2012

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

Going Mobile

With

Warehouse Skateboards

Ricardo A. Valea

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 Operation Management

University of North Carolina Wilmington

2012

Advisory Committee

Dr. Ron Vetter, Chair	Dr. Jeff Brown

Dr. Bryan Reinicke

Abstract

Going Mobile with Warehouse Skateboards, Author: Valea, Ricardo, 2012. Capstone Paper, University of North Carolina Wilmington.

This paper describes software design and development considerations needed to implement and distribute an iOS application for an established web based company called Warehouse Skateboards. The purpose of this application is to blend social media with traditional web content to allow the end user to have an enriching and mobile driven experience. In this paper you will find detailed information on the application development process, including actor diagrams, data model and user interface design documents. Also included is a discussion of the technical challenges encountered while integrating social media with more traditional web content into a modern mobile application. The paper concludes with the design of an evaluation form that product stakeholders can use to assess the application.

List of Figures

Table of Contents

Chapter 1: Introduction

Warehouse Skateboards is a huge warehouse with all the gear skaters' need from the most innovative skate companies to the top brands in the industry. Warehouse Skateboards is a one-stop shop for all skateboarders' needs. Skaters can get the latest decks and wheels to the coolest threads. Warehouse Skateboards primary goal is customer satisfaction. The founder and president Mike Duncan feels that "Your customer satisfaction is our top priority. We want to make sure your experience with Warehouse Skateboards is positive so you'll tell your friends about us! Now get out there and rip it up!"

Figure 1 – Warehouse Skateboards Homepage

Warehouse Skateboards are always looking for new innovative ways to help promote the company and further enhance clientele satisfaction. With the birth of smart phones such as the

Apple iPhone and Google Android, Warehouse Skateboards is looking to conquer a new method of marketing towards the skater community – mobile marketing. The goal of this project is to develop an iOS mobile application that will integrate the company's social network and traditional web site with the company's newest and most popular products. The application is intended to be a cool, interactive, and fun way to communicate with Warehouse Skateboards.

1.1 Purpose of Project

There are three key goals that need to be achieved. The first goal of the project is to help improve customer interaction, satisfaction and sales with Warehouse Skateboards (WS). Currently, customers can interact with WS through multiple online social network (SNs) mediums. These SNs are Facebook, Twitter, YouTube and Blogs. However, due to the size of the iOS devices being used for implementation and the complexity of the application, Facebook and/or Twitter are the only two mediums incorporated into the software for this project.

Another major goal for the project is to make the software reproducible for other Sage Island clientele. This capability gives Sage Island the ability to market additional features to their already extensive web development and marketing packages. The key term for this goal is reproducible; meaning in a short amount of time the software can be altered to fit another web site's needs. Since Sage Island incorporates social media into a majority of their clienteles web sites, the Facebook and/or Twitter functionality is a must have for all versions of this software.

Since these applications are intended to be freely downloadable from Apple iTunes Application Store (App Store), we needed to provide ways that the software would enhance WS sales. In order to accomplish this goal, we incorporate company specials, coupons and new arrivals within the application. These features change in real time and give the web site

administrator the ability to control the content that is being display by the software. One of the most important design considerations was to have the ability to update app content without having to resubmit the software application back to the App Store.

1.2 Significance of the Project

Presently, there are a number of web-based retail providers for skateboard paraphernalia. They all provide different levels of customer satisfaction, different amounts of product support, and different levels for ease of use with the websites. WS's currently provides top-level customer satisfaction with all current brands related to the skateboard industry, as well as an easy to use interface for the clientele to shop with. More importantly WS's provides a fully functional social media medium for customers to: tweet about current products, write reviews and answer customer satisfaction surveys about the site and interact with other clients and staff within the WS community. Some competitors mimic WS's website functionality thus diluting WS's web presence. One way to make WS stand out from the crowd is to be on the front line of technology advances and integrate their existing online resources with mobile technology.

1.3 Research Questions

The research project explored a couple of questions of interest to WS, namely,

1) Will current users of WS embrace the mobile platform?
2) Is the iOS platform suitable and appropriate to satisfy WS business requirements (e.g., better customer satisfaction, improved customer interaction, increased sales)?
3) Can a single software framework be developed for WS and then effectively be applied to other web based businesses?
4) Is the mobile content updatable without having to re-upload the app to the store?

Chapter 2: Background Technology

2.1 Xcode Environment

Xcode is a suite of development tools designed by Apple for the creation of Mac OS X and iOS software. These tools are required to run on an Intel based Apple computer and are available to registered Apple developers. In order to become a registered developer, you need to apply at http://developer.apple.com/programs. The cost of the program is $99/year. Alternatively you can purchase a stand-alone copy of Xcode through the Mac Application Store for $4.99. Interface builder, Instruments, Graphical Debugger and Zombie detection are just a few example tools included in the Xcode Environment.

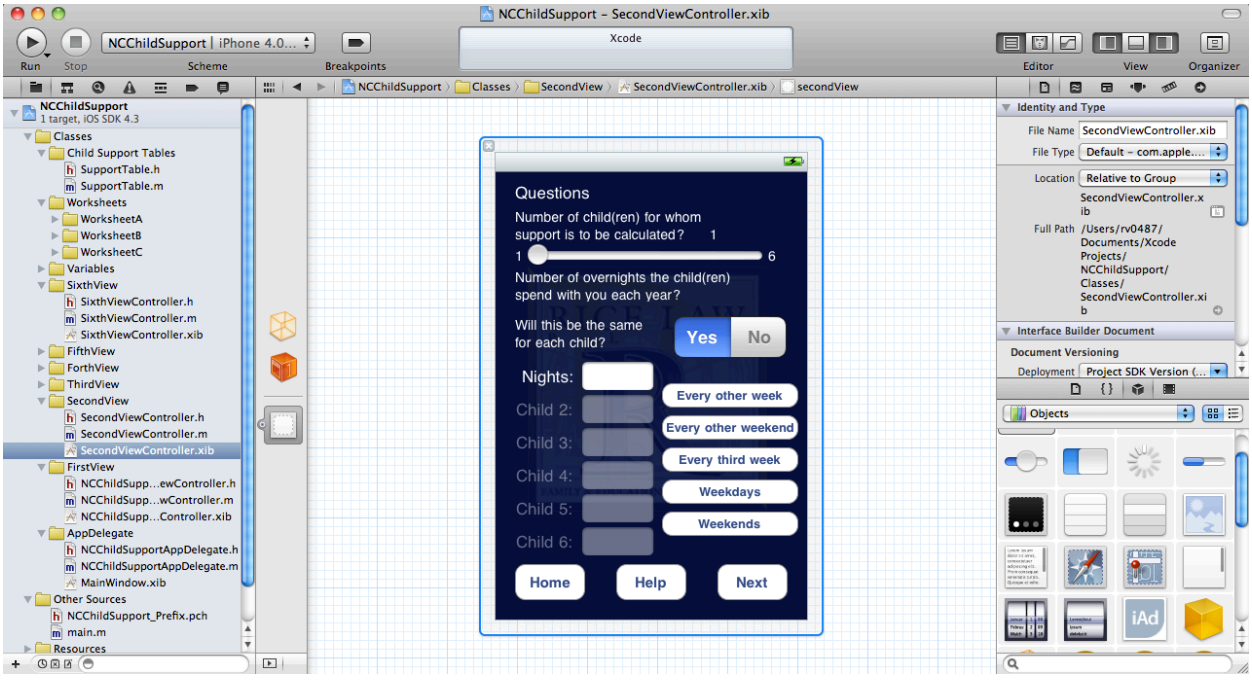


Figure 2 – Xcode Environment

2.2 Objective-C

Objective-C is an object-oriented programming language that's blends C programming with Smalltalk-style messaging. Objective-C (Obj-C) is the primary language for Apple's Mac OS X and iOS platforms. These two environments are based on the OpenStep standards although they are not compatible. Obj-C is a small step on top of C; thus making it possible to compile any C program with an Obj-C Compiler.

"Obj-C requires that the interface and implementations of a class be in separately declared code blocks. By convention, developers place the interface in a header file and the implementation in a code file. The header files, normally suffixed .h, are similar to C header files while the implementation (method) files, normally suffixed .m, can be very similar to C code files." [08]

2.3 Facebook Connect API

Facebook Connect (FBC) is a set of API's provided by Facebook that enables third party applications to interface with the Facebook platform [06]. This API allows developer to give their users access to their Facebook accounts in order to share and read information with their current application. The creation of this API brought social media directly to the web site and mobile application industry.

2.4 Twitter API

The Twitter API consists of three parts: two REST APIs and a Streaming API. The REST APIs were originally part of Summize, Inc. [07]. Summize provided search capability for

Twitter data, which was later acquired and rebranded by Twitter. Twitter's main goal is to unify all their current APIS into one codebase but that is currently a daunting task. The Streaming API is different from the two REST APIs because Streaming supports long-lived connections on a different architecture.

"The Twitter REST API methods allow developers to access core Twitter data. This includes update timelines, status data, and user information. The Search API methods give developers methods to interact with Twitter Search and trends data. The concern for developers given this separation is the effects on rate limiting and output format. The Streaming API provides near real-time high-volume access to Tweets in sampled and filtered form." [07]

2.5 Web Driven API

The back end environment, also known as the Web API is proprietary information for WS. However, WS did provide a high level overview of how their system works and what portion of the system the application would communicate with. WS has an ongoing deal with Eastern Skate Supply (ESS). ESS is the east coast's largest wholesale warehouse for any skate shop's needs. ESS constantly gets new products, better deals, and an ever-growing inventory. WS created a web site and CMS that integrates with ESS database. More importantly any orders made on WS are forwarded directly to ESS; leaving no inventory for WS to manage. The portion of WS that the application will be communicating to is the web service. This communication will be occurring when the device contacts the REST service portion of the web service with a GET request. This request will in turn return the data to the device as described earlier in this document. To obtain a high level understanding of the Web Driven API, refer to figure 3 below.

Figure 3 – Web Driven API

2.6 RESTful Web API

Representational state transfer also known as REST is a style of software architecture for distributing media through a medium such as the World Wide Web. REST is primarily used to broadcast its information through the Hypertext Transfer Protocol HTTP or the Secured Hypertext Transfer Protocol (HTTPS). A RESTful web service is a simple web service implemented using HTTP and the principles of REST. There are four key features you need to have in order to have a RESTful web service:

1. A base URI for the web service. http://www.warehouseskateboards.com/resources/
2. The ability to display Internet media through the web service. An example of this would be JavaScript Object Notation (JSON) or Extensible Markup Language (XML).
3. The use of REST requests through HTTP. These methods can be POST, GET, PUT and DELETE.
4. Lastly, all content needs to be driven by HTTP or HTTPS.

Typically, REST architectures consist of clients and servers. A client will initiate a REST request to a server via an HTTP request. This request can look like http://www.militarydiscounters.com/getrss2.aspx?lat=34.270000&lng=-77.920000&rad=25. Once the request is made to the server, the server processes the information give and returns the appropriate responses. These responses are built around a resource that can be easily manipulated into meaningful data. A good example of a resource would be a database.

This RESTful web service will be applied to WS via the base URI of www.warehouseskateboards.com/rest/enterbaseurihere. Once you complete the REST request, whether it is a GET, POST, PUT or DELETE, the server will be queried and return you standardize XML. The device for specific actions then reads the returned XML.

2.7 Summary

Developing an iOS application for the iPhone and/or iPad platform requires a solid understanding of Obj-C and object oriented programming skills. Additionally, developing an Obj-C application requires the use of the Xcode Software Development Kit. This environment is the one utility available to develop these applications. In order to obtain this environment you

must join the Apple Developer Program or obtain Xcode through the App Store on Mac computers. In order for the success of this application, FBC and the Twitter API needed to be researched and clearly understood for the successful implementation of social media to the application. All these features will successfully make the WS Application an eye-catching application for WS clientele and staff.

Chapter 3: Gathering User Requirements, Analysis, and Design

3.1 Software Development Life Cycle (SDLC)

SDLC is the process of creating, editing and modeling software components that people use to develop complex solutions. Under SDLC there are many types of software development methodologies. These methodologies help create a framework for the developers to plan and control during the creation of the software. SDLC typically consist of five phases: Planning, Analysis, Design, Implementation, and Maintenance. To manage this level of complexity many methodologies such as waterfall model, spiral model, rapid prototyping model, incremental model and unified process model were created. SDLC is a key part to the success of any software implementation.

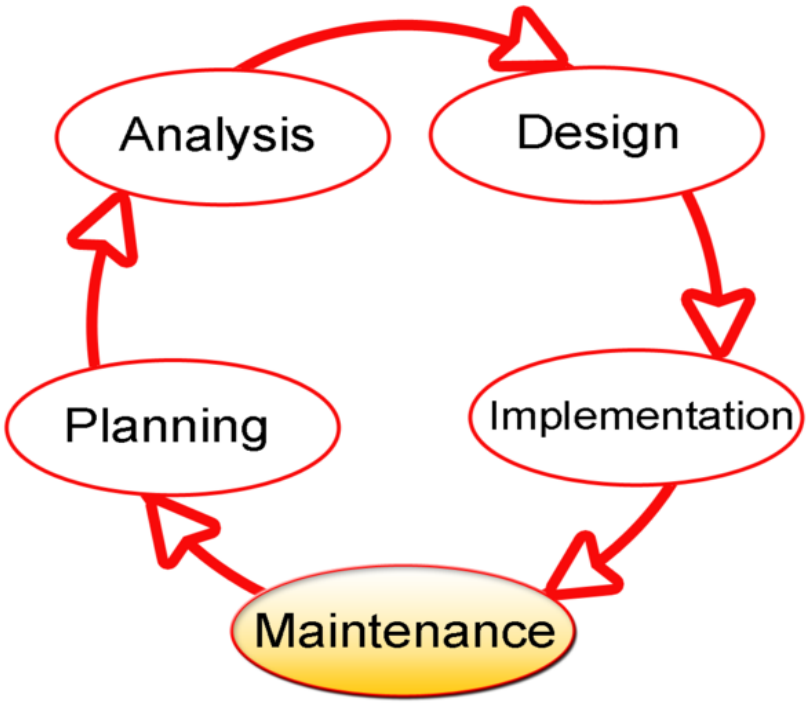


Figure 4 – Software Development Life Cycle

3.1.1 Spiral Model

The Spiral methodology was chosen for this project because it combines both prototyping and waterfall design modeling efforts. Barry Boehm defined the spiral model in 1986. Each iteration of this process was envisioned to take at least 6 months to complete. Each iteration starts with a design goal and ending with a client review. The purpose of the Spiral methodology approach is to involve the client during the development process of the application. Each time a task was completed the client would review and purpose additional features and modifications that are required for the project.

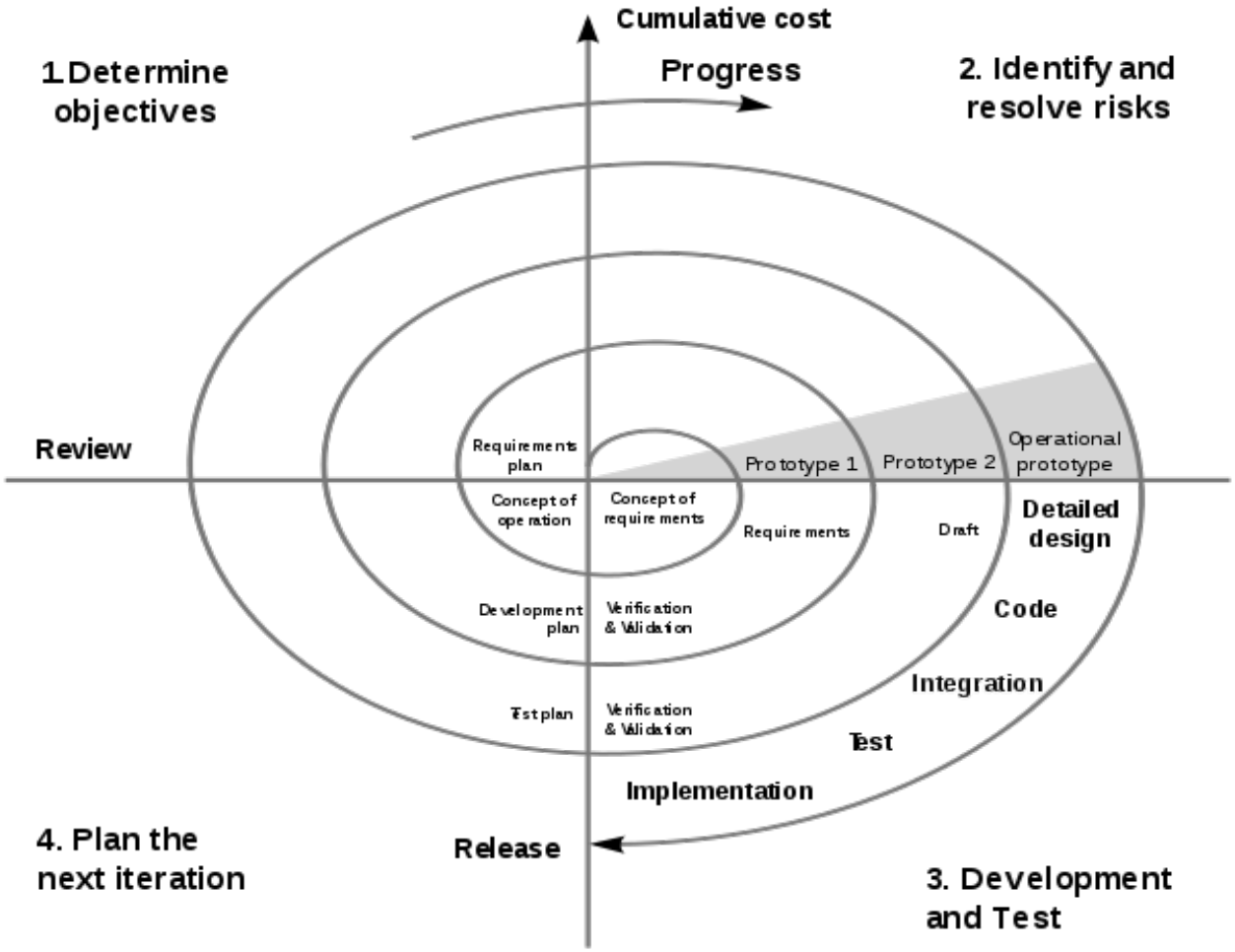


Figure 5 – The Spiral Model

3.1.2 Reasons for Selected Methodology

The spiral model methodology offers and a step-by-step process that allows the client to interact with the development team during the life of the project. This SDLC methodology also offers the developer and the project manager to revisit areas of development if needed. Many times projects are fully developed and created well before testing and client review is done. If an error arises the process has to start all over. Whereas, with the spiral model development a part of the software is developed, at one time, then reviewed and tested, followed by the next part and so on. This allows for sound programming decisions and good testing practices to occur.

3.2 Desired Functionality of the Software

After multiple meetings with WS it has become clear that the proposed project solution requires the following functionality:

1. Integration with WS Facebook and/or Twitter page
2. Integration with WS blog.
3. Displaying current WS coupons.
4. Displaying current WS sales.
5. Displaying WS new arrival products.

3.3 System Description

In order to implement this application, a tab bar controller (TBC) template must be used. A tab bar controller is useful when you want to organize an application along a functional line. The key concept for a TBC is the presence of a tab bar view on the bottom of the screen. This

bar is used to navigate between different application views. To successfully implement a tab bar interface is to create a TBC object. This object will manage the custom views associated for each page of content that is being displayed. Each of these custom views is then designated as the root view controller (RVC) for the associated tab. Once a tab is selected, the tab bar object will display the view assigned to it with its corresponding RVC.

Figure 6 – Sample Tab Bar Controller

Using the TBC as the backbone for the implementation of this application, we associate each tab of the application with the corresponding functionality requested by WS. In total there will be a maximum of five tabs displayed at any given time. If deemed necessary the 6th tab item and subsequent items will be store in the 5th tab item as a table of additional features. For clarification, see the image in Figure 7 below.

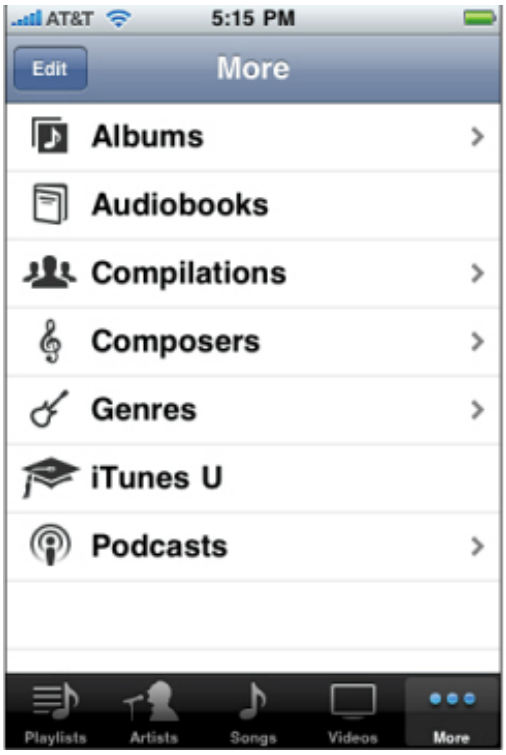


Figure 7 – Tab Bar Controller with 6 Items

3.4 Actor Diagrams

This Unified Modeling Language (UML) is a key documentation technique where Actor diagrams are used to visually represent user interaction between system components and the dependencies attached to them. The "Actors" can be anything that interacts with the system. These actors can be humans or other computers. Components are the specific parts of the system in which an actor can interact with. Actor diagrams are shown in Appendix A

3.5 Data Flow Diagram

A data flow diagram (DFD) is a graphical representation of the flow of data between systems. DFD's are typically used for visualization of the data process for the current structural design. Typically on a DFD, data items flow from some external data source to an internal data

source via an internal process. DFD's do not provide information about system components, system timing, or if a system is parallel or in sequence. Below is the current DFD for WS Application. The iOS device (iPhone) is requesting data from an Internet based XML feed that is generated from WS current Database and Web Server.

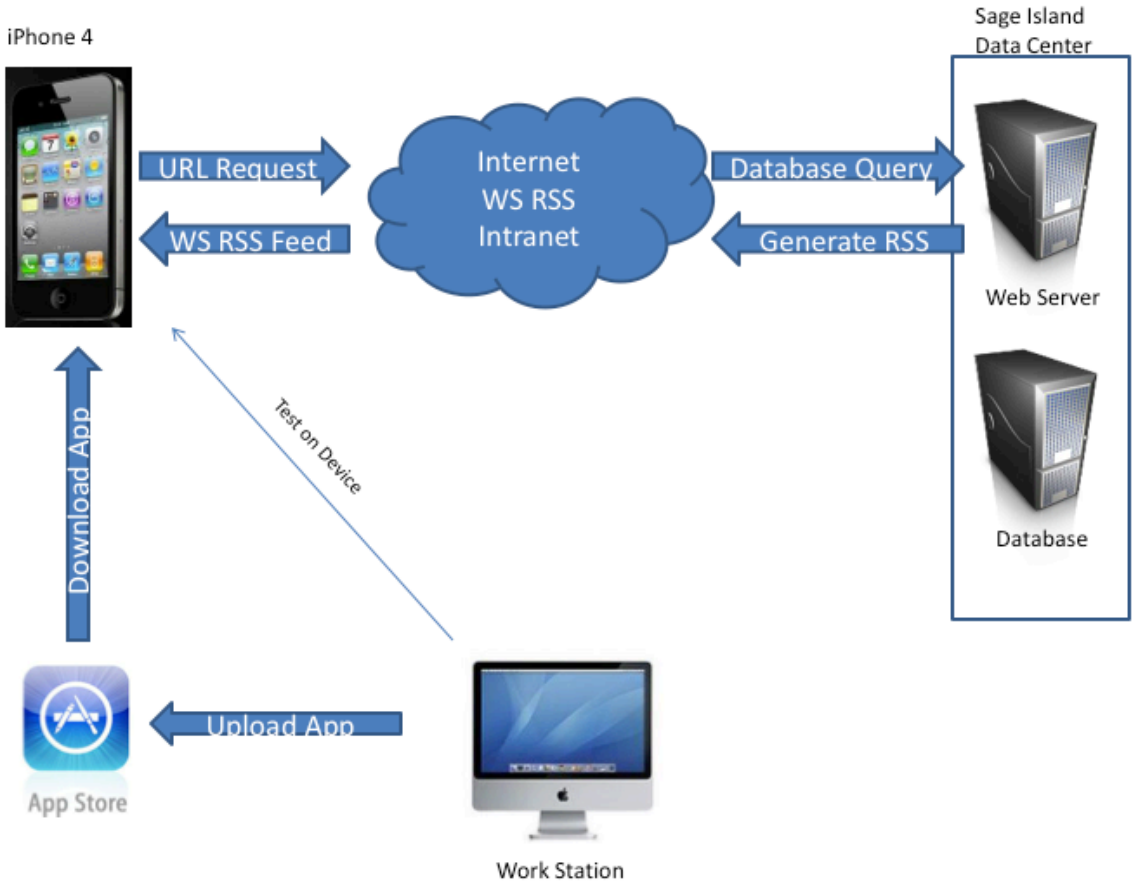


Figure 8 – System Data Flow

3.6 User Interface Design

Developing an iOS application is not similar to developing a piece of software for a typical computer. The mobile developer must work with many additional constraints that a typical software developer does not encounter. Some constraints for the iOS platform are: screen resolution, designing simplistic user interfaces, communicating with the end user about networking issues. Additionally, the developer must follow the rules of the Human Interface Guidelines (HIG). The HIG documentation tells the developer specifically how users must be informed about the processes that occurring during software activity. An example of such HIG rules would be displaying and Activity Indicator when loading a website on the device. All these constraints and restrictions are what make a successful iOS application. Lastly, there were additional user interface guidelines that needed to be followed from the storyboard developed by WS.

3.6.1 Warehouse Sales & Just In View

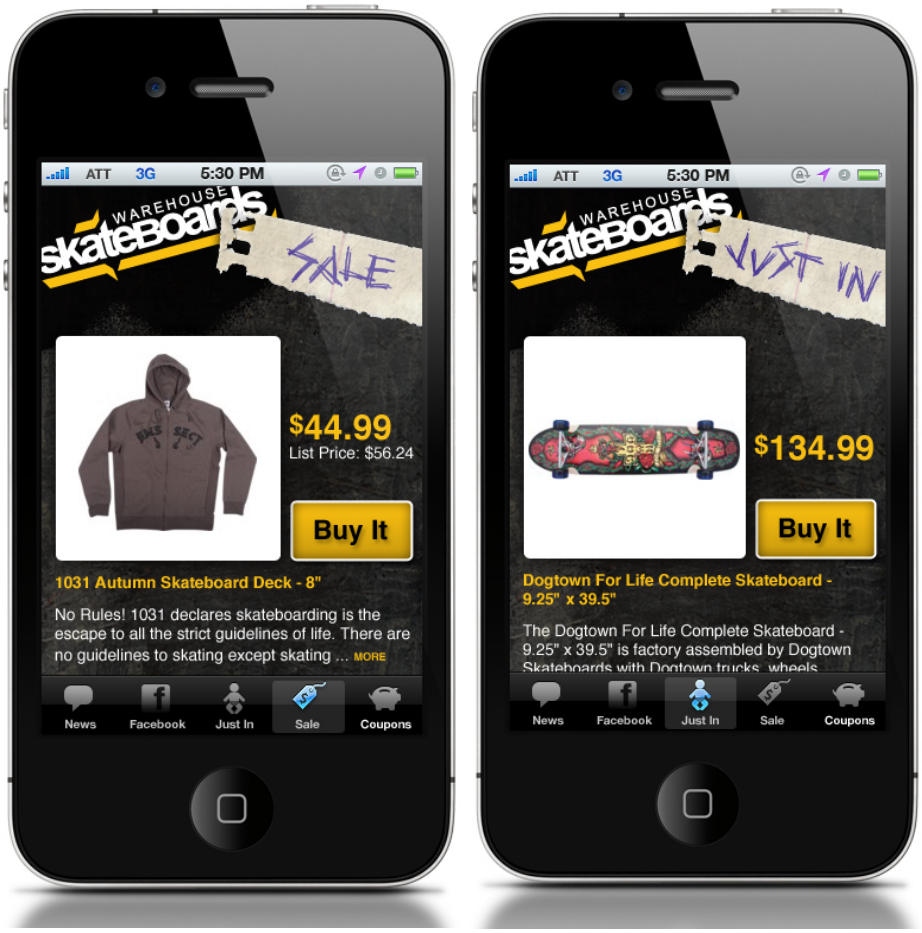


Figure 9 – Sales and Just In Views

The Sales and Just In views have identical UI functionalities and layouts. The only difference between the two views is in the upper right hand corner where prices are displayed. Each view has horizontal scrolling, which will enable users to easily transition from one item to the next. This design feature is similar to viewing selected images in the photos section of the iPhone. The description for the product is vertically scrollable as well. This will give the user s the ability to read a lengthy description of the product when needed. If the user selects the item's

image it will transition to a larger display image of the current product. When the buy now button is pressed, a transition to a web view for the product will occur within the app.

3.6.2 Warehouse News View

Figure 10 – News View

The UI functionality of the news view has similar properties to the Sales and Just In views. However, the gesture functionality of this screen is different. Instead of enabling horizontal scrolling to progress through the feeds, the user can vertically scroll the page. This decision was based off of the text messaging scroll functionality found in the iPhone. Each entry "news item" displayed in this page could have and image attached to it at the end. Just like the sales view if the image is selected, the image will enlarge and be displayed nicely for the user.

On each entries title, if the data received contains a link to a main article, will become a button that will launch an in app web view of the main article.

3.6.3 Warehouse Facebook View

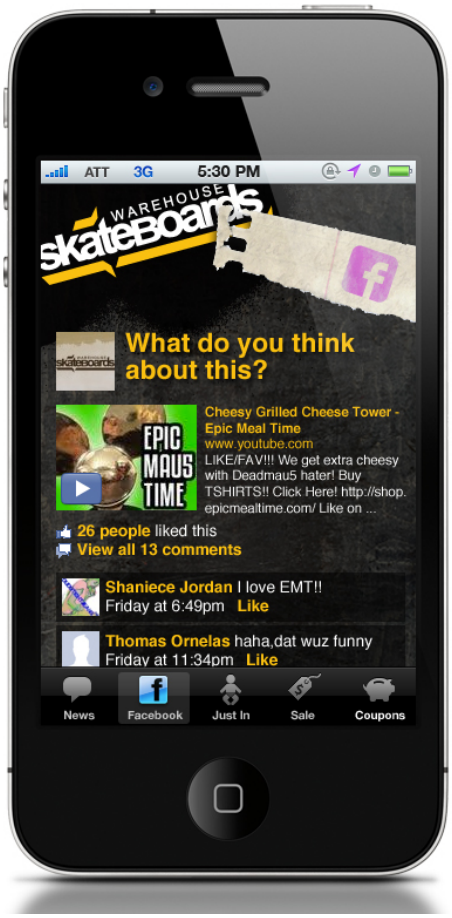


Figure 11 – Facebook View

The Facebook view is the most complicated of the five views due to the demanding interface users must interact with. The standard vertical scrolling gesture is added to be able to read multiple posts on the WS Facebook wall. Users have the ability to view comments on a single post and track the number of people who favored the post. If a video or image is added the user is given the ability to enlarge or watch the current media content. At this time users are not allowed to post to the WS Facebook wall as this would require every user to authenticate their

Facebook account with the application. This feature is desired but will come at a later date due to the complexity of the Facebook API. If the application cannot pull from data from Facebook, a default to a wipe out graphic "The site is undergoing repairs come back later" is shown.

3.6.4 Warehouse Coupon View

Figure 12 – Coupon View

The coupon view is simplistic. When a user selects this view, one if not many coupons will load. The user will have the ability to scroll horizontally between each coupon as desired. The coupons are currently UIImages. There is no button overlay, email capability, or links associated with a coupon. These deals are intended to be applied to large orders on the flagship

site. If no coupons are available, a base coupon or please check back soon image will be used. This will give users an active feel in the UI even when no data is there.

3.7 Description of Software Development Activities

3.7.1 Determine Objectives Phase

The purpose of the determine objectives phase was to establish a well-defined product that will benefit both client and customer needs. This phase includes recognizing key requirements for WS project, defining the scope for WS project and developing the estimated software development timeline. During this phase detailed discussions between the developer and client need to take place in order to discuss what ideas can be successfully developed in a reasonable amount of time.

3.7.2 Identify and Resolve Risks Phase

For each of the identified project risks, a detailed analysis is carried out and steps must be taken to reduce the risk. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed. A project risk simply means something that can go wrong.

3.7.3 Development and Test Phase

A development model for the system was chosen after risk assessment and objectives were set. The model included the initial prototype that contained the basic features required. After a few iterations of development and testing, the desired software product was produced.

3.7.4 Plan the Next Iteration Phase

The project iterations were reviewed and decisions were made by the client on whether to continue with a further loop of the spiral. When it was decided to continue, plans were drawn up for the next phase of development.

Chapter 4: Development and Implementation Considerations

4.1 Xcode Source Code Management (SCM) and Testing Environment

The Xcode programming environment provides the ability to integrate with Subversion or GIT. Both of these are used for source code management during the development life of an application. Additionally, I unit test any methods that were not UI driven events with Xcode's built in unit testing framework known as SenTestingKit. This framework helped to strengthen the code as it was written.

4.1.1 Source Code Management

The Xcode IDE (integrated development environment) provides several ways to mange versions of your project. You can take snapshots to save the current state or workspace for restoration later. You can integrate to SCM repositories such as GIT or Subversion for tracking of individual changes of a file/s. Lastly, you can archive packages of your product or distribution when needed. For this project, Xcode's integration to Subversion was used.

Why use source control? When working on a software project, it can be useful to use SCM to keep track of changes in the code base. An SCM system, will save multiple versions of each file at a specified location, storing metadata about each version of each file in a location known as an SCM repository. [12] An SCM system can help you reconstruct past versions of the software during its development life. During your daily work period, when you make any major change, the intent is you will commit the changes made back to the repository for versioning. If during your development bugs arise that were not there before you can compare versions of the software and see the differences. If the worst-case scenario occurs, you can always revert to an earlier version and start from where you left off in the previous version. These actions apply to

both GIT and Subversion for SCM systems. The only justification for using Subversion during the development of this project was because it was already instantiated in the environment used by the developer (author).

The Xcode IDE provides a user friendly UI for interacting with Subversion and GIT. This UI allows developers to manage multiple repositories, it allows them to commit, push, and merge versions with the push of a button. This UI replaces the need for the traditional terminal commands used in the past. The UI keeps track of each files state. This means that if a file is modified, updated, added, deleted, ignored, replaced, or marked for not under source control it will be graphically represented with a specific symbol. The UI also allows the developer to navigate the file structure of a repository visibly. That way if you need to check a specific file instead of the entire folder you can. This Xcode utility was developed specifically for SCM, it has made the process easier and more intuitive than it has been in the past. Please see Figure 13 for a sample of the interface.

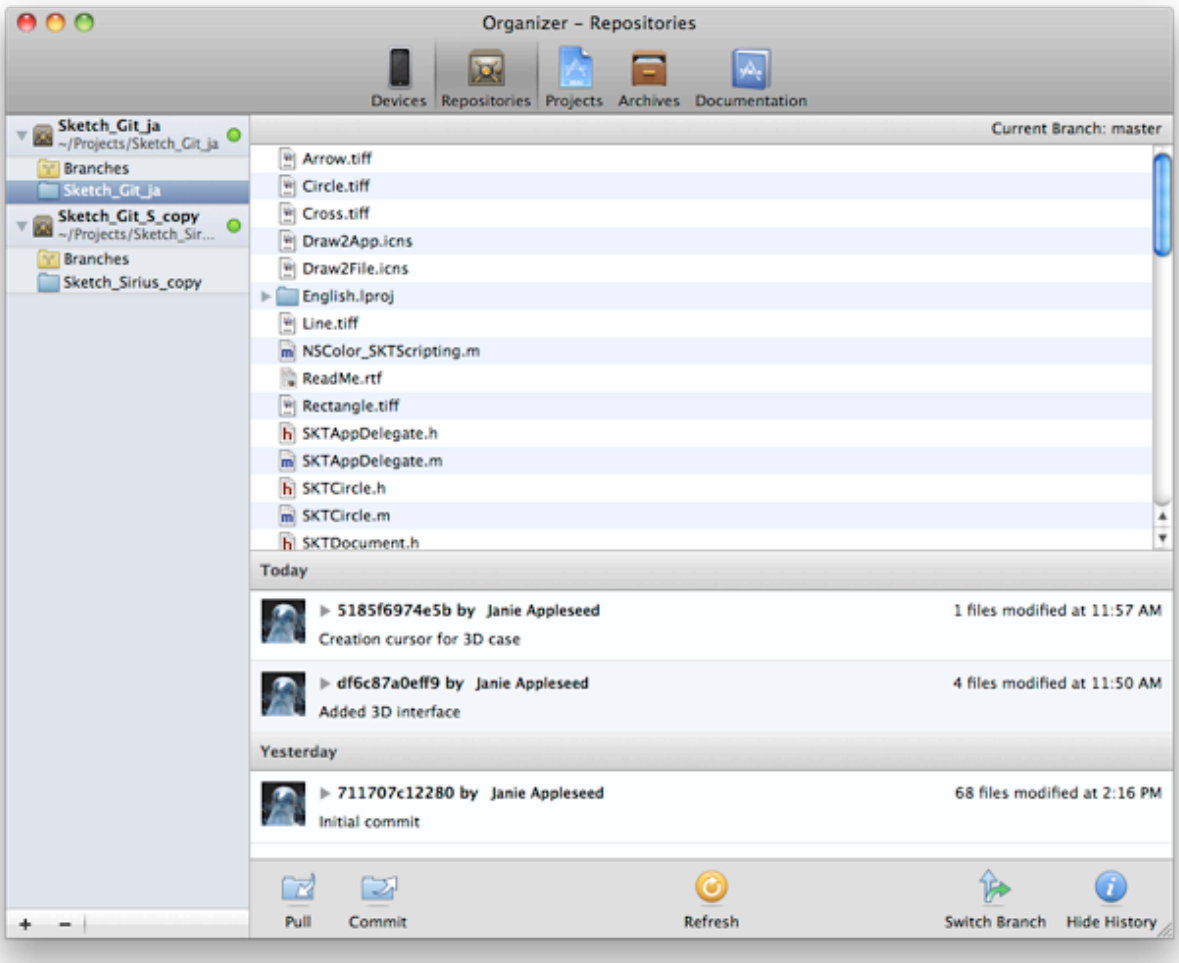


Figure 13 – Xcode Source Code Management Interface

4.1.2 Unit Tests

What is unit testing you might ask? Unit testing lets you specify behavior that your code must exhibit to ensure that its functionality remains unchanged as you modify it. [13] A “unit of code” also known as a method in a class is the primary target for unit testing. When testing a method a can write multiple test cases. A test case is a singular exercise of said method. It ensures that the method behaves in a specific way; if the results of the test are different than what is expected, the test case will fail.

Unit test are essential for test-driven development, this is a style of writing code where you write test cases before writing code. This development approach enables the developer to create a more vested application. When not using test driven development, unit tests can help reduce the introduction of bugs in your code. You can incorporate unit tests in a working application to ensure future changes do not modify the app's behavior. As you fix bugs, you can add test cases to verify the fix.

Xcode's SenTestingKit framework offers two types of unit tests: Logic tests and application tests. Logic tests are used to check the correct functionality of a unit of code and not the application. With logic tests you can perform stress testing and make specific tests for each unit. These tests will help produce a robust code base that works correctly when used in ways that were not anticipated. Application tests are used to check units of code in context to the application. You can use application tests to ensure that the connections of your user-interface controls remain in place. These tests are typically used to perform hardware testing, such as obtaining a GPS position or coming in and out of activity states.

4.2 Discussion of Implementation Decisions

4.2.1 Data Access

There are many different ways to access data from an external database. Some developers choose to communicate directly with the database using embedded SQL queries. Others prefer REST requests. Some even choose to have a local copy of the database reside on the phone. All three of these paths are capable of obtaining and displaying current data from WS.

Embedded SQL queries in Obj-C can be a difficult path to take. Initially when Obj-C was being molded for the iOS platform, Apple decided to not allow direct access to databases.

Their view of clean code and simplistic methods clashed with the complex structure of database syntax. Over time and with increased demand, Apple decided to allow developers to establish connections directly to databases. However, this is not a trivial task. There is little to no documentation on how to access a database. One developer discovered that you could create an internal instance of the database and clone the external database on request. After reading more about embedded SQL through Obj-C, it was decided to not venture any further into this topic.

Another solution for obtaining data would be to pre-populate the application with the relevant information. This solution is by far the easiest to implement. You will not have to access any external data source. On the other hand, there are many drawbacks to this path. Any changes that are required to be made, such as updating the information, changing prices, swapping out coupons or removing items will require an update to the application. This process takes a minimum of 2 to 4 days with apples approval process. That being said, this was not the appropriate path to take.

As explained earlier in the paper the solution of choice was to use a RESTful service that would return xml-formatted information to the device. This approach is a 4-step process. First, the device must make a GET request to the web server over HTTP. The web server will receive the request and kick a process off. This process will communicate with the database and generate data. Once the web service gets the data, it is returned back to the device that initially established the request. This process is one of the most widely used processes to communicate with a database over the web for iOS and or mobile development.

4.2.2 Data Processing

There are typically two ways to parse and handle XML, the Tree-Based parsing and the Event-driven parsing. Both approaches have their strengths and weaknesses. Tree-based parsing, maps an XML document into an internal tree structure that conforms to the logical structure described by a schema. Once mapped, it allows the developer to manipulate and traverse each node in that tree. An example of a tree-based parse would be the DOM parser that is widely used in Java. Event-drive parsing is used when data is constantly being feed in. In this approach, the parser reports parsing events to an application as it encounters them. This reporting is typically accomplished through callbacks implemented by the application to handle the type of event. Event driven parsing consumes less memory and it is ideal for situations where performance is a goal and modifications of parsed XML is not. However, Apple's NSXMLParser uses and NSXMLParser object that sends messages to its delegate. It sends different messages for each type of event. A similar example the Event-driven parsing would be the Java SAX parser. This parsing style is sometimes referred to as stream parsing.

Since data will be coming in as a stream to the device, the logical choice would be the Event-drive parser known as NSXMLParser. To better understand how callbacks occur, please review the following block of XML code.

```
<?xml version= "1.0" encoding="UTF8">
<article author="John Doe">
<para>This is a very short article.</para>
</article>
```

The parser would report the following events to its delegate.

1. Started parsing document.
2. Found start tag for element article.
3. Found attribute author of element article, value “John Doe”.
4. Found start tag for element para.
5. Found characters “This is a very short article”.
6. Found end tag for element para.
7. Found end tag for element article.
8. Ended parsing document.

Since the data is a stream, we will need to only worry about one line at a time as the data comes into the device. This will eliminate the need to “download” the whole document and parse it after it is obtained. In addition to reporting parsing events, the NSXMLParser also verifies that the XML is well formed. If not, the parsing event is stopped and the delegate is informed.

4.2.3 Data Storage

There are many different ways data can be stored in Obj-C. They can be stored in Arrays, Dictionaries, Core Data, SQLite DB’s, Objects, etc. There is truly no perfect path when choosing how to store data. In order to narrow my choices I reviewed Core Data, SQLite, and an Array list of Objects. All three of these paths developed reasonable results.

Core data is one of Apples most powerful data management framework. Core Data has a mature code base and is rigorously tested via unit tests by Apple. Due to the popularity of core data over other data management frameworks, Apple has spent ample amounts of time optimizing it. Simply put, Core data is scalable, robust, efficient, and intended for large amounts of data. Core Data provides an infrastructure for change management and for saving objects to and retrieving them from storage. SQLite is one of many persistent store types. Core data however, is not a database on its own. When using SQLite, you can either use it in conjunction

with core data or standalone. SQLite has similar properties to core data in regards to scalability, robust and efficiency. However, you as the developer need to verbose in SQL and C syntax computing languages. Additionally, SQLite is also intended for a large volume of data that changes constantly. Since Core data and SQLite are intended for actively large amounts of changing data it is best to stick with a simple array of objects, there is no need to add such a heavy framework on top of this application.

4.2.4 Network Connectivity & Network Access

An asynchronous connection was used when establishing a connection to the data source. The reasoning is to give users the ability to navigate to other pages while the data is being loaded. Asynchronous connections force the connection into the background thread of the application. Synchronous connections establish the connection on the main thread. Synchronous connections also prevent users from navigating to another page while a connection transaction is occurring.

Since we established a connection to the World Wide Web, Apple requires every developer to instantiate some form of a Reachability test. Reachability is a term that was coined by iOS developers. Reachability is the monitoring of the network state of an iPhone, iPod, or iPad with the System Configuration framework. More specifically it demonstrates how to know when traffic can be routed through a Wireless Wide Area Network such as EDGE or 3G. If connectivity is spotty or poor any requests that are made by the device will be held in a queue for a short period of time before being dropped by the devices main monitoring system.

In order to complete these tasks, the use of an external framework known as “All-seeing “I”nteractive (ASI) will be used. “ASI is an easy to use wrapper around the CFNetwork API that

makes some of the more tedious aspects of communication with web servers easier. ASI is a suitable external framework for basic HTTP requests and interacting with RESTful-based services such as GET, POST, PUT & DELETE" [14]. For more information about the ASI framework please reference Appendix D for an in-depth overview.

4.2.5 UI Implementation Decisions

WS specified the overall functionality of the application. As the primary developer it was up to me to make their ideas flow nicely as an application. Many ideas were tossed around in how the application could flow. It could have been created as a custom UIView with a custom homepage or I could have created the application as a tiered list of data using UITableViews with a custom UIView attached to the end cell. The custom UIView approach would have been costly, overly complicated, with high graphical content and apple like features. The UITableView approach would require little to no artwork, it would consist of mostly of tables and the over all application would lack the "WOW!" factor that WS desired. These reasons are what lead me to decide to implement a UITabBarController. It provides the organization of a tab bar while still giving me the ability to incorporate customizable views. Navigation between views is already handled by the controller and more importantly; it retains the apple and graphic appeal that WS desired.

When developing a mobile application the number of clicks to complete a path is key. This application can boast that it requires a minimal amount of clicks to view news/blog feeds, WS Facebook wall and cool products. More importantly the application only has a depth of one view from any selected tab. Navigation between each screen is seamless due to the implementation of a UITabBarController. If a user ever finds the need to return to the main level

of a page, a navigation bar will appear allowing the user to return to the previous screen. Note: This will only occur when you navigate to the second level of a screen (Selecting a web link or pressing the "Buy Now" button). It will not occur when you switch between the tabbed views.

4.3 Design of an Application Evaluation Plan

In order to properly assess the applications potential, target demographic and user interface a structured evaluation plan is required. Evaluation plans can come in many ways. For this iteration of the application, a guided questionnaire will be perfect for assessing the state of the application and its features. WS has specific target markets that need to be accessed as well as application functionality.

After WS reviews the application they will be asked to complete a twenty-question survey about it. The survey listed in Appendix C will help assess many different aspects of the application. Perspective users is a key area to assess, this is because our main goal is to develop an application where current users are happy enough with the application to help promote it to other friends. This action is known as making something go viral in the app store. We are also interested in seeing if we have reached out to our target market as well. Another key area that WS is focused on is the graphical appeal of the application. This application is all about how nicely we can display data and give the users an experience that will leave an impression. If issues arise during the evaluation period, users can notify us via the evaluation form or email.

For the business side of the evaluation, we focused on satisfying WS needs for the application, if there are any more features that are needed and if the iOS platform should be the sole platform to target. A major question on the form, question 19 asks WS and Sage Island if they feel that the applications core functionality and general design can be applied to other areas.

If so, we ask our evaluators to list their thoughts. This information will be utilized to explore new markets of interest.

The 20-question evaluation form will be used to target several areas of information. We will be targeting responses for Perspective Users, Application Design, Satisfying Business Requirements, Reusability, & Content Management. Each area with the exception of Reusability and Content Management has at least 5 questions revolving around the topic area. Users will be able to select their answers based off a scaled response from 1-5. These responses can also be answered in a written response for more clarity. Each question targets a specific area within its respective topic. For example, question 7-13 all target different areas of application design. This form is intended to be used once the application is launch and ready for inclusion in Apple's iTunes App Store.

Chapter 5: Summary and Conclusions

WS Mobile was a fantastic opportunity for me to work closely with Sage Island and the WS Staff. As a development group, we managed to create one of the coolest apps out on the market. Personally, I feel that the skateboarding consumer society will be delighted with this application and utilize it to stay current with WS incoming products, deals, and discounts. We hope that this application goes "Viral for WS" and can potentially increase revenue for WS. Upon approval of the application on the App store we hope to closely monitor its success and potentially apply this concept to other sites.

While both WS and I are confident that we accomplished our goals, the product is far from finished. We have many more ideas and plan to grow the application with hopes of giving the skateboarding community a more interactive application. Ideas for additional features are abundant. For example, we can add more social media links into the application. Give the user a block breaker game based off of product logos. Integrate with the current WS "active" catalogue. These are just a few of the endless opportunities to expand this project. This current iteration of WS Mobile will provide a solid platform in which the application can expand on.

Earlier in the paper I had mentioned one of my major goals for this project was to be able to make the software reproducible. The way the application was designed enables me to swap graphics, log in information & data feeds quickly. With that said, I am more than capable of turning this application around for another client within a week. I know this because I designed the application to be readable, adaptable and easy to access any constraints from one file. I wrote the parser to take an array of tokens for parsing. This makes it easier to update a singular spot in the code instead of bouncing around the entire parser. These features and many more are what

make the application capable of being reproduced for other clientele. In order to successfully complete this reproduction, minor tweaks for parsing the data and small adjustments for UI's are required. More specifically, the developer needs to update the xmlTags.plist file with the expected tags for the new restful service; additionally, the developer could add views into the tab bar controller for more application features. By using these design technique, I am confident that I could reproduce a similar application quite easily.

I was recently asked, "What the easiest and hardest part of the project was?" this is a tough question to answer. Since I have been developing iOS application over the past few years I have been able to handle almost any request a client gives me. I would say the easiest part of the application was developing the user interface to be interactive for the user. My favorite part of the WS design was to utilize a swipe gesture to scroll through products on an endless screen. The hardest part of the application was working with other people. Being reliant on the client to provide content for the application is a risky process. This can be late, inaccurate, or not what you requested. These can turn into serious problems when developing the software. My primary problem during the life of the application was developing a product that interacts with a live data stream that did not exist until the end of development. This forced me to create mock data work around said problem until the content was available. That being said, it made me a stronger developer and taught me to not always expect resources to be readily available during development.

If I were to develop this application again I would do a few things differently. I recently was taught the Scrum Agile development style. I felt that the use of sprints and product backlogs in Agile development would have been a great way to manage the development process. I would also prefer to manage my time better by utilizing project management tools such as Jira and

Confluence. These would help me produce a better paper trail and force my thoughts to be more organized and well constructed. Outside of that, I thoroughly enjoyed developing such a cool application for WS and cherished this opportunity.

References

[01]: Nutting, Jack, N., David, Mark, D., & LaMarche, Jeff, L. (2011). *Beginning iphone4 development: Exploring the ios sdk*. Apress.

[02]: Fowler, Martin, F. (2004). *Uml distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional.

[03]: John, Robert B., J., & Stephen D., S. (2004). *Object-oriented analysis and design: With the unified process*. Course Technology Ptr.

[04]: Summerville, Ian, S. (2007). *Software engineering*. 2007.

[05]: Mark, M., Dave, D., & LamMarche, Jeff, L. (2009). *More iphone 3 development: Tackling iphone sdk 3*. Springer.

[06]: *Facebook developers*. (2012, April 27). Retrieved from http://developers.facebook.com/docs/guides/mobile/

[07]: *Twitter developers*. (2012, April 27). Retrieved from http://dev.twitter.com/pages/api_overview

[08]: *Objective-c*. (2012, April 27). Retrieved from http://en.wikipedia.org/wiki/Objective-C

[09]: *Warehouse skateboards*. (2012, April 27). Retrieved from http://www.warehouseskateboards.com

[10]: *Mobile roadie*. (2012, April 27). Retrieved from http://www.mobileroadie.com

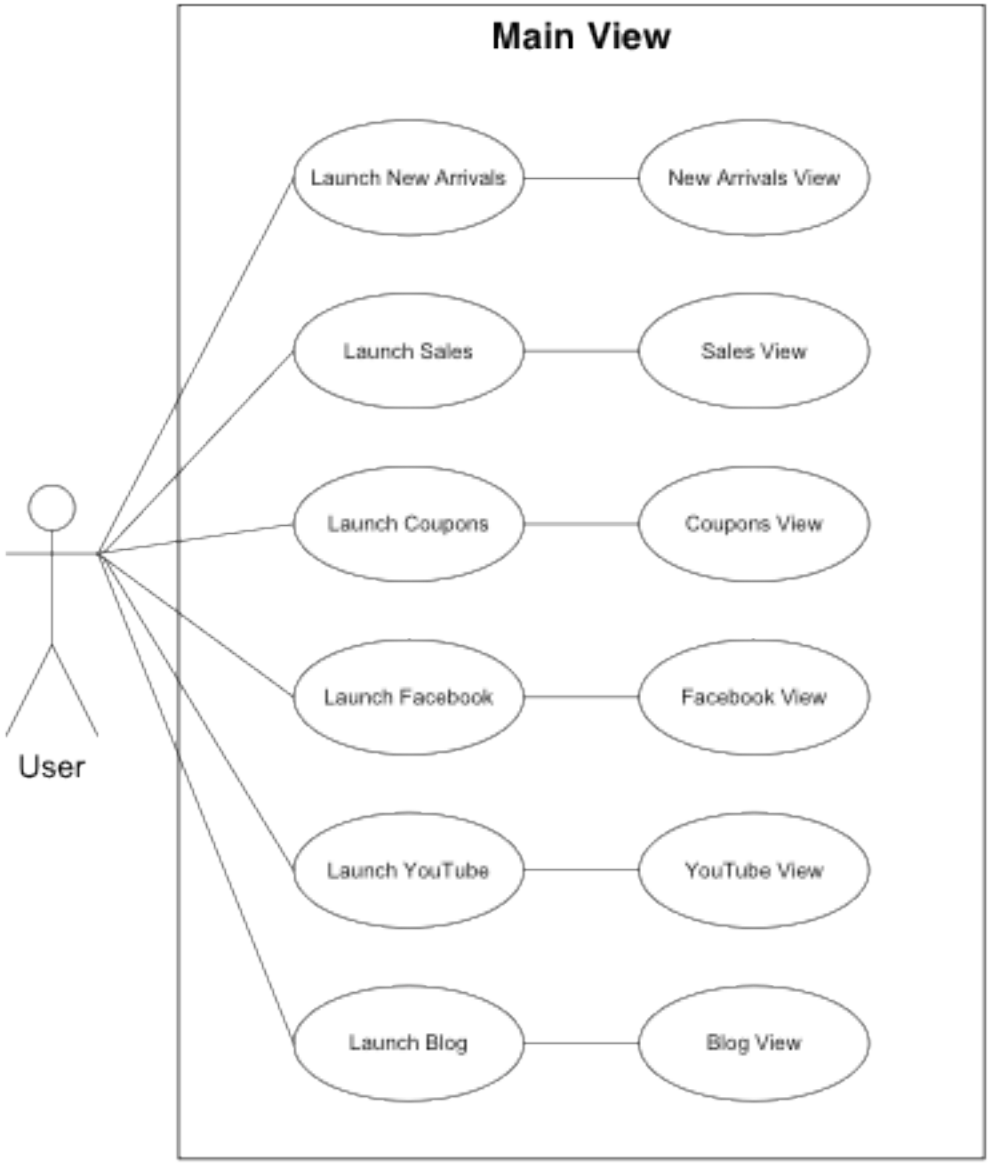
[11]: *Xmlparsing*. (2012, April 27). Retrieved from http://developer.apple.com/library/mac/

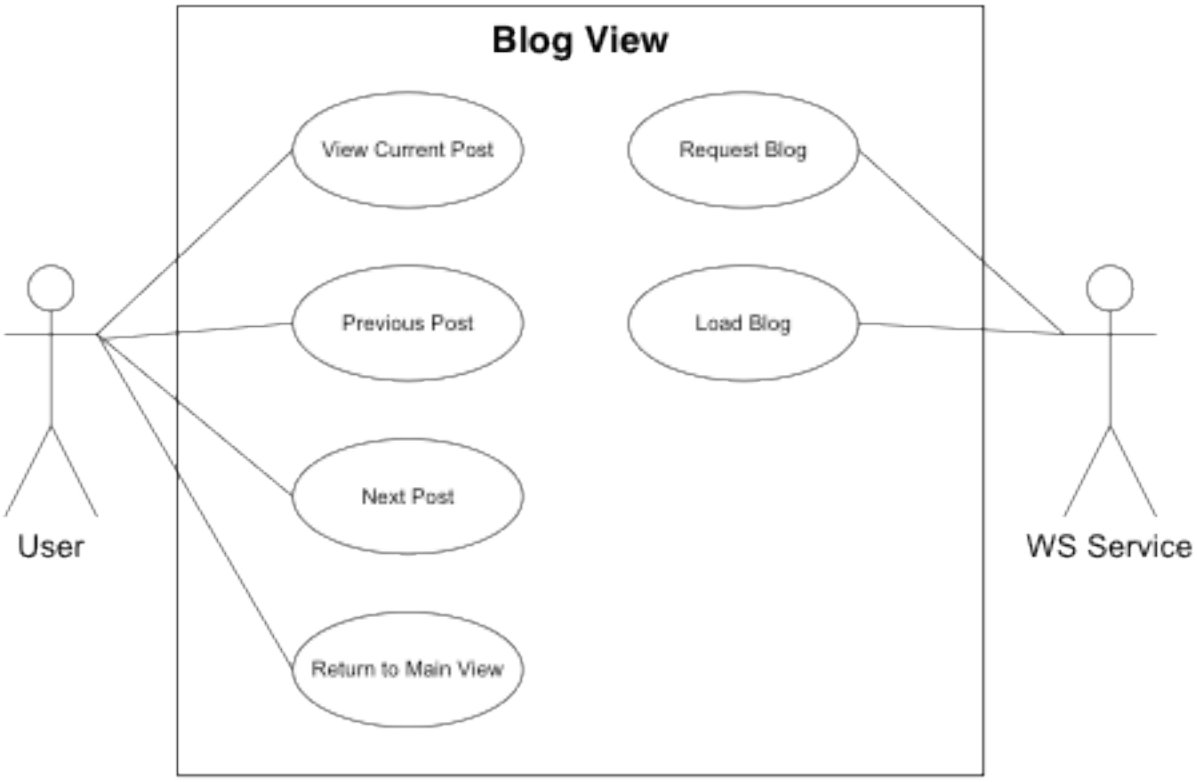
[12]: *Source code management*. (2012, April 27). Retrieved from http://developer.apple.com/library/mac/

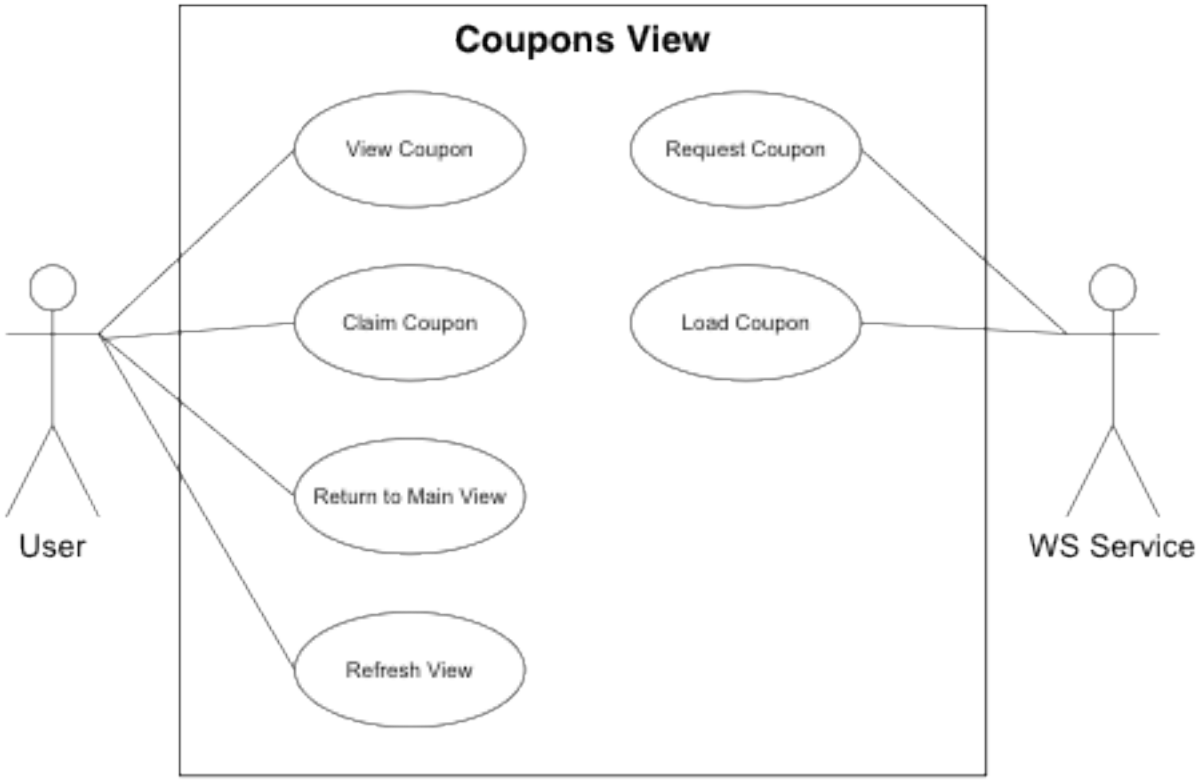
[13]: *Unit testing*. (2012, April 27). Retrieved from https://developer.apple.com/library/mac/

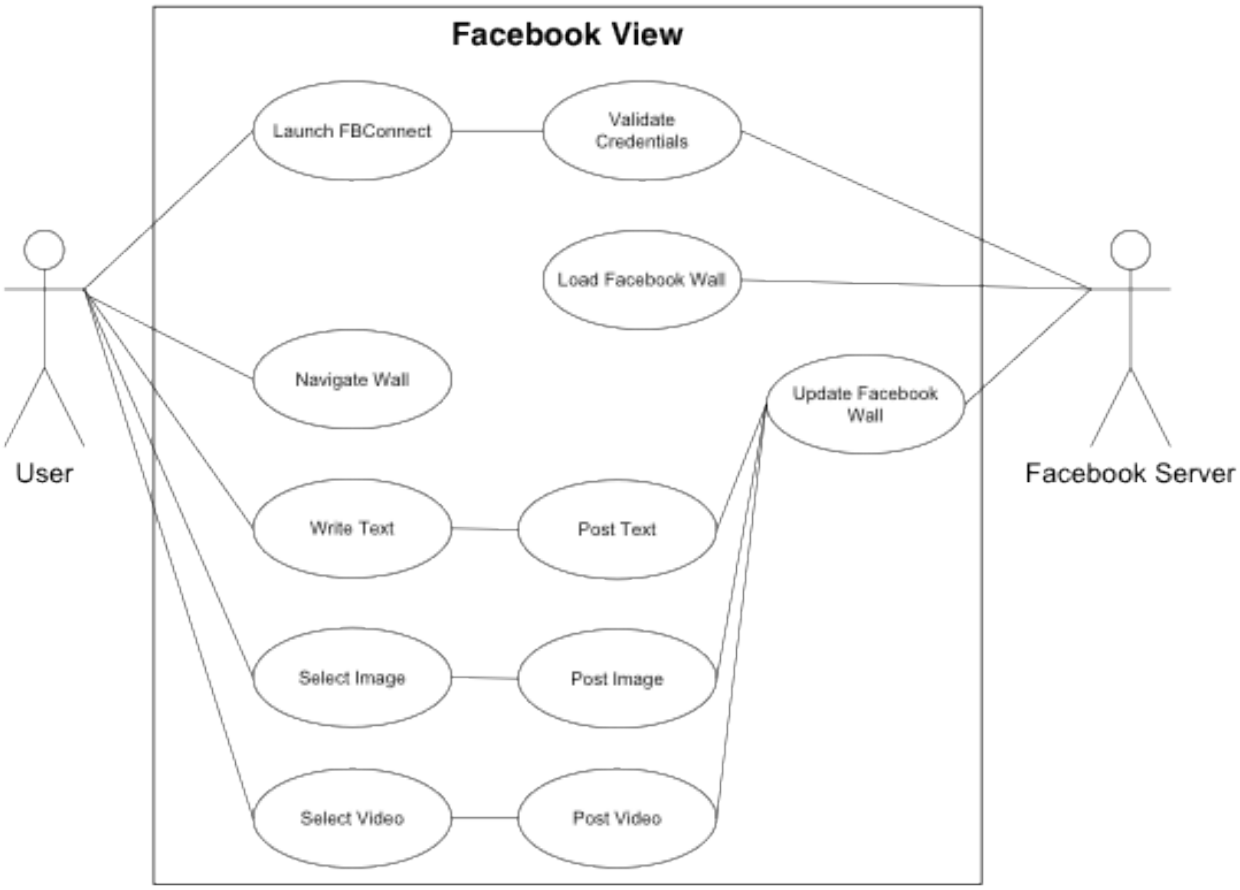
[14]: *All-seeing I*. (2012, April 27). Retrieved from http://allseeing-i.com/ASIHTTPRequest/

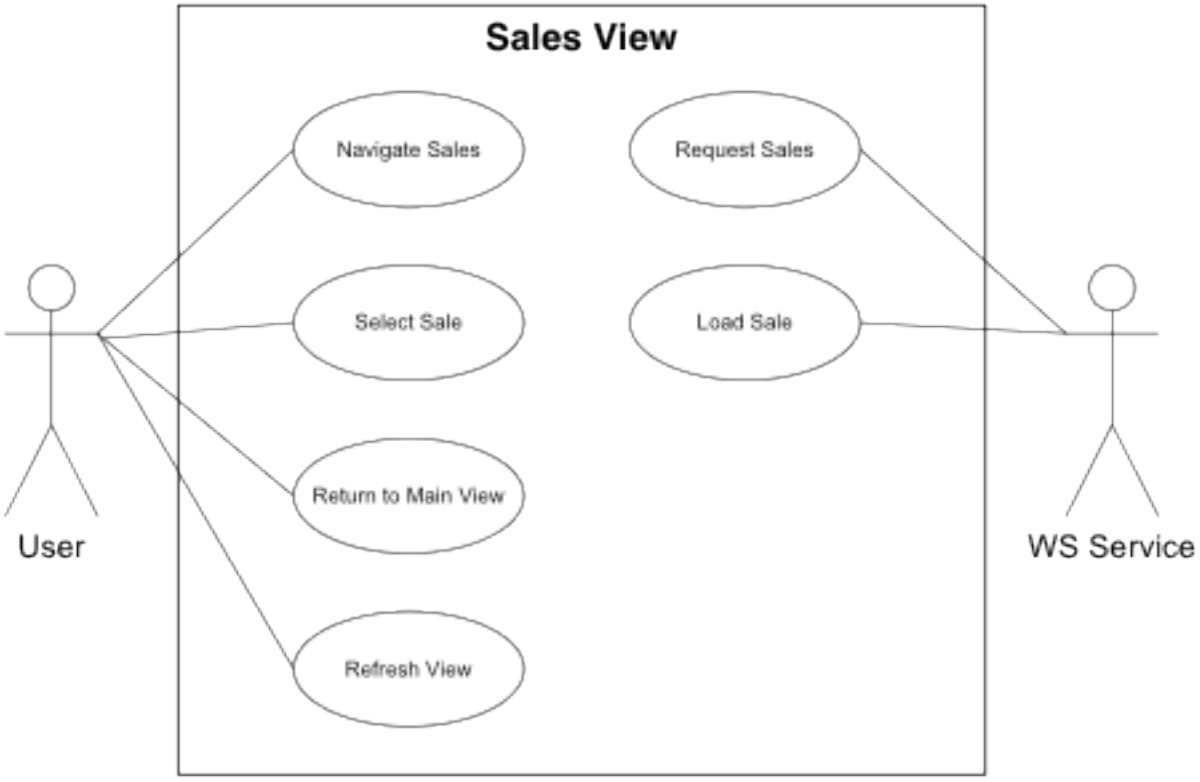
Appendix A

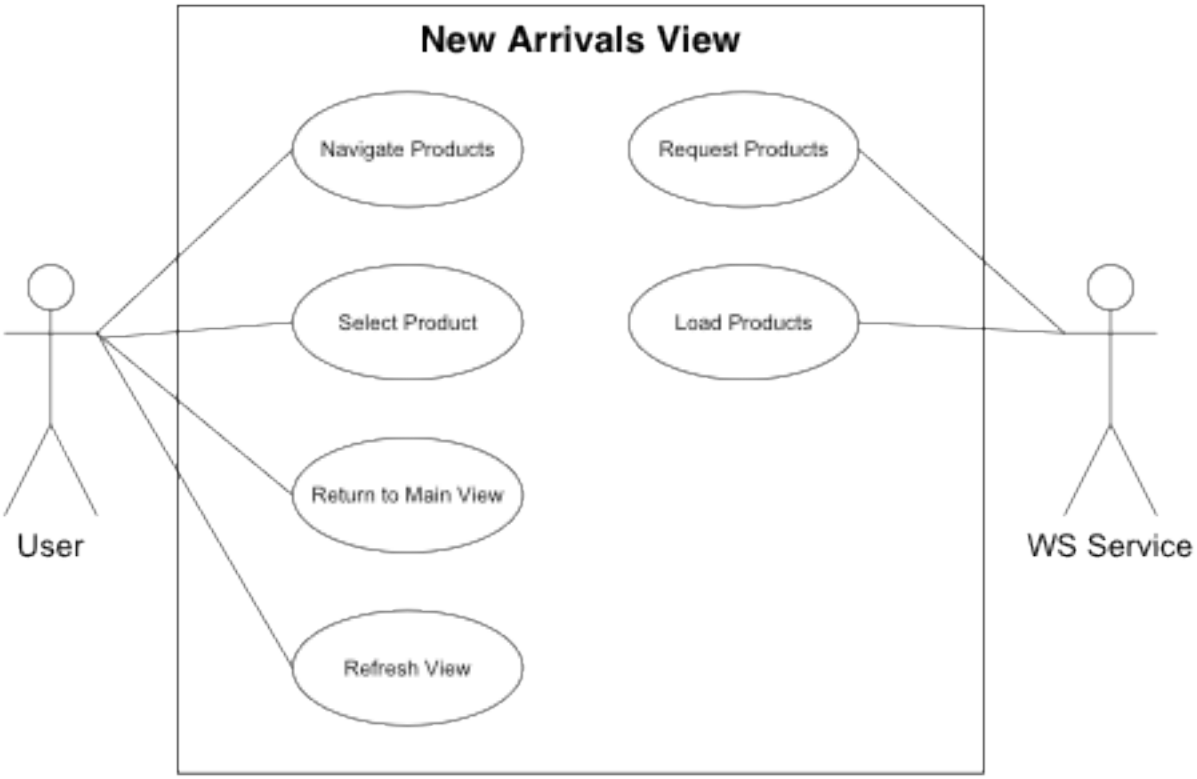

Main View
Launch New Arrivals
New Arrivals View
Launch Sales
Sales View
Launch Coupons
Coupons View
Launch Facebook
Facebook View
Launch YouTube
YouTube View
Launch Blog
Blog View
User

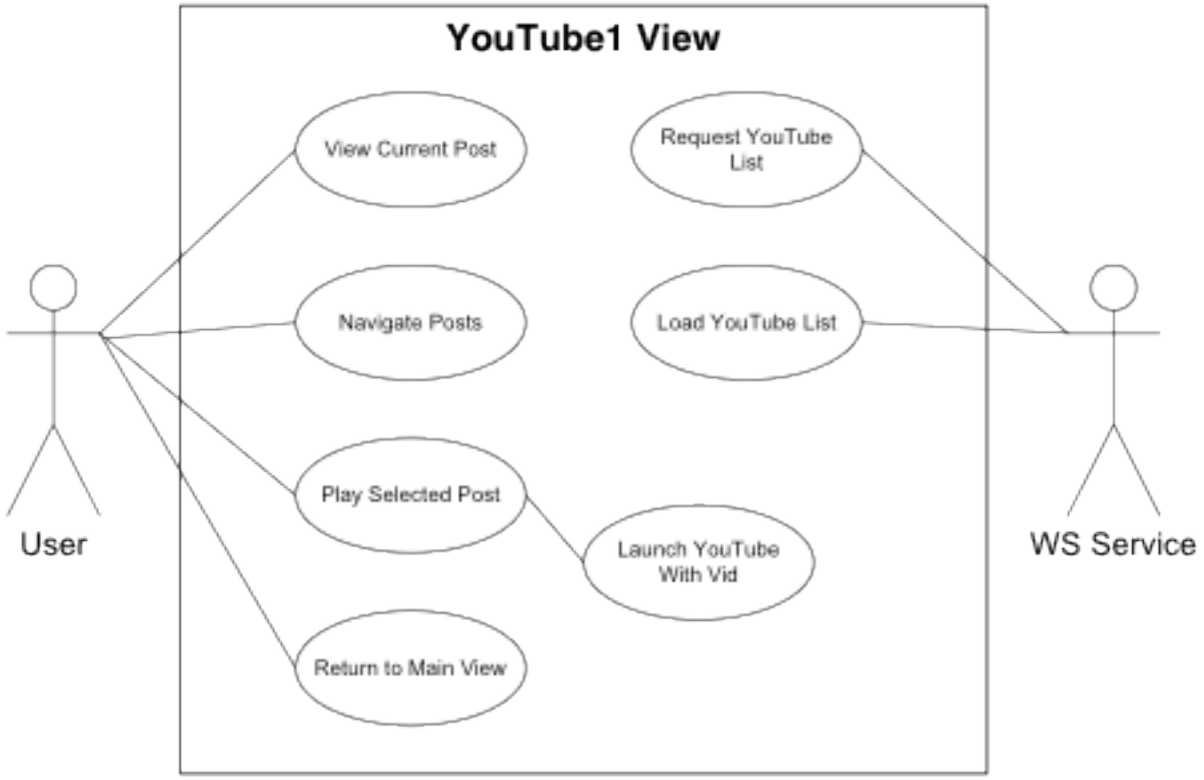

Blog View
View Current Post
Request Blog
Previous Post
Load Blog
Next Post
Return to Main View
User
WS Service

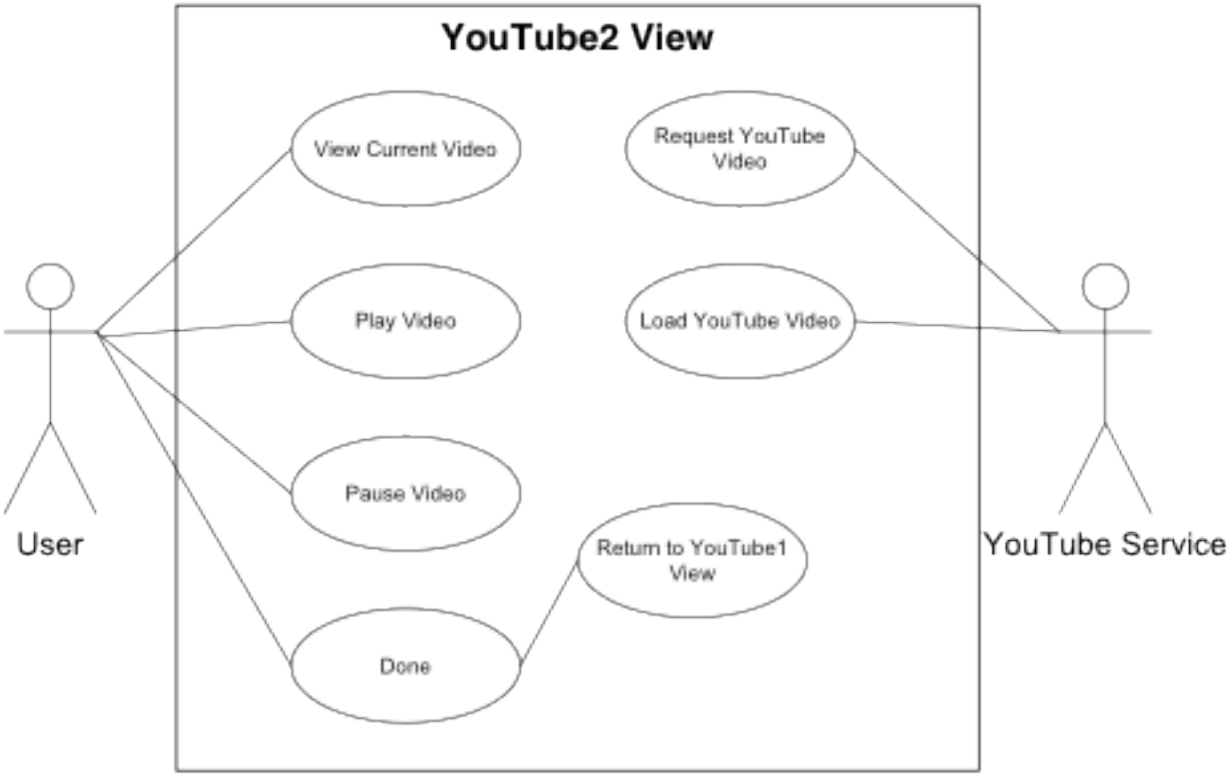

Coupons View
View Coupon
Request Coupon
Claim Coupon
Load Coupon
Return to Main View
Refresh View
User
WS Service


Facebook View
Launch FBConnect
Validate Credentials
Load Facebook Wall
Navigate Wall
Update Facebook Wall
Write Text
Post Text
Select Image
Post Image
Select Video
Post Video
User
Facebook Server

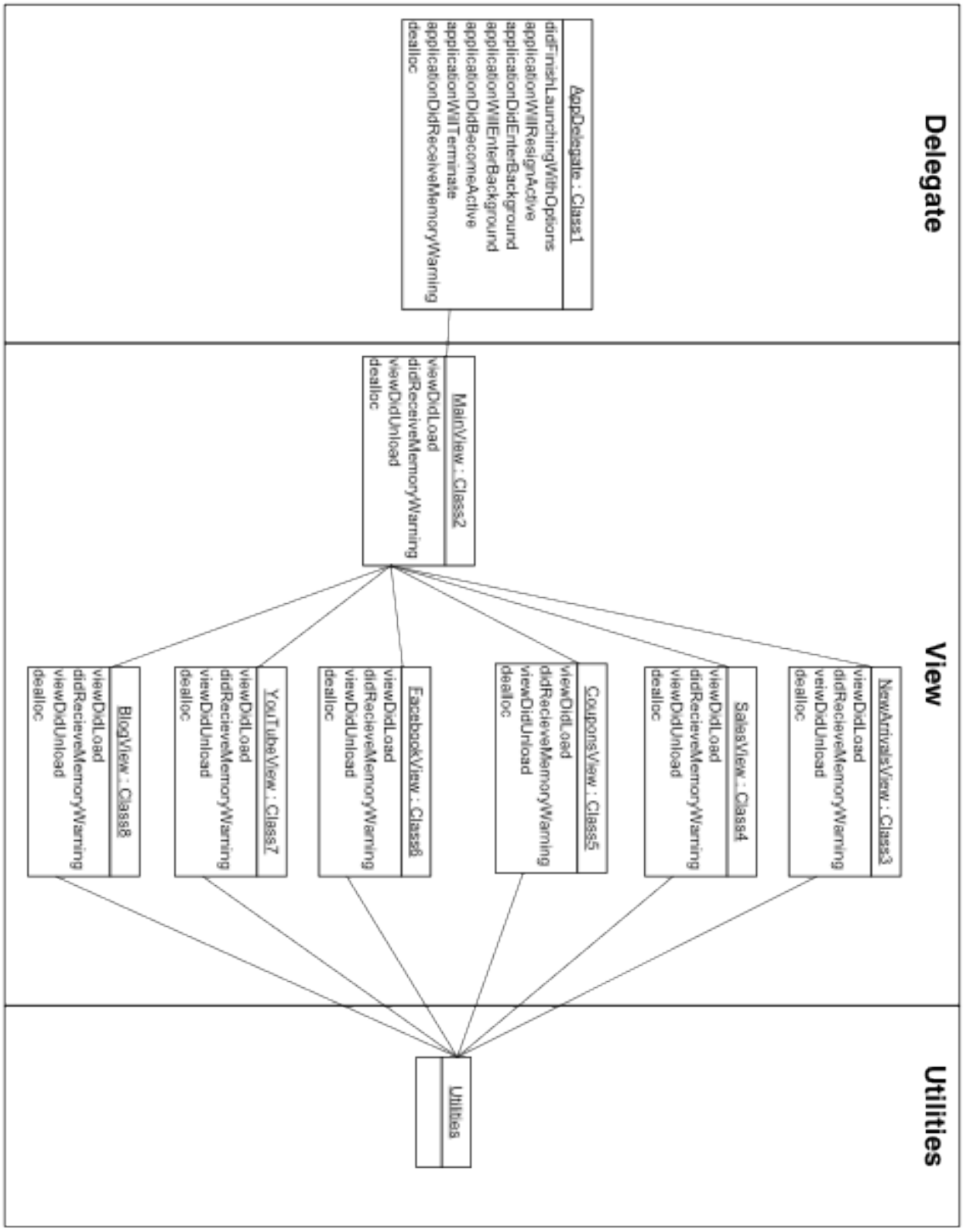

Sales View
Navigate Sales
Request Sales
Select Sale
Load Sale
Return to Main View
Refresh View
User
WS Service


New Arrivals View
Navigate Products
Request Products
Select Product
Load Products
Return to Main View
Refresh View
User
WS Service


YouTube1 View
View Current Post
Request YouTube List
Navigate Posts
Load YouTube List
Play Selected Post
Launch YouTube With Vid
Return to Main View
User
WS Service


YouTube2 View
View Current Video
Request YouTube Video
Play Video
Load YouTube Video
Pause Video
Return to YouTube1 View
Done
User
YouTube Service

Appendix B


Delegate
View
Utilities
AppDelegate : Class1
didFinishLaunchingWithOptions
applicationWillResignActive
applicationDidEnterBackground
applicationWillEnterBackground
applicationDidBecomeActive
applicationWillTerminate
applicationDidReceiveMemoryWarning
dealloc
MainView : Class2
viewDidLoad
didReceiveMemoryWarning
viewDidUnload
dealloc
NewArrivalsView : Class3
viewDidLoad
didRecieveMemoryWarning
veiwDidUnload
dealloc
SalesView : Class4
viewDidLoad
didRecieveMemoryWarning
viewDidUnload
dealloc
CouponsView : Class5
viewDidLoad
didReceiveMemoryWarning
viewDidUnload
dealloc
FacebookView : Class6
viewDidLoad
didRecieveMemoryWarning
viewDidUnload
dealloc
YouTubeView : Class7
viewDidLoad
didRecieveMemoryWarning
viewDidUnload
dealloc
BlogView : Class8
viewDidLoad
didRecieveMemoryWarning
viewDidUnload
dealloc
Utilities

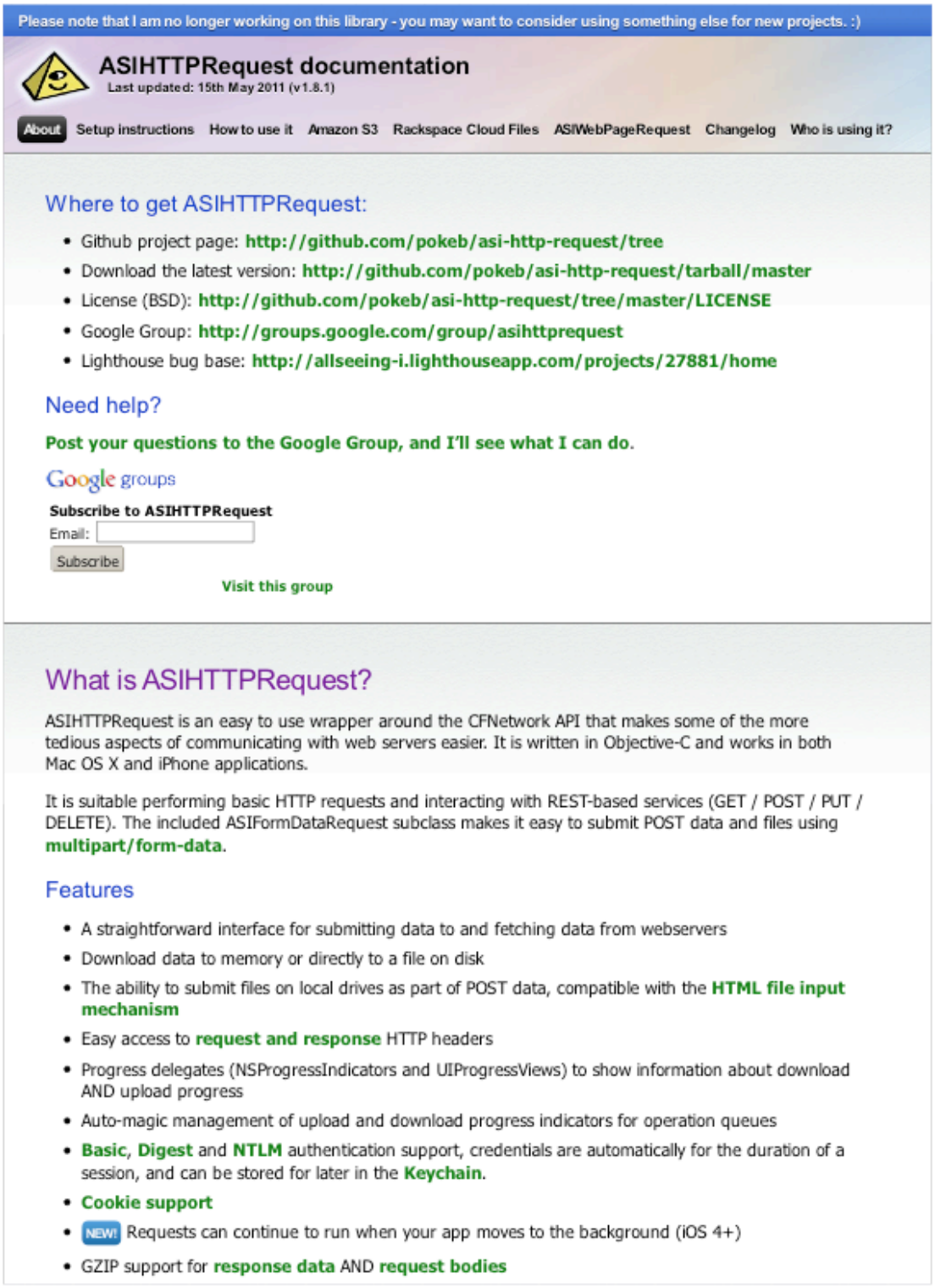
Appendix C

Evaluation Form
For
Warehouse Skateboards Mobile

Notice: This evaluation form is anonymous; its feedback will be used in the paper "Going Mobile with Warehouse Skateboards"

1. Do you feel perspective users will embrace this application?
   1-Strongly Agree 2-Agree 3-Neutral 4-Disagree 5-Strongly Disagree
2. As a perspective user was this application a topic of interest?
   1-Strongly Agree 2-Agree 3-Neutral 4-Disagree 5-Strongly Disagree
3. As a user will you promote this application to other people?
   1-Strongly Agree 2-Agree 3-Neutral 4-Disagree 5-Strongly Disagree
4. Do you have any of the following Social Media accounts?
   1-Facebook 2-Twitter 3-MySpace 4-LinkedIn 5-Other
5. Have you visited www.warehouseskateboard.com?
   1-Yes 2-No
6. Have you ever developed a software application?
   1-Yes 2-No
7. Is the application User Friendly?
   1-Strongly Agree 2-Agree 3-Neutral 4-Disagree 5-Strongly Disagree
8. Does this application provide sufficient access to social media?
   1-Strongly Agree 2-Agree 3-Neutral 4-Disagree 5-Strongly Disagree
9. Does the overall UI design match the main Warehouse Skateboard site?
   1-Strongly Agree 2-Agree 3-Neutral 4-Disagree 5-Strongly Disagree
10. Does this application utilize current iPhone functionalities such as swipes, icons, gestures, buttons, and navigations?
    1-Strongly Agree 2-Agree 3-Neutral 4-Disagree 5-Strongly Disagree
11. Are you please with overall complexity of the application?
    1-Strongly Agree 2-Agree 3-Neutral 4-Disagree 5-Strongly Disagree
12. Are you able to obtain the data you desire easily?
    1-Always 2-Often 3-Sometimes 4-Rarely 5-Never
13. Did the application crash during your session?
    1-Yes 2-No
14. Was the iOS platform suitable for this application?
    1-Strongly Agree 2-Agree 3-Neutral 4-Disagree 5-Strongly Disagree
15. Should we develop an Android version?
    1-Strongly Agree 2-Agree 3-Neutral 4-Disagree 5-Strongly Disagree
16. As a Warehouse Skateboard Staff Member were you satisfied with the application?
    1-Strongly Agree 2-Agree 3-Neutral 4-Disagree 5-Strongly Disagree
17. Do you feel this application will help promote Warehouse Skateboards?
    1-Strongly Agree 2-Agree 3-Neutral 4-Disagree 5-Strongly Disagree
18. Which of the features below would you add to application on its next version?
    1-Youtube 2-Twitter 3-Cataloge 4-Shopping Cart
19. Do you feel this concept can be applied to other commerce websites?
    1-Strongly Agree 2-Agree 3-Neutral 4-Disagree 5-Strongly Disagree
20. How often do the application and the main site go out of sync with each other?
    1-Always 2-Often 3-Sometimes 4-Rarely 5-Never

Appendix D
[From – http://allseeing-i.com/ASIHTTPRequest/]



Please note that I am no longer working on this library - you may want to consider using something else for new projects. :)

ASIHTTPRequest documentation

Last updated: 15th May 2011 (v1.8.1)

About | Setup instructions | How to use it | Amazon S3 | Rackspace Cloud Files | ASIWebPageRequest | Changelog | Who is using it?

Where to get ASIHTTPRequest:

- Github project page: **http://github.com/pokeb/asi-http-request/tree**
- Download the latest version: **http://github.com/pokeb/asi-http-request/tarball/master**
- License (BSD): **http://github.com/pokeb/asi-http-request/tree/master/LICENSE**
- Google Group: **http://groups.google.com/group/asihttprequest**
- Lighthouse bug base: **http://allseeing-i.lighthouseapp.com/projects/27881/home**

Need help?

Post your questions to the Google Group, and I'll see what I can do.

Google groups

Subscribe to ASIHTTPRequest

Email:

Subscribe

Visit this group

What is ASIHTTPRequest?

ASIHTTPRequest is an easy to use wrapper around the CFNetwork API that makes some of the more tedious aspects of communicating with web servers easier. It is written in Objective-C and works in both Mac OS X and iPhone applications.

It is suitable performing basic HTTP requests and interacting with REST-based services (GET / POST / PUT / DELETE). The included ASIFormDataRequest subclass makes it easy to submit POST data and files using **multipart/form-data**.

Features

- A straightforward interface for submitting data to and fetching data from webservers
- Download data to memory or directly to a file on disk
- The ability to submit files on local drives as part of POST data, compatible with the **HTML file input mechanism**
- Easy access to **request and response** HTTP headers
- Progress delegates (NSProgressIndicators and UIProgressViews) to show information about download AND upload progress
- Auto-magic management of upload and download progress indicators for operation queues
- **Basic**, **Digest** and **NTLM** authentication support, credentials are automatically for the duration of a session, and can be stored for later in the **Keychain**.
- **Cookie support**
- NEW! Requests can continue to run when your app moves to the background (iOS 4+)
- GZIP support for **response data** AND **request bodies**

- The included **ASIDownloadCache class** lets requests transparently cache responses, and allow requests for cached data to succeed even when there is no network available!
- NEW! **ASIWebPageRequest** - download complete webpages, including external resources like images and stylesheets. Pages of any size can be indefinitely cached, and displayed in a UIWebview / WebView even when you have no network connection.
- **Easy to use support for Amazon S3** - no need to fiddle around signing requests yourself!
- **Full support for Rackspace Cloud Files**, contributed by Mike Mayo of Rackspace.
- NEW! **Client certificates support**
- **Supports manual and auto-detected proxies, authenticating proxies, and PAC file auto-configuration**. The built-in login dialog lets your iPhone application work transparently with authenticating proxies without any additional effort.
- **Bandwidth throttling support**
- Support for persistent connections
- Supports synchronous & asynchronous requests
- Get notifications about changes in your request state via delegation or NEW! blocks (Mac OS X 10.6, iOS 4 and above)
- Comes with a broad range of unit tests

ASIHTTPRequest comes with a example applications for Mac and iPhone that demonstrate some of the features.

ASIHTTPRequest is partly based on code from **Apple's ImageClient** code samples, so if it doesn't meet your needs, take a look at their CFNetwork examples for more.

ASIHTTPRequest is compatible with Mac OS 10.5 or later, and iOS 3.0 or later.

An overview of the classes

Main classes

ASIHTTPRequest.h

ASIHTTPRequest.m

ASIHTTPRequest

Handles the basics of communicating with webservers, including downloading and uploading data, authentication, cookies and progress tracking.

ASIFormDataRequest.h

ASIFormDataRequest.m

ASIFormDataRequest

A subclass of ASIHTTPRequest that handles x-www-form-urlencoded and multipart/form-data posts. It makes POSTing data and files easy, but you do not need to add this to your project if you want to manage POST data yourself or don't need to POST data at all.

ASINetworkQueue.h

ASINetworkQueue.m

ASINetworkQueue

A subclass of NSOperationQueue that may be used to track progress across multiple requests. You don't need this if you only need to perform one request at a time, or prefer to track the progress of each request individually.

ASIDownloadCache

This class allows ASIHTTPRequest to transparently cache responses from webservers. Requests can be configured to use cached content when remote data has not been updated since it was last downloaded, when the network is available, or whenever cached data is available.

If you do not wish to use caching, or have written your own cache, you do not need to include this class.

Support classes

You will not generally need to use these classes directly, they are used behind the scenes by the main classes above.

ASIInputStream

A helper class used by ASIHTTPRequest when uploading data. You must include this class in your project to use ASIHTTPRequest.

ASIDataDecompressor

A helper class used by ASIHTTPRequest to inflate (decompress) gzipped content. You must include this class in your project to use ASIHTTPRequest.

ASIDataCompressor

A helper class used by ASIHTTPRequest to deflate (compress) content. You must include this class in your project to use ASIHTTPRequest.

ASIAuthenticationDialog

This class allows ASIHTTPRequest to present a login dialog when connecting to webservers that require authentication, and authenticating proxies. It is required in all projects targeting iPhone OS, though not for Mac OS projects.

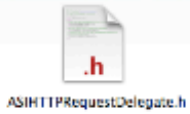
Reachability

This class was written by **Andrew Donoho** as a drop-in replacement for **Apple's Reachability class**. It allows ASIHTTPRequest to be notified when the network connection changes from WWAN to WiFi, or vice-versa. You must include this class in iPhone projects, but not in Mac projects.

You may find this class useful in detecting the status of network availability in your own applications - **find out more**.

Protocols and configuration

You must include all these files in your project.



ASIHTTPRequestDelegate

This protocol specifies the method that a delegate of an ASIHTTPRequest may implement. All these methods are optional.

ASIProgressDelegate

This protocol lists the methods that an uploadProgressDelegate or downloadProgressDelegate may implement. All these methods are optional.

ASICacheDelegate

This protocol is used to specify the methods that a download cache must implement. If you want to write your own download cache, make sure it implements the required methods in this protocol.

ASIHTTPRequestConfig.h

This file defines global configuration options that are set at compile time. Use the options in this file to turn on various debugging options that print information about what a request is doing to the console. **Don't forget to turn these off in shipping applications!**

Appendix E

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