Bachelor's Thesis

Information Technology

2013

Samuel Chukwumah

DEVELOPING A CROSS-PLATFORM MOBILE APPLICATION: MY SHOPPING LIST

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR´S THESIS | ABSTRACT
TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Networking

June 2013| 47

Instructor: Almurrani Balsam

Samuel Chukwumah

DEVELOPING A CROSS-PLATFORM MOBILE APPLICATION: MY SHOPPING LIST

This thesis examines the development of a mobile application that works on multiple mobile operating systems with the same source code. The application was implemented with HTML5 programming and wrapped with PhoneGap (an open source cross-platform development framework).

A shopping list application was designed and implemented to enable users to manage a list of items before, during, and after shopping. Six shopping list-related applications from two app stores were downloaded and analyzed based on their functional requirements and user feedback. As a result, an application that meets the usability requirements most users would expect was created. A number of features, including direct access buttons and defensive design were designed to enhance the user experience of the application.

An iterative development model comprising of requirement analysis, design, implementation, testing, and evaluation was also used to guide the project from start to completion. Testing was carried out on Androids and Windows Phone real and virtual devices.

The outcome of the thesis validates HTML5 programming as a considerable approach to develop cross-platform mobile application provided that the associated constraints are acknowledged.

KEYWORDS:

Cross-platform, mobile application, html5, phonegap

DEDICATION AND ACKNOWLEDGEMENTS

I dedicate this thesis to the memory of my late mother Blessing Chukwumah who could not witness my B.Eng. graduation. Her loving care and support from my childhood till she passed away helped me face the challenges in my career.

I would like to thank my thesis supervisor, Balsam Almurrani. His support helped this thesis to reach the goal of completing the thesis within the estimated time. I thank my teacher Poppy Skarli whose course on academic writing helped to improve my writing skills which was valuable to the thesis. I express my gratitude to the Finnish government. Their policy of free education for all enabled foreign students including myself to acquire a prestigious higher education degree with zero tuition fees.

A big thank you to my parents and siblings for the sacrifices made to support me financially and spiritually. My warmest thanks belong to my dear wife, Anniina, who supported and encouraged me throughout my thesis. I also appreciate my parents-in-law for the support and encouragement for my relationship and career.

Finally, I thank my God for his faithfulness and loving-kindness towards me. I asked His help when it was difficult to carry on with my thesis and He gave me the inspiration to accomplish it.

CONTENTS

1 INTRODUCTION **4**
1.1 Project Goal 4
1.2 Project Scope 5
1.3 Thesis Overview 5
2 BACKGROUND ON MOBILE APPLICATIONS **6**
2.1 Rising Demand for Mobile Applications 6
2.2 Strategies for Developing a Mobile Application 7
3 REQUIREMENT ANALYSIS **9**
4 PROJECT MODEL AND DESIGN **12**
4.1 Project Model 12
4.2 Design Features 14
4.2.1 Direct Access Buttons and Defensive Design 14
4.2.2 Simplified Shopping 15
5 IMPLEMENTATION **21**
5.1 Implemented languages 21
5.2 Implemented Libraries and Frameworks. 22
5.3 Implementation of Screen Views 27
5.4 Storage Implementation 28
5.5 Adding Item to Shopping List 29
5.6 Editing, Deleting, and Sorting Items 30
6 TESTING AND RESULT **32**
7 CONCLUSION **34**
7.1 Discussion 34
7.2 Future Work 34

REFERENCES **35**

APPENDICE

Appendix 1. HTML code for form without twitter bootstrap 37
Appendix 2. HTML code for form with twitter bootstrap 38
Appendix 3. Item template derived from mustache 40
Appendix 4. Data storage 40

Appendix 5. Add and save item 42
Appendix 6. Item deleted with primary delete button 43
Appendix 7. Handle screen view change 44

FIGURES

Figure 1. Use case diagram that captures the requirements of the shopping list app 12
Figure 2. Iterative model diagram 13
Figure 3. Indirect access buttons 14
Figure 4. Direct access buttons 14
Figure 5. Shopping list screen view 16
Figure 6. Activated Add Item button 17
Figure 7. Items not bought 19
Figure 8. Items bought 19
Figure 9. Secondary delete button 20
Figure 10. Items List screen view 21
Figure 11. Form coded without bootstrap framework 24
Figure 12. Form coded with bootstrap framework 24
Figure 13. Displayed Item template 25
Figure 14. Mobile operating systems supported on PhoneGap framework. 25
Figure 15. Supported features on PhoneGap (PhoneGap 2013). 26
Figure 16. PhoneGap Build compiles and deploys mobile application (Pete 2012). 27
Figure 17. Notification of existing item name. 29
Figure 18. Notification of null item name. 30
Figure 19. Deleting a collection of items using the secondary delete button. 32

TABLES

Table 1. Worldwide Devices Shipments by Operating System (1,000 of Units) (*Gartner Press Release 2013*). 6
Table 2. Native code language fragmentation 8
Table 3. Details of analyzed apps that relates to the thesis project. 10
Table 4. Testing on different devices and platforms 33

LIST OF ABBREVIATIONS (OR) SYMBOLS

API	API stands for Application Programming Interface. An API is a set of function that is used to work with a software component , application or operating system (Microsoft msdn 2013).
CSS	CSS stands for Cascading Styling Sheets
HTML	HTML, which stands for Hypertext Markup Language, is the predominant markup language for web pages.
IDE	Integrated Development Environment.

JSON	JavaScript Object notation. It is a lightweight data-interchange format.
OS	Operating System
RIM	Research In Motion, a company that manufactures blackberry smartphones
SDK	Software Development Kit
SQL	Structured Query Language
UI	User Interface
app	application
iOS	brand name of Apple’s Mobile Operating System

1 INTRODUCTION

The development of mobile applications took a new turn when Apple and Google launched the Apple app store and Android market (now Google Play) in 2008. Both app stores were set up to allow third party developers to publish apps for consumers of each mobile operating system platform. Companies, such as Nokia, RIM, Microsoft and others, joined this trend to catch up with the competition for market share.

However, an app published in Apple app store is only native to the Apple's iOS. This means it is written in Objective C language and developed in Apple's SDK. To publish the same app product on Android would require a different language and SDK. The challenge for developers is that the majority of the mobile operating system requires a different language and SDK to develop apps native to a specific platform. Therefore, publishing the same app product on different platforms can be a challenging task because it can lead to differences in user experience. The good news is that technologies to allow develop once app and publish with same user experience across multiple platforms are emerging. The concept of cross-platform application is writing an app with a language that is supported on a cross-platform SDK. A Cross-platform SDK does not necessarily support all available mobile operating systems but can support at least the major ones. Another constraints is that a feature that is accessed through a native API may not be supported on a cross-platform SDK. Hence, when developing a cross-platform app, it is important to check if the constraints mentioned above do no limit a user experience across the targeted platforms.

This thesis is focused on a developing an application that gives the same user experience on majority of the mobile operating system platforms.

The author has chosen to create a shopping list app to assist a user to manage items that are to be shopped. Shopping is an activity that many people engage in daily. A well-planned shopping list will assist a shopper to focus on the essential items that are needed. It also saves the shopper’s time and helps keep to a budget.

1.1 Project Goal

The goal of this thesis project is to develop a shopping list application that provides the same user experience across multiple platforms. The application should run on more than one mobile operating system platform without making changes to the source code. It also hopes to produce an application with a simple user interface in order to make it easy for a user to operate.

1.2 Project Scope

The thesis covers the development of the application on PhoneGap development framework. At the end, the working features of the app are:

- Two screen view of items called Shopping list and Item list.
- Ability to build a personalized list of items.
- Ability to prioritize an item on the shopping list.
- Ability to delete an individual item, items not bought and bought items.

1.3 Thesis Overview

An overview of the rest chapters of the thesis is given below:

Chapter 2. Background on mobile applications

This chapter gives the reader an overview of the increase in the use of mobile devices especially smart phones which has led to high demand for mobile application development. The reader is also introduced to the different approaches of developing mobile application.

Chapter 3. Requirement Analysis

This chapter covers the process of how the app features in the thesis project was developed. A use case of the shopping list application is presented to the reader to capture the basic requirements upon which the app is built.

Chapter 4. Project model and design

This chapter details the iterative development process that is adapted in developing the thesis project. It also covers the various design features to ensure a good user experience of the application and the reasons for choosing some of the features.

Chapter 5. Implementation

This chapter mentions the choice of languages, libraries and framework that were used in implementing the thesis project. The reader will understand how data storage is implemented in the application as well as adding, editing and sorting of items in the shopping list application.

Chapter 6. Testing and results

This chapter presents the details of virtual devices and real devices that were used during the test phase and how the test was carried out. The limitations of the test are also mentioned as well as a brief report of the author's experiences on setting up and installing the test devices.

Chapter 7. Conclusion

This chapter discusses the benefits and limitations encountered in developing a cross-platform mobile application based on the author's experience with the project. It concludes the thesis writing with possible features that could be added as future work to the project before it is submitted to an app store.

2 BACKGROUND ON MOBILE APPLICATIONS

2.1 Rising Demand for Mobile Applications

Mobile applications usage is becoming an increasingly popular aspect of daily life around the world. Similarly, users of smart phones have also increased significantly over the last six years. The proliferation of smart phones targeted towards consumers has led to a rapid expansion of mobile applications development. This new era in mobile applications development creates a new business for developers in two ways. One way that generates business for developers is to publish apps to consumers through app stores. The developers earn their income by selling the apps or offering them as free and generate income from advertisement. Another way that generates business for developers is developing apps for organizations to offer services to their clients. For example, a bank may have a mobile banking app through which their clients can engage in bank transactions.

There has been a massive increase of iOS and android mobile operating system platforms which has been a major factor in the rise of mobile applications development. Android and iOS account for 87.6% of the smart phones worldwide shipment in 2012. While Blackberry, Symbian, Windows Phone and other platforms accounted for the rest 12.4% of smart phones shipped worldwide. (*IDC press release* 2013).

The most recent statistics at the time of writing this thesis indicate that 1.875 billion mobile phones are expected to be sold in 2013, a 7.4% increase compared to 1.746 billion mobile phones sold in year 2012. Of the expected mobile phones to be sold this year, 1 billion units will be smart phones. (*Gartner Press Release 2013*).

Table 1. Worldwide Devices Shipments by Operating System (1,000 of Units) (*Gartner Press Release 2013*).

Operating System	**2012**	**2013**	**2014**	**2017**
Android	497,082	860,937	1,069,503	1,468,619
Windows	346,457	354,410	397,533	570,937

Table 1. Worldwide Devices Shipments by Operating System (1,000 of Units) (*Gartner Press Release 2013)* (continue).

Operating System	**2012**	**2013**	**2014**	**2017**
iOS/MacOS	212,899	293,428	359,483	504,147
RIM	34,722	31,253	27,150	24,121
Others	1,122,213	871,718	702,786	396,959
Total	**2,213,373**	**2,411,796**	**2,556,455**	**2,964,783**

The facts stated above indicate that the demand for application development for these various mobile operating system platforms is on the increase.
Prior to the introduction of iPhones by Apple, the majority of the mobile applications developed were sold with mobile devices as predefined packages of mobile devices before selling them to consumers.

2.2 Strategies for Developing a Mobile Application

Many people define mobile application as any application that runs on a mobile device which is usually a mobile web application or a native application. This thesis defines mobile application as native application*. "A native app is one for which the library has gone extra step to create an app that is available for purchase in mobile stores (e.g., iTunes, Android market). (*La Counte 2011*)".*

A mobile application can be developed using one of the three main coding strategies defined by Eran Zinman (Eran 2012) . They are

- Native code
- HTML5 code and
- Hybrid code.

Native code mobile application.

A native code mobile application uses the native code of the targeted mobile device operating system platform for development. Table 2 shows a list of the native code languages for each mobile operating system. Applications developed with native codes have the best look and performance. They also have the fastest graphics which is very important when playing graphic intensive games. Developing an application with native code provides a developer with access to all the built-in features required to provide a user experience that most people expect on that device.

Native code-based mobile applications require an IDE environment which provides the tools for debugging, managing the project etc. A comprehensive knowledge of the native code language is very important when developing an application. Hence, it is not uncommon for a developer to be limited to just one mobile device platform. (Korf and Oksman 2013).

Table 2. Native code language fragmentation

Mobile OS Type	**Native Code Language**
Apple iOS	C, Objective C
Google Android	Java
RIM Blackberry	Java {J2ME flavored}
Symbian	C, C++, Python, HTML/CSS/JS
Windows Mobile	.NET
Window 7 Phone	.NET
HP Palm webOS	HTML/CSS/JS
Meego	C, C++, HTML/CSS/JS
Samsung bada	C++

HTML5 Code Mobile Application

An HTML5 code mobile application is an application developed by using HTML5 code combined with JavaScript and CSS. The greatest advantage with an application purely based on HTML5 is that it can easily be adapted to various mobile operating system platforms. Most developers are already familiar with using HTML5, JavaScript and CSS, thus , it is relatively easy to use it to develop a mobile application. The drawback of developing an application with pure HTML5 coding is that it cannot access all the built in features that are available on a device. HTML5-coded mobile apps can be developed on any of the cross-platform software development platform such as PhoneGap, MoSync, Apcellerator etc. The thesis project adapts this strategy to develop a mobile application in order to produce an application that can run on multiple mobile device platforms.

Hybrid Code Mobile Application

A hybrid code mobile application is an application that is developed by combining native code and HTML5 code. The purpose of integrating native codes with HTML codes is to leverage on the best of both strategy of app development. Integrating native codes can be used to address the limitations that are faced with HTML5 limitations. However, a developer would still require native code language skills to successfully implement an hybrid code approach. Just like HTML5 code mobile apps, hybrid apps can be developed on any of the cross-platform SDK , such as PhoneGap, MoSync and Apcellerator.

3 REQUIREMENT ANALYSIS

The requirement analysis encompasses the tasks involved in determining user expectations for a new or modified product or software application (Rouse 2007). The tasks involve getting to know the functional requirements which means what the system is required to do. Thus, a good understanding of the application requirements is needed in order to determine the specific features that should be implemented. These features must be quantifiable, testable, and relevant to the identified requirements and defined to a level of detail sufficient for designing the application. (*Requirements analysis* 2010, p. 1).

Existing applications related to the thesis project app were analyzed to identify the functional requirements. In addition, user feedbacks on the applications that were analyzed were also reviewed. This helps to develop the features of the thesis project app in line with what users expect.

The author downloaded three shopping list-related application from Windows Phone app market (App Highlights) and another three from Google play to carry out the requirements analysis. Apps from the Apple app store could not be analyzed because the author could not find anyone available during this phase of the project. However, it was determined that this would not affect the outcome of the analysis. The apps chosen were all from the list of highly rated apps and were chosen randomly. Details of the apps can be seen in Table 3.

Table 3. Details of analyzed apps that relates to the thesis project.

App Highlights	Version	Publisher	Star Ratings	Number of ratings	Release Date	Features
My Shopping List	V1.0.0.0	H3G SpA	3	2	16.02.13	Extended
Shopping List	V2.2.0.0	valuePhone GmbH	4	49	23.02.13	Basic
Simple Shopping List	V1.3.0.0	Cesar Meira	3,5	48	25.02.13	Basic
Google Play						
Shopping list	V2.7.0	Drama Productions	4,3	8104	26.03.13	Basic & extended options
Shopping List	n.a.	maloil	4,3	2065	06.02.13	Basic
OIShopping list	n.a	OpenIntents	4,4	11483	16.10.12	Extended

The features of the existing apps can be categorized into basic features and extended features. Basic features meet the requirements of the activities a day to day shopper would do. In simple terms, basic requirement features revolve around entering the items intended to be purchased into a shopping list. Whatever item(s) that is shopped is marked to show it has been purchased. It can also be cancelled from the list if the user wants to discard the item from the shopping list.

Extended features consist of the basic and extra features that enable the user to access a range of resources in the mobile device in order to provide an advanced shopping experience. Some of the apps in this category enable a user to take pictures of an item, input the item name through a barcode scanner, input price, append note and much more. These are great features but not all users may utilize them. The most basic of the apps which is Simple Shopping List developed by Cesar Meira had 48 ratings and a 3.5 stars. In comparison, My Shopping List developed by H3G SpA had 2 ratings and 3 stars at the time the analyses were performed. As seen from Table 3, both apps have close release dates. The shopping list app by Drama productions requested the user to select a basic or extended feature version when installing it. A review of the customer feedback on the app revealed that many users preferred the simple version.

There are also some downsides to having lots of extended features in an app. A lot of users do not have time to spend on interacting with an app. They want straight to the point features. One user complained that it takes three clicks to delete an item. The developer might have implemented this to prevent deleting an item unintentionally. However, this does not fit well with customers' expectations. The solution to this kind of problem is designing an intuitive user interface. The good

ratings and high number of user ratings observed in some apps with basic features do not reflect that all users want apps with just basic features. Instead, it means there is a market for users who are satisfied with an app that carries out the functions of the basic activities a user would do without the app. An app is also expected to simplify these activities and not make things complicated.

These observations lead to another fact that is observed from user behavior towards a mobile application. Some apps with extended features also have good ratings and a high number of user ratings. Although these apps had extended features which are complex to develop nevertheless, they had intuitive design. An intuitively designed app makes an app to be simple and easy for a user to interact with even if the user may not need all the features.

It can be observed from Table 3 that there is significant difference in the number of ratings from the Google play store and App Highlights. This is attributed to the huge difference in the market share of both apps. Apps on Google play app store are available to all devices running android whereas apps on App Highlights are sometimes limited to the Windows phone version. At the time of writing this thesis, some apps were specifically designed to run on just Windows phone 8 version of the Windows mobile operating system. This factor might also account for the overall number of ratings of apps from App Highlights which was downloaded on Windows phone 7.

The requirement analysis formed the basis upon which the requirements of project is captured and leads to creating the use case for the app project.

Use Cases

A use case captures the functionality that the application provides in fulfilling one or more of the users requirements (Miles and Hamilton 2006, p. 37). The use case diagram in Figure 1 is an instrumental model in the beginning of the project because it describes what actions the app users can perform in an overview. However, it is important to point out that it does not provide details that are sufficient in the actual designing and exact understanding of how the requirements will be met (Miles and Hamilton 2006, p. 45). More details and changes will be needed later during the design and implementation stage.

Shopping list mobile application requirements

The use case diagram in Figure 1 can be detailed into understandable terms as given below:

- The user shall be able to create a shopping list of items
- The user shall be able to edit/delete or mark an item as important: Marking an item as important will prioritize the item in the shopping list.
- The user shall be able to buy/unbuy an item: A user buying an item may return the item. Unbuying the item allows the user to change the state of the

- The app shall enable a user to save an item for one time shopping in the shopping list or store it in a list from where it can be ordered in the future.
- The user shall be able to order an item into the shopping list from a list of stored item.

An item ordered from the stored list of items can only be ordered once into the shopping list except if it has been bought.

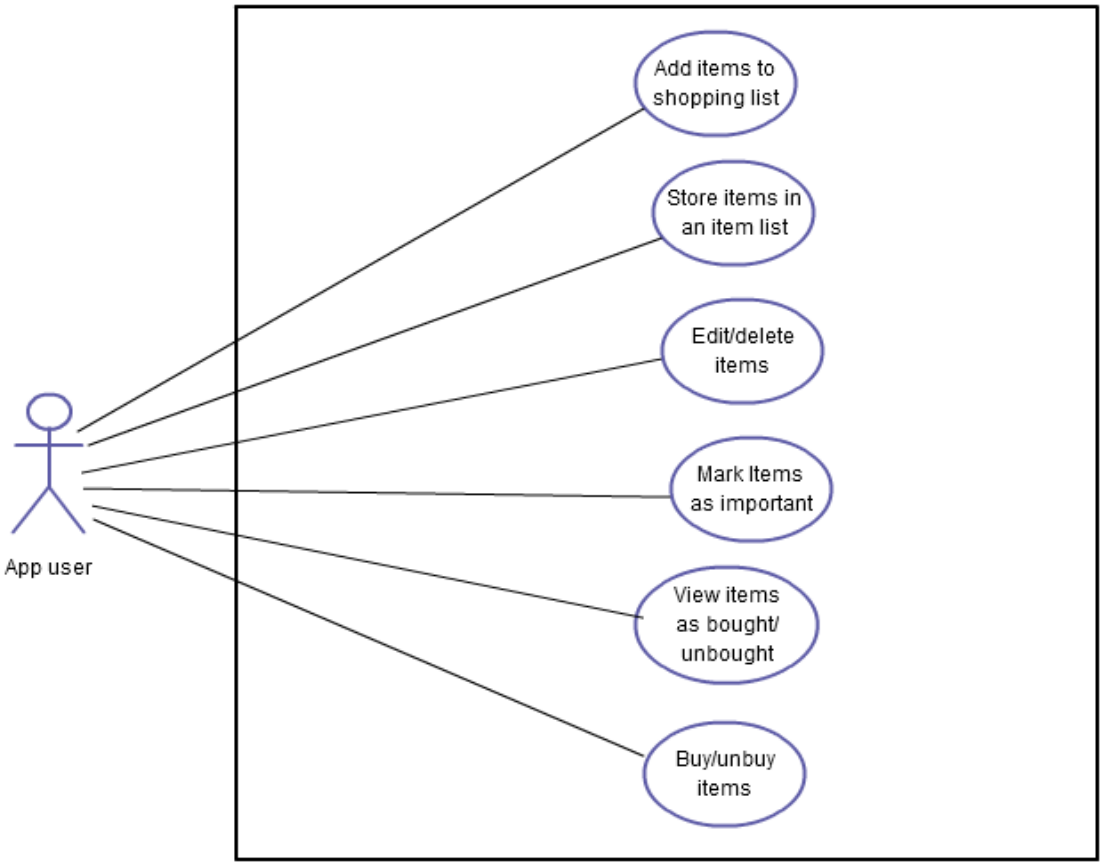

Figure 1. Use case diagram that captures the requirements of the shopping list app

4 PROJECT MODEL AND DESIGN

4.1 Project Model

The approach to the design of the project is iterative development. As seen in Figure 2, there is a cycle of requirements, design, implement, test and evaluate. These five components make up the model that is adapted in developing the thesis project. The potential success of the mobile apps lies in the functionalities and good user experience. It is also important to distinguish the app in

certain aspects from other apps available. Hence, the cycle of the five model components iterates till there is a reasonably successful outcome of an application that fulfills the goal of the thesis project. The discussions on design and implementation will focus on the final version of the thesis project. A brief overview of the five model components of the thesis project is give below:

Requirements: Details of this component have been discussed in the literature review chapter. Iterating through this stage produces further requirements till the final specification for the app is developed.

Design: This involves a series of storyboards, sketching and mockups that were used in detailing the screen view and user interactions of the app. Several iterative changes were made before the actual Implementation: This helped to save time, however, the process is revisited several times after a need for changing some design features is discovered during evaluation. This chapter is focused on the design stage

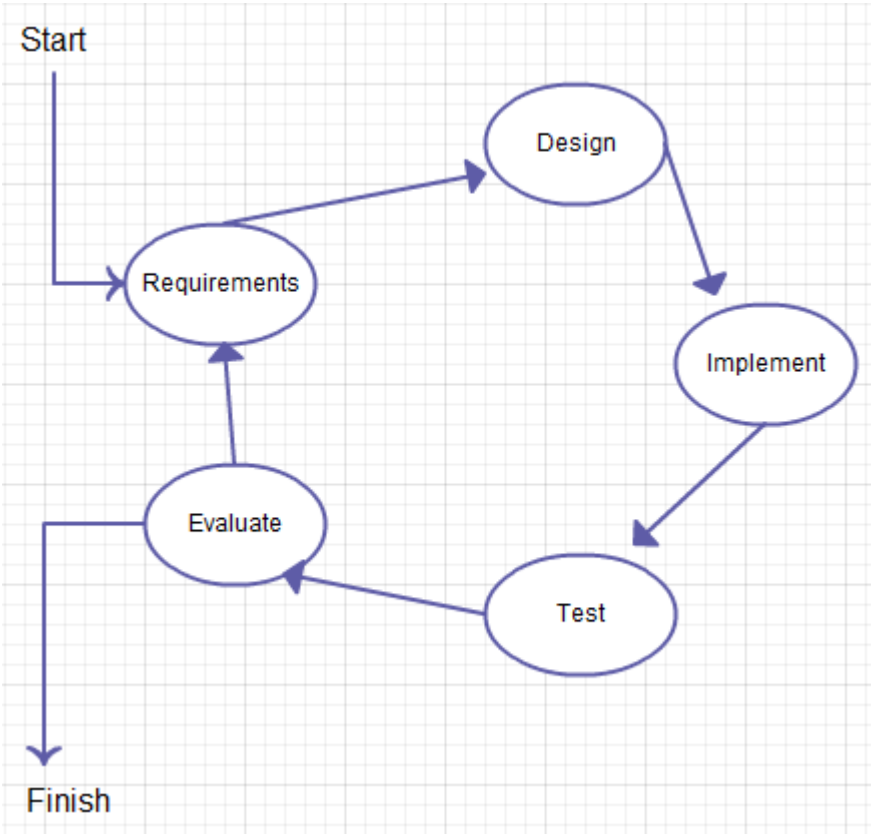

Figure 2. Iterative model diagram

Implementation: This involves the activities that are undertaken to execute the requirements and design.

Test: Testing is performed to verify the requirements and design specification is met.

Evaluate: The product of the implementation is examined to determine if some additional features should be added or removed.

4.2 Design Features

4.2.1 Direct Access Buttons and Defensive Design

The app design enables the user to have direct access to the primary actions that are associated to a shopping activity. The primary actions identified are buying, editing, prioritizing, deleting of an item and navigation. All these actions can be performed directly on the screen view. This design approach helps to simplify the usability of the app as well as reduce the number of times to perform an action. Figure 3 describes how a user performs an action on an item by first tapping on the **>** button. This would require the user to perform a minimum of two taps. Figure 4 describes how the same actions on an item can be performed by a minimum of one tap by providing direct access to the actions in a single view.

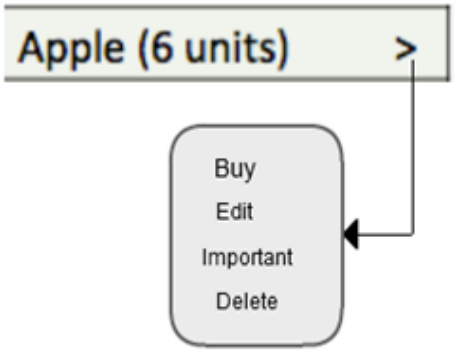

Figure 3. Indirect access buttons

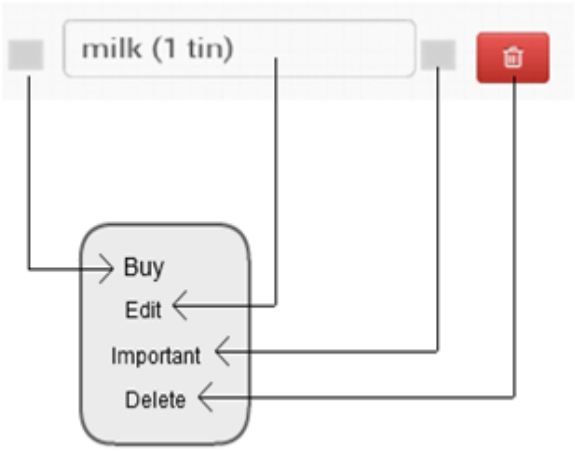

Figure 4. Direct access buttons

It can be noticed that the direct access buttons create a flaw that allows the user to easily mistap an unintended action. For example, an item can be mistakenly deleted with a mistap. This is where the defensive design comes in. Defensive design refers to the idea that the interface is designed in such a way that potential mistaps will not force the user to make an unintended action. (Pro-Logix

2011). One likely action that can be a potential mistap is deleting an item. The implementation chapter covers the defensive design approach that is undertaken.

4.2.2 Simplified Shopping

Many of the shopping list apps available in the app stores enable users to set up multiple shopping lists. A lot of users appreciate this feature. However, there are many others that may be fine with just one shopping list when they go shopping. Setting up multiple shopping lists may help to group items together for certain reasons but it may also be challenging to manage. One reason a user might want to create two shopping lists may be to have one list for essential items that must be purchased and another list for items that are nonessential to purchase. In the case that all essential items cannot be purchased from one store, the user would most likely look for them in another store. Having only one shopping list might as well achieve this purpose. The design of the app can take care of such kinds of potential reasons that make a user to create multiple shopping lists. The goal of this design is to simplify the shopping experience and still satisfy the needs of many users. Hence, one shopping list is designed to be implemented.

The advantage of a single shopping list is given below:

- It enables the central management of all items that are to be shopped.
- It reduces the time spent in engaging with the app. Navigating from one shopping list to another to perform an action takes more time than having just one list.
- It is simple to implement.
- Less storage space is required; hence the app can run faster.

The proposed solution to achieve the set requirements for the shopping list app is made up of two screen views. They are:

- Shopping List
- Items List

Shopping list screen view

The shopping list page serves as the home page of the app. It is made the home page because the primary objective of the user is to engage in shopping activities. The shopping list screen view is divided into three parts as observed in Figure 5. They are:

- Top
- Body
- Bottom

Top

The top screen has buttons of the actions that the user is mostly likely to perform when starting the app. They are:

- Add item
- Shopping list
- Item list

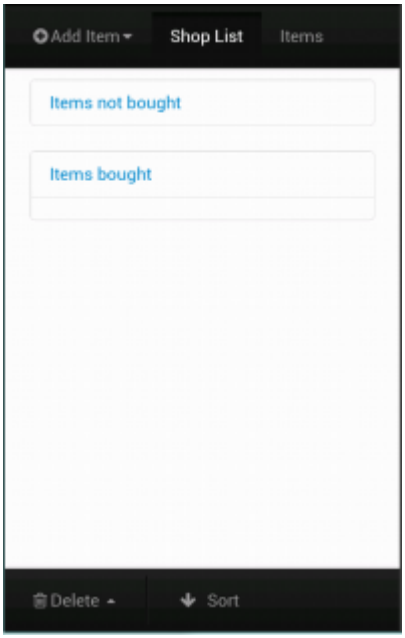

Figure 5. Shopping list screen view

Add Item: This button is used to add items into the shopping list. As seen from Figure 6, when an item is to be added, the user can set two attributes in addition to the name of the item by marking it as important and detail the description. These attributes are not compulsory for an item to be added to shopping list. At the time of adding an item to the shopping list, the user might be sure of specifying the item as important. Giving this option to the user helps to detail an item's priority at the point of adding the item. The important option solves the need to create multiple shopping lists of items to distinguish their priority. Having the option to specify an item as important is one of the features that distinguish this project app from those that the author analyzed from the app market.

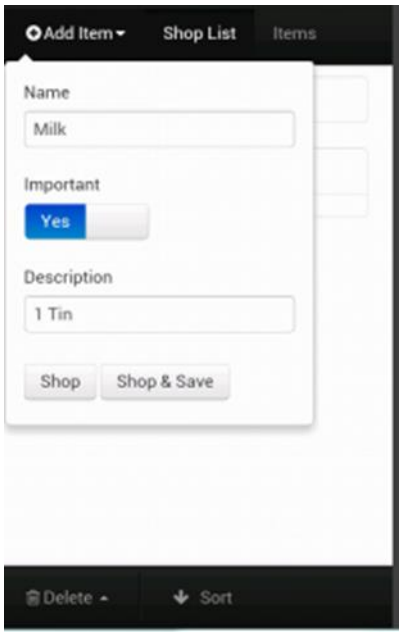

Figure 6. Activated Add Item button

The description text allows the user to provide details such as price of the item, amount, and item unit. For example kg, kilo, crate etc. can be specified by the user in the description to refer to the unit of an item. Most of the apps with extended features analyzed during the requirement analysis stage provided users with options to specify the amount, units, category etc. For instance, one app enabled the user to input the price of item(s) which it uses to determine the shopping cost. The author made the decision to drop these features for the following design reasons. Firstly, having so many options to specify an item description will take much of the user's time when choosing the suitable options for an item. Secondly, the units or description which a user uses to identify an item may not be among the options. Giving the user to option to type the description enables the user to specify a more personalized description.

After a user enters an item’s information, there are two options to save the item in the shopping list. They are:

- Shop
- Shop & Save

Choosing the shop option should save the item to the body of the shopping list screen view under the list of item not bought as seen in Figure 6 . This option is preferred when adding an item that is not often shopped into the shopping list. In this case, the user may want to shop the item but not save it in a list from where it can be selected in future shopping.

Choosing the Shop & Save option saves the item to the shopping list and stores it in items list. Only the item name is passed on to be saved on the items list. This manner of saving an item in the item list allows the user to manually select the items to be shopped later. Hence, the user can gradually build a personalized list of items that are often shopped.

The other two buttons at the top screen are the shopping list and items list which help to simplify navigation for the user. When a user navigates to one of the screen views, the button of that screen is to be highlighted to signify which screen is active.

The Body

The body section of the shopping list screen categorizes the items into a list of Items not bought and Items bought. When an item is added to the shopping list, it is first listed under the Items not bought. There are means by which an item is added into the shopping list. One way is through the add item button in the shopping list screen that is discussed previously. The other way is adding an item from the items list screen. More details on adding an item through the items list is covered under the section items list screen view.

When an item is added to the Items not bought list as seen in Figure 7 and 8, it has a buy (check) button starting from the left; a text that contains the item name and description within bracket; another check (important) button and a trash-can icon that signifies delete. As seen from Figure 7, the first two items under the Items not bought list have a blue background and a checked button on the right. The blue background signifies that the item is selected as important In other words it is essential. The check button can be set at the time of adding an item, or after the item has been added to the shopping list. The important items are also placed at the top of the list in the shopping list screen view so that the user can easily notice if the item was not purchased. This same rule applies to items under the Items bought list.

The user enlists an item as bought by checking the buy button and the screen is updated to reflect this change. All items under the items bought list have their left check button checked to identify that they have been purchased. It is not uncommon for a user to mistakenly buy an item or return a bought item. In such situation, the item already marked as bought can be unbought by unchecking the buy button of the item. The important button is the check button close to the delete icon. It allows the user to specify an item as a must buy and should move the item to the top of the screen.

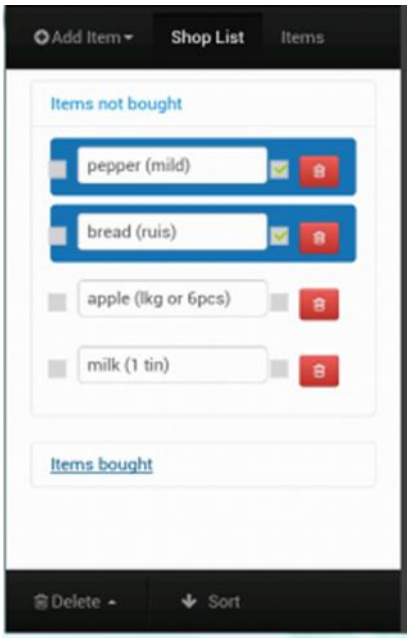

Figure 7. Items not bought

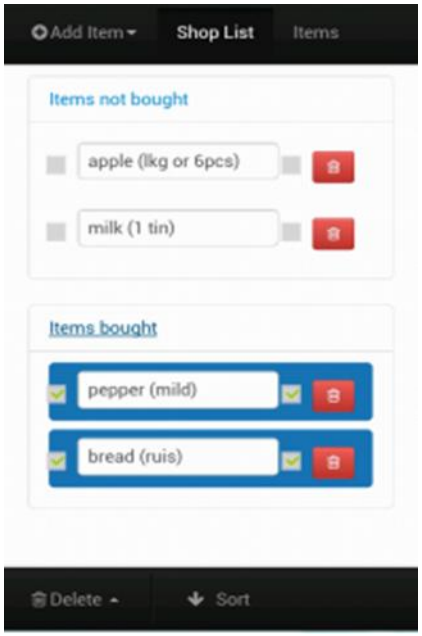

Figure 8. Items bought

Bottom

The bottom of the shopping list screen view hosts two buttons which are used to perform secondary actions. The buttons are delete and sort. This secondary delete action is different from

the delete icon associated with each item. As seen from Figure 9, there are three delete options. "Delete All" deletes all the items in the screen when selected. “Delete Bought” deletes all bought items and “Delete Not Bought” deletes all not bought items. These options are very useful if a user requires clearing the items in a list. Although this feature was not included in the initial design because most of the analyzed apps in the requirement analysis stage did not have it a, further review on what users require highlighted the relevance of this feature.

The sort button is used to sort items alphabetically in increasing or decreasing order. More details on how it works are given in the implementation.

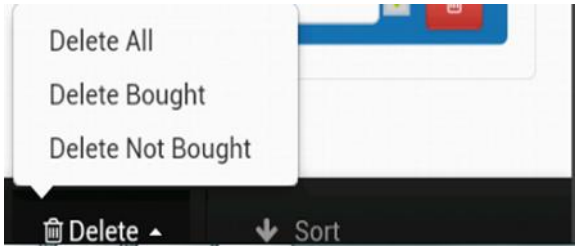

Figure 9. Secondary delete button

Items list screen view

The items screen serves as a repository of items that the user has saved for future shopping. The user can add items to the shopping list by a single tap on the buy button item. This design helps to prevent the user from adding an item more than once to the shopping list because each item is checked when added.

The screen is also divided into three sections similar to the shopping list screen view which are the top, body and bottom as seen in Figure 10. The user can access the add item's button from the top section and also navigate to the shopping list screen. The body simply contains the items in the repository and the bottom has a sort button to enable the user to sort the items in alphabetical order.

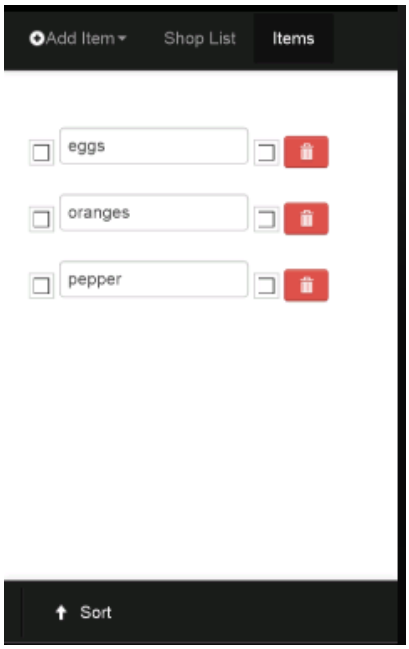

Figure 10. Items List screen view

5.1 Implemented languages

The key principle behind the development of the cross-platform app is using the HTML markup language to structure content, the CSS style sheet language to style the content and the JavaScript scripting language to determine the behavior and logic of the app. An overview of these three technologies in relation to the thesis project is given below:

HTML5

HTML is the publishing language of the World Wide Web (Smith 2013). The latest version of HTML is HTML5. It is currently supported by virtually mobile browsers in the market today. As HTML5 is a cross-platform, this makes it a good choice to build a cross-platform application. HTML5 has new added tags and attributes that did not exist in previous versions of HTML. HTML5 also supports some APIs that work with it but are not included in its specification (Kyrnin 2012, p. 5). An example of those APIs is the local storage API which stores data locally on the client device. A number of these features were implemented in the thesis project. However, only the most relevant features that were used in the thesis project are discussed in this thesis.

CSS

CSS is responsible for styling how the structure of HTML contents should look. HTML elements can be styled individually but this approach is very limited in design options and cumbersome when

changes are to be made. Subsequent changes to the HTML style will need to be made on individual HTML elements. CSS offers an efficient way to style individual HTML elements by creating a style in one sheet that is applied to the individual elements of all HTML pages associated with the sheet. The project makes use of CSS3 which is the latest version of CSS. A summary of the advantages why CSS is used in the styling is given below:

- Styles are easier to read and maintain.
- CSS offers a central point of changes to the styling of multiple pages by editing a single file.
- CSS offers better control to the styling of page elements (Powers 2012, p. 3).
- The new modules introduced in CSS3 work very well with HTML5 to bring out the best page look.

JavaScript

JavaScript is chosen as the scripting language that will add interactivity or behavior to the app for the following reasons:

Extensive support: JavaScript is the primary script language of HTML5. It is also the most common client side scripting language that is supported in available browsers.

Simplicity: JavaScript does not require a special development environment for it to be executed. It is executed in web browsers which are present in virtually all devices. This makes it easy to write and implement.

Extensive Libraries: JavaScript has so many libraries that make it easier to develop and extend its capabilities. There are lots of pre-written libraries that provide simple solutions to many complex or common tasks written in JavaScript code (Sawyer Mc Farland, 2011, p. 18). Libraries such as Jquery greatly reduce the inconsistencies in JavaScript across browsers. Most of the web technologies have built JavaScript libraries that are integrated into developed applications to achieve a higher level task. An example is YUI JavaScript Library used to build sleek and interactive user interface for web apps.

5.2 Implemented Libraries and Frameworks.

A library is a group of data and programming code that is used to develop software programs and applications while a framework is a platform for developing software applications (Cory 2013; *Framework* 2013). This chapter section discusses the libraries and frameworks that were implemented to achieve the objective of the thesis project.

Jquery

Jquery is one of the most widely used JavaScript libraries and was the choice of this project for reasons below:

Compatibility: Jquery has been tested and recommended in the developer's community as the library that ensures consistent functionalities across browsers. Jquery also supports old and new browsers; therefore, more concentration is given to the functionality of the app and not inconsistencies in the browser.

Speed: The size of the library is relatively small which makes it work fast.

Easy to learn: There are a lot of materials and documentations on Jquery. The learning curve is really short because it is easy to understand and most developers are familiar with it, thereby making it easy to get assistance.

Lots of Plug-ins: There so many Jquery plug-ins available for free that make it easier to reuse nice features and effects already created. There is no need to re-invent the wheel. Some features and effects in this thesis project are courtesy of plug-ins.

Compatible with CSS3: It works pretty well with CSS3 styling sheet that is used in the helps to produce good user interface and design.

Twitter Bootstrap

Twitter Bootstrap is a framework that provides simple and flexible HTML, CSS, and JS for sleek UI components therefore, making the creation of websites and apps easier, faster and better in general(David 2013; Bootstrap 2013). Integrating Bootstrap into the project helped to save the large amount of time spent in styling the structure of the HTML. Bootstrap comes with plenty of readymade classes that can be used for design templates. Building the HTML selectors with the appropriate bootstrap classes reduces the amount of CSS styles that is written. For example the initial form that was used to add items into the shopping list was made without using the Bootstrap framework. It required a separate styling to structure the form as seen in Figure 11. The HTML source code can be found in Appendix 1. Although the form in Figure 11 looks almost like the form in Figure 12, it would be challenging and time consuming to style each and every component of the HTML structure to achieve the intended UI design. On the other hand, the form in Figure 12 looks nice with a cool button for the user to select an item as important. No additional styling was required because the Bootstrap readymade classes were sufficient to achieve the design of the form. The HTML source code can be found in Appendix 2.

Another advantage of using the Bootstrap framework is that the results produced are consistent across different platforms. Furthermore, Bootstrap is design-responsive. This means the UI design results easily adapts to the different screen sizes of the devices that will run the application.

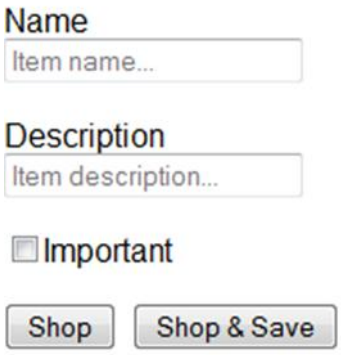

Figure 11. Form coded without bootstrap framework

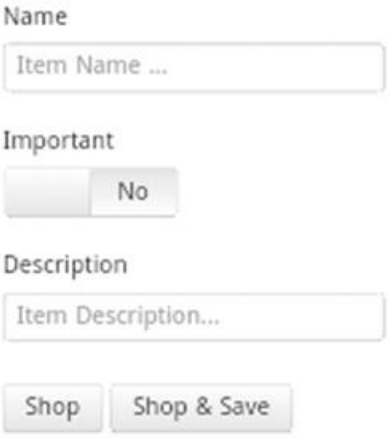

Figure 12. Form coded with bootstrap framework

Mustache.js

The Mustache.js library is a simple mustache template system that is implemented in JavaScript (Github 2013). Figure 13 shows the format in which all the items will be displayed. Each item in the list is distinguished by what attributes are set in the format such as name, description and checked buttons. By using Mustache, a template for the items is created in the HTML source code and we can render the variables or parameters of each item using one template. The source code of the item's template can be found in Appendix 3. The double curly braces ({{<parameter>}}) in the source code specify the parameter that is rendered in HTML from the JavaScript. Making use of

the Mustache template reduces the amount of HTML and JavaScript codes because there is no need to write codes to for each item displayed on the screen.

Figure 13. Displayed Item template

PhoneGap

PhoneGap, also referred to as Cordova, is a mobile development framework that enables a mobile app to be developed with HTML, CSS, and JavaScript. The sets of device API in PhoneGap allow a mobile developer to access native device features such as the camera or storage using JavaScript. JavaScript APIs are built on web standards that are consistent and widely supported across various device platforms (Apache Cordova 2013). The shopping list app developed in the thesis project is implemented using JavaScript APIs within the PhoneGap framework. As a result, the app can run across the mobile operating systems that are supported on PhoneGap as seen in Figure 14. However, the native features that can be implemented on a mobile device using PhoneGap are limited to the features that are supported by PhoneGaps' API. Figure 15 shows the API features that are currently supported by PhoneGap. It can be seen in Figure 15 that Notification (Alert) and Storage APIs are circled with a red line. The red line marks them as the core APIs that are implemented in the thesis project. They have to be widely supported in order to ensure a successful implementation of the app across multiple device platforms.

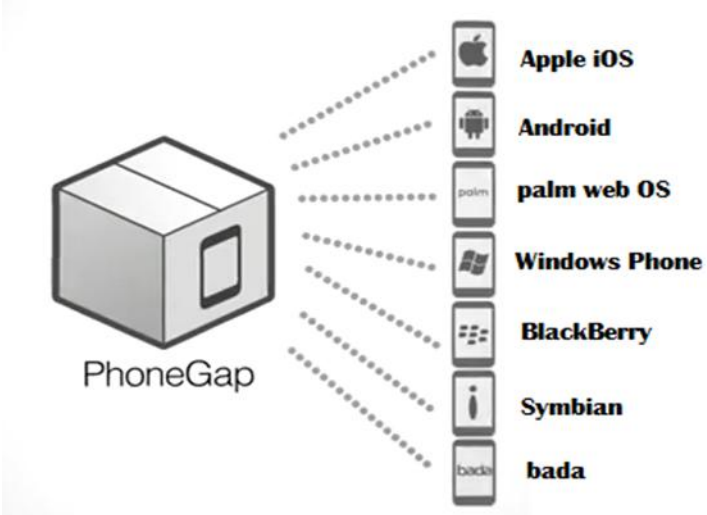

Figure 14. Mobile operating systems supported on PhoneGap framework.

	iPhone 3G	iPhone 3GS & newer	Android	BlackBerry OS 5x	BlackBerry OS 6.0+	WebOS	Windows Phone7 + 8	Symbian	Bada
Accelerometer	✓	✓	✓	✓	✓	✓	✓	✓	✓
Camera	✓	✓	✓	✓	✓	✓	✓	✓	✓
Compass	X	✓	✓	X	X	✓	✓	X	✓
Contacts	✓	✓	✓	✓	✓	X	✓	✓	✓
File	✓	✓	✓	✓	✓	X	✓	X	X
Geolocation	✓	✓	✓	✓	✓	✓	✓	✓	✓
Media	✓	✓	✓	X	X	X	✓	X	X
Network	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Alert)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Sound)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Vibration)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Storage	✓	✓	✓	✓	✓	✓	✓	✓	X

Figure 15. Supported features on PhoneGap (PhoneGap 2013).

There are other open source mobile development framework such as Rhodes, Sencha Touch, Appcelerator Titanium and MoSync that are available for free. However, the thesis project is implemented in PhoneGap because of its high level of popularity and support among the developers community

Implementing within the PhoneGap framework also allows the source codes to be easily compiled and deployed. The source code of the app is simply uploaded in a single zip file into a cloud service called PhoneGap Build. PhoneGap Build then generates a native app for each platform that it supports. Prior to September 2012, a PhoneGap app was compiled on each native SDK platform of the targeted device. For example, an app that is targeted to run on iOS, Android and Windows phone would require XCode, Eclipse and Visual Studio respectively. Furthermore, developing an iOS app requires an Apple computer and a Windows phone a Windows computer. Therefore, using PhoneGap Build reduced the resources and complexities that would have been required in implementing the thesis project. An overview of deploying an app with PhoneGap Build is seen in Figure 16.

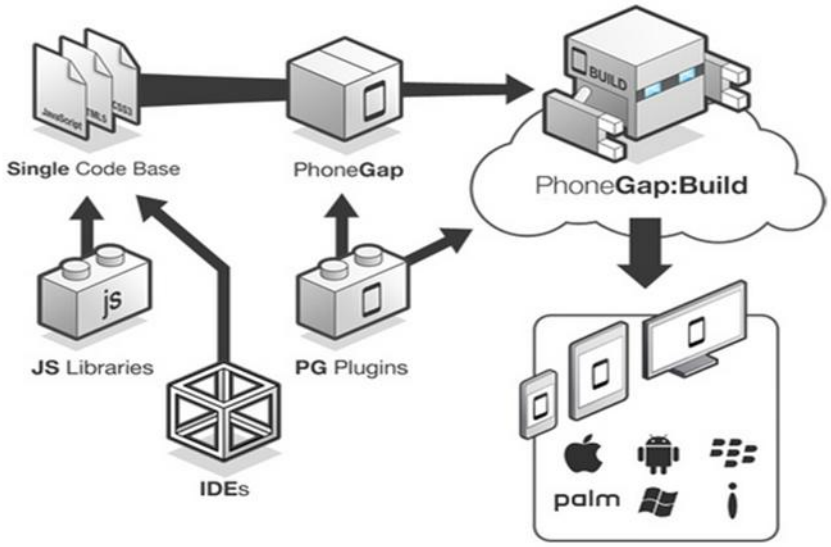

Figure 16. PhoneGap Build compiles and deploys mobile application (Pete 2012).

The thesis project started off with MoSync mobile development framework but it was later converted to a PhoneGap project. The choice to use MoSync initially was because it came with a tool called MoSync Reload. Mosync Reload enables a developer to test an app simultaneously on multiple devices. Hence, the developing and testing of an app is easier and faster than on any other mobile development framework. However, developing the app on MoSync was dropped in favor of PhoneGap because the author did not have the experience to deal with some bugs that caused the app to malfunction.

5.3 Implementation of Screen Views

Mobile devices come in varying screen sizes. It is essential that the screen view of the app adapts to the screen size of the user's mobile device. Thus, the app can have same look irrespective of the screen size. To address this problem, the statement <meta name="viewport" content="width=device-width, initial-scale=1.0"> is included within the head tag of the index.html file.

The design chapter detailed the components of the Item list screen view and Shopping list screen view of the app. The reader should notice we refer to them as screen views and not pages. Screen view is the appropriate word that conveys the implementation of what the user refers to as a page in the context of the mobile app. In the context of the thesis project, it implies that only some parts of the index.html page are presented on the screen for the user to see. All the parts of the two screen views are defined within div tags in the index.html. A JavaScript function is defined to display the parts that make up the active screen view and hide the rest parts of the index.html.

Hence, we are able to reuse an HTML code by referring to the id of its tag and thereby reduce the amount of HTML code required to run the app. The source code for handling the change in screen view can be found in Appendix 1.

The icon images of the Add item, Delete button and Sort button were sourced from the Twitter-bootstrap library. The Bootstrap library made it easy to include these images by including the class name of the image that is referred to within an HTML tag.

5.4 Storage Implementation

The data of the application is stored locally on the mobile device. The PhoneGap StorageAPI facilitates the storing of data locally on a device that supports HTML5 web browser. The storage API is one of the new APIs that is supported by HTML5 but not included in its specification. Prior to the introduction of local storage API, cookies were the only means to store data locally on a device. However, cookies were allowed to store data of maximum 4KB. Unlike cookies, the local storage API can store persistent data to a limit of 5MB or 10MB on a device. Hence, the author leveraged the Storage API in the implementation of data storage.

PhoneGap documentation specifies that the Storage API supports the implementation of database, SQL objects and Local Storage on the following platforms (PhoneGap Documentation 2013):

- Android
- BlackBerry WebWorks (OS 6.0 and higher)
- iPhone
- webOS
- Tizen

This would have enabled a database implementation to manage the list of items in the shopping list app. However, database and SQL objects are not supported on Windows Phone platform. Since the project resources were limited to Android and Windows Phone platform, implementing a database and SQL transactions were dropped in favor of Local Storage object implementation. PhoneGap documentation specifies that the Storage API supports the implementation of Local Storage in the platforms listed above as well as Windows Phone 7 & 8.

It is important to note that the Local Storage object can only store data in form of a string. Since the mobile app data that is stored involves storing an item's data as an object. Hence the JSON data interchange format is used to convert the item's object data into a string before it is stored. Then

the stored string is parsed from a string to an object when retrieving the item's object data. The source code for storing and retrieving data can be found in Appendix 4.

5.5 Adding Item to Shopping List

In line with the app design, a user first adds an item to the shopping list using the Add Item button. The implementation prevents an item with same item name or null value to be entered into the shopping list. Item names are case sensitive. The user is then alerted if the item name already exists or is null as seen in Figure 7 and 8. If these criteria are met, an object of the item that is stored in the local storage is created. It also creates an object which holds the attributes to render the item details on the screen. These attributes are plugged into the HTML item template and then displayed to the user on the screen. Each created object has a unique ID. Hence, a specific item can be queried with Jquery by referring to the ID. The detailed source code on the item object and adding an item with the add button can be found in Appendix 5.

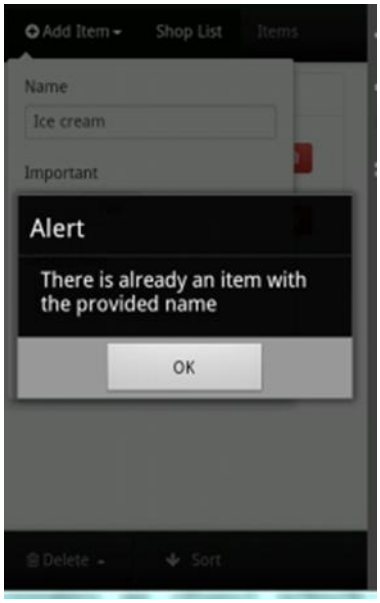

Figure 17. Notification of existing item name.

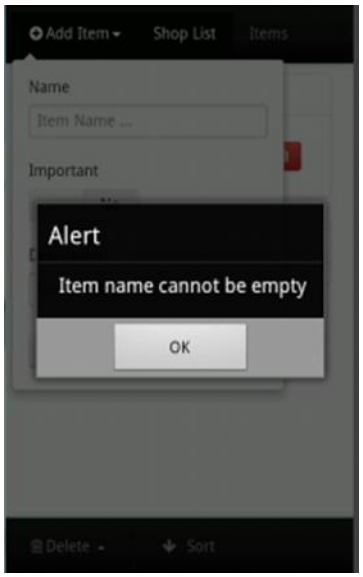

Figure 18. Notification of null item name.

When a user orders an item from the items list into the shopping list, the item in the items list becomes a parent to its newly created copy in the shopping list. The newly created copy of item in the shopping list is referred to as a child item. If the parent item was marked as important, the child item appears on top the List of unbought items in the shopping list. As discussed in the design, the check button used to order an item allows just one order to be placed on an item. At first, this design feature felt sufficient to prevent the user to order more than one item. However, after it was implemented, it was noticed that a user can create more than one child item from one parent item. This means that more than one same item can exist in the shopping list. This can be done by unchecking and checking the order button. This behavior is not consistent with the design to have unique items in the shopping list. Therefore, an additional feature was added. In the event that the user unchecks the order button, it means the user wants to remove the item from the shopping list. Hence, the item is deleted from the shopping list. This behavior was implemented in order to preserve the consistence of having unique items in a shopping list. A user may wish to change the importance priority placed on an item after it has been ordered. If the change is made on the child item, it automatically reflects on the parent item and vice-versa.

5.6 Editing, Deleting, and Sorting Items

Editing Items

A user can edit an item by tapping the item name and description area. This action triggers an event that brings focus to the item name and description which allows the item to be edited. If there is no editing activity, the triggered event assumes the tap was not intentional, thus, the focus

expires and the name and description remains unchanged. This implementation is consistent with items in the shopping list and items list.

Deleting Items

The shopping list app has two methods of deleting an item. The first method of deleting an item in the app is through the direct access delete icon that is associated with every existing item. The second method is a secondary action of a deleting group of items as mentioned in the design chapter.

Deleting Items with the direct access delete icon

It is essential to implement a defensive design in order to prevent a user from accidentally deleting an item. During the requirement analysis, it was noticed that many users did not like the approach of displaying an alert before they could delete one item. The problem with using an alert for individual items is that it interferes with the user interaction with the app. The user cannot perform any other action until the alert is closed. It also feels like a burden when a user has to deal with each alert associated to a delete action. On the other hand, some apps enable a user to turn off an alert notification when deleting an item. Although this approach eliminates any interference to a user's experience, the defensive design offered by an alert notification is removed.

To solve this potential problem, the implemented delete action requires the user to tap the same delete icon two consecutive times within a period of two seconds. The first tap on the delete icon triggers the background of the item to be highlighted as red. The change in item background-color alerts the user to the delete action without interrupting the user's interaction with the app. Hence, the user is aware that the delete icon was tapped accidentally and will avoid the second tapping. If there is no second consecutive tap within two seconds, the red background-color of the item reverses to the previous background-color. The source code of deleting an item using the delete icon can be found in Appendix 6

Deleting Items with the secondary action delete button

The delete button at the bottom of the shopping list screen enables a user to delete any of the following: all bought items; all unbought items; all items in the shopping list. For example, after a user has finished shopping and wishes to clear the shopping list, the delete all items comes handy to clear the shopping list.

The defensive design implemented with the secondary action delete button takes a different approach to that of deleting an item with a delete icon. This time, it is necessary to alert the user with a notification alert because each delete action deletes a collection of items as seen in Figure 19.

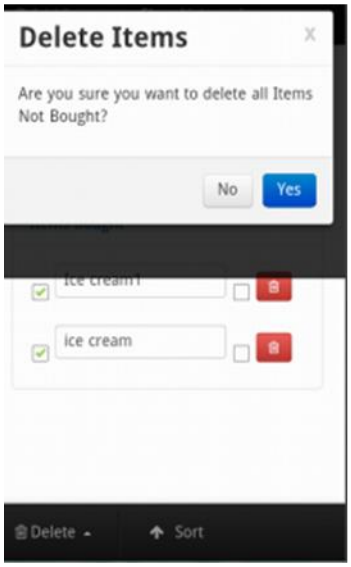

Figure 19 a.

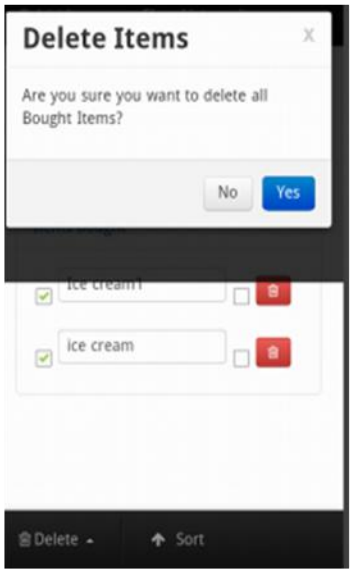

Figure 19 b.

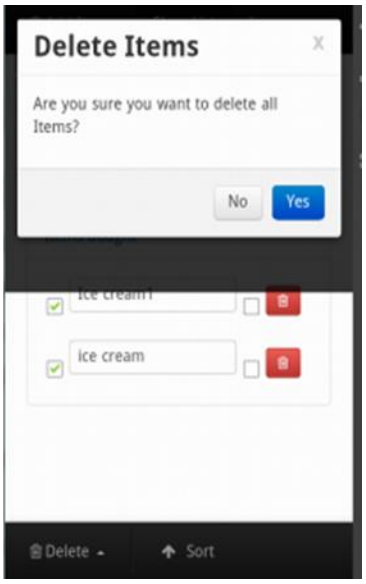

Figure 19 c.

Figure 19. Deleting a collection of items using the secondary delete button.

Sort Items

By default, items are logged into the various lists of items by the order in which they are added to the list. For example, the latest item added as an important item under the Items not bought list appears at the top of the important items. It does not matter if it was added from the Add Item button or the Items list. Further review of the user feedback on the apps that were analyzed revealed that some users prefer their items to be logged into a list without any arrangement. There was some contradictory feedback that appreciated that they could sort their items alphabetically in any order. Hence, the author added the sorting feature. With the sort button, a user is able to sort items within a list of items in ascending or descending alphabetical order. The sort button can also change the list of items to the default order before they were sorted.

6 TESTING AND RESULT

The application was tested based on the requirements and overall user interface design. The result of the test showed that the project performed well and the requirements and design specification were met. However, more improvements can be made and will be discussed in the future work section of the Conclusion chapter.

Due to the limited available resources, the testing was only carried out on Android and Windows Phone mobile operating system platforms. The application file for each platform generated by

PhoneGap Build was installed on an emulator (virtual device) and a real device as specified in Table 4. Set up and installation of the application was faster on an emulator than on a real device for both Android and Windows application. The set up for testing an application on a Windows phone device was the longest. It required the device to be registered with Microsoft Development Center as well as an annual subscription fee. Fortunately, the fee was waived because the author was a DreamSpark student at the time the Windows Phone was registered. Using an emulator enabled the test to be carried out on Android 2.3.3 to Android 4.2.2 versions. The screen shots used in this thesis writing were captured from the emulators.

Emulators also have their limitations. They work very slowly and their performance may not represent the true performance of an app on a device. On the other hand, testing the app on a mobile device produces reliable results in terms of performance and usability. However, only two mobile devices were available for this thesis project which is fairly limited.

Although the app was also tested on desktop browsers, it is specifically designed to run on mobile device platforms. The nature of the app makes it suitable for just mobile devices. Testing the app on a desktop browser revealed that the alert notifications did not work. It was later discovered that the method for defining an alert function on a mobile device platform differs from that of a desktop browser. Nevertheless, few lines of code were added to the source code to apply the right method of notification alert if the platform running the app is a desktop browser.

Table 4. Testing on different devices and platforms

Emulator	**Platform Version**
Android SDK	versions 2.3.3 to version 4.2.2
Windows Phone	Windows Phone 7
Device	
Nokia Lumia 800	Windows Phone 7
Samsung Galaxy Gio	Androids 2.2

7 CONCLUSION

7.1 Discussion

Developing a cross-platform mobile application based on HTML5, JavaScript and CSS saves time and resources. A developer is not limited to one mobile operating system platform due to lack of knowledge of the native interface language of the platform. By using a mobile development framework such as PhoneGap, MoSync and SenchTouch, a native application that runs on various mobile device platforms was built without the use of Java, C# and other native coding languages.

The requirements and design of the thesis project has distinct features in comparison with the apps that were analyzed from two app stores. This validates requirement analysis as an effective process for producing a unique or well modified software design.

The thesis project achieved its goal and met the set requirements of the developed cross-platform mobile application. However, some challenges and limitations were encountered. Despite the right decision to migrate to a reliable cross-platform mobile development framework PhoneGap, there exist certain limitations. Some features of a device can only be accessed through the device's native language. Hence, this limits what feature can be implemented on a cross-platform app except a developer understands the native interface language of the targeted mobile device platform. One limitations encountered in the project was, it avoided implementing a database because PhoneGap does not support the feature on Windows Phone as mentioned in the implementation chapter. Nevertheless Windows Phone has a database feature that can be implemented its own native language C#.

The author recommends mobile application developers to assess what choice of development framework is more suitable for each cross-platform mobile application project.

7.2 Future Work

More features can be added to improve the thesis project.

- Improve the native look and feel of the UI: The look of the feel of the UI is slightly inferior to that developed on a native mobile development framework. Cross-platform mobile applications are often criticized for this reason. However, it can be improved by experimenting with several front end frameworks to determine which is most suitable for the app.

- Add a feature that allows users to share shopping list with someone else: This feature requires the app to be registered in the app store for it to work. It would allow a user to send a shopping list to another person as a text message or email. For example, a user can send the shopping list to his/her spouse.

The author hopes to reach out to users that will test and give feedback on the user experience of the project. Then more improvements can be made to the project in the future based on this feedback. The scope of the testing should also be expanded to other mobile OS platforms supported by PhoneGap aside Windows Phone and Androids. More resources will be required to accomplish this task, such as different mobile devices for each platform and developers' registration fee for some platforms. Nevertheless, submitting an app to any of the app stores requires testing on its mobile OS platform.

REFERENCES

Apache Cordova 2013, *About Apache Cordova,* Apache Software Foundation, accessed 11 May 2013, <http://cordova.apache.org/>

Bootstrap 2013, *Twitter bootstrap,* Twitter, accessed 10 May, 2013, <https://twitter.com/twbootstrap>.

Cory, J 2013, *Software library,* Janalta interactive, *accessed 14 May 2013,* <http://www.techopedia.com/definition/3828/software-library>.

David, N 2013, 11 Reasons to use twitter bootstrap, SitePoint, accessed 10 May 2013 <http://www.sitepoint.com/11-reasons-to-use-twitter-bootstrap>.

Eran, Z 2012, *Native, HTML5, and Hybrid Mobile App Development: Real-Life Experiences,* online video, accessed on 16 May 2013, <http://www.youtube.com/watch?v=We0byPckthQ>.

Framework 2013*, Techterms.com,* accessed 14 May 2013, <http://www.techterms.com/definition/framework>.

Gartner press release 2013, Gartner Says Worldwide PC, Tablet and Mobile Phone Combined Shipments to Reach *2.4 Billion Units in 2013,* Gartner, accessed 12 April 2013, <http://www.gartner.com/newsroom/id/2408515>.

Github 2013, *mustache.js - Logic-less {{mustache}} templates with JavaScript,* Github inc., accessed 11 May 2013, <https://github.com/janl/mustache.js/#readme>.

IDC Press Release 2013, International Data Corporation, accessed 12 April 2013, <http://www.idc.com/getdoc.jsp?containerId=prUS23946013>.

Korf, M and Oksman, E 2013, *Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options,* accessed 18 May 2013, <http://wiki.developerforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options>.

Kyrnin, J 2012, *Teach Yourself HTML5 Mobile Application Development in 24 Hours,* Pearson Education, Indiana.

La Counte, S 2011, Going Mobile: Developing Apps for Your Library Using Basic HTML Programming, American Library Associations, Chicago.

Microsoft msdn 2013, *What is an API,* Microsoft, accessed 20 April 2013, <http://msdn.microsoft.com/en-us/library/office/aa141380%28v=office.10%29.aspx>.

Miles, R and Hamilton, K 2006, Learning UML2.0, O'Reilly.

Pete 2012, *What's So Good About Phone Gap,* accessed 11 May 2013*,* <http://www.blue-leaf.co.uk/what-is-phonegap>.

PhoneGap 2013, *Supported Features,* Adobe Systems, accessed 13 May 2013, <http://phonegap.com/about/feature/>.

PhoneGap Documentation 2013, *Storage*, Adobe Systems, accessed 13 May 2013, <http://docs.phonegap.com/en/2.7.0/cordova_storage_storage.md.html#Storage>.

Powers, D 2012, *Beginning CSS3*, Apress, Berkley.

Pro-Logix 2011, *Get defensive: Defensive web design and how will it help your website,* ProLogix technologies, accessed 30 April 2013, <http://www.pro-logix.net/blog/web-design/get-defensive-defensive-web-design-and-how-will-it-help-your-website>.

Requirements analysis 2010*,* The University of Architecture, Civil Engineering and Geodesy Bulgaria, accessed 26 April 2013, <http://www.uacg.bg/filebank/acadstaff/userfiles/publ_bg_397_SDP_activities_and_steps.pdf>.

Rouse, M 2007, requirements analysis (requirements engineering), SearchSoftwareQuality, accessed 26 April 2013 <http://searchsoftwarequality.techtarget.com/definition/requirements-analysis>.

Sawyer Mc Farland, D 2011, *Javascript & Jquery: The Missing Manual*, 2nd edn, O'Reilly Media, California.

Smith, M 2013, *What is HTML?* W3C*,* accessed 20 April 2013, <http://www.w3.org/html/>.

Appendices

Appendix 1: HTML code for form without twitter bootstrap

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<style>
body {
margin: 0;
width: 100%;
height: 100%;
padding: 0;
font-family: Arial, Helvetica, sans-serif;
color: #000;
background-color: #FFF;
}

label {
display: inline-block;
width: 50px;

}
.button{
display:block;
float:right;
width: 40px;
background-color: #DC17E8;
border-radius: 5px;
margin:10px;
}
</style>
</head>
|
|//missing tags
|
<body>
```

```
<form id="item-input">
    <p>
        <label for="item" >Name</label></br>
            <input type="text"  id="add_item" placeholder="Item name...">
    </p>
    <p>
        <label for="Description">Description</label></br>
            <input type="text"  id="item-description" placeholder="Item description...">
    </p>
    <p>
        <input type="checkbox" id="item-important" value="important">Important
    </p>
    </p>
        <input type="submit"  value="Shop" id='btn-add-item-to-shopping-list'> 
        <input type="submit"  id='btn-add-item-to-items-list' value="Shop & Save">
    </p>
</form>
```

```
                |
                |               //missing tags
                |
</body>
</html>
```

Appendix 2. HTML code for form with twitter bootstrap

```
<!doctype html>
<html           lang="en">
<head>
                <meta charset="utf-8">
            <meta name="viewport" content="width=device-width, initial-scale=1.0">
                                <link           href="css/bootstrap.css"       rel="stylesheet">
                                <link           href="css/bootstrap-responsive.css"          rel="stylesheet">
                                <link href="css/bootstrapSwitch.css" rel="stylesheet">
</head>
<body>
                |
                |               //missing tags
                |
```

```
<form class="form-horizontal" id="item-input">
    <div class="control-group">
        <label class="control-label" for="item-name">Name</label>
        <div class="controls">
            <input type="text" id="item-name" name="item-name"
                placeholder="Item Name ...">
        </div>
    </div>
    <div class="control-group">
        <label class="control-label" for="item-description">Important</label>
        <div class="controls switch" data-on-label="Yes"
            data-off-label="No">
            <input type="checkbox" id="item-important">
        </div>
    </div>

    <div class="control-group">
        <label class="control-label" for="item-description">Description</label>
        <div class="controls">
            <input type="text" id="item-description"
                name="item-description" placeholder="Item Description...">
        </div>
    </div>
    <div class="control-group">
        <div class="controls">
            <a href="#" class='btn' id='btn-add-item-to-shopping-list'>Shop</a>
            <a href="#" class='btn' id='btn-add-item-to-items-list'>Shop
                & Save</a>
        </div>
    </div>
</form>
```

| //missing tags
|
| //missing tags
|
</body>
</html>

Appendix 3: Item template derived from mustache

```
<div class="item-template" id="item-template">
    <li id="{{id}}" class="item-element" data-item-status="{{itemStatus}}"
        data-name="{{name}}" data-description="{{description}}"
        data-parent-element="{{parentElement}}" data-parent-id="{{parentID}}"
        data-is-template={{isTemplate}}>
        <ul>
            <li class="item-data-first item-data"><input type="checkbox"
                class="btn-shop" data-parent-id={{id}}></li>
            <li class="item-data"><input type="text"
                value="{{name}} {{description}}" class="input-editable"
                data-parent-id={{id}} /> <span id='item-display-name-{{id}}'
                style="display: none;">{{name}}</span> <span
                id='item-display-description-{{id}}' style="display: none;">{{description}}</span>
            </li>
            <li class="item-data"><input type="checkbox"
                class="btn-important" {{important}} data-parent-id={{id}}>
            <li class="item-data-last item-data"><a href="#"
                class="btn btn-danger btn-small btn-delete" data-parent-id="{{id}}"><i
                    class="icon-trash icon-white"></i> </a></li>
        </ul></li>
</div>
<!-- /item-template -->
```

Appendix 4: Data storage

```
/**
 * Get local storage entry holding items. Create one if it does not exist.
 *
 * @returns The Local storage entry holding all items
 */
function getStorage() {
    if (window.localStorage.getItem(ITEMS_KEY) == null)
        window.localStorage.setItem(ITEMS_KEY, "{}");

    return JSON.parse(window.localStorage.getItem(ITEMS_KEY));
}

/**
 * Update local storage entry with new data
 *
 * @param storage
 *            The new data to be saved.
 */
function updateStorage(storage) {
    window.localStorage.setItem(ITEMS_KEY, JSON.stringify(storage));
}
```

```
/**
 * Save an item in the Local Storage.
 *
 * @param id: The item ID
 * @param parentElementID: The DOM element ID where this item is displayed
 * @param name: The Item name
 * @param important: The item importance
 * @param description: The item description
 * @param parentID : The parent item ID (not to be confused with paremenElement ID).
 *           This will be set only if the item was created from a template item.
 * @param itemStatus: The item status (possible values are bought AND not-bought)
 * @param isTemplate: If true this item is a template item.
 */
function persistItem(id, parentElementID, name, important, description, parentID, itemStatus, isTemplate) {
    // Get the storage and build the JS object to persist the item
    var items = getStorage();
    items[id] = {
        'id' : id,
        'parentElementID' : parentElementID,
        'name' : name,
        'description' : description,
        'important' : important,
        'parentID' : parentID,
        'status' : itemStatus,
        'isTemplate' : isTemplate
    };
    updateStorage(items);
}
```

Appendix 5: Add and save item

```
// Add item to the shopping list using the Add Item Button.
$('#btn-add-item-to-shopping-list').on(eventName, function() {
    var name = $('#item-name').val().trim();

    if (name.length == 0) {
        navigator.notification.alert("Item name cannot be empty", function() {
        });
        return false;
    }

    var important = $('#item-important').is(':checked');
    var description = $('#item-description').val();
    var id = new Date().getTime();

    if (itemExists(name)) {
        navigator.notification.alert("There is already an item with the provided name", function() {
        });
        return false;
    }

    storeItemInList(id, '#shoplist-item-list', name, important, description, null, 'not-bought', 'no');
    persistItem(id, '#shoplist-item-list', name, important, description, null, 'not-bought', 'no');

    // Clear form values
    $('#item-name').val('');
    $('#item-description').val('');

    // Hide the "Add Item" form.
    $('#add-item-toggle').dropdown('toggle');

    return false;
});
```

Appendix 6: Item deleted with primary delete button

```
$('body').on(eventName, '.btn-delete', function() {
    var id = $(this).data('parent-id');
    var item = $('#' + id + " ul");
    var currentTap = new Date().getTime();

    if (lastTap != -1 && (currentTap - lastTap) < tapThreshold) {
        // The use has tapped twice the delete button. See if it's the same
        // item and if it's then delete it.
        if (id == currentItemID) {
            clearTimeout(unhightlightDeleteElement);
            item.removeClass('highlight-delete');
            lastTap = -1;
            currentItemID = -1;

            if (item.data('parent-id') != undefined) {
                var parentItem = $('#' + item.data('parent-id'));
            }
            deleteItem($('#' + id));
        } else {
            // For now do not allow the user to delete if an item is
            // already hightlighted
        }
    } else {
        // This is the first tap. We just need to highlight the element, and
        // unhighlight the previous one (if any).
        item.addClass('highlight-delete');
        currentItemID = id;
        lastTap = new Date().getTime();
        if (currentItemID != -1)
            clearTimeout(unhightlightDeleteElement);
        setTimeout(unhightlightDeleteElement, tapThreshold);
    }
});
```

Appendix 7: Handle screen view change

```
$(document).ready(function() {

    // Handle view change. This function activates. the items screen view
    $("a[href='#items']").on(eventName, function() {
        $("a[href='#shoplist']").parent().removeClass('active');
        $("#shoplist").hide();
        $("a[href='#items']").parent().addClass('active');
        $("#items").show();
        $('#remove-items').hide();
    });

    // Handle view change. This function activates the shopping list screen view.
    $("a[href='#shoplist']").on(eventName, function() {
        $("a[href='#items']").parent().removeClass('active');
        $("#items").hide();
        $("a[href='#shoplist']").parent().addClass('active');
        $("#shoplist").show();
        $('#remove-items').show();
    });
    |
    |
    |
    <missing code>
```