen mode	Yul Gok video plays	
30	Allow Yul Gok video to end	Video should close and return to patterns view	Application crashes	Movie player file path incorrect
31	Quit Yul Gok video	video should close and return to patterns view	video closes successfully	
32	Press Joong Gun play button	Joong Gun video should play in full screen mode	Joong Gun video plays	
33	Allow Joong Gun video to end	Video should close and return to patterns view	Video closes successfully	
34	Quit Joong Gun video	video should close and return to patterns view	video closes successfully	
35	Press Toi Gye play button	Toi Gye video should play in full screen mode	Toi Gye video plays	
36	Allow Toi Gye video to end	Video should close and return to patterns view	Application crashes	Movie player file path incorrect
37	Quit Toi Gye video	video should close and return to patterns view	video closes successfully	
38	Press Hwa Rang play button	Hwa Rang video should play in full screen mode	Hwa Rang video plays	
39	Allow Hwa Rang video to end	Video should close and return to patterns view	Video closes successfully	
40	Quit Hwa Rang video	video should close and return to patterns view	video closes successfully	

Appendix H - App Development Logs

Objective-C Development log

Section 1: UI Creation

Task ID	Description	Problems encountered	Solution
1	Create table view in the rootviewController for the basic layout of the main page of the App	The stack wouldn't initialise properly	Inside the inspector of interface builder was the proper settings section for the stack
2	Connect the Datasource, delegate and view to the file owner.	None	
3	Create the UIWebView for the homepage access	None	
4	Connect the view to the file owner	None	
5	Create the view for the google maps page	None	
6	Connect the view to the file owner	None	
7	Create the scroller for the patterns view page	Problems setting the scroller to scroll beyond the boundary of the page	Separate onload function had to be created defining the length of the scroller in pixels.
8	Create the buttons for each video playback element	None	
9	Create the text view objects for the patterns descriptions	None	
10	Populate the pattern description text views	None	

Section 2: Development

Task ID	Description	Problems encountered	Solution
Main Page			
1	Create table view object	None	
2	Create the patternData object containing the mainDetails, findSchools and patterns objects each containing the dictionaries holding the necessary information to populate the cells and handle requests	The complicated process of creating the array of arrays of dictionary objects caused several code errors	Resolve each line of code and each object individually until all correctly populate the table view

Task ID	Description	Problems encountered	Solution
3	Populate table view object cells with page names, mainDetails page first, findSchools and patterns page. Create a switch case function to order the cells properly.	None	
Home Page			
4	Synthesize the web-view object and the url passed from the main page	None	
5	Load the url in the web-view to access the website from within the view	None	
Map Page			
6	Synthesize the mapView object and the url to be passed from the main page	None	
7	Load the url in the mapView to load the proper map location and search parameters	Map fails to load properly	Alter url to create proper location view
Patterns Page			
8	Synthesize the scroller object and set the scroll length to allow user interaction of all buttons.	None	
9	Create Saju Jirugi video action object to load and play the file	File wont play	Had to fix the file url in the movie player object
10	Create the Saju Jirugi video finishing function to handle closing the player and release the memory	None	
11	Create Saju Makgi video action object to load and play the file	None	
12	Create the Saju Makgi video finishing function to handle closing the player and release the memory	None	
13	Create Chon Ji video action object to load and play the file	None	
14	Create the Chon Ji video finishing function to handle closing the player and release the memory	None	
15	Create Dan Gun video action object to load and play the file	None	

Task ID	Description	Problems encountered	Solution
16	Create the Dan Gun video finishing function to handle closing the player and release the memory	None	
17	Create Do San video action object to load and play the file	None	
18	Create the Do San video finishing function to handle closing the player and release the memory	None	
19	Create Won Hyo video action object to load and play the file	File wont play	Had to fix the file url in the movie player object
20	Create the Won Hyo video finishing function to handle closing the player and release the memory	None	
21	Create Yul Gok video action object to load and play the file	None	
22	Create the Yul Gok video finishing function to handle closing the player and release the memory	Application crashes	Movieplayer remove from superview release function was incorrectly targeted
23	Create Joong Gun video action object to load and play the file	None	
24	Create the Joong Gun video finishing function to handle closing the player and release the memory	None	
25	Create Toi Gye video action object to load and play the file	None	
26	Create the Toi Gye video finishing function to handle closing the player and release the memory	None	
27	Create Hwa Rang video action object to load and play the file	None	
28	Create the Hwa Rang video finishing function to handle closing the player and release the memory	None	

QuickConnectiPhone Development log

UI Design and Development

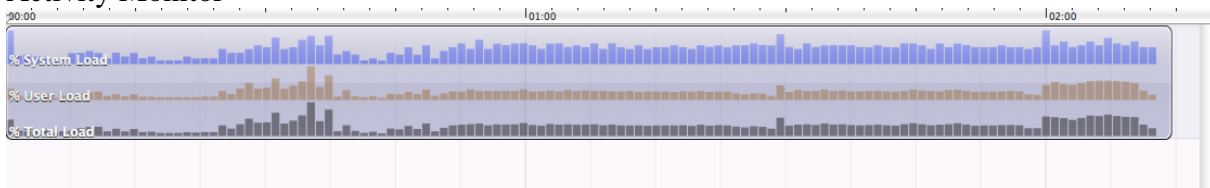
Task ID	Description	Problems encountered	Solution
Main Page			
1	Create the content tags for the view 1 division	None	
2	Create the stylesheet references to the view 1 elements to create the desired view	Objects not properly displayed	Trial and error process of tweaking the parameters of the style elements
3	Create the anchor to the uktf home page and assign it to the desired button	None	
4	Create the anchor to the map view functionality to load the Google map and assign it to the desired button	None	
5	Create the button to activate the view swapper function in the javascript file	None	
Home Page			
6	Load url to access page	None	
Map Page			
7	Load url to load map with proper location and parameters.	None	
Patterns Page			
8	Create the stylesheet references to the view 2 elements to create the desired view	Objects not properly displayed	Trial and error process of tweaking the parameters of the style elements
9	Create the paragraph and video elements for the patterns page	None	
10	Create the stylesheet references to the patterns page elements to create the desired views	objects not properly displayed	Trial and error process of tweaking the parameters of the style elements
11	Code the video tag urls to match the directory listing of the source video files.	None	
JavaScript file			

Task ID	Description	Problems encountered	Solution
12	Create the load function to display view 1 and hide view 2 when the app opens	None	
13	Create showView function to swap out the views when the patterns page button is clicked by altering the style values of the two views to change their visibility and push them to the back when not selected	Styling issues	trial and error process of tweaking the parameters of the style elements

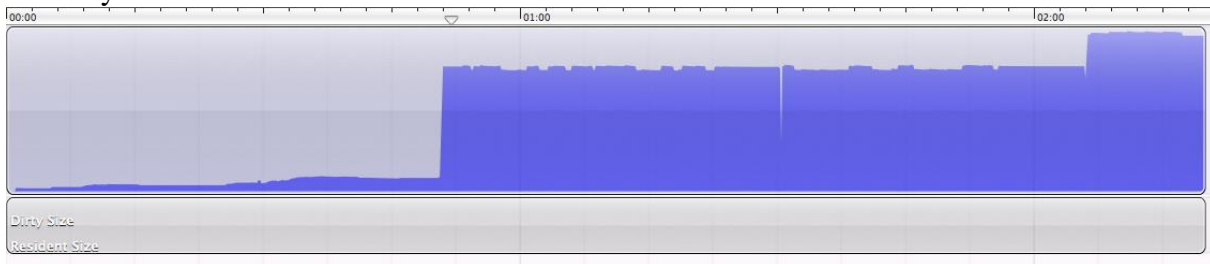
Appendix I - App Instruments Benchmark Test Data

Objective-C App

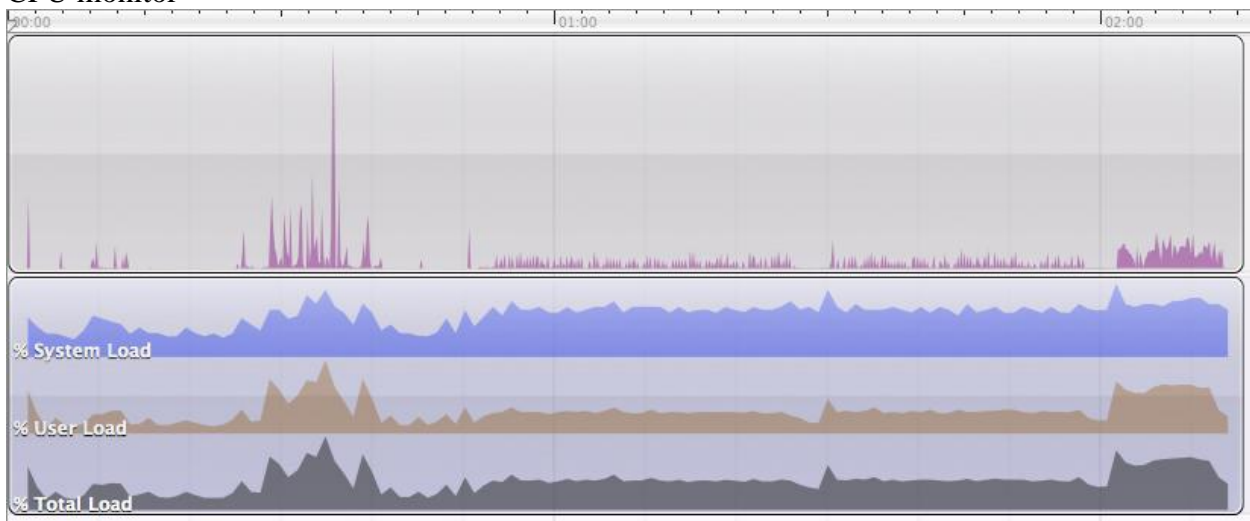
Activity Monitor



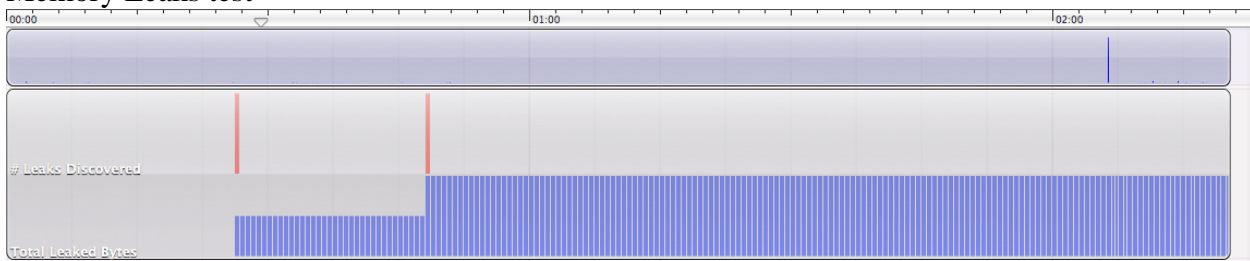
Memory Allocation test



CPU monitor

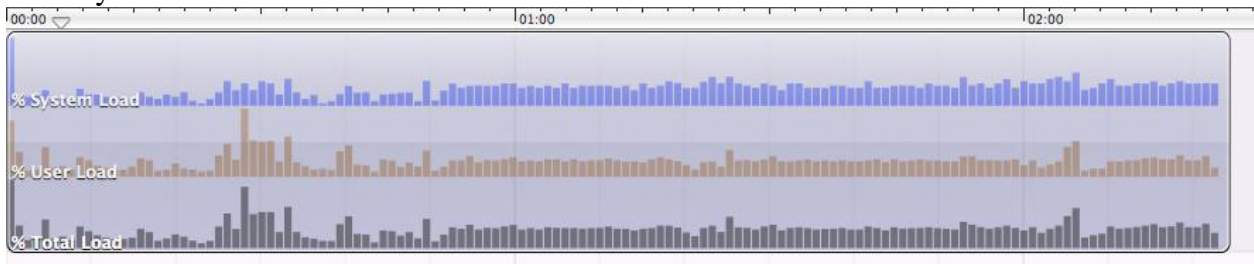


Memory Leaks test

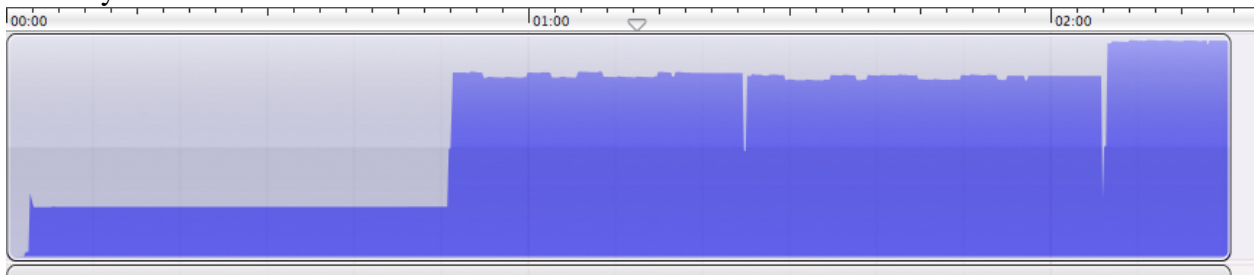


QuickConnectiPhone App

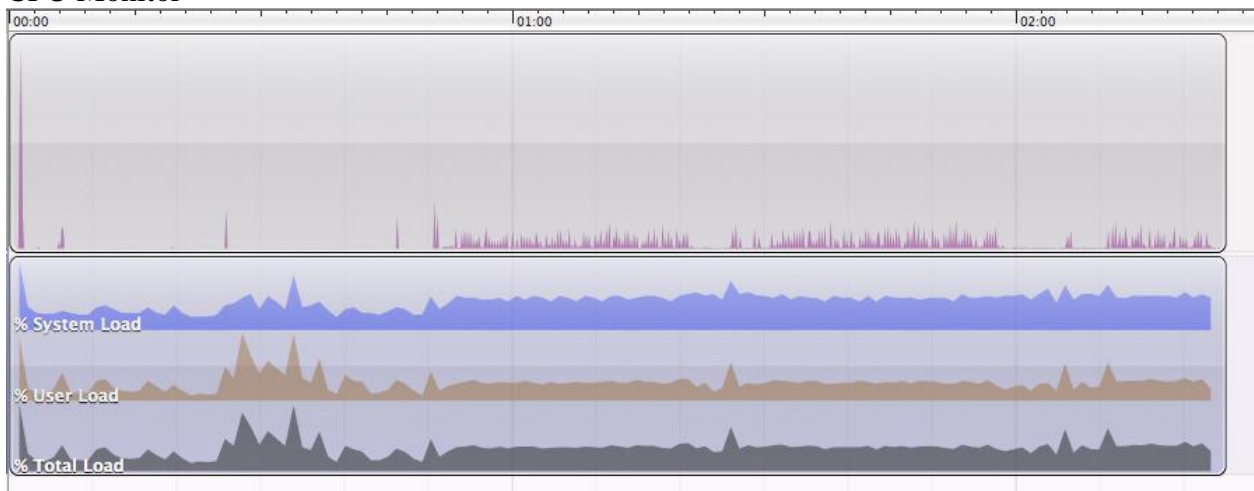
Activity Monitor



Memory Allocations test



CPU Monitor



Memory Leaks Test

