Annals of the

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

http://www.csb.uncw.edu/mscsis/

“myATMC”: THE DESIGN AND IMPLEMENTATION OF AN IPHONE WEBAPP

Candace Burwell

A Capstone Project Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science
Department of Information Systems and Operations Management

University of North Carolina Wilmington

2012

Approved by

Advisory Committee

Dr. Bryan Reinicke	Dr. Laurie Patterson

Dr. Ron Vetter, Chair

Abstract

"myATMC": The Design and Implementation of an iPhone WebApp. Author: Burwell, Candace, 2012. Capstone Paper, University of North Carolina Wilmington.

This paper discusses various options available to develop iPhone applications. The design decisions made on development procedures are based on the overall project to create an iPhone application for ATMC, called myATMC. This application will allow customers to login and view their wireless usage total since the last billing cycle. Web applications and native applications are discussed as well as several tools available for hybrid mobile web development. Explanations of the development approach in addition to various design elements are discussed.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

This project was selected based on my research interests and the needs of my employer, Atlantic Telephone Management Cooperation (ATMC). In 1955, ATMC was established as a nonprofit cooperative *(owned by its members)* to bring telephone service to the rural Brunswick County community. To keep up with the ever-changing environment and needs of its customers, ATMC has expanded its services from the initial landline telephone to also include wireless communications, cable television, broadband, and security.

ATMC Wireless works with AT&T to provide wireless service to its subscriber community; however there are limitations with regards to our partnership. The number one goal for ATMC is to provide outstanding customer service, and despite undying efforts, ATMC does not have the enough staff available to provide every feature that other nationwide companies can. Where ATMC may be limited in their ability to provide every custom feature the larger companies can, they certainly gain an advantage by providing local and personalized customer service.

A local newspaper, The Brunswick Beacon has held a Best of Brunswick contest each year that started in 2008, of which ATMC has been elected "Best in Customer Service" four years in a row. "To be voted Best Customer Service for one year, let alone four straight years, is an honor and testament to the effort all of our employees give to providing customer service excellence each day. It's really what separates us from the national companies", commented Allen Russ, ATMC CEO/General Manager [20].

In attempts to provide its customers with added convenience and keep up with the current trends of larger wireless providers, ATMC would like to offer a smart phone app that can easily display a customer's monthly wireless usage summary. As most wireless carriers already provide this functionality, ATMC will not gain a competitive advantage

from this development. It will advance ATMC's goal to level the playing field and most of all heighten the level of customer service that its employees strive so hard to improve upon.

In this project, I developed an iPhone application for ATMC that will eventually be available to the public via Apple's iTunes AppStore. The application, myATMC, will provide customers with various usage counts per their individual billing cycle. The login credentials are based at account level and usage rates are available for any wireless number tied to the authenticated account. This application has been modeled after myAT&T and myVerizon which are currently available in the AppStore.

A goal of the project included the research of various options available for iPhone application development, to determine the most efficient and valuable resource(s) to utilize. The myATMC application has been developed initially for the iPhone only, yet ATMC would like to expand its customer base to include the iPad and other mobile phone platforms in the future. Considerable research and experimentation was required to determine the most appropriate development environment for each of the most popular hardware platforms (i.e., iOS and Android).

This project also investigates whether native apps or web-based apps are most appropriate for the myATMC application. Investigation of the benefits and drawbacks of several authoring tools available in the marketplace today has been carried out to accurately make this comparison. After determining the application development environment, I performed extensive analysis and design to outline the development process.

Chapter 2: Background and Previous Work

2.1 Mac vs. PC

In the technical world there are two main types of computers available and the rival between these brands is so significant it has given way to a never-ending argument, which is be the Mac or the PC. When the facts are explored, the technical difference lies within the operating systems: Apple's, Mac OS X versus Microsoft's, Windows. The difference between these two operating systems and their public perception is only further emulated by the late Steve Jobs and Bill Gates. This personality comparison eventually triggered a *"chapter in their intense, decades-long rivalry: The Apple television ads pitting a cool Mac Guy against a dorky PC Guy… hammer away at the message that an Apple computer is more fun, less complicated and maybe worth the trouble of deserting your old faithful PC."* [19]. When comparing the two platforms each certainly has its own advantages and disadvantages, therefore the true champion really depends on the needs of the user and their preferences.

Figure 1 - Mac Guy vs. PC Guy *left* [18] *right* [19].

It is not uncommon that a business professional may live in a Windows world for their desktop computer needs while honing an iPhone to fulfill their mobile needs. Thus, the question has been posed, can iPhone applications be developed using a Windows environment? Although this may seem impossible, research proves quite the contrary. It is absolutely possible, for the right price, and under certain limitations.

2.2 Definition of Terms

Throughout this research, various terms referencing different authoring tools available to assist in development have been used. It was difficult to determine the difference between these terms because they are so similar and are even used interchangeably. Their relationships can be compared to that of squares and rectangles. For example, a square is a rectangle but a rectangle is not a square. It is important to note that these terms can be used in a broad manner and resemblance lies in the overall concept of a collective group of tools. To help resolve this confusion, some key terms related to this reading are described below for reference.

2.2.1 Library. In computer science, a library is a collection of resources used to develop software. These may include pre-written code and subroutines, classes, values or type specifications.

2.2.2 Toolkit. A toolkit is a compilation of multiple libraries that are designed to work well together. The key characteristic of a toolkit is integration, thus setting it apart from a group of independent libraries.

2.2.3 Framework. The central concept surrounding a framework is that the custom methods previously defined by the user will often be called from within the framework itself at a later point, rather than from the user's application code. The framework often plays the role of the main program in coordinating and sequencing application activity. This inversion of control gives frameworks the power to serve as extensible systems. The methods supplied by the user tailor the generic algorithms defined in the framework to suit particular application's requirements thus creating an extension.

2.2.4 Hollywood's Law. "Don't call us, we'll call you." The term Hollywood's Law later became known as The Hollywood Principle, where an objects (or components) initial condition and ongoing lifecycle is handled by its environment rather than by the

object itself [7]. This inversion of control is a common phenomenon that you come across when extending frameworks.

2.2.5 SDK. The abbreviation stands for Software Development Kit. SDK is a bigger concept as it can include libraries, frameworks, documentation, tools, etc. used to develop software. It may be something as simple as an application programming interface (API) in the form of some files to interface to a particular programming language or include sophisticated hardware to communicate with a certain embedded system.

2.2.6 SDK vs. Framework. An SDK is expected to offer tools to program against a certain system resource or feature, whereas with a Framework this is not necessarily true. A framework consisting solely of libraries could be developed, but if it is called an SDK, it will be expected to offer something to additional capabilities for development.

2.2.7 IDE. Interface Development Environment is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, compiler and/or an interpreter, build automation tools, and debugger. The boundary between an IDE and other parts of the broader software development environment is not well-defined and can vary depending on the IDE.

2.2.8 API. An Application Programming Interface is a particular set of rules ('code') and specifications that software programs can follow to communicate with each other. It serves as an interface between different software programs and facilitates their interaction. An easy way to conceptualize this would be to relate it to the way a user interface facilitates interaction between humans and computers.

2.2.9 Web Service. A web service is a software system identified by a URI [RFC 2396], whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact

with the web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols [17].

2.2.10 Document Object Model. The DOM is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML, and XML documents [21].

2.3 Windows SDKs

There are several SDKs available today for the Windows operating systems that offer developers a chance to create iPhone applications without ever having to touch a Mac. Some of these SDKs include DragonFireSDK (http://www.dragonfiresdk.com/), Marmalade (http://www.madewithmarmalade.com/), Corona SDK (http://www.anscamobile.com/corona/), Unity3d (http://unity3d.com/), ShiVa3D (http://stonetrip.com/), and Appcelerator Titanium (http://www.appcelerator.com/). There are demos available for most of these products that allow developers to explore the SDK over a set period of time using a limited free trail. Nevertheless, while all of these SDKs are available for Windows, none of them are free when it comes to developing and compiling an application from start to finish. A vast majority of these SDKs are marketed towards game development, with useful APIs to handle graphics and complex click events. However, in most cases the developer has to work with the APIs provided by the SDK and cannot step out of that environment. While some SDKs have more APIs than others, there are several limitations that are apparent. One notable outlier, Marmalade, offers an EDK *(Extensions Development Kit)* to developers with the opportunity to wrap existing iOS and Android libraries provided by 3rd parties, and design their own extension APIs as depicted in Figure 2.

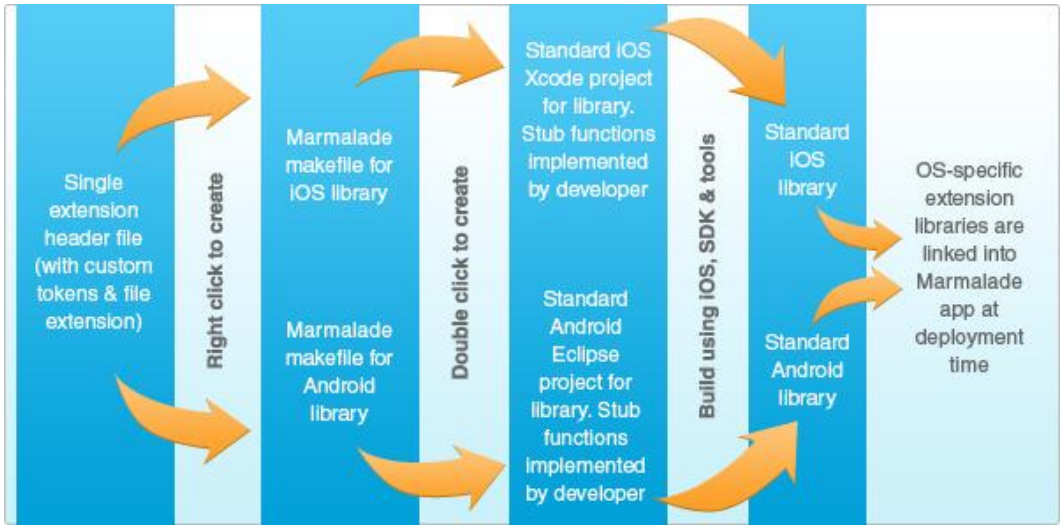


Figure 2 - How the Marmalade EDK Works [9]

Some requirements of the myATMC application, including database access and user authentication, are difficult if not impossible with many of these SDKs. For instance, the creator of DragonFireSDK states "our internet functionality is…limited." To compile the application with DragonFireSDK, the developer must send code back to company, where it is compiled in Objective-C, and returned as a fully packaged app. A common advantage of these SDKs is their ability to allow a developer to write generic code for an application, write some additional platform specific sets of code, and then compile particular combinations to generate apps for multiple devices. Marmalade, for example, has the ability to use your code to create an iPhone app and with some minimal tweaking create an Android app from almost the same C++ code. This is a great feature, promoting code reuse while maintaining a single code base used for maintenance or updates.

Although, many SDKs do offer a way to develop applications within Windows, most are quite limited in their functionality, and there is the potential for many bugs when working in one language and transferring that code to another language. While it is certainly possible to develop an iPhone app using these products, this researcher has determined that it may be more trouble than it is worth when it comes to the myATMC project requirements. The myATMC application will involve user authentication and

communication with a web service that requires more technical tools than those used to simply display game related graphics and responds to touch events. While some of the more established SDKs do offer a higher level of API interaction, it is important to consider that all of these SDKs require a yearly investment for licensing. Many of the licenses are sold on a per seat basis and include dramatic price inflation when used commercially. This was an expense that ATMC was not willing to pay. Therefore, the option of the development on a Windows operating system using a SDK was not the best solution for this project and therefore other alternatives were considered.

Following the decision to develop the iPhone application on a Mac, the question still remained: native or web? A native application runs solely on the device itself, in this case the iPhone. Of course, this application has access to the web but most of the functionality and processes performed are run on the device itself within the application. A web application runs entirely through the browser on the iPhone and does not require an installation to run. A third option that is commonly referred to as a web app, consists of a "shell-like" native app that holds some standard code used to trigger and communicate with a customized web portion of the app. Technically this create a native app but from a developers point of view it can be considered a web app based on the code development required. This format allows the application to be distributed through the AppStore instead of requiring the user to add a "bookmark" to their home screen to obtain the app icon. This example is for all intents and purposes a "web app" per say and will be referred to as such throughout this paper.

2.4 Native Applications

Native applications are widely popular in the current app market and a significant percentage of the apps available for the iPhone today are, in fact, native. iPhone applications are written in Objective-C using the developer tool Xcode. One undeniable

perk of a native application is the ease of user access provided by *"the ability to reach hundreds of millions of customers simply by uploading your app to a store"* [15]. This form of application does provide developers with certain performance advantages, due to the fact that the code is typically compiled and run on the device itself rather than interpreted like JavaScript, for example. *"With native code, we paint pixels directly on a screen through proprietary APIs and abstractions for common user-interface elements and controls"* [1]. While it is important to consider performance variability apparent in 3D gaming or image processing apps, there is a negligible performance penalty in business web applications that are designed properly [1]. An advantage native apps bring to the table revolves around the user interface, where all native platforms have their own paradigm for common user interface controls and experiences. Web toolkits currently available do a fantastic job of mimicking the device user interface. However, there are still small differences that have not yet been conquered, for example, *"on iOS, the CSS position property does not properly support a value of 'fixed'"* [1].

It is important to remember that the advantages provided with native apps are not without cost. If you intended to reach a broad audience, then building a different app for each platform is expensive. *"There are different tools, build systems, APIs, and devices with different capabilities for each platform"* [1]. While this price may be achievable to some who require the advantages presented by native applications, for a Telco cooperative based out of Brunswick County, NC it would have meant single platform development for the iPhone only. Thus, native apps may be the best solution for some circumstances and not for others. Developers must weigh the advantages and disadvantages that web apps have to offer as well, before an ultimate decision can be made.

2.5 Web Applications

A web-based application, *"which involves technologies such as HTML5, CSS, JavaScript, and related frameworks"* [15] has recently become an exciting alternative to native app development. This is partially due to the increase in network speed available through the wireless carriers and the growth of available wireless networks. Many mobile device platforms have various HTML5 APIs available for web development and *"given that the W3C has a Device API Working Group (http://www.w3.org/2009/dap/), it's likely we will be seeing many more APIs reach the browser in the near future"* [1]. For the time being, these APIs only provide access to some of the phone's features such as, geolocation. Thus, the overall feature accessibility still falls short when compared to native application development.

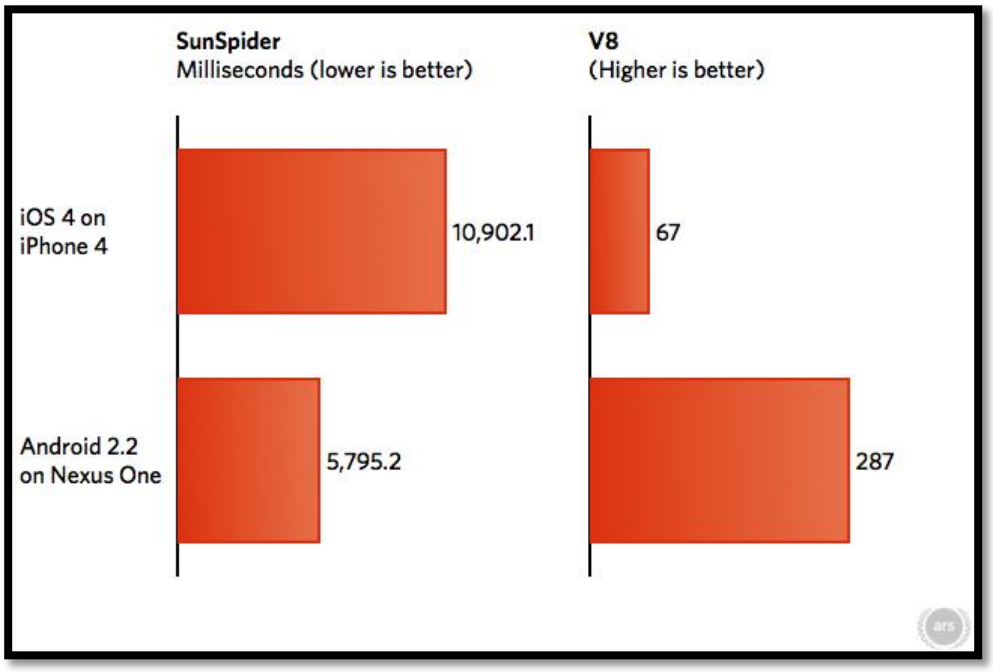
A combination of both web and native might be the solution since *"many developers see hybrid approaches as a natural migration path for developing cross-platform code that can run in a device-independent way across multiple hardware platforms"* [15]. PhoneGap has made great strides in the web application market, by offering a unique solution that, as mentioned before, will encapsulate your web app in a "shell-like" native app. This allows not only AppStore distribution but also provides developers with a collection of APIs developed by PhoneGap that can access various functionalities on the phone itself that were previously only available to native apps. These APIs are currently available for iPhone, Android, Blackberry, and Windows Mobile platforms with variations of capabilities depending on the platform. They utilize JavaScript and thus the speed comparison is between JavaScript and other compiled languages, *"JavaScript is rapidly getting faster—so fast, in fact, that HP Palm webOS 2.0 rewrote its services layer from Java to the extremely popular node.js platform (http://nodejs.org/), which is built on Google's V8 engine to obtain better performance at*

a lower CPU cost (and therefore longer battery life). The trend we're seeing is the Web technology stack running at a low level, and it's in production today on millions of devices" [1].

The race to develop the best JavaScript virtual machine technology has become quite competitive; the benchmark by Ara Technica shown in Table 1 is an example of this competition. SunSpider and V8 are both JavaScript benchmarks that test the overall speed of JavaScript engines. The V8 benchmark produces a score, not raw time sets thus the higher the score the better performance whereas with other benchmarks the lower the score the less time was required thus representing better performance. Despite the constant evolution and success of the JavaScript engines, currently they are still slower than their native counterparts and in turn the web app is lacking in that comparison. Of course, depending on the applications requirements the performance difference can be quite small and as time passes this gap in performance is shrinking.

Web applications and Native applications each have their own advantages and disadvantages, and it is important to consider them all before deciding which form of application is best suited, Table 2 specifically outlines the differences between these application models. Web applications fall short in certain areas such as user interface features and performance yet this difference is not extreme. One major advantage to web application development is the ability to program logic that will be functional *(minus a few tweaks per platform)* on multiple devices and require far less maintenance than multiple native apps would. Therefore, if a project goal is to market an application to multiple devices, it would be extremely convenient to use web apps while saving a great deal of time and money in the process. Developers must determine resources available, the application requirements, how these requirements will affect the final application, and if potential disadvantages are worth the advantages provided.

Table 1- *JavaScript performance: Android 2.2 vs. iOS 4*



Test is not directly indicative of the real-world performance you'll see in your everyday activities, they provide a standard example of how the browsers on specific devices perform against each other [8].

2.6 Frameworks

PhoneGap has been mentioned as one promising technology so far, but what is this development tool and how can it be utilized for this project? PhoneGap is an open source Framework, used to create "native" applications that are developed primarily from the web side via HTML, CSS, and JavaScript. *"From within that WebView we can call native code from JavaScript. This is the hack that became known as the PhoneGap technique pioneered by Eric Oesterle, Rob Ellis, and Brock Whitten for the first iPhone OS SDK at iPhoneDevCamp in 2008"* [1].

Using the JavaScript APIs provided by PhoneGap, the native code referred to can be called, perform a function, and/or pass information back to the app running in the WebView container. *"Of course, the Web technology stack (HTML/CSS/JS) is itself implemented in* information back to the app running in the WebView container. *"Of*

Table 2 - *How Do Native Apps and Web Apps Compare?*

The Issues	Native Apps	Web Apps *(without PhoneGap)*
Internet access	Not required	Required, except for rare apps with offline capability
Installation/updates	Must be deployed or downloaded	Hit refresh
User interface	Native apps are responsive and functional	Browsers can be clunky, but new advancements in JavaScript like jQuery Mobile are catching up fast
Device compatibility	Platform-dependent, hardware-dependent	Platform-agnostic, content can be reformatted with CSS to suit any device
Animation/Graphics	Fast and responsive	Web apps are getting closer, but will probably always lag
Streaming media	Few problems with audio and video. Flash works, but only if the device supports it	Flash works where supported. Browser-based audio and video are getting there, but still beset by compatibility headaches. Give it a year or two
Fonts	Tight control over typefaces, layout	Almost on par, thanks to advancements in web standards. Give it six months
Is my content searchable?	Not on the web	By default
Sharable/Tweetable	Only if you build it in	Web links are shared freely. Social APIs and widgets allow easy one-click posting
Discussion and collaboration	Only if you build it, and it's more difficult if data is disparate	Discussion is easy, all data is stored on a server
Access to hardware sensors	Yes, all of them: camera, gyroscope, microphone, compass, accelerometer, GPS	Access through the browser is limited, though geolocation is common
Development	Specific tools required for some platforms (like Apple's). You have to build a new app for each target platform	Write once, publish once, view it anywhere. Multiple tools and libraries to choose from
Can I sell it?	Charge whatever you want. Most app distributors take a slice, up to 30%	Advertising is tolerated, subscriptions and paywalls less so. No distribution costs beyond server fees
Distribution	Most app stores require approval and you have to wait	No such hassle
Outside access to your content	No, the reader must download your app	Yep, just click a link
Advertising	Control over design (though limited in iAds) and rate	More choices for design, plus access to web analytics. Rates vary widely

Two roads diverge on a tablet screen. One is the path to the native app, the other leads to the open web [16].

course, the Web technology stack (HTML/CSS/JS) is itself implemented in native code.

The distance between the native layer and the browser is just one compile away. In other

words, if you want to add a native capability to a browser, then you can either bridge it or recompile the browser to achieve that capability" [1].

Another way to picture this architectural layout would be to consider the code developed as a web application that is ultimately wrapped within a native application shell. The native application will host the web application previous mentioned on the phone itself via a browser or WebView thus allowing access to features that otherwise would not have been accessible via the traditional web application. This concept is critical because without access the device features and sensors an ordinary web app has no chance of standing up to its competitor's native app that inherently would have access to such things.

Because the application in reality is a native application that uses web technologies to provide a great deal of the functionality, applications developed with PhoneGap can be offered through the Apple AppStore. However, this also means that you must have access to a Mac, Xcode, and an Apple Developers License to develop iPhone applications using PhoneGap. Applications developed using PhoneGap can also be customized for the Android and even Blackberry devices rather quickly. This convenience allows the developer to reuse a significant amount of code when developing cross-platform applications. The true logic behind this communication relies on the APIs developed by PhoneGap and the universally accepted web technologies utilized. These APIs allow the coordination between multiple platforms using the PhoneGap APIs as the intermediary. For instance, a user selects "take picture" from the user interface, the web container will invoke the Camera.getPicture function of the PhoneGap API via JavaScript, PhoneGap's API will communicate with the specific platforms native API(s) to open the device's default camera application so that the user can take a picture. Once the photo is taken, the camera application closes and the original application is restored.

PhoneGap's API will return the image contained within an object to the web container for further use within the WebView. This relationship is depicted in Figure 3 using the myATMC project as an example.

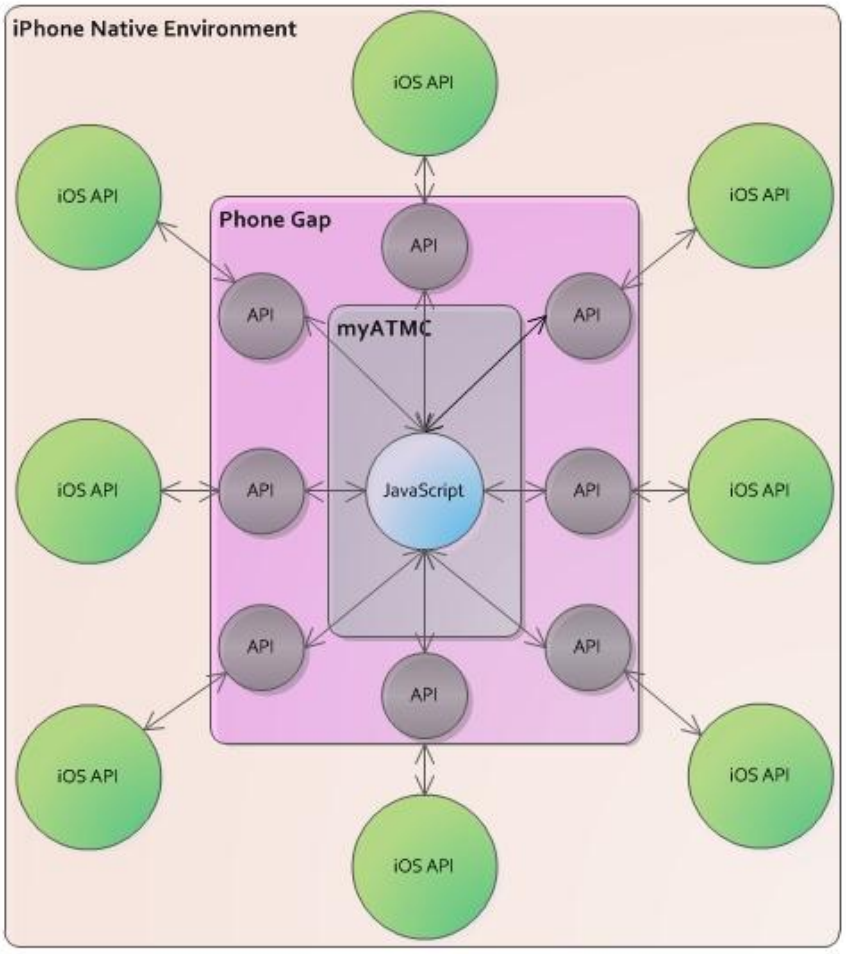


Figure 3 - PhoneGap Overview

Some platforms have begun to develop web friendly APIs that offer access to certain native information, such as geolocation. In this case PhoneGap's *"API is based on the W3C Geo location API Specification. Some devices already provide an implementation of this spec. For those devices, the built-in support is used instead of replacing it with PhoneGap's implementation. For devices that don't have geolocation support, PhoneGap's implementation should be compatible with the W3C specification"* [11]. As previously noted, the user interface plays an important role in application

development. PhoneGap and web development as a whole can circumvent this hurdle through the use of various Toolkits that have been created for mobile web development.

2.7 Toolkits

There are many toolkits available today that can be used in the development of mobile applications. These toolkits can provide various functionalities, the most common of which is the user interface design. Toolkits have been around for quite some time and are commonly used in standard web development, for example the Dojo Toolkit. They allow screen real-estate optimization, ease of user interactions, and much more. *"When we first started tapping into these next-generation mobile browsers, no framework worked properly across devices. Today there are more than 20 mobile frameworks, and support is being rapidly added to existing DOM (Document Object Model) libraries"* [1]. Most of the mobile toolkits have JavaScript driven logic combined with CSS3 style sheet aesthetics. Some commonly used mobile toolkits include, but are not limited to, jQTouch, Sencha Touch, jQuery Mobile, and DojoX Toolkit. All of these toolkits provide a plethora of visual design tools design for use on the iPhone, iPad, Android, and often more. Their differences lie in the unique features provided, event handling, transition styling and speed.

One Toolkit available, jQTouch provides a progressive-enhancement approach by providing the iPhone-esc experience layered on top of appropriate HTML base. Performance issues with regard to page transitions that are jumpy or even missing is one reported problem with this Toolkit [12]. Another option, jQuery Mobile is a relatively new Toolkit that has quickly gained attention since announced in August, 2010. jQuery Mobile is quite similar to jQTouch; however, additional UI controls and styles are available. Some problems experienced include delay in response to tap event rendered animations [12]. Sencha Touch and Dojo Mobile take a much different approach. Instead

of enhancing pre-existing HTML, it generates its own DOM based on objects created in JavaScript [12]. Sencha Touch provides a significant number of features that its competitors do not. For instance, they tackle one of the biggest challenges in developing cross-platform mobile applications which are the different resolutions found across devices. Sencha Touch employs a groundbreaking method to eliminate this problem, allowing developers to change the overall scale of their interfaces on the fly. This way, buttons are always big enough to tap, regardless of the screen they are on [13]. Sencha Touch is equipped with storage and data binding facilities in addition to a robust animation system that surpasses many competitors with simple yet flexible animations between screens and views [12]. "For all that apparent extra weight, Sencha performed noticeably better and more reliably than either jQTouch or jQuery Mobile in my test, with the exception of initial load time" [12]. DojoX Mobile Toolkit is another option available that immediately took my attention due to my standing familiarity with their web toolkit already used by ATMC. Starting with Dojo 1.5, dojox.mobile JavaScript libraries have been included to provide lightweight themes and assistance for mobile developers, as well as common transition effects. Dojo 1.5 also includes a number of HTML5 features, with many more options breaking out in 1.6. Embracing close support for PhoneGap the DojoX Mobile Toolkit include Native Mobile application support like Google Maps with Location track, Enterprise Charts for Mobile, Mobile Gauge etc.

The toolkits reviewed represent only a portion of the various options available for mobile web development; however, they are among the leaders in the market. myATMC will start as a fairly straightforward app that does not require much complexity but over time could grow to perform more and more functions. I selected DojoX Mobile Toolkit for the development of myATMC due to the extended capabilities provided, user reviews, and most of all the IS Department's current relations with the Dojo Toolkit as a whole.

DojoX Mobile Toolkit provided an effective and esthetically pleasing user interface while avoiding significant levels of debugging that are known to be experienced with other toolkits.

2.8 IDE and SDKs

The SDKs used for iPhone development are the iOS SDKs found within the Xcode suite provided by Apple that operates on an iOS operating system. The most recent suite available is Xcode 4.3.2 which includes the Xcode IDE, Instruments, iOS Simulator, and the latest Mac OS X and iOS SDKs [14]. The various iOS SDKs allow developers to create applications against the headers and libraries of different operating system versions. For example, an app for Mac OS X version 10.4 can be built while running on Mac OS X version 10.6 [14]. The Xcode suite includes a series of APIs used to access different aspects of the phones native environment in addition to many other APIs with useful functionalities. Most of the provided APIs coordinate with native applications however Xcode does provide an HTML5 API that allows access to limited features via the web. As mentioned previously, there are some SDK's available to develop iPhone applications in Windows however they are limited in their functionality, generally require subscription fees, and ultimately your code will be transformed into Objective-C then compiled within Xcode. Regardless of the initial development path before an application can be distributed through the AppStore it must have been compiled with Xcode and approved by Apple.

2.9 myATMC Approach

While considering each approach, it is important to acknowledge that *"while mobile Web apps are proliferating, there will be an active market for native apps for some time to come. A good understanding of the advantages and limitations of both development approaches will be important to successful application deployment and a*

positive user experience" [15]. The iPhone application myATMC has been developed with PhoneGap utilizing web development techniques while technically remaining a native application. This decision was based on several factors.

1. The overall speed of this application does not play a major role due to basic display functionality it performs. There are no intensive graphic requirements therefore the user interface display and interactions will not affect the applications speed at a significant level. In addition, any delay experienced by the retrieval of data from our internal database will be apparent for both the native and web application forms therefore that factor was irrelevant.

2. Web development will offer the opportunity to expand this application to multiple platforms using a similar code base in the future, reaching more customers and providing a higher level of customer service. This opportunity has not come at a major cost to the user interface as there have been significant developments in the mobile toolkit industry allowing developers to create user friendly web based applications with ease.

3. The user's access will not suffer provided the app is developed with PhoneGap because distribution through the Apple AppStore will be available for easy access.

Chapter 3: Design and Implementation

ATMC will offer an iPhone application, myATMC through the iTunes AppStore. To develop this application, careful planning was required. Following the requirements analysis of myATMC, a web application hybrid was selected for development using PhoneGap; thus, the design and implementation planning was be done to provide specifics to assist in the development process. This chapter reviews the analysis done to outline the desired look of the user interface, a use case diagram, and details surrounding the security architecture.

3.1 User Interface Design

There are many prominent wireless providers that offer an iPhone application similar to myATMC. Due to the preference of ATMC's IT manager, myATMC has been modeled in a similar fashion to myAT&T and myVerizon. This decision was made with the assumption that project teams were likely assigned to design myAT&T and myVerizon, thus regenerating those efforts was unnecessary. Screen shots of these user interfaces can be seen in Figure 4 and Figure 5. The basic layout consists of a login screen, home screen, and feature specific screens for any options available from the home screen. The only noticeable difference of myATMC is the account level user authentication as oppose to the authentication available for each individual wireless line. Account level username and password credentials are already maintained for my.ATMC.net functionality and have been mirrored in the mobile application. This authentication format required the development of a "Wireless Usage" screen. This screen is triggered by the selection of the wireless usage option from the home screen and allows the user to select a specific wireless phone line attached to a user's account. Upon section of a specific phone line the "Usage Detail" screen will then be displayed with

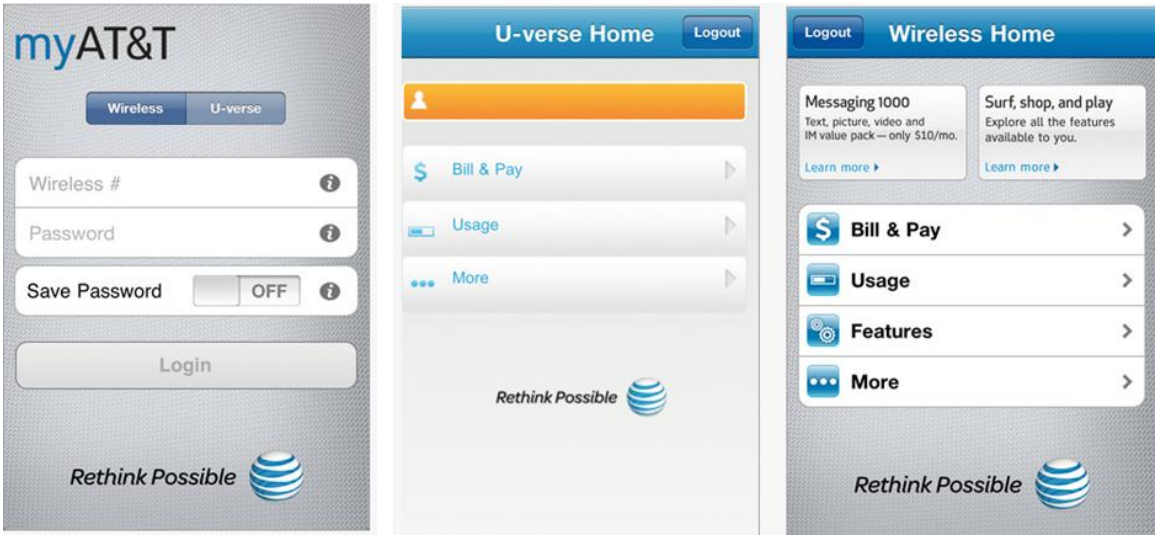


Figure 4 - myAT&T User Interface

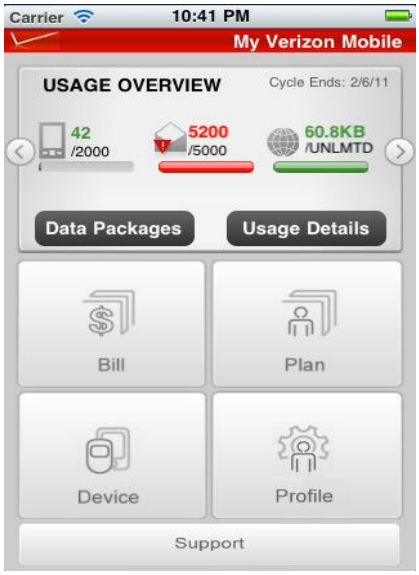
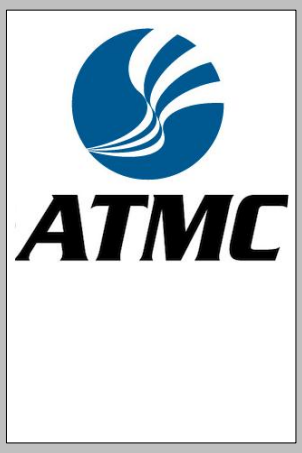


Figure 5 - myVerizon User Interface

data relevant to wireless phone number selected. The myATMC iPhone application screen shots are shown in Appendix A. The App Icon will contain the ATMC logo and are pictured below with the splash screens as well in Table 3. There are specific pixel size requirements for the icon and the splash screen depending on which device you will be developing for and whether that device has a Retina display or not. Although the dimensions for the iPad are not listed in Table 3, note that there are also varying pixel size requirements for the standard versus Retina iPad display screens as well.

Table 3 -*Required Icon and Splash Image Formats*

	iPhone Display Format	File Name	Pixel Size	Image *(Icons actual size)*
ICON	Standard	icon.png	57 x 57	
	Retina	icon@2x.png	114 x 114	
SPLASH	Standard	Default.png	320 x 480	
	Retina	Default@2x.png	640 x 960	

3.2 Use Case Diagram

The relationship between the mobile user and myATMC iPhone application has been outlined below in the use case scenario. The ATMC customer will interact with the myATMC application through login authentication, selection of a wireless number related to their account, view usage data, switch wireless phone number selected, and logout of the application. This relationship is depicted below in Figure 6.

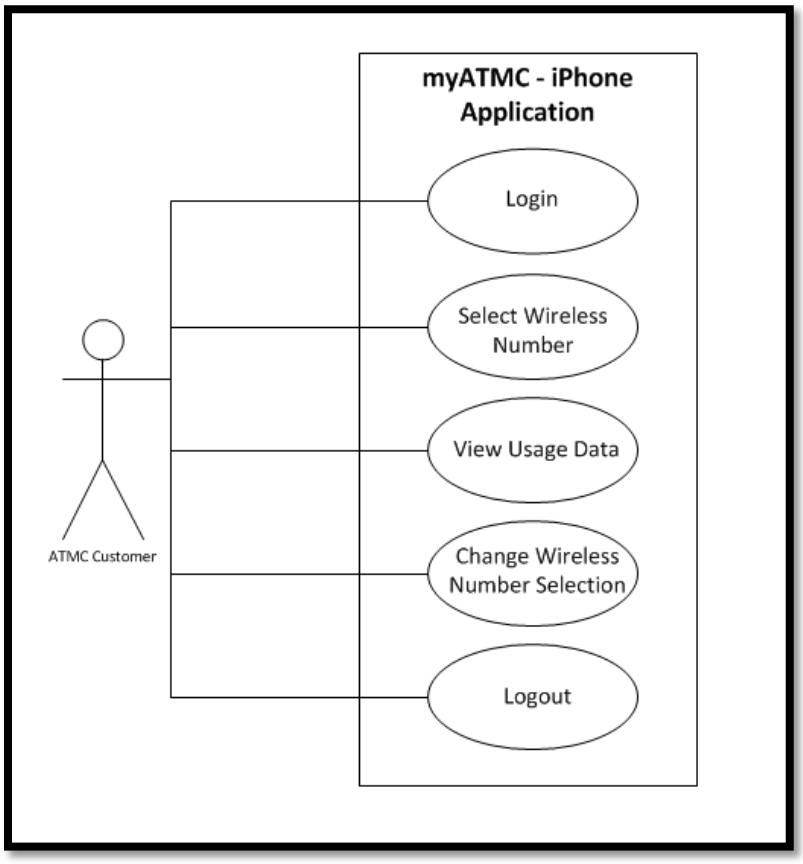


Figure 6 - Use Case

3.3 Security Architecture

The communication line between the mobile application and customer data within the ATMC database (AS400) was a security concern that had a large influence on the overall design of the architecture. An active website my.ATMC.net already existed, allowing customers to access not only their usage information but also view account details, pay bills online, setup automatic payments, and even view previous bills in PDF format. Thus, a web application (CSC) and a web service (SOWS) were previously in place within the ATMC architecture to support my.ATMC.net. CSC is used to host the my.ATMC.net website and SOWS is used to provide a secure connection to a limited amount of customer specific data on the AS400 database via encrypted XML communication. The supporting architecture for the myATMC mobile application has been built to mimic the same pattern by utilizing its own customized web application

(CSCMobile) in the place of CSC and calling customized methods within SOWS when necessary. Although a custom web application has been developed, both CSC and CSCMobile are hosted on the same server within the DMZ. This relationship is depicted in Figure 7.

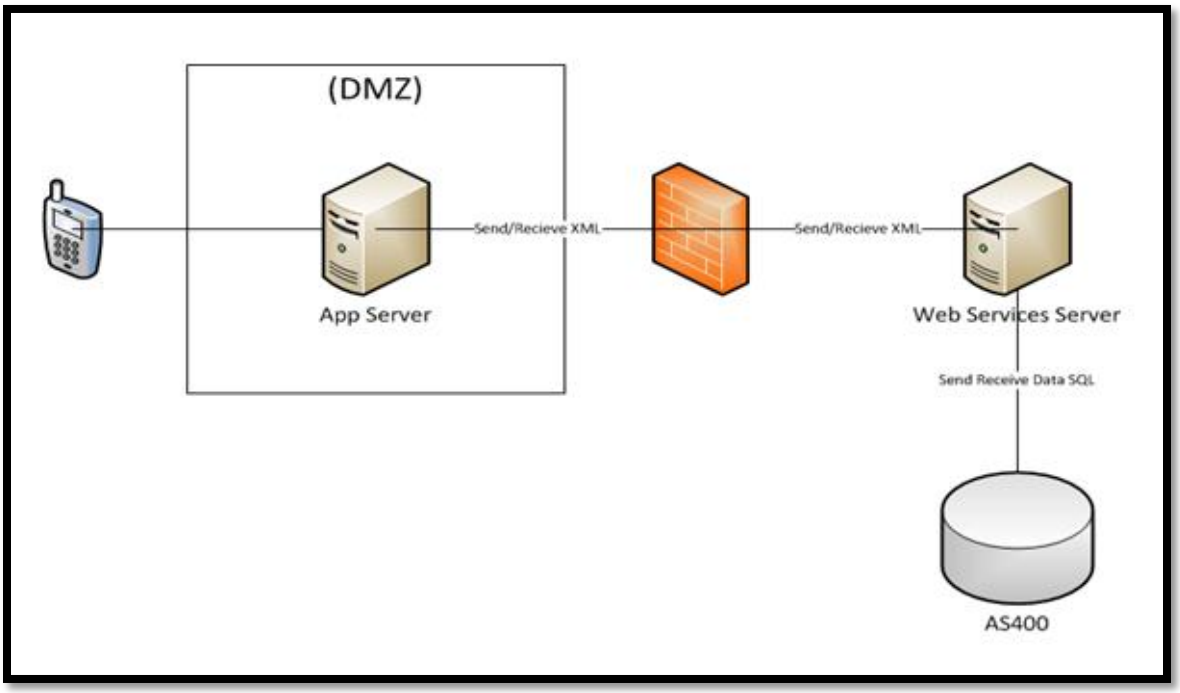


Figure 7 - Communication Architecture

The authentication request and customer information are transmitted through the firewall via XML packet communication. Consider the following chain of events surrounding a customer's use of my.ATMC.net as an example of this communication structure. First a user submits a request from their web browser to the web application hosting the website, CSC. The customer has access to CSC because it is running on an application server outside of the firewall that prevents unauthorized access to ATMC's internal network, also known as the demilitarized zone (DMZ). CSC then communicates via encrypted XML packet transmission through the firewall to the web service SOWS running on a server inside the ATMC network. The web service SOWS will process the request and following successful authentication, access customer specific data stored in the AS400

database then return a response through the firewall to the web application CSC via XML packet transmissions. The web application CSC can then interpret the XML and send the user a response to their initial request, whether it is usage information, billing information, or even a login error message.

Although, the web application CSC and web service SOWS mentioned above already exist. The mobile application myATMC will require the development of an entirely new web application CSCMobile that will communicate with the mobile device via Ajax. The web application currently used for my.ATMC.net hosts the webpage while also sending and receiving data between the user's browser and the ATMC database. In contrast, the myATMC mobile application view layer will consist of a webpage hosted by the application installed on the mobile phone itself, yet still submit request for data to a web application located on the same server as the standard my.ATMC.net web application. Therefore, the myATMC mobile application will host the HTML view layer itself using the mobile phone's localhost while concurrently submitting AJAX request to a server within the ATMC DMZ that is hosting the CSCMobile application.

Generally, cross domain communication violates the same origin policy stating that, *"If a Document or image was served over the network and has an address that uses a URL scheme with a server-based naming authority, the origin is the origin of the address of the Document or the URL of the image, as appropriate"* [21]. However, PhoneGap loads the WebView through a local html file using the file:// protocol and therefore can perform the said actions without violating the same origin policy. The myATMC mobile application submits HTTP requests via AJAX in the form of a JSON object, thus the web application receiving, unwrapping, rewrapping, and responding to such requests was custom designed to handle this sort of communication.

The web service that provides the connection between the application running in DMZ and the database (SOWS) did not require complete re-engineering however, some additional development was required. The only methods that were previously available suited the my.ATMC.net application, returning much more information than actually required for the myATMC mobile application. Therefore, new methods were developed to return a limited data set containing only information required the by myATMC mobile application. These additional methods help to heighten security, by establishing a lower security clearance for the mobile application users. Furthermore, the reduced dataset will increase processing speeds by limiting the amount of data retrieved from the database, minimizing the number of packets transmitted, and reducing the data filter processing that would have been required otherwise.

Given the project requirements there were two areas on the backend or server side that required development to support the myATMC mobile application. First, development of a separate web application was required for the DMZ server to segregate mobile users from my.ATMC.net users. Additionally new methods were required within the currently operational web service to gather customer data from the ATMC database specific to the mobile application's need. The decision to create additional methods was based on the vast differences between view layers data requirements as well as the desire of ATMC management to limit the overall data access of the mobile application users.

Chapter 4: Discussion

4.1 Development Process/Issues

To begin the development process I decided to create a web application within RAD (Rational Application Developer) using HTML and JavaScript only for the WebView and writing custom Java on the backend that could later be reused as the web service located in the demilitarized zone. This decision was made considering two main arguments. First, I was not familiar with Xcode or the iOS platform and wanted to get started developing in a comfortable environment that I was used to working in. Second, I had yet to obtain a Mac Mini and it would be a few weeks before one was available to begin the Mac side of my development process.

After looking over the various mobile toolkits available to create an iPhone GUI (Graphic User Interface) on the web side, I had decided to use DojoX Mobile Toolkit 1.6. This decision was greatly based on the previous decisions of ATMC to use the Dojo Toolkit as a user interface tool to assist with our general web development. By selecting this toolkit I would be able to fulfill my user interface needs and work with a company that was already trusted and familiar to the ATMC Information Systems department. I began the development of a web application prototype in Windows and decided that before I even tried to move my work over into Xcode I would like to achieve the following goals. First, I had to setup a GUI that was attractive to the user, functional, and styled to give the impression of an iPhone native application. In addition, the backend Java servlets must developed and deployed on my Windows PC localhost (IBM WebSphere Application Server) to receive, interpret, and respond to JSON request. Finally, the JavaScript code development was also required to send, receive, and interpret JSON objects sent to and from the Java servlets via XMLHttpRequests. I had predicted

that the Java servlet programming would not take the majority of my efforts as I have experience with this sort of development. However, I was unsure as to whether the JavaScript or HTML development would prove to be a sizeable obstacle. Although I had experience programming with the Dojo Toolkit I was not versed in the DojoX mobile libraries that were required for both the HTML as well as the JavaScript development.

Although I was not able to begin developing on a Mac Mini at first, I did seek out the registration of a developer's account with Apple. To register as a developer you must first have an Apple ID. Additionally, to setup an iPhone you must have an Apple ID and since I am the proud owner of an iPhone, I already had an Apple ID. Thus I was able to skip straight to the developer registration of which there were several options available. To register merely as an Apple Developer was free. With this membership you can download Xcode and begin development. However, you cannot publish to the Apple AppStore with this Developer status, or even deploy your application on a device. You can only run tests with the iOS 5 Simulator provided with Xcode. To publish your application with the AppStore or on a device you must be a member of an Apple Developer Program. There are three programs offered that include the Developer Program ($99/year), Enterprise Program ($299/year), and University Program (Free, pending qualification). As of yet, I am still only enrolled as an Apple Developer with the standard free developer's license. However, I have informed my boss him that when the application was ready for extensive testing and AppStore publishing, we would need to join one of the programs offered.

Once the Mac Mini equipped with an Intel processor had arrived at ATMC I was able to bring it home to download and install Xcode 3.2.1 as well as PhoneGap 1.4.1. Later, I would be required to bring the Mac Mini back to ATMC to allow easy access to the ATMC network and in turn my Windows PC that would be hosting the web

application, CSCMobile previously mentioned. The next step would be to copy the HTML and JavaScript files from the web application I had started my development with and paste them into a new PhoneGap project within Xcode. I also had to remove any code from the Java servlets originally designed to host the HTML files that had been relocated. Aside from my research thus far, I was in no way certain exactly how this transformation would play out. There was plenty of documentation available for developing with PhoneGap, Xcode, and Dojo Toolkit however none of this documentation suited the exact chain of events my application development process followed. After forming this general plan of attack I set out to create the iPhone web application myATMC. I worked through the bugs one after another until I reached a functional result that adheres to the applications requirements.

After my application had reached the prototype stage, sought help from the university to obtain the required certificate and provisioning license to temporarily deploy my prototype onto a device for presentation purposes. The certificates and provisioning licenses required to deploy an application onto a device and submit that application for publishing via the Apple AppStore are shown below in the screen shot of the Key Chain application on my Mac Mini. This Key Chain application is used to display any certificates or provisions obtained via the Apple Developer Portal online. To access the Apple Developer Portal you must be a member of an Apple Developer Program. The certificates and provisions work together to link the sponsoring Apple Developer ID (with Program Membership), your own Apple Developer ID, and your Xcode mobile application. They are used in conjunction with Xcode, iTunes, and eventually your iPhone to install the mobile application. Instructional documentation links are provided on the portal pages to walk you through this process.

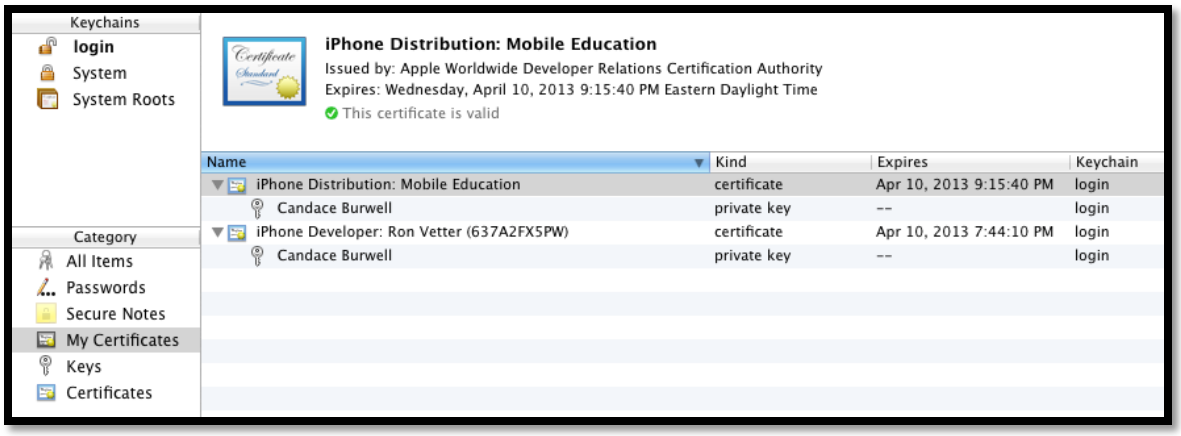


Figure 8 - Certificates and Provisioning Required for AppStore Publishing

4.2 Obstacles Experienced

From the beginning to end of my development process I encountered errors and odd circumstance that limited and occasionally halted my progress for days at a time. It is difficult to explain the irritation combined with consuming hopelessness I felt when attempting to solve a problem with inadequate documentation. I found a decent amount of information available online that helped with most of the issues I experienced. However, given that PhoneGap and especially the DojoX libraries are fairly new technologies, the documentation available was not always comprehensive. In turn, the assistance provided by documentation was inconsistent. The application myATMC, had unique requirements and at times the help documents available proved to be too vague. A snippet about the general subject matter was often available but not specifics surrounding my particular error or issue. Consequently, the blogs and discussion posts of issues or bugs experienced by fellow programmers proved to be the most helpful body of information available yet, still with some errors I was left wanting more.

4.2.1 Xcode and PhoneGap Installation Issues

To even begin the development process it was necessary to install Xcode and PhoneGap as well as import the Dojo Toolkit into my project workspace. The installation

of Xcode was relatively simple. The only real hiccup I experienced surrounded registration as an Apple Developer followed by my pursuit of a free copy of Xcode that occurred on January 9, 2011. The process of registering and locating an Xcode download without joining one of the Apple Developer Programs proved slightly confusing. Registration as an Apple Developer outside of a Program membership is free and on January 9th 2011, Xcode 4.2.1 was also available for free, whereas the download of Xcode 4.3 (Developer Preview) required a Mac Developer Program membership. While browsing the internet to locate a free Xcode download I keep finding myself directed to the Xcode 4.3 (Developer Preview) webpage and it took a bit of searching to find the free solution that was available. This search was not a tremendous struggle however I was left with the feeling that it was not as straightforward as it could have been. Xcode 4.3.2 has become available since my initial download offered free of charge to any registered Apple Developer and Xcode 4.4 (Developer Preview) is now available to any Program member. Xcode 4.3.2 is now simple to locate and grouped with its predecessors all the same webpage for easy access. Once Xcode was successfully installed I moved on to the PhoneGap installation. The download was easy to find and a "Get Started Guide" (http://phonegap.com/start/) was also made available through the PhoneGap website. There were a few steps that I found slightly confusing that surrounded copying the "www" directory into the project via the finder view as well as instructions to drag and drop this copied folder into the apps project folder. This is fairly simple but to ensure I understood the directions clearly without apprehensions I searched YouTube.com and found numerous tutorial videos walking through the exact same steps and more. Since the YouTube videos found were typically home videos posted by amateur developers not a recognized research source, any concepts or ideas gleaned had to be proven before relied upon. I found the plethora of videos available exceptionally helpful by allowing me to

visualize the programming of some basic PhoneGap applications in Xcode, two environments I had absolutely no coding experience with. Furthermore, the videos provided visual insight pertaining to basic navigation around a Mac, yet another environment that I had little to no experience using. Figure 10 shows a screen shot of the final folder structure of the myATMCmobile project from Xcode after PhoneGap was installed and development had reached a prototype stage.

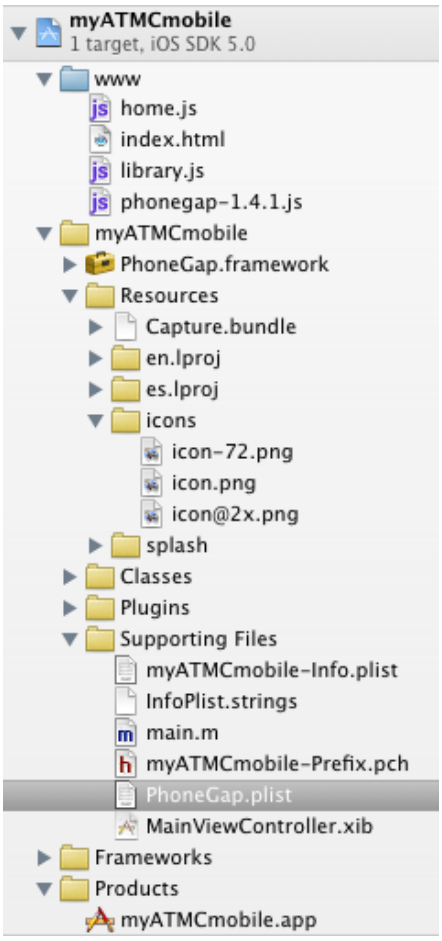


Figure 9 - myATMC Xcode Application Architecture

4.2.2 Dojo Installation Issues

Before I began the backend development of the myATMC web application I had setup on my Windows PC to get started. I wanted to setup a user interface using only HTML and the Dojo Toolkit. The latest Dojo Toolkit available is Dojo 1.7 that includes the mobile JavaScript libraries DojoX. I soon realized that Dojo 1.7 was in a sense the

same as the Developer Preview versions of Xcode, meaning that errors were still apparent in this release. In the same respect although some errors existed within the 1.7 version there were also additional features and solutions to other issues never fully resolved in version 1.6.

Additional research led me to the realization that Dojo Toolkit would most likely always have some problems here and there in their latest releases, however nothing that would prevent general use of the Toolkit. They release a new version prematurely depending on fellow developers to find and report bugs. The minor simple fixes are corrected whereas some more complex bugs and additional feature requests are scheduled for the next version release that will inevitably offer new features and new bugs. Upon downloading and importing the Dojo Toolkit 1.7 I found multiple errors in the JavaScript files that Rational Application Developer would not allow me to easily ignore. I then tried Dojo Toolkit 1.6 and found fewer errors, however some still existed.

For a temporary fix I decided to use the link to the Dojo Toolkit cloud hosting through google.axis as simple work around to get started. I have continued using this functionality in Xcode in attempts to continue moving forward with my prototype development. Eventually, I will need to load the Dojo Toolkit into Xcode to prevent the need to access these files via the internet and permit me to remove the relative External Hosts entries that I will explain in greater detail later. Being that Xcode runs in the iOS environment, I feel the implementation of Dojo should be more cooperative as the SDK will not attempt to validate all of the Dojo libraries like RAD.

Another hurdle of the Dojo installation process surrounded the multiple different downloads available from the dojo release webpage all varying in size and name. These options included one link with the named dojo-release-1.7.2 and the others including this same root name plus -src, -demos, and -shrinksafe. Each of these options came with the

file extensions of .tar.gz, .tar.gz.md5, .zip, .zip.md5 which made more since to me being the various compression file formats .tar.gz being the newer tighter compressed option commonly referred to as gzip versus the .zip file that represents the classic compressed file extension. The .md5 stands for a message-digest algorithm a widely used cryptographic hash function in some cases used to check data integrity [23].

While I do have experience coding with the Dojo Toolkit libraries, this had been my first attempt to download, decompress, and install a toolkit from start to finish and the ill labeled files had caused unnecessary confusion. After looking around the Dojo website for some relative help documentation, I soon found exactly what I had needed with a simplified download selection accompanied by descriptions that assisted in my selection (http://dojotoolkit.org/download/). Clearly the second download page found was intended for the novice developer and the initial page was for the experience patron.

The Dojo Toolkit 1.7.2 Release (3.7mb-5.3mb) was described as the correct download for immediate deployment containing the complete Dojo, Dijit, and DojoX projects with any non-essential files removed including test and demonstrations. The Dojo Base (31kb) only contained the base Dojo APIs which clearly was not the correct option for me. The third option was the Source download (18mb-21mb) noted as ideal for development containing all tools needed to build, document, test, and develop with Dojo. I started off with the Source download before transitioning to the use of the cloud hosting however, the first option mentioned will eventually be the download I use due to the reduced size and my projects requirement of the DojoX tools.

4.2.3 Session Management

Due to the authentication requirements of myATMC, a user session must be created and managed each time a customer logs into the mobile application. Initially, I used the session management features already incorporated into the RAD web application

project that I began development with before transferring the view layer to Xcode. By selecting a web application project type my initial project was automatically setup to work with the web server, WebSphere 7.0 to manage session tracking and timeouts. At first I assumed that this functionality would not carry over when I was required to transfer the view layer into my Xcode project. However, this assumption was incorrect and I was pleasantly surprised when I discovered that in fact the default settings originally applied to the CSCMobile web application were capable of managing this service for me even when using the XMLHttpRequest functionality to submit requests to the servlets. This meant that I did not have to add any customization to CSCMobile or my Xcode application to manually write cookies, track timeout rates, or record a session id.

4.2.4 Dojo Toolkit Issues

The Dojo Toolkit utilizes a customized Document Object Model (DOM) which is different from many of the other mobile WebKits available. While this method offers certain advantages to their model it also proved to be somewhat tedious to use as a beginner. I had to learn how to use the various objects to set, create, and update their attributes. Although these objects followed a standard methodology for the most part there were a few differences in the handling here and there that took time to work through. For example, setting the label of a header bar dynamically to show the phone number of the corresponding usage data made the back arrow disappear. This issue will subsequently prevent the user from navigating back to a previous menu. A work around I used was to simply add a RoundRect div within the primary div and set that area dynamically instead, which actually turned out to be more visually appealing anyway. This was a quick fix but highlights the same sort of issues I experienced throughout development with the Dojo Toolkit.

4.2.5 PhoneGap Issues

When an application is developed with PhoneGap any server accessed outside of the iPhone environment must be added to the External Host list found within the PhoneGap.plist file. From past experiences at work to access the local host on my machine I would typically use the machine name followed but the context root of whichever application or specific servlet I was attempting to access. After hours of trial and error I finally determined that instead of my machine name, the actual IP address of my PC was required to allow the connection without triggering whitelist errors within the mobile application, preventing the XMLHttpRequest transmission. In addition to this the external host IP address had to be entered as 000.000.000.000* instead of 000.000.000.000/* or 000.000.000.000/Login or 000.000.000.000/Login?username=candace&pw=whatever. The asterisk character represents a wildcard, allowing any of the various file paths after that character to be accessed. This contributed to my confusion and the drawn out debugging process. These two issue combined made it difficult to determine the root of the problem. I used the Wireshark application to view network activity as well as research online to finally determine the correct solution. An example of the correct entry is show below in Figure 9 where I have replaced the actual IP address with zeros in relation to my previous reference.

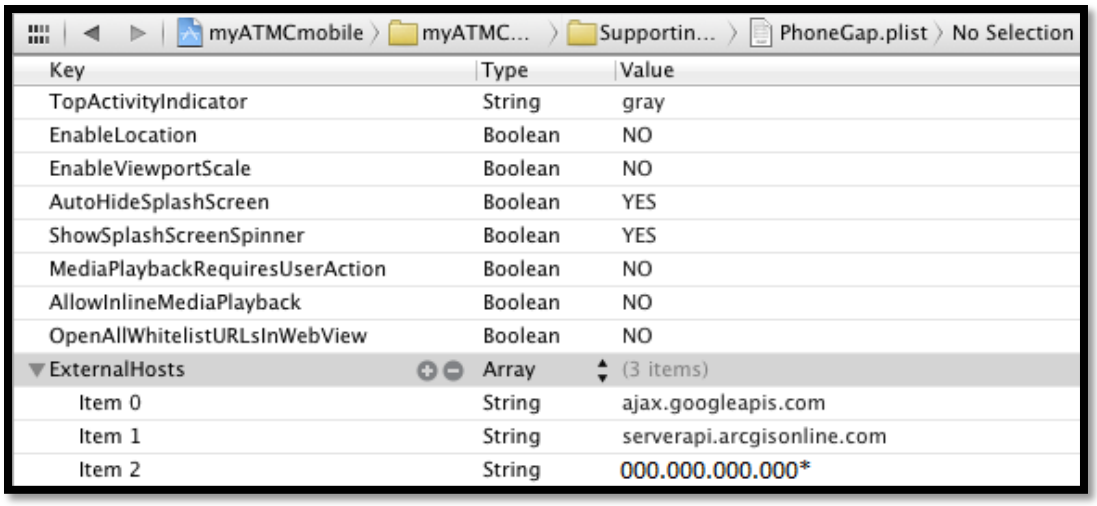


Figure 10 - PhoneGap External Host Requirement

Another issue I experienced while moving the view layer of my application into the Xcode application using PhoneGap surrounded the window.location command in JavaScript used to forward the view to a new page. This page forward reloaded the view to and in essence cancelled any events that were lingering in the background. I soon realized that with PhoneGap the window.location command was not quite so simple and loading another page after the initial page load proved unnecessarily difficult. In attempts to move forward I simple changed the new page to that standard div transition I was already using later in the application. Another, encounter along the same concept surrounds the inability to open a Safari to load from an external URL. Upon researching this issue I found a proposed solution to open within application if URL is listed in the Default Host records of the PhoneGap.plist file, otherwise default to the Safari web browser. I felt this was a good suggestion and will hopefully be applied in a future release of PhoneGap.

4.2.6 Click Event Management

After the user has successfully logged in, the list items within the usage div are dynamically created with a click event trigger attached to each. Upon selection, the list

item should submit a request to the database for relevant usage data. Then the usage detail content will be destroyed if necessary and created with the relevant usage details requested. The procedure to create the wireless telephone number list was relatively straight forward. However, the process of attaching a click event dynamically after the page load had already occurred was not so simple. To attach click events to list items that are setup from the initial page load was not problematic but to attach this event function to the list items dynamically I had to use a dojo.connect approach which proved to be confusing.

I spent a great deal of time first attempting to just set the click event reference while creating each individual list item instance, I also tried to connect the click event reference to the list items after they had been created by referencing each list item's id property, and then I attempted all sorts of combinations of the dojo.connect and dojo.hitch functions. The various attempts resulted in click events that would not trigger at all, always trigger the first list item's click event regardless of actual selection, or trigger each one of them one after another. I determined the two main causes of these results were either the click event was never added to the list items at all or the click events were created with an improper scope. The incorrect assignment of a click event can cause issues with bubbling. When a click event handler is set for an object that has been clicked, the event is triggered. However, if no event handler is set for that object, the click event bubbles up (like a bubble in water) to the objects parent. The final solution I found, was to build and attach the unified list with all list items required, then perform a query on unified list that would return each list item node found within the referenced unified list, then I could loop through each list item object and attach the click event using the dojo.connect functionality. This process attached the click event to each list item node individually with the correct scope and did not cause any bubbling issues. The

length of troubleshooting required can be attributed to my lack of experience with the HTML DOM Event object behaviors but also to the lack of proper referencing functionality provided for the customized DojoX unified list object. I found that the functions developed to interact with the different DOM objects within the DojoX Toolkit were analogous for the most part however the random inconsistencies where irritating.

Overall from my experience with this project and as a software developer in general I can acknowledge that while writing code you will inevitably encounter obstacles and errors caused by your own negligence. Once realized, the underlying issue can seem quite trivial, that misspelled word you overlooked for hours or a careless dash drastically underestimated. This process can be comical at times and absolutely disparaging at others nevertheless, it is an inevitable experience. The simple poem A Programmer's Lament sums it up so clearly stating, “I really hate this damned machine; I wish that they would sell it; It never does quite what I want; But only what I tell it” [22]. On this project I wasted hours on dash placed in an External Host entry. I had tried removing the dash but not before changing another variable to the equation, thus leaving me to believe the dash was not the underlying issue. This project has proved to yet again remind me that I should always be aware of the finicky nature of computer programming. This awareness will help to minimize similar experiences in the future when troubleshooting and debugging an application.

4.3 App Review and Testing

Although the mobile application myATMC has not yet been submitted to Apple for approval, this is the ultimate goal of the project. The additional work required to allow this includes developing catches to handle each sort of possible error that may arise, extensive testing to prove the functionality and security of each piece that has been developed, and publishing the web application on a web server in the demilitarized zone.

Once all of these requirements have been fulfilled I will finally be able to submit myATMC to Apple for their final review and potential publishing. However, since this will be my first AppStore submission I'm predicting that it will be a trial and error process as Apple may deny my request and require additional changes.

I have tested this application throughout development and have proven the overall functionality. However, the application is still in a prototype state of development and adequate measures have not been taken to catch each potential error or odd circumstance that may arise. Some of these issues include handling an unavailable response message (web application in DMZ available but cannot communicate with internal ATMC database), repetitive invalid login attempt limitations, session timeouts, appropriate session termination, various XMLHttpRequest responses possible, and any other potential issues found from additional testing. While I have started to develop some of these features sufficient testing has not been done to prove their reliability. At this time the application has only be installed and tested using the iOS 5 Simulator and my personal iPhone 3GS device. Additional, testing must be done and corrections made before submission to the AppStore can be considered. The validation of myATMC will involve test cases laid out to verify the login, data usage display, error handling, and logout functionalities. These test procedures will be performed on multiple devices from the iPhone 3GS to the iPhone 4 in addition to a range of iOS versions to ensure myATMC is effective in the various iPhone environments. These tests must be performed on a range of accounts with various levels of active wireless line counts. My manager has requested that extensive testing be done to prove that there are no holes that would allow someone to exploit the web application and its access into the ATMC internal network. Furthermore, satisfactory completion of these testing procedures will help to ensure a

great customer experience using the application once it is available through Apple's AppStore.

Currently, the web application that will provide the communication link across the firewall and into the ATMC internal network is being hosted on my work PC. This affects the mobile apps ability to communicate with the said web application. Until this migration is complete myATMC will not allow the user to login unless the iPhone device has been linked to the ATMC network via a VPN (Virtual Private Network) or authorized wireless connection. After additional error handling and debugging has been completed, the web application will be moved onto a web server within the DMZ so that additional testing can be performed with the project architecture setup in true form.

There are some specific tests that the mobile application must pass in order to receive the required approval by Apple to publish the application on the iTunes AppStore. Some of these tests include matching icons from (store vs. home screen), violations of the Human Interface Guidelines, button consistency, limited bandwidth usage over cellular networks, appropriate popup notification when no network is found, false claims of missing network, and OS compatibility for every version support claimed. These are only a few examples of the many ways an iPhone application will be tested by Apple for AppStore approval. Being aware of the complexity of Apple's review process warrants extensive testing in preparation for an AppStore submission to prevent or at least limit any changes requested by Apple.

Chapter 5: Summary and Conclusions

The project of developing a mobile application for my current employer, ATMC was inspired by the applications that are currently available to the customers of our competitors. The functionality I sought out to mimic was the ability for customers the check their wireless usage information via a mobile phone application. The first phase of this project addressed the iOS platform and specifically iPhone devices with hopes of expanding service iPad as well as the Android platform. ATMC holds customer services to the utmost importance and strives to constantly improve in this area. The motives for developing this mobile application were to fulfill customers' needs while also exploring a subject that was completely foreign yet fascinating to me. Because mobile application development was a genre I had no experience with, I was able to approach this study and report on my experiences as a beginner. While there are topics reviewed that may seem elementary to an experienced developer, this paper is intended to give a general overview of the mobile development subject as a whole. Initially I reviewed the development options available along with their associated advantages and disadvantages. After I had derived a knowledge base I adopted a method of development and proceeded to design and implement the myATMC mobile application.

The widespread technical requirements of this application lead to the incorporation of multiple programming languages, development tools, and hardware platforms. For example, some programming languages required by this project include HTML, JavaScript, XML, Java, and Objective C. I have also used multiple different development tools that include but certainly are not limited to Xcode, PhoneGap, Rational Application Developer, Dojo Toolkit, and Ajax. Upon beginning this project I had not used a Mac for more than few hours over the last decade, yet the interaction with this platform was unavoidable and a key assets of this project. Needless to say it was

extremely difficult for me to adapt to using a Mac not only for programming purposes but also for simple navigation. I realized almost instantly how different the two platforms were and how ingrained my habits of simple everyday tasks were to the "Windows way." To summarize the results, I performed a detailed analysis of the trials and tribulations personally experienced throughout the development cycle of this project accompanied by their purposed remedy.

Despite the ongoing trial and error of the myATMC development process I was able to complete a functional prototype of the application to defend. Although the current model of the mobile application is functional, it is not ready to submit to Apple for review. The application has yet to reach the final stages of development and additional testing procedures must be executed. While there is still work to be done, the myATMC prototype has been successfully installed on my personal iPhone 3GS device and fulfills the basic functionalities outlined in phase one of the myATMC mobile application project. The functional processes of prototype include user authentication, navigation through screens, dynamic creation of a wireless number list, the display of relative usage data, and more. The application performs these functions with a user interface that predominantly upholds the look and feel of a native application. Overall I'm incredibly pleased with the progress so far and feel that this app will be appealing to a large customer base once it is ready for deployment. This project is extremely important to me and the Information Systems Department of ATMC. Therefore despite the additional work and time involved, the application will be completed and published as soon as possible.

The evolution of this project will undoubtedly continue over time. The long term goals include a customer satisfaction survey and updates to the application itself. These

updates may include suggestions from the survey, platform expansion, and additional features.

Once the mobile application myATMC is finally completed and published in the AppStore, I intend to perform further evaluation to ensure that the web application meets the needs of ATMC's customers. A usability study will be performed to determine the applications difficulty of use and the overall customer satisfaction rating. The results of this study will be based on a satisfaction survey that will be distributed among a small test group of ATMC employees of various age, sex, and technical ability. The results of this survey will be analyzed and taken into consideration upon future development. If the survey results show high customer satisfaction and usability ratings, it will ultimately prove that for this circumstance a web application was able to replace the need for native application effectively.

Over the long term it is my goal to improve the mobile application as a whole while expanding the customer usage base. This will be done through update functionalities to incorporate any suggestions found from the customer satisfaction survey as well as the addition of features similar mobile applications provided by competitors. The option of feature additions is limitless but some of the possibilities include the option to view more account data outside of wireless information, the ability to add new services to accounts, mobile billing, as well as dynamic advertising for marketing campaigns or new products that could appear in screens with available real estate. These additions will be incremental and some will require additional security considerations to be made.

Furthermore, I intend to expand the mobile platform availability to both iPad and Android devices. These platforms are both supported by the Dojo Toolkit as well as PhoneGap. At first I would like to perform an update for the iPad. This update will

require a change to the project Targeted Device Family record to include the iPad device type. This referenced file is standard in all Xcode applications and denotes the compatibility of the application with each of the various iOS devices. PhoneGap will require iPad appropriate icons and splash screens with pixel ratios to suit the different iPad models being supported. Finally, the user interface will also require additional testing to ensure that difference in screen size does not affect the overall appearance or functionality. It is common for some additional tweaking to be required when using mobile Toolkits to style both an iPhone and iPad application. However, using the Dojo Toolkit this should not cause any significant issues. These are some of the considerations I will face when the time comes to move into the iPad market.

The potential for future development is endless, only extended by the choice to develop the mobile application with PhoneGap and web development principles. This development approach has extended the scope of this application with regards to additional platform development. As a whole I was very pleased with the PhoneGap Framework and feel the benefits provided make it a viable choice for mobile application development.

References

1. Andre Charland and Brian LeRoux. 2011. Mobile Application Development: Web vs. Native. *Queue* 9, 4, Pages 20 (April 2011), 9 pages. DOI=10.1145/1966989.1968203 http://doi.acm.org/10.1145/1966989.1968203

2. Tommi Mikkonen and Antero Taivalsaari. 2011. Reports of the Web's Death Are Greatly Exaggerated. *Computer* 44, 5 (May 2011), 30-36. DOI=10.1109/MC.2011.127 http://dx.doi.org/10.1109/MC.2011.127

3. Tommi Mikkonen and Antero Taivalsaari. 2008. *Towards a Uniform Web Application Platform for Desktop Computers and Mobile Devices*. Technical Report. Sun Microsystems, Inc., Mountain View, CA, USA.

4. Antero Taivalsaari, Tommi Mikkonen, Dan Ingalls, and Krzysztof Palacz. 2008. *Web Browser as an Application Platform: The Lively Kernel Experience*. Technical Report. Sun Microsystems, Inc., Mountain View, CA, USA.

5. Mache Creeger. 2011. ACM CTO Roundtable on Mobile Devices in the Enterprise. *Queue* 9, 8, Pages 10 (August 2011), 9 pages. DOI=10.1145/2016036.2016038 http://doi.acm.org/10.1145/2016036.2016038

6. Clément Quinton, Sébastien Mosser, Carlos Parra, and Laurence Duchien. 2011. Using multiple feature models to design applications for mobile phones. In *Proceedings of the 15th International Software Product Line Conference, Volume 2* (SPLC '11), Ina Schaefer, Isabel John, and Klaus Schmid (Eds.). ACM, New York, NY, USA, , Article 23 , 8 pages. DOI=10.1145/2019136.2019162 http://doi.acm.org/10.1145/2019136.2019162

7. Richard E. Sweet. 1985. The Mesa programming environment. *SIGPLAN Not.* 20, 7 (June 1985), 216-229. DOI=10.1145/17919.806843 http://doi.acm.org/10.1145/17919.806843

8. Ryan Paul. July 7, 2010. Android 2.2 demolishes iOS4 in JavaScript benchmarks. Ars Technica. http://arstechnica.com/gadgets/news/2010/07/android-22-demolishes-ios4-in-javascript-benchmarks.ars

9. Team Marmalade. (2011, October 04). Made with Marmalade, Trading name of Ideaworks3D Limited, 1998. Retrieved 04 2011, October, from Made with Marmalade: http://www.madewithmarmalade.com.

11. PhoneGap. (2011, November 11). PhoneGap, Copyright 2011 Adobe Systems Inc. All rights reserved. Retrieved 11 2011, November from PhoneGap: http://phonegap.com.

12. David Feldman. January 23, 2011. Comparing Mobile Web (HTML5) Frameworks: Sencha Touch, jQuery Mobile, jQTouch, Titanium. Copyright 2010 Powered by WordPress, PHP, and MySQL.

13. Sencha. (2011, November 11). Sencha, Copyright 2011 Sencha Inc. All rights reserved. Retrieved 11 2011, November from Sencha: http://www.sencha.com/.

14. Apple. (2011, November 11). Apple, Copyright 2011 Apple Inc. All rights reserved. Retrieved 11 2011, November from Apple: http://www.Apple.com.

15. Vetter, R., Challenges and Opportunities in Mobile Web and App Development - Guest Editor's Introduction, Computing Now, Nov. 2011. Web - http://www.computer.org/portal/web/computingnow/archive/november2011

16. Michael Calore. (2011, August 17). Michael Calore, Copyright Wired.com 2011 Condé Nast Digital. All rights reserved. Retrieved 25 2011, November from Web Moneky: http://www.webmonkey.com/2010/08/how-do-native-apps-and-web-apps-compare/.

17. D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard, W3C Web Services Architecture Working Group. February, 11 2004. Web Services Architecture Requirements. W3C, Copyright 2004. http://www.w3.org/TR/wsa-reqs.

18. S. Stevenson. Mac Attack Apple's mean-spirited new ad campaign. June 19, 2006. Slate is published by The Slate Group, a Division of the Washington Post Company. The Slate Group, LLC, Copyright 2012. http://www.slate.com/articles/business/ad_report_card/2006/06/mac_attack.html

19. M. Nizza. Those Real-Life PC and Mac Guys, Blogging the News with Robert Mackey. May 31, 2007. Retrieved 01 2012, April from The Lede Blogs NY Times: http://thelede.blogs.nytimes.com/2007/05/31/those-real-life-pc-and-mac-guys/

20. Daphne Yarbrough-Jones. ATMC Voted “Best of Brunswick” for Fourth Consecutive Year. Feburary, 2 2011. Retrieved 01 2012, April from ATMC: http://online.atmc.net/corporate/index.php?id=93df141546946026b5ed09cc65ebc234&display=detail

21. Retrieved 12 2012, April from W3C: http://www.w3.org/: W3C. http://www.w3.org/.

22. Van Tassel, D. and Van Tassel, C.L. The Compleat computer. Science Research Associates 1983. ISBN 9780574214157.

23. R. Rivest. The MD5 Message-Digest Algoritm; Request for Comments: 1321. MIT Laboratory for Computer Science and RSA Data Security, Inc. April 1992. Retrieved 12 2012, April from W3C: http://tools.ietf.org/html/rfc1321

Appendix A:

myATMC Application Screen Shots

Figure 11 - Login

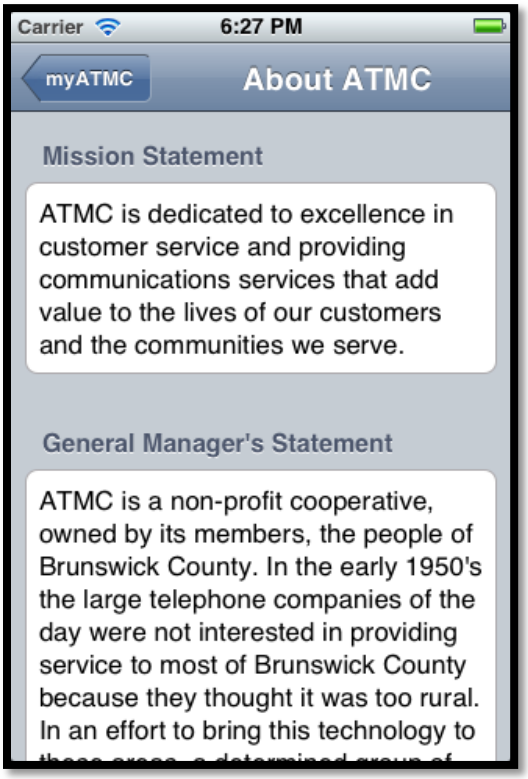


Figure 13 - About ATMC Screen

Figure 12 - Home Screen

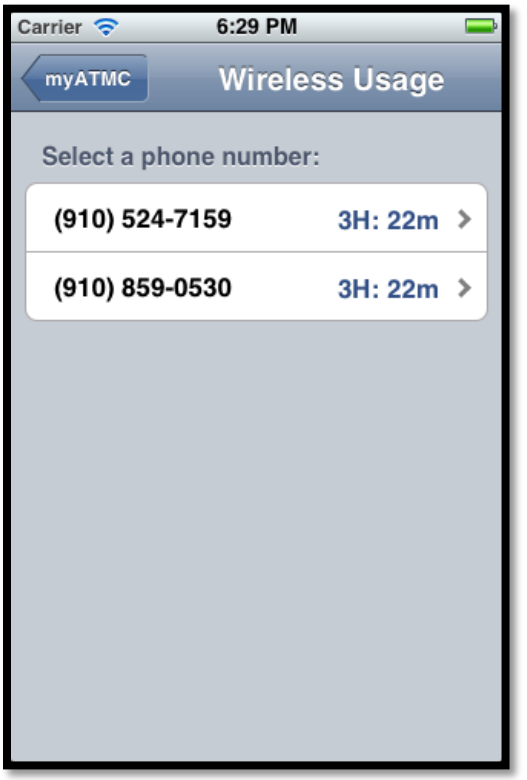


Figure 14 - Wireless Usage Screen

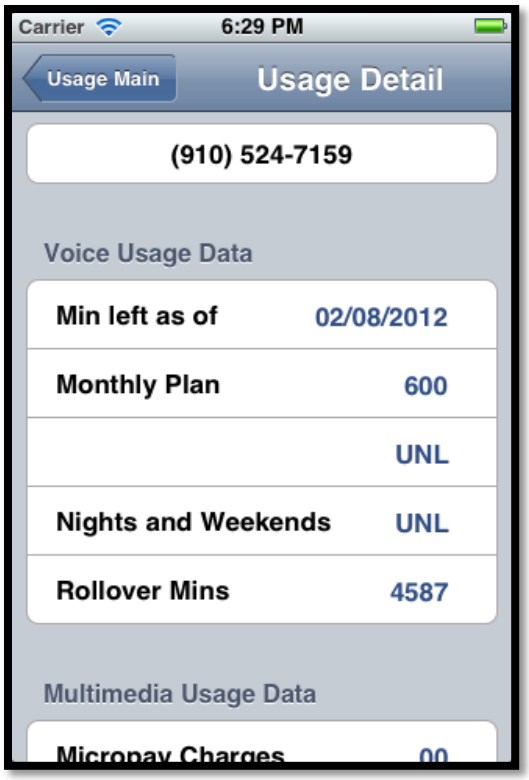


Figure 15 - Usage Detail

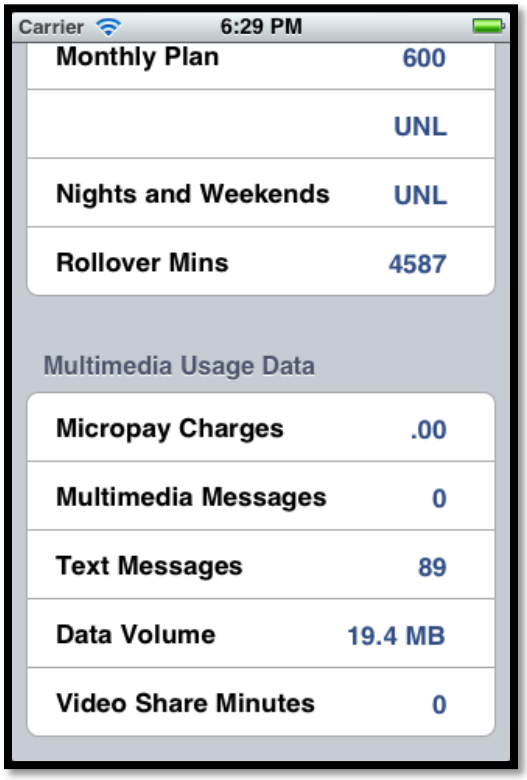


Figure 16 - Usage Detail Cont...

Appendix B: Source Code

This appendix has been removed due to potentially confidential / proprietary content.