THE STUDY OF WEB APPLICATION DEVELOPMENT VERSUS
NATIVE APPLICATION DEVELOPMENT ON IPHONE

A Thesis
Presented to the
Faculty of
San Diego State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Sunee Waleetorncheepsawat
Fall 2010
SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Sunee Waleetorncheepsawat:

The Study of Web Application Development Versus Native Application Development on iPhone

Roger Whitney, Chair
Department of Computer Science

Kris Stewart
Department of Computer Science

Zohir Chowdhury
Graduate School of Public Health

09/07/2010
Approval Date
Copyright © 2010

by

Sunee Waleetorncheepsawat

All Rights Reserved
DEDICATION

This thesis is dedicated to my family and friends who always support me.
ABSTRACT OF THE THESIS

The Study of Web Application Development Versus Native Application Development on iPhone
by
Sunee Waleetorncheepsawat
Master of Science in Computer Science
San Diego State University, 2010

Applications for smart phones are growing in an importance. However there are different mobile platforms. Developing a native app for each mobile platform is expensive. So people promoting developing web app for mobile phone. But also some issues with web applications: performance, accessing the device hardware, and access to full set of UI widgets of the platform. We compare web applications versus native applications on the iPhone. We see how close web applications are to native applications based on the access to the native widgets, data access, hardware access, and performance.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>TABLE OF CONTENTS</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>viii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>ix</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>xi</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2 IPHONE BACKGROUND</td>
<td>4</td>
</tr>
<tr>
<td>2.1 Hardware</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Software</td>
<td>6</td>
</tr>
<tr>
<td>2.3 The Limitation of the iPhone OS Platform</td>
<td>9</td>
</tr>
<tr>
<td>2.4 Types of iPhone Application</td>
<td>13</td>
</tr>
<tr>
<td>3 NEW APPROACHES FOR WEB DEVELOPMENT</td>
<td>15</td>
</tr>
<tr>
<td>3.1 HTML 5</td>
<td>15</td>
</tr>
<tr>
<td>3.1.1 Geolocation Service</td>
<td>16</td>
</tr>
<tr>
<td>3.1.2 Client-side Data Storage</td>
<td>17</td>
</tr>
<tr>
<td>3.1.2.1 Local Storage and Session Storage</td>
<td>17</td>
</tr>
<tr>
<td>3.1.2.2 Client-Side Database</td>
<td>17</td>
</tr>
<tr>
<td>3.1.3 Offline Application Cache</td>
<td>18</td>
</tr>
<tr>
<td>3.1.4 Scalable Vector Graphics (SVG)</td>
<td>19</td>
</tr>
<tr>
<td>3.1.5 Canvas</td>
<td>19</td>
</tr>
<tr>
<td>3.1.6 SVG vs Canvas</td>
<td>21</td>
</tr>
<tr>
<td>3.2 Cascading Style Sheet</td>
<td>21</td>
</tr>
<tr>
<td>4 DEVICE FUNCTIONALITY ACCESSIBILITY</td>
<td>24</td>
</tr>
<tr>
<td>5 IPHONE WEB DEVELOPMENT FRAMEWORKS</td>
<td>27</td>
</tr>
<tr>
<td>5.1 iUI</td>
<td>34</td>
</tr>
<tr>
<td>5.2 iWebKit</td>
<td>39</td>
</tr>
<tr>
<td>5.3 jQTouch</td>
<td>42</td>
</tr>
</tbody>
</table>
5.4 Conclusion ...............................................................................................................43

6 PERFORMANCE TEST .............................................................................................46
   6.1 Launch Time ....................................................................................................47
   6.2 Execution Time ...............................................................................................52
   6.3 Database Access Time .....................................................................................55
       6.3.1 Online Data Access Time ....................................................................55
       6.3.2 Offline Data Access Time ......................................................................58
       6.3.3 Online Data Access and Save to Local Database ...............................59

7 SUMMARY .................................................................................................................63

BIBLIOGRAPHY ....................................................................................................................66
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2.1</td>
<td>iPhone Gestures</td>
<td>10</td>
</tr>
<tr>
<td>Table 3.1</td>
<td>Functions for LocalStorage and SessionStorage</td>
<td>17</td>
</tr>
<tr>
<td>Table 3.2</td>
<td>Functions to Open and Manipulate Database</td>
<td>18</td>
</tr>
<tr>
<td>Table 3.3</td>
<td>CSS Style Properties</td>
<td>23</td>
</tr>
<tr>
<td>Table 4.1</td>
<td>Device Functionality and Accessibility from a Web App</td>
<td>25</td>
</tr>
<tr>
<td>Table 5.1</td>
<td>Open-Source iPhone Web Development Framework</td>
<td>32</td>
</tr>
<tr>
<td>Table 5.2</td>
<td>Features and Controls Supported in iUI, iWebKit, and jQTouch</td>
<td>34</td>
</tr>
<tr>
<td>Table 5.3</td>
<td>Size of Source Code for iUI, iWebKit, and jQTouch</td>
<td>44</td>
</tr>
<tr>
<td>Table 6.1</td>
<td>Hardware Specification for iPod Touch and iPhone</td>
<td>48</td>
</tr>
<tr>
<td>Table 6.2</td>
<td>Size of Tables in Expense Database</td>
<td>50</td>
</tr>
<tr>
<td>Table 6.3</td>
<td>Average Launch Time of Apps on iPhone and iPod Touch</td>
<td>52</td>
</tr>
<tr>
<td>Table 6.4</td>
<td>Summary of Execution Time Test</td>
<td>54</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 2.1. iPhone home screen showing a touch screen area.................................5
Figure 2.2. Safari browser on iPhone in portrait mode and landscape mode. ........7
Figure 2.3. Notification alert messages.................................................................8
Figure 2.4. Virtual keyboard on iPhone with a magnified view of the text.............11
Figure 2.5. Navigation control in the Settings application.................................12
Figure 3.1. Geolocation web application.............................................................16
Figure 3.2. Shapes and paths created from SVG script on Safari browser...........20
Figure 3.3. Calculator laid out with CSS.............................................................22
Figure 5.1. Edge-To-Edge list from iUI’s sample project.................................28
Figure 5.2. HTML form from UiUIKit’s sample project.................................28
Figure 5.3. HTML form from iWebKit’s sample project.................................29
Figure 5.4. Rounded-rectangle layout from WebApp.Net’s sample project.......30
Figure 5.5. Sliding page sample........................................................................30
Figure 5.6. Single-page navigation from iUI’s sample project..........................30
Figure 5.7. Comparing the toolbar when scrolling between iPhone User Guide and Digg web application.................................................................32
Figure 5.8. Expense native app’s listing tab........................................................33
Figure 5.9. Search dialog in iUI........................................................................36
Figure 5.10. Expense native app.....................................................................38
Figure 5.11. Expense web app created with iUI.............................................38
Figure 5.12. Music list from iWebKit’s sample project.......................................39
Figure 5.13. Look and behavior of form elements created with iWebKit...............41
Figure 5.14. Comparing iWebKit’s popup dialog and a native app action sheet...41
Figure 6.1. Expense application......................................................................49
Figure 6.2. Application launch time on iPhone device.....................................51
Figure 6.3. Applications launch time on iPod device........................................51
Figure 6.4. Execution time of switching segment from weekly to monthly on Expense native app on iPhone.................................................................53
Figure 6.5. Execution time of switching segment from weekly to monthly on Expense web app on iPhone.

Figure 6.6. Execution time of switching segment from weekly to monthly on Expense native app on iPod.

Figure 6.7. Execution time of switching segment from weekly to monthly on Expense web app on iPod.

Figure 6.8. Average of execution time of switching segment from weekly to monthly on Expense native app and web app on iPhone and iPod.

Figure 6.9. Online database access time from iPhone using Wi-Fi network.

Figure 6.10. Online database access time from iPhone using 3G network.

Figure 6.11. Online database access time from iPhone using EDGE network.

Figure 6.12. Online database access time from iPod using Wi-Fi network.

Figure 6.13. Offline database access time from iPhone.

Figure 6.14. Offline database access time from iPod.

Figure 6.15. Average of all database access times on iPhone and iPod.

Figure 6.16. Time to access database remotely and save to local database from iPhone using Wi-Fi network.

Figure 6.17. Time to access database remotely and save to local database from iPhone using 3G network.

Figure 6.18. Time to access database remotely and save to local database from iPhone, EDGE network.

Figure 6.19. Time to access database remotely and save to local database from iPod using Wi-Fi network.
ACKNOWLEDGEMENTS

I am very grateful to Dr. Whitney for all the help, support, guidance, and encouragement from the very first stage until the submission of the thesis. Many thanks must also go to Dr. Stewart and Dr. Zohir for being a part of my thesis committee.

Finally, the completion of this thesis might not have been possible without constant support, trust, and encouragement from my family and friends.
CHAPTER 1

INTRODUCTION

iPhone, the immensely popular cell phone, despite having only been out in the market for just two years, has already sold 33.75 million units worldwide by the end of fourth quarter, 2009 [1]. iPod Touch, MP3 player, which has the same ability to install applications like iPhone, also has already been sold for 20 million as of September 2009 [2]. Both of these devices enable users to easily download and install applications without having to connect a device to a computer. Users can choose to access the Apple app store from their device or from iTunes application on their computer machine. More importantly, the purchasing process requires no long form of entry fields that need to be entered. The total number of downloads has proved Steve Jobs’s statement that the app store reinvents the user experience with a mobile device and obtains user preference [3]. On September 28, 2009, Apple announced that more than 2 billion apps have been downloaded from the Apple app store and more than 85,000 apps were available for users in the store since its opening date of July 10, 2008 [3]. There were 125,000 developers in the iPhone development program [3].

Creating an iPhone app using the iPhone SDK is not a simple task. Only having knowledge of the SDK itself is not enough, developers also need to learn various tools and frameworks. They also require learning Objective-C, the programming language Apple chose for their applications’ development. Additionally, developers might need to have some knowledge of Cocoa programming, since it is the root of Cocoa Touch, the one used in iPhone development. This is not included the extra requirement, such as OpenGL or Quartz 2D for game development or Map kit for location base service applications. Developing a native app turns out to be an expensive route to start with.

Even if developers already are familiar with the language, developing a native app could also take time. One screen might require one to implement up to three files: one for a view itself, another one for a view controller, and the XIB file for User Interface (UI). In addition, a complex view can include multiple sub views that lead to more classes to
implement. For a rich UI, the background color alone might not be enough. Images may need to be added for sophisticated views or components.

SDK is not an only option for developing an application for iPhone. A web app is also available for developers. In fact, iPhone development originally started with only web app development. Developers only need to know HTML, CSS, and JavaScript to implement iPhone web apps. JavaScript is far more popular language than Objective-C based on langpop.com’s charts [4]. Therefore, web development should not take longer time of studying than Objective-C.

Developing a web page is simple and it can be done on any number of text editors. The ease of deployment has always been an advantage in web development. Many features and functions have been added into CSS3. As a result, creating a richer UI can be done easily by defining in CSS. Safari for iPhone OS 3.0 also already supports the HTML5. The emergence of HTML5 improves web apps’ user experience allowing web apps closer competing with native apps in the way that many developers might not have thought of.

The iPhone OS is not the only OS in the smart phone market. Among the others are Symbian, Rim Blackberry, Windows Mobile, Google Android, and Palm OS. The iPhone OS is ranked third in the global market share; whereas Symbian is ranked first [6]. If an application is intended to support many platforms, developing it for each OS using the particular SDK is expensive. Developing a web app has the additional benefit of software portability. With some small changes or no change at all, a web application can be ported to run on different platforms.

It certainly costs less to develop an application on the web than in a native application environment. However, based on the number of applications in the app store, developing a native app is still a popular choice.

This thesis focuses on study of the possibility that a web app can replace a native app. Three main issues that will be studied include hardware accessibility, user interface and design, and performance.

Hardware accessibility has always been the advantage of developing a native app over a web app. The SDK allows applications communicate with accessories that attached to the phone, such as Bluetooth, dock connector, or sensors. In this thesis, I will point out what a native app can do, and whether or not it can be achieved in a web app environment.
User interface is the important factor affecting user experience. UIKit has come with some view controllers that are mainly used on iPhone-preloaded applications. Apple suggests developers to use their templates. One of the reasons is that users have get accustomed to the design [5]. Failing to do so could result in poor user experience. Interface Builder, the tool that is used for User Interface design, also has most components available for developers. It also allows us to modify the appearance easily. Using Interface Builder and UIKit allow developers to ensure that the components’ appearance is standardized. I will point out tools and methods that make it possible to mimic the iPhone app look and feel in a web app. Users will then have a similar experience when using a web app.

In mobile environments, resources are expensive. Draining battery life is a behavior that all applications should avoid. Using less processing time is a high priority developers should consider of. Native applications run directly on CPU unlike web applications that run on a browser. The launch time of native apps should be faster than web apps if we only consider this issue. The same factor applies to executing time. A command that is called directly through the OS should be executed faster than a command that needs a browser to interpret before make a call to the OS. However, SunSpider’s benchmark reported that the beta version of Safari mobile 4.0 browser running on iPhone OS 3.0 was three times faster than the pervious version [6]. Two applications being developed for both environments with similar functionalities will be tested with regards to the performance issue.
CHAPTER 2

IPHONE BACKGROUND

In this chapter, I will introduce background information about iPhone development. Before one can build an application for iPhone, one needs to know the basic information about the device. Knowing hardware of a device, one will recognize its restriction, know what are available for an application to take advantage of, and know to design appropriately to gain user experience. Developing a native app requires developers to know the iPhone SDK that integrated with tools for building a native app. Many of the iPhone functionalities are accessible from a native app as well. Next, I will introduce constraints of the device that developers should consider when building an application. Lastly, three types of applications will be defined. The two of which I will focus on in this thesis are a native app and a web app.

2.1 HARDWARE

iPhone, the smart phone that combines three main functions together: a phone, an mp3 player, and a computing device. We can make a phone call, listen to music, and do various things with applications on the phone. The main interface connecting to users is its screen. The touch screen area is defined in Figure 2.1. Users control the device by touching the 480 x 320 pixel multi-touch screen with their fingers. Unlike the previous touch-screen devices, they usually come with a stylus, a sharp tip stick, providing us the more precise touch point in the screen. The iPhone app’s user interface is determined based on the fact that a fingertips is not as accurate as a stylus.

While current desktop and laptop computer processors are already multi-cores and exceed more than the 2 GHz speed barrier, the iPhone 3GS is a small device that comes with only a 600MHz processor and 256 MB of memory. Although the processor seems to be rather slow compared to computers, it is one of the faster processors of today’s smart phones [7].
Figure 2.1. iPhone home screen showing a touch screen area.

The iPhone is a multi-touch screen device. Its wide screen size is 480 x 320 pixels. It can be either 480 pixels wide on landscape mode or 480 pixels high on portrait mode, depending on the orientation. The iPhone 3G and 3GS supports Wi-Fi, 3G, and EDGE network. It will automatically switch to the best of the three networks that is available at the time, ensuring that users always get the fastest connection. With stereo Bluetooth, iPhone can now connect with stereo supported device. Additionally, in the latest generation of the iPhone, Bluetooth allows a phone to connect directly with another phone without having a network connection. Games can now use peer-to-peer connectivity to allow multiple players to compete with each other. iPhone 3GS comes with a built-in 3-Megapixel camera. The new autofocus camera allows shooting in either portrait or landscape mode. One can take a photo, then upload the photos to a server, and share them with his/her friends via MMS or e-mail, or save them in their photo library to view later. Users can also switch to record a 30-fps VGA video. Many applications have integrated photo-taking features into their app. It allows users to take pictures to use in that particular application. For example, the Facebook application
enables users to add a photo to their profile by either picking a photo from the Photo Gallery or taking a new photo. The phone embedded with GPS enabling a real-time tracking. GPS can find the current location accurately. Along with a built-in compass, the map application can match whichever direction users are heading.

iPhone comes with three sensors. One sensor used for power saving is a proximity sensor. As soon as the phone is placed next to the ear, it will turn off the display for battery saving and preventing from an accidently touch the screen when users are using a phone. iPhone uses the ambient light sensor to adjust the brightness of the display based on its current environment. Another sensor is the accelerometer. A 3-axis accelerometer can detect motions on any direction, which lets the phone know the device orientation. Therefore, it can switch the display between portrait mode and landscape mode based on the phone’s alignment, shown in Figure 2.2. Like in Photo Gallery, when users view photos, they can switch display mode to portrait or landscape by rotating the phone. Accelerometer also senses the shaking gesture, which many applications have taken advantage of using shaking for input to control certain behaviors of the application. For example, AllRecipe app, a recipe searching application, users shake to control the slot. When users shake a phone, the app will spin the ingredient slots and return the random result of 3 ingredients. Another sample is AntiStress Painter, the application that allows users to paint their feelings. It also allows user to easily the paint by shaking the phone. In addition to shaking gesture, some games use tilt-and-turn the device as an input when moving an object.

2.2 SOFTWARE

iPhone SDK provided by Apple is for developers to develop applications for iPhones. Tools that come along with the SDK consist of xCode, Interface Builder, iPhone Simulator. Developers need to know how to use these tools to build a native app. xCode is an Integrated Development Environment. It is used for debugging, building, and running an app in a simulator. Interface Builder is used for designing applications’ interface. It has all the controls that support in iPhone SDK, for example: text field, label, view, etc. The designed interface is saved in a nib file, which is then merged into application binary file by a compiler at a compile time. iPhone simulator simulates iPhone’s behavior. Once the ‘Build & Go’
button is hit in xCode, the compiler launches the application into the simulator. The functionality. However, some features are not supported in the simulator. For example, the accelerometer is not available there. Developers will need to test on a real device only if their application require motion sensation feature. Not only can the simulator be used to test a native app, it can also be used to test a web app. Safari in the simulator works the same as the real device. Users can enter a URL, pinch to zoom in or zoom out using alt key, as well as, checking on landscape or portrait mode. The GUI of any web sites looks identical to the result launching in an iPhone device. In addition to that, using the simulator, developers to test behavior of the application running on different versions of OS too.

The new iPhone SDK for iPhone OS 3.0 is the upgraded version that supports functionality of the new OS. The developers have gained a number of accesses to API added in the new OS. Among all the features, the important ones are Push Notification Service, Peer-to-Peer Connectivity, Maps, iPod Library Access, Open GL ES 2.0, Video Recording, and Compass [8].

Push notification service is provided to developers in order to reduce the previous version lack of allowing background processes running. This allows a chat application, for example, to be able to send a message to users when a new message is coming even when the application is not running. Because of battery life concern, Apple continues to disallow more than one third-party applications running at the same time. Instead, Apple gives applications...
opportunity to alert users about new data waiting for them. The service is provided through their Push Notification Server. The notification server will maintain connection to a device. When an application is not running on a device, and its server has a new data arrive, it will then send a notification to a notification server, which is then being passed to the device. Since the processes are running on a server instead of on device, the limited power of a device will not be impacted by a number of applications sending a notification. On the other hand, if applications were allowed to run in the background, many of them could run at the same time and that could drain the battery. Applications have three options to sending a notification: alert message—an alert dialog box showing some message, a numbered badge on home screen icon showing the number of new information arrived, and a sound alert notifying of a new coming data. In Figure 2.3, the left side shows two application icons with a badge number in the corner. The right side shows an alert message of Textfree application, when a message arrives when a user is in another application.

Figure 2.3. Notification alert messages.
With Game Kit Framework, developers are introduced to a new communication network between nearby devices. When the distance is short enough that devices can communicate via Bluetooth, the Peer-to-Peer connectivity becomes another choice that allows developers to enhance a game battle. It allows users to compete one-on-one with another user, or multi-users. An application establishes a session to communicate with other devices. A device can be served as a server, a client, or both.

Map Kit framework, which is available in the new SDK, makes it easy to add a map into an application. The framework uses Google Map Services to provide us map data information. Pinning down a location in a map becomes possible. The framework also provides developers a way to customize a pin, and call out popup. With Geocoding service, developers can retrieve the address from a given coordinate.

Accessing iPod library using Media Player Framework enables applications to access users’ music library. The developers can add a feature to play users’ songs in the background. They can also retrieve information about a current playing song, such as title, album, artist, and genre, etc. Media Player Framework also provides developers playback control over audio, music, and video files.

2.3 The Limitation of the iPhone OS Platform

iPhone is a mobile device. Unlike desktop computers, iPhone’s resource is limited. It is a portable device that serves user daily usage. Its main purpose is for using as a mobile for iPhone, or a music player for iPod touch. There are various differences between an iPhone and a desktop computer that developers should consider when developing an application for iPhone OS.

It is a tiny computer. iPhone screen size is only 480 x 320 pixel. With that size, there is not enough space to put everything in an application. Developers need to design an application interface so that the screen space has the most use out of it. The unnecessary user interface needs to be cut off. Controls should be obvious that users know how to use right away. Any long description should be avoided, instead, put on a short and obvious label.

It is a multi-touch screen. The main interaction to users is via the touch screen. Users actually use fingers to create different gestures to interact with applications. For example, users tap on a home screen icon to launch an application. Because human fingers are not
pinpoints, the touching area becomes less accurate. Based on this fact, controls in iPhone applications need to have a proper size, which allows user to tap on them easily. The controls need to be spread out enough that users do not need to pay attention when tapping.

Applications should follow the common use of the gestures because users already familiar with them, so they could instantly figure out how they can control or interact with an application. The more common the gesture is, the easier for them to use. Table 2.1 shows a list of gestures users can interact with an iPhone [9].

Table 2.1. iPhone Gestures

<table>
<thead>
<tr>
<th>Gesture</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tap</td>
<td>To press or select a control or item (analogous to a single mouse click).</td>
</tr>
<tr>
<td>Drag</td>
<td>To scroll or pan</td>
</tr>
<tr>
<td>Flick</td>
<td>To scroll or pan quickly</td>
</tr>
<tr>
<td>Swipe</td>
<td>In a table-view row, to reveal the Delete button.</td>
</tr>
<tr>
<td>Double tap</td>
<td>To zoom in and center a block of content or an image.</td>
</tr>
<tr>
<td></td>
<td>To zoom out (if already zoomed in).</td>
</tr>
<tr>
<td>Pinch open</td>
<td>To zoom in.</td>
</tr>
<tr>
<td>Pinch close</td>
<td>To zoom out.</td>
</tr>
<tr>
<td>Touch and hold</td>
<td>In editable text, to display a magnified view for cursor positioning.</td>
</tr>
</tbody>
</table>

There is no physical keyboard on iPhone or iPod touch. Keyboard will appear on the screen when users tap on a control requiring them to enter information. Taping on a little key is not as simple as typing on a keyboard of a desktop computer. Deleting a character of the entered text is an uneasy task, as it requires users to tap and hold to turn on the magnified view and drag a cursor to a particular position that needs to modify (shown in Figure 2.4). It requires a lot of user attention and effort. To enhance user experience, it is best if an application requires as minimal entry fields as possible. If other controls, such as a picker is replaceable, then it is better to user it. For example, use date picker to gather date information from users instead of having an entry field for user to type in.

Memory is limited. It is only 128 MB RAM on iPhone 3G, and 256 MB RAM on the new iPhone 3GS [7]. Plus, the iPhone OS does not have disk space to swap out memory. Its
virtual memory is very limited according to its limited physical memory. Once memory gets full, and there is no more unused pages or read-only content pages to remove, it will start to warn a running application to clean up memory. If the application fails to do so, the OS will eventually shutdown the application. Developers should implement methods that are required to handle the memory warning appropriately. As a result, it is essential for developers to carefully design an application that consumes less memory, and ensures that the memory is released properly when it is no longer needed [10].

iPhone is made of a slower clock rate processor as compared to current desktop computers. iPhone 3GS comes with 600 MHz processor, and iPhone 3G only has a 412 MHz processor [7]. With that clock rate, iPhone’s execution speed is much slower than the desktop computers. Developers cannot expect an iPhone application to perform as well as it would be on a desktop computer. Therefore, avoiding adding unnecessary code in an application is an important factor to consider for iPhone developers. According to the iPhone Human Interface Guideline, iPhone applications should be simple and only focus on one single task [11].
iPhone OS cannot display multiple windows at the same time. Developers need to design applications that will navigate through windows sequentially. A navigation view controller is used for this purpose. UINavigationController controls navigation of views to move back and forth. It navigates views in hierarchy order. The deeper the view is, the more specific content it is in related to the previous view. The navigation controller also includes a navigation bar. The bar holds the current view’s title, some other custom control, and a back to a previous view button. The good example of this view controller is a setting application (see Figure 2.5). Under setting screen, if a user taps on Wi-Fi, then Wi-Fi setting information will be shown on the next screen. Another important view controller is the UITabBarController. The tab bar controller holds together views that can be switched to display in non-order. Users can tab on a different tab at anytime to reveal different content. Views in tab are usually not closely related to each other, unlike the navigation controller.

![Figure 2.5. Navigation control in the Settings application.](image)

For third-party applications, iPhone OS is a single tasking operating system. Apple only allows one application run at a time. Switching to another application means quitting the
current running one. Therefore, developers should certainly tune the launching time that it is as fast as possible, so that users will not see much difference compared to application switching on multi-tasking systems. Developers should be aware that users could switch to another application at any time, so they should save data as soon as it is changed [11]. If developers decide to use a system setting application for setting an application preference, they should keep in mind that the setting application is another application that users need to quit a running application to get to it. Thus, the preference that can be changed often should not be put in the OS setting.

The main purpose of iPhone is to be used as a mobile phone. People use it anytime, and tends to get interrupted while using the device also. As a result, applications should be easy to use, easily to communicate with users, and users know right away what they need to do to accomplish a task they want. Controls should be obvious and labels should be brief and clear. An application should only focus on one task, so it is not complicated to use. Reducing users entries would facilitate usability. Furthermore, it is significant to develop applications that convey the same meaning of gestures as any other applications do [9].

2.4 TYPES OF iPHONE APPLICATION

Based on iPhone Human Interface Guidelines, iPhone applications are divided into three categories: native application, web-only content, and hybrid application [12].

A native application only runs on iPhone OS device. This application is implemented in Objective-C language using iPhone SDK. The application is put in the App Store for users to download and install into their device. They can only run specifically on iPhone OS. This application can take advantage of the iPhone OS.

A web-only content application is developed mainly with HTML, CSS, and JavaScript. The web app is viewed in a browser. With little modification, iPhone audiences can view an existing web app. Developers have a choice to enhance users’ experience making a web app looks like a native app. iPhone OS has added a featured that allows users to create a home icon for bookmark a web site. Users can click on the icon, and then it will automatically launch Safari and open the site for them.

A hybrid application is a native application that relies on content on the server. The hybrid app takes advantage of iPhone OS user interface controls, while its data is
downloaded from the server. iPhone SDK is packed with web view control. The control can use to view web content just like Safari does. However, developers should still ensure that users do not aware that the application is a network-dependency application.
CHAPTER 3

NEW APPROACHES FOR WEB DEVELOPMENT

In this chapter, I will describe new approaches that make the iPhone web development closer to native app development. Safari on the iPhone uses Webkit as its internal engine, which supports many features of HTML 5, such as client-side local storage, geolocation API, offline cache, canvas, etc. It also includes a number of Webkit’s specific CSS styles for enhancing the User Interface and SVG.

3.1 HTML 5

The three browser vendors: Apple, Mozilla, Opera formed a group called WHATWG to work on the new HTML specification. HTML 5 allows developers to make a powerful web page with less effort. Prior to HTML 5, the intense interactive user interface and animation are relied on 3rd party plug-in such as Flash or Silverlight. These plug-ins need to be installed in a browser before users can view those file formats. Another problem with Flash or Silverlight is that most search engines do not read text embedded in those files, which leads to bad indexing in search results. More importantly, Safari in iPhone does not support Flash. Also, no matter how good a web app was, it would still never overcome a native one, since the app became useless when there was no Internet connection.

Several interesting features are embedded in HTML 5 standard. In addition to new layout tags such as <header> and <footer>, several APIs have been added into the standard. Some of the important ones that improve the mobile web app are canvas, geolocation, local storage, and offline cache. Canvas allows web apps to have a rich interactive interface that all are rendered in JavaScript and also are controllable by JavaScript. A web app can now be active without an Internet connection thanks to local storage and offline cache. Combining all these features together, developers can now build web applications with better performance than ever before. With those new added features on HTML 5, the gap between web app and a native app has narrowed. The improvement of JavaScript allows what was once impossible to implement in web apps now becomes common in web apps.
3.1.1 Geolocation Service

Location Base Service, LBS applications has gained a lot of attention lately, since most browsers now have supported geolocation. It has enhanced the user experience because applications can get access to user current location information, and then be able to populate service or data displayed to a user based on that location.

Browsers retrieve location information from three sources. The most accurate one is GPS, equipped with most mobiles. The less accurate one is from the cell towers. When users use their phones, the information goes through the tower. The last and least accurate location data is from Wi-Fi access points. From this available information, browsers can calculate the approximate location of the cell phone using triangulation [13].

In order to get the current location in JavaScript, one can call functions on ‘navigator.geolocation’ class. The function `navigator.geolocation.getCurrentPosition` is a one-time location request. The function `navigator.geolocation.watchPosition` will call a callback function periodically.

Both functions will execute a callback function that pass in as first parameter in case it is successful retrieve user position, otherwise it will call the second callback function. When a geolocation function is called, browsers will first ask user permission whether or not it will allow an application to access the information. On the left side of Figure 3.1 shows a browser asking a user a permission to access the users current location data. The right side shows the map app displaying current location.

![Figure 3.1. Geolocation web application.](image)
3.1.2 Client-side Data Storage

Before HTML5, a web app cannot store much data on client side. The only available option for a web app to store data locally is cookies. HTML5 has added more options to client-side data storage. One can use local storage, session storage, or client-side database.

3.1.2.1 LOCAL STORAGE AND SESSION STORAGE

Local Storage and Session Storage are very similar. The only difference is its lifetime. Session Storage only lasts until the window is closed and data is only accessible in that one window. The Local Storage data are stored for a period of time. Data is saved, and accessible even when the window is closed. Data for Local Storage can be shared to other windows or tabs that open the URL of the same domain name, protocol, and port. In Table 3.1 is a list of functions used to manipulate data on both these features.

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>To save value</td>
<td><code>setItem('email', 'xxx@xxx.xxx')</code>;</td>
</tr>
<tr>
<td>To read value</td>
<td><code>getItem('email')</code>;</td>
</tr>
<tr>
<td>To delete value</td>
<td><code>removeItem('email')</code>;</td>
</tr>
<tr>
<td>To clear value</td>
<td><code>clear()</code></td>
</tr>
</tbody>
</table>

3.1.2.2 CLIENT-SIDE DATABASE

Following the concepts of Google Gears, HTML5 has added client-side database into its specification. By integrating with SQLite engine, browsers allow developers to store structure data in a database file locally. SQLite is an embedded Relational Database Management System on browsers. It is designed to be used internally in a main application, so it runs in the same process as an application to avoid latency from inter-process communication. It does not require installation, or configuration. Each database is stored in a single file. SQLite supports most of SQL-92 standard [14]. The default maximum size for each database is 5 MB. Once it is exceeded the max size, browsers will confirm if users allow the app to expand the database size.

Local database is another reason that web apps can now work offline. When connection is unavailable, an offline web app can still access saved data in the database. The
well-known example, that already takes advantage of this feature, is Gmail on mobile. Gmail creates several tables locally to store contacts and emails. Each user that login to Gmail on the browser has its own database. The application also stores user actions in the database, so when it is back online, then it can perform those actions to data on the server. With the use of client-side database and offline cache, which will be discussed later, Gmail can successfully bring the app offline. The app allows users to read emails even when they are on an airplane mode, which turns off Wi-Fi, 3G, and EDGE, or when a connection is unreachable.

Developers can execute a SQL statement to manipulate with data by just using JavaScript. HTML5 has specified interface for accessing database. In order to access the database, developers need to call ‘openDatabase’. This function will call the database API to open the given database name so we can query, insert, delete, or update data later. To execute a SQL statement, we call a function ‘executeSql’. The database API is designed to execute a transaction asynchronously so that it would not block the main process. When executing an SQL statement, developers have an option to specify callback functions that will be called when the execution succeeds or fail. In Table 3.2 is a list of functions used to open database and execute query.

### Table 3.2. Functions to Open and Manipulate Database

<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>openDatabase(databaseName, databaseVersion, displayName, estimatedSize)</td>
</tr>
<tr>
<td>executeSql(sqlStatement, arguments, callback, errorCallback)</td>
</tr>
</tbody>
</table>

#### 3.1.3 Offline Application Cache

Browsers have been using cache to help expedite web page load time and reduce bandwidth consumption for years. However, these caches were not intended for an offline app. They were wiped out if the device is turned off. Besides, iPhone will only cache a file that less than 15KB and total cache size is limited to 15 MB. Browsers will not check for update if cache is still fresh—the expiration date has yet reached. The desire to make web applications accessible offline leaded to new specifications added to HTML 5. Before the implementation of HTML 5, the benefit of the native app that the web app could never
overcome was that native app is always available to users even when a device has no Internet connection. Web apps, on the other hand, were crippled as soon as it went offline.

The AppCache API is now available on many browsers, such as Safari, Chrome, and Firefox. To take advantage of this feature, developers need to tell the browser which files need to be cached by listing them in a manifest file. The file must have a text/cache-manifest mime type. Also, in the web page the manifest attribute, which value is the name of the manifest file, needs to be added into the html tag. When a user accesses a page for the first time, a browser will download and save all the files listed in the manifest files. When a user accesses the page later, the browser will first look for cached files, if they are available then run it. If a manifest file is changed, the browser will download all files listing in a manifest file again. With cache manifest, an app is accessible even when a device is on offline mode.

3.1.4 Scalable Vector Graphics (SVG)

SVG is not a part of HTML5 specification. It is an open standard has been introduced since 1999. SVG is a file format for browsers to render vector graphics. Vector graphics has the advantage that it will not lose quality if the image is scaled. Resizing a bitmap image requires an algorithm to cut or add pixels into the actual image, which could reduce image quality. SVG, on the other hand, can scale images potentially without losing image quality. SVG is stored in an XML-based file that has been standardized by W3C. It can be edited on any editor just like HTML. Most browsers except IE support SVG directly, including mobile Safari on iPhone. More importantly, unlike flash or Silverlight, SVG is searchable by search engines and does not require a plug-in.

There are a number of tools for SVG editor [15]. Among those is an open source project called Inkscape. It facilitates the process of creating complex paths, modifying nodes, and so on. It allows a graphic designer to focus on a design, and then Inkscape will generate SVG tags for us. The following picture demonstrates the result of a simple implementation of SVG that draws shapes and paths. Figure 3.2 displays shapes created with SVG script.

3.1.5 Canvas

One of the new HTML elements added into HTML5 specification is Canvas. Canvas allows programmatically drawn bitmap graphics using JavaScript. It can be used for drawing graphs, creating animation, or building games on the web page. Canvas is another alternative
to Flash, and just like SVG it does not require users to install a plug-in for their browser. Canvas renders a bitmap image, whereas SVG is used for vector graphic. At the moment most browsers only support 2D canvas. However, Opera is already experimenting with 3D [16].

The authors can write JavaScript code to generate a diagram or chart on the fly. For example, the site can design to gather user votes, and then display the result in the chart right away. We can also use Canvas to generate fancy user interactive web sites, as good as ones which use Flash. Now creating 2D-graphic games using JavaScript is also possible.

To implement Canvas, one needs to add a canvas tag into the html body specifying a drawing area, and then use JavaScript to generate graphics. In JavaScript, one needs to get reference to a context using getContext. Each pixel of color data in Canvas is stored into an array.

Canvas is not a simple task for developers who are new to Computer Graphics. Therefore, a group of people has implemented the JavaScript library called Processing.js
[17]. It facilitates the use of Canvas making it easier to learn and produce a quality graphic. It also handles cross-browser problems.

### 3.1.6 SVG vs Canvas

Both SVG and Canvas are used for drawing a graphic. The overlapping functions between Canvas and SVG make it hard for people to choose between the two. Most people think that Canvas will be replacing SVG since most browsers support Canvas [18]. However, the two are non-replaceable. One needs to understand usability on both techniques to be able to choose one over another depending on the situation. Both of them let us access data via JavaScript. However, SVG stores data in a form of DOM elements, whereas Canvas stores data in an array.

A group of people from University of Helsinki had studied on whether or not SVG and Canvas can be used replaceable [18]. The result was interesting, as it turns out that for pixel-base drawing application, Canvas is the winner. Due to the fact that Canvas uses bitmap graphics where drawing a pixel is very simple; whereas a vector base like SVG is not designed to support drawing a pixel. Therefore, using SVG results in creating each DOM element to represent a brush dot for each pixel. The DOM can rapidly become massive. They also experimented creating vector drawing application using both approaches. As expected, since SVG is originally supported vector graphic, therefore it is capable to handle shape objects better than Canvas. In Canvas, just to add mouse listener to a drawn shape is not an easy task, different from SVG, that each shape has its own DOM element. Also, exporting the draw vector graphic from SVG is easier, since there are a few applications, such as Inkscape or Illustrator, which supports SVG file format. As a result, SVG is good for graphic interface, shapes, user interaction; while Canvas is appropriate for pixel-level graphic, like adding effect and filter, manipulating with pixel, etc. [18]

### 3.2 Cascading Style Sheet

Cascade Style Sheet, CSS, defines the style of an HTML page by separating it out from the main content making the page more manageable. It also reduces the redundancy of style attributes embedded into HTML tags and facilitates the process of style modification as developers now only need to change style sheet for the user interface in one place.
Figure 3.3 shows an example of a calculator web app that made use of CSS. The background of the page uses a gradient background by using background-webkit-gradient. The entry area’s border is rounded by using -webkit-border-radius. The button style is taking advantage of the border image feature by using -webkit-border-image.

CSS has been introduced to web development for many years. Web developers already know how to use it. I will focus on only perperties that are added into Safari for Mobile. Many features are available especially for developers to enhance the iPhone web app’s user interface making it possible to mimicking a native app’s look and feel. These include rounded borders, image borders, multiple backgrounds, texts or box shadows, etc. Table 3.3 lists CSS properties supported by Safari on iPhone [19].
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rounded borders</td>
<td>Make corners of a border of a box rounded.</td>
<td>-webkit-border-radius</td>
</tr>
<tr>
<td>Border Shadows</td>
<td>Drop shadow effect to the border of the box</td>
<td>-webkit-box-shadow</td>
</tr>
<tr>
<td>Animation</td>
<td>Combines animation properties in one property</td>
<td>-webkit-animation</td>
</tr>
<tr>
<td>Reflections</td>
<td>Defines a reflection of a box</td>
<td>-webkit-box-reflect</td>
</tr>
<tr>
<td></td>
<td>Defines a variety of mask properties within one declaration.</td>
<td>-webkit-mask</td>
</tr>
<tr>
<td>Perspective</td>
<td>Gives depth to a scene, causing elements farther away from the viewer to appear smaller.</td>
<td>-webkit-perspective</td>
</tr>
<tr>
<td>Transform</td>
<td>Specifies transformations to be applied to an element.</td>
<td>-webkit-transform</td>
</tr>
<tr>
<td>Transition</td>
<td>Specifies a transition, when should it start, how low should it last, what property to be transitioned, how should it occur, ex: ease-in, ease-out.</td>
<td>-webkit-transition</td>
</tr>
<tr>
<td>Clip background</td>
<td>Specifies the clipping behavior of the background of a box.</td>
<td>-webkit-background-clip</td>
</tr>
<tr>
<td>Marquee</td>
<td>Defines properties for showing content as though displayed on an electronic marquee sign.</td>
<td>-webkit-marquee</td>
</tr>
<tr>
<td>Text size adjust</td>
<td>Specifies a size adjustment for displaying text content in Safari on iPhone.</td>
<td>-webkit-text-size-adjust</td>
</tr>
<tr>
<td>Text stroke</td>
<td>Specifies the width and color of the outline (stroke) of text.</td>
<td>-webkit-text-stroke</td>
</tr>
<tr>
<td>Draggable content</td>
<td>Specifies that an entire element should be draggable instead of its contents.</td>
<td>-webkit-user-drag</td>
</tr>
<tr>
<td>Content modifiable</td>
<td>Determines whether a user can edit the content of an element.</td>
<td>-webkit-user-modify</td>
</tr>
<tr>
<td>Touch callout</td>
<td>Disables the default callout shown when you touch and hold a touch target.</td>
<td>-webkit-touch-callout</td>
</tr>
</tbody>
</table>
CHAPTER 4

DEVICE FUNCTIONALITY ACCESSIBILITY

One advantage that a native app has over web app development on the iPhone is the ability to access device hardware and functionalities. They are accessible via the iPhone SDK, which includes many features such as, OpenGL ES, Mapkit, compass, and video recording [8]. Those frameworks allow developers to access hardware functionalities added to the iPhone 3GS.

The iPhone device contains various hardware, for instance, camera, phone, audio, video, GPS, sensors, etc. Most applications on the App Store exploit these features of the device. Many applications take advantage of embedded camera. Some apps like Facebook allow users to take and post a photo simultaneously. One can build a location based service application retrieving GPS data for a users current location by using Mapkit framework. Yelp, a restaurant reviewing application, uses user current location to limit an area to search for restaurants. With a compass on the iPhone 3GS, applications can now track which direction a device is heading. Accelerometer sensor is used to detect the movement of the device; applications can get that value and do something with it. For example, a game application uses accelerometer to control direction. Proximity sensor is one that will turn the screen off when it detects that the device is close to a face or any surface. An application can make a use of this sensor to turn on and off the device so that it can extend the battery life.

In Table 4.1 is the list of hardware and its accessibility from a web app. Thanks to Apple; many features have been enabled for developers to access thru JavaScript. For example, handling an “onorientationchange” event, a web app can detect a device rotation from portrait to landscape, and vice-versa. Adding this feature, it will be able to, for example, present a different layout for a different orientation. Still, a native app would be able get more detailed information from accelerometer, with x & y coordinate, one can find an angle using atan2 function [20]. As I mentioned in previous chapter, a web app can now get a users current location in JavaScript code. iPhone SDK integrated with OpenGL ES, a lower level application programming interface that connects a software with a graphic hardware which is
Table 4.1. Device Functionality and Accessibility from a Web App

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Webapp Accessibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer sensor</td>
<td>Partial (Portrait/Landscape)</td>
</tr>
<tr>
<td>Address book</td>
<td>N</td>
</tr>
<tr>
<td>Audio</td>
<td>Y</td>
</tr>
<tr>
<td>Camera</td>
<td>N</td>
</tr>
<tr>
<td>Compass</td>
<td>N</td>
</tr>
<tr>
<td>Cut, Copy &amp; Paste</td>
<td>N</td>
</tr>
<tr>
<td>Database</td>
<td>Y</td>
</tr>
<tr>
<td>GPS/Triangulation</td>
<td>Y (Geolocation API)</td>
</tr>
<tr>
<td>Graphics</td>
<td>Y</td>
</tr>
<tr>
<td>Multi-touch</td>
<td>Y</td>
</tr>
<tr>
<td>Network</td>
<td>Y</td>
</tr>
<tr>
<td>Phone</td>
<td>Y</td>
</tr>
<tr>
<td>Photo library</td>
<td>N</td>
</tr>
<tr>
<td>Proximity sensor</td>
<td>N</td>
</tr>
<tr>
<td>Push notification</td>
<td>N</td>
</tr>
<tr>
<td>Video</td>
<td>Y</td>
</tr>
</tbody>
</table>

good for 3D gaming graphics. Fortunately, HTML5 has introduced Canvas, which uses hardware to perform graphics. Therefore, a web app has an option of using Canvas. Webkit, the Safari rendering engine, has been working on WebGL that will make possible 3D graphics on a web app [21]. Both native app development and web app development are both using SQLite. As we will see in the next chapter, the web app and a native app can use the same database structure and it will work on both. Although a web app does not have access to the Phone functionality, it can trigger an app to dial a number [22]. When the anchor tag links to a tel: URL scheme, Safari will launch a Phone application to dial that number when the link is tapped. More URL schemes that are supported include iTune app, Youtube, SMS, Mail, and Google map. One can also embed an audio or a video in the web page and Safari will use a native app to play it. Just like a native app, a web app can handle multi-touch event by using a TouchEvent.
On the other hand, some hardware and functionalities are not supported in the web app development. A native app can enable and get the status of a proximity sensor on their app, whereas it is impossible to do that on a web app. Hardware that is not accessible from a web app including proximity sensor, camera, and compass. In addition, a web app has no access to contacts on users’ address book or photos on the device. Moreover, push notification, a new feature on iPhone OS 3.0 that will display a message when an app is not running, is not available for a web app either. Finally, cut, copy, & paste event is not supported on a web app. Although users can copy and cut the web content, a web app has no way to catch that event.
CHAPTER 5

IPHONE WEB DEVELOPMENT FRAMEWORKS

In this chapter, I will introduce some iPhone web app user interface design standards. Later in the chapter, I will present a study on some of the iPhone web development frameworks. Several of them are open sourced projects. The ones I will point out in detail are iUI, iWebKit, and jQTouch. I will do an experiment on building the same web app from these three frameworks to determine whether they are easy to implement practically, how well the performance is, how well it can mimic a native app user interface, and what the limitations I encounter.

Building an iPhone web app is not just about building any web app; web developers also need to know the constraints of the iPhone device, user experience, and user interface standards to ensure its consistency and make it easy to use for iPhone users. The app needs to return the result that users expect when interacting with a particular widget. Apple has contributed good guidelines for iPhone web development that developers should follow. The guidelines include a suggestion on the size of a button, table row, text field, toolbar, and font [23]. It recommends the edge-to-edge list—a list of equal sized rows for the best user experience on the iPhone as shown in Figure 5.1. All these standards are put together in frameworks by a group of developers to lessen other developers’ workload.

iPhone web development framework is a toolkit that provides a good starting point for developers to build a web app that has the iPhone’s look-and-feel. A web app framework is a combination for HTML, CSS, JavaScript, and images that are put together to create a template for emulating an iPhone native app user interface such as single column list, rounded-corner table view. HTML form controls such as a button, radio button, select box, text fields, etc are styled to fit well on the iPhone screen as the way Apple suggests. The controls sizes are set to make it easy for users to interact with them. In Figure 5.2 is a sample from UiUIKit showing the HTML form controls that follows Apple’s suggestion. The controls sizes are set to make it easy for users to interact with them. Some frameworks also customize HTML form controls that would make them look like native iPhone app widgets.
For example, iWebKit customizes the checkbox’s style to look like a switch control—a control with only a YES or No option in iPhone app. iWebKit’s demo in Figure 5.3 shows form elements that mimic a native app’s widgets. A set of radio buttons is put together in a rounded rectangle block. A select box only displays a selected option with the down arrow on the right side. Most frameworks load with useful icons and images including images for
buttons, bar, and iPhone blue-stripe background, etc. More importantly, frameworks are also taking the iPhone screen size and the available space for web content into account. Frameworks create buttons and table rows that meet the Apple guideline that ensure they are wide enough for a users’ finger. The mimicking of the built-in iPhone widgets help form elements in web apps gain ease of use and consistency in the user experience. Most frameworks support edge-to-edge list layout. Another regularly supported layout is the rounded-rectangle design. It is suggested to use for grouping the related information together or for a list of options for users to select. Figure 5.4 is a sample from WebApp.Net displaying the rounded-rectangle layout [24]. Each rectangle block groups together options for users to select. An arrow indicator at the end of the row suggests that a new page that contains more information related to that particular option would be opened when users tap on it. Moreover, most frameworks will handle sliding in a new page, which resemble that of the native app’s navigation view. Figure 5.5 is the screen capture of the Weather application on iPhone showing the screen sliding when going from one screen to the next. A custom navigation control such as a back button is also added to the toolbar for specific cases such as single-page navigation. Figure 5.6 illustrates single page navigation from iUI’s sample. In the Music list, when users click on Artists, the Artists screen will be slide in to replace the Music screen. When users tap on the artist name, the screen of a listing of that artist’s albums will
Figure 5.4. Rounded-rectangle layout from WebApp.Net’s sample project.

Figure 5.5. Sliding page sample.

Figure 5.6. Single-page navigation from iUI’s sample project.
replace the Artist screen, and once users click on an album name, the list of the songs screen will be displayed in the place of the previous screen. With style sheet and JavaScript, a single page can be separated into different pages in the users’ view. Users’ pages are actually pointed to the same URL resulting in malfunctioning of Safari browser’s built-in back button in users’ perspective. The page users see when tabbing on a back button is not the same page that they expect because a browser’s history only can go back to the previous URL not the previous section of the page. With the back button on the toolbar, frameworks would handle navigation history themselves, and they will direct users to their previous page disregarding whether it is from the same URL. Additionally, the back button would display a title of the previous page, so that users know which page it would direct to.

There are many iPhone web development frameworks that are available now. Table 5.1 shows a list of available open-source iPhone web development frameworks. Each of the frameworks may have different advantages over the others. For example, jQTouch provides a rich user interface and also comes with two default themes. It is a jQuery plug-in. For developers who are familiar with jQuery, jQTouch would be a good choice for them. iUI focuses on the navigation workflow on JavaScript and CSS. iUIUIKit is a CSS framework that only cares for the iPhone-like user interface. Its example covers a lot of widgets. The most recent one is PastryKit from Apple. One claimed that the application it made is being the best iPhone web app on the world [25]. PastryKit supports fling event, allowing users to spin a long list smoothly. It accomplishes scrolling the content of the HTML div instead of the whole page, leaving the toolbar in a fixed position shown in Figure 5.7. The bar does not move along with the whole page when scrolling, like it does on other frameworks. On the left side of Figure 5.7 is iPhone User Guide web app is created with Pastry framework, showing the tool bar still stay in the same position when scrolling to the end of the page. On the right side is Digg application, a demo in iUI framework, showing that the toolbar is disappeared when scrolling. Although PastryKit is the most interesting one, Apple has not released the framework to public yet.

Next in this chapter, the experiment will be done on each of these three frameworks to build an iPhone application called Expense (Figure 5.8). The goal is to create a web app that has the same user interface as the native one by using these frameworks. The Expense is a native app that stores and displays one’s daily expense records.
Table 5.1. Open-Source iPhone Web Development Frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>iWebKit</td>
<td><a href="http://iwebkit.net/">http://iwebkit.net/</a></td>
</tr>
</tbody>
</table>

Figure 5.7. Comparing the toolbar when scrolling between iPhone User Guide and Digg web application.

The main tab includes:

1. Toolbar: displays title and buttons
2. The main area which includes:
   a. The summary view: display the total expense, budget and balance, and the selected date.
   b. Segment control: allows switching between different listings.
   c. Listing view: lists all expense of the selected date, week, or month.
3. The Tab bar: allows user to switch between different tabs.
Figure 5.8. Expense native app’s listing tab.

The trial will only focus on the first two tab of the app including the main listing tab and the “New” tab. The main attention is to create mimicking screen of those two screens and the tab switching from a Listing tab to a new tab. The determination of ease of implementation of these frameworks will be reported.

WebKit engine on Safari browser has well support CSS3, which is the main factor that allows developers to be able to build a potential web app. Most frameworks take advantage of the transformation animation to create slide-in page effect like the navigation control as the native app does as shown in Figure 5.5. The border image is an efficient way to create a button because no matter what size the button is, the border image will be the same size that is needed to cover the border part, and the browser will stretch the image around the border to fit the text or specified size of the element. Frameworks usually take care of the little things developers might not consider to enhance performance. For example, some frameworks preload images so they are ready when needed.

When developing a native app, Apple provides the tool to build user interfaces called Interface Builder (IB). IB has many standard interfaces ready for developers to take advantage of. For example, a navigation bar is a tool bar placed on top of the screen to display a title and a back button on the left side allowing users to navigate back to the previous page. The right side of the bar is available for other control buttons. The toolbar in
IB is placed on the bottom of the screen for holding more buttons. The tab bar is displayed at
the bottom of the screen to display tab bar items. When clicking on a tab bar item, it will
introduce a different view. A segment control is an element that divides into segments that
each segment behaves like a button and it will introduce different actions, such as displaying
a different sub view when choosing a different segment. Table 5.2 demonstrates whether
these three frameworks support the interfaces IB provided.

<table>
<thead>
<tr>
<th>Widgets</th>
<th>iUI</th>
<th>iWebKit</th>
<th>jQTouch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigation bar</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Tool bar</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Tab bar</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Table</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Segment control</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Picker</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Date Picker</td>
<td>N</td>
<td></td>
<td>N</td>
</tr>
<tr>
<td>Round Rect Button</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Text field</td>
<td>Y</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>Text view</td>
<td>Y</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>Switch</td>
<td>Y</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>Slider</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Progress view</td>
<td>N</td>
<td></td>
<td>N</td>
</tr>
<tr>
<td>Activity indicator view</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page control</td>
<td>N</td>
<td></td>
<td>N</td>
</tr>
<tr>
<td>View</td>
<td>Y</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>Search bar</td>
<td>N</td>
<td></td>
<td>N</td>
</tr>
</tbody>
</table>

5.1 iUI

iUI was the first iPhone Web App framework. Later, the author, Joe Hewitt, moved the
project to Google open-sourced project. It is still an active project and will have a new
version release soon. The current version on the project’s site is Release 0.40 Development Test Release 2. iUI’s original purpose was to provide the nice and smooth single-page navigation that had the native iPhone app look-and-feel. The single-page navigation that most web apps implement breaks the browser’s history as mentioned above. When users hit a back button, they would expect to see the previous page on the screen, not jumping two or three pages back. iUI also animates horizontally slide in and out a screen. It also supports vertically sliding page animation.

Although, its project’s site has yet to provide good documentation, there are a number of tutorials for iUI. Both “iPhone in Action” and “Safari and WebKit Development for iPhone” books have a chapter about iUI that has enough detail allowing one to get started.

iUI overwrites HTML links’ behaviors, so it can handle downloading a page via AJAX itself. For an internal link, the anchor tag is pointed to the id attribute of the element in the same page. When the link is clicked, iUI will then hide the current showing element, display the element that is pointed from the clicked link and slide the element in as if it’s opening a new page. For the external link, although the anchor tag is pointed to the external file normally, iUI, however, intercepts link-click event, and instead, calling AJAX to download the linked page, which needs to be only a fragment of an HTML document. iUI will add that fragment into the current DOM and follows the same steps as it displays a link pointing an element in a same file. Because the downloaded file is only a fragment of the HTML, it reduced bandwidth consumption while allowing the page to be loading separately from the main page to avoid first-time loading overhead. It is a better solution than single-page navigation in case putting everything in a file results in a large chunk of data to download in one time. The longer the download period, the more likely users will leave the page. Another kind of a customized link iUI supports is to open a modal dialog. With a simple CSS class set, developers can direct iUI to open a dialog on top of the current page instead of opening a whole new page when users tap on the link. In Figure 5.9 is a search dialog in iUI. The modal dialog only covers the top of the screen and make the rest an opaque layer cover the previous screen. The bottom space is left out so that when a visual keyboard is displayed it will not cover any content of the dialog.

The later version of iUI has supported theme customization. All style sheet and images for each theme are placed into its own folder. This allows developers to switch
themes effortlessly by referencing to the folder name of the theme they like. Creating a new theme is also simplified by only need to create a new folder that hold the images and the customized CSS file.

iUI is coded in the way that every top-level layer will always be hidden by default, unless it has a “selected” attribute set to “True”. It is convenient because CSS will hide all top-level elements automatically for developers, but that makes it complicated for the layout that does not follow a single-page strategy like the Expense application.

To customize the page for my Expense app using iUI, I learned that iUI is not flexible enough for a custom view like a listing tab of the Expense app. Most views in Expense need to be customized by hand including the summary area, the segment control and the list header. The only view that is likely to be supported in iUI is the toolbar and the rounded-rectangle listing area, a part of the main page. The toolbar is handled well by iUI and it does not require any modification to make it work. However, I found that making a rounded rectangle design for the expense list is not an easy task as I had expected. There is no documentation for it; therefore, I needed to study both iUI’s JavaScript code, and the CSS file. Also, it has no sample on how to make a row with the comment on the right side and make the whole row clickable that will open the next page. The only drill-down examples it provides is a single column list with an arrow on the right side. The rounded-rectangle page
Another obstacle is the tab bar at the bottom of the screen. iUI does not have a tab bar built-in. Moreover, iUI has a style setting that every direct child element of the body element is hidden, unless it explicitly specifies “selected” attribute that would tell iUI to show that element. Because I have three direct elements to the body tag: the toolbar, content area, and the tab bar. All of those elements need to be visible on an absolute position. Toolbar is fixed on the top. The tab bar is placed at the bottom. The rest of the area in the middle is left for content and it is scrollable. Toolbar is handled correctly with iUI, and I can also set the content area selected attribute, since it will be the only page I will display. However, the tab bar is not supported in iUI, and is hidden by default. The display and position property of the tab bar CSS style need to overwrite iUI default style that would hide unselected components and place it right under the toolbar. In addition, the overwritten iUI’s link behaviors are not applied to the tab bar. Therefore, I modified JavaScript to catch a link from a tab that would then replace the content of the loading file into the main content area and swap the highlight tab.

Another problem with Expense web app using iUI is to display Save and Reset buttons on tool bar of the “New” page. The buttons only need to show on that page not on the main listing page. However, iUI does not provide mechanism to handle display buttons on toolbar only on a specified page. Modifying the framework to hide button and explicitly set an attribute of the element, so it hides the button solved the problem.

Regarding the style sheet, iUI provides a good starting tool for especially developing the list, however, its style sheet is not flexible enough. Most of the CSS selector depends on the order of child tags, which makes it more complicated for custom views like the “Expense” application. Tab bar introduces another problem. When users click on the browser’s back button, the selected tab does not switch back to match the displayed page. The problem has yet to be solved, as it was not a goal of my experiment.

To sum up, iUI provides a web app structure that would work for some applications or at least a good sample for developers to find resolutions for some iPhone web app development problems, but in practice, one cannot expect it to apply to all applications without customization. Less than 20% of the Expense app is using iUI without style or script
changes needed. Figure 5.10 and 5.11 show screenshots of the Expense native app and the Expense web app respectively. The left side of both pictures is the main listing view, whereas the new item view is on the right side.

Figure 5.10. Expense native app.

Figure 5.11. Expense web app created with iUI.
5.2 iWebKit

iWebKit was created to take advantage of Safari new features. Its goal is to provide a simple and robust way for developers to create a beautiful iPhone web app. iWebKit has a well-documented User Guide. The content includes both the basic HTML for beginners and information about how to follow the framework structure to create UI it supports. The iWebKit’s UI is separated into the top bar, content area, and footer.

In the top bar, one can add a title, buttons, or navigation arrows—navigation buttons connect together the links to pages. The navigation arrow let users jump to any page in its hierarchy. The title will be placed on the middle of the bar. The buttons and navigation arrow can be either left or right side. Figure 5.12 is the music list sample from iWebKit demo.

![Figure 5.12. Music list from iWebKit’s sample project.](image)

In content area, the framework supports multiple designs of list such as, the list with icons, image, arrow, or with extra comment. In its demos, it provides examples of a simple list and a music list. All CSS classes used in these list demos are already specified in its CSS file. iWebKit makes it flexible for developers to mix different styles of list together. Additional, developers can instruct iWebKit to initially load only some number of items and then load more items into the list later when users click on the load-more link. This feature make a list manageable, because sometimes the list might be too long that it is too
overwhelming to users. However, the fact that iWebKit internally downloads all items in the list the first time the page is loaded, but uses JavaScript to show and hide only a number of items specified, it does not relieve the consumption of the bandwidth bottle-neck, especially when the list is lengthy. iUI employs the better approach in this case, it actually downloads only a number of items that will be displayed when the page is loaded. Once users request to load more, then it will use AJAX to gather more from the server.

iWebKit has done a fine job on styling HTML form elements to simulate the native widgets as shown in Figure 5.13. For example, the checkbox is styled and coded so that it has a native app switch control interface. When a checkbox is checked, the switch is on otherwise it will be off. iWebKit add script to control the switch to slide on and off smoothly when users toggle the control. A group of radio buttons is displayed in a block of options with the check mark indicating which one is selected. iWebKit uses JavaScript to capture which option is chosen and adjust the background image’s position so that a check mark is displayed at the end of the row. The select tag in HTML displays a down arrow on the right side of the row implying that it is a dropdown menu. On the left side of Figure 5.13 displays both on and off Check boxes, a radio button that one option is selected, and a closed select box. The screen shot on the right side displays the options spinning that is a default behavior when a user taps on a select box within iPhone Safari browser.

iWebKit also provides a popup dialog which is a simulation of an action sheet in a native app as shown in Figure 5.14. The popup can holds menu items, any list, or any links. The left side of Figure 5.14 is the popup dialog from iWebKit demo. On the right side displays the action sheet from a native app.

Implementing Expense web app is not much different using iWebKit because the most parts in Expense needed to implement by hand. Those areas include the tab bar, the segment control, and the summary area under the tool bar. I can use the same HTML and style sheet as the one implementing using iUI. The tool bar is also as simple as one in iUI. The style on the left and black buttons already set in iWebKit. It only requires changing the background images of the buttons to match the color of the toolbar of Expense app.
Another advantage of iWebKit is its flexibility. When implementing the main list in the rounded rectangle design using iWebKit is easier when compared to iUI. iWebKit’s list item has default styles to support the text on the left side, and allow an extra comment on the right side, or an arrow indicator at the end of the row. These work well with Expense list. It is convenient that developers can choose to add layouts as needed. For Expense, the main list is
the only needed name on the left side and price on the right side, and leave out the arrow indicator. No code structure changes require implementing the list.

There is a problem when implementing tab switching. iWebKit does not support AJAX loading a fragment of the page like iUI does. To be precise, there is no function call for AJAX at all in the library. For tab bar implementation, JavaScript code for switching a selected tab and downloading a HTML fragment of the new page to replace in the current displaying content is needed. Moreover, it also needs to swap the selected tab to the new selected one.

Another problem is regarding the tool bar’s title. iWebKit does not support changing the tool bar title when navigating to different pages but in the same HTML document because iWebKit does not manipulate the link click event like iUI does. The link is assumed to link to a new page with the whole new HTML document, not a fragment of the HTML page like iUI does.

In conclusion, the features that iWebKit supports are made flexible enough, so that developers can easily adjust to match ones’ need. However, iWebKit is still lacking some of the useful features that exist in iUI.

5.3 jQTouch

jQTouch is a jQuery plug-in for mobile web development. Web developers who are already familiar with jQuery will be able to use jQTouch effectively. jQuery is a lightweight JavaScript library that saves developers’ time on coding JavaScript code. It also frees developers from having to handle cross-browser compatibility. The advantage of using jQuery is its convenience to access a group of DOM elements by just using CSS selector, which would reduce a number of lines of code. jQuery also supports animation, AJAX, event handling. Moreover, it allows developers to add functionalities to the library via plug-in and share to the others to gain reusability. A lot of plug-ins are available for developers to take advantage of. jQTouch is a plug-in that help developers implement a web app for iPhone while exploiting features from jQuery. It also makes it easy to port to other mobile platforms such as Palm pre or Android. Moreover, it is able to detect a swipe event, the action when a user place a finger on the screen and move it to the side.
jQTouch makes it easy to turn on and off its default setting, just like how jQuery does. Developers allow overwriting its default behavior. The startup screen which will be loaded when an application first launched in order to enhance user experience can be easily turned on by set the plug-in argument. The application icon is also set the same. The default status bar color can be changed to either black-translucent or black. jQTouch adds more flexibilities by letting developers specify a class selector for its animation effects. For example, one can direct jQTouch to flip a page if its class is matched that is set in the setting.

jQTouch set style for any unordered list to display a rounded rectangle block. These do not work well with Expense since it uses unordered lists to various sections in the page, which should not be rounded. Therefore, Expense style sheets that are used on both iUI and iWebKit would not work in jQTouch. The style sheet for UL tag in jQTouch needs to be modified or overwritten. This requires more work compared to the other two frameworks. Between the two themes it provides, Apple’s theme is a better starter kit for Expense application since the background is already in blue-stripe and font size and row size is already matched well for iPhone. The change is needed to adjust rows’ size in rounded-rectangle design since they are not equally as iPhone suggested. Although jQTouch allows developers to specify the type of transitions to open each page either slide-in, flips, or slide-up, it handles opening a page just like iUI does. It captures the click event to manually use AJAX to load the page, and then insert that page into the DOM document. Therefore, implementing a tab selecting on jQTouch is nearly the same as iUI.

It turned out that it took me more time to work on Expense app using jQTouch than the other ones that I tried when I have limited experienced with jQuery. The more convenient default behaviors the framework provides, the more difficult it becomes for customization. The most work in jQTouch is spending on customization its CSS. The JavaScript code is easy to deal with, since it exploits jQuery features.

5.4 CONCLUSION

Choosing the framework to use is not an easy task to begin with. One needs to know what the frameworks can do, and what one wants to do. One needs to be able to modify the framework to achieve one’s goal. Most of the work developers need to do is to customize the page layout. Most frameworks only provide samples on doing a simple app. The iPhone blue-
stripe background, iPhone native apps default background, is too common. It won’t get much attention from users. Then, developers have to deal with a performance issue. For example, the previous version of iUI was using JavaScript’s timer to control the sliding page effect. It moved the left position of the page by 20% every 3 milliseconds until it reaches the leftmost edge of the screen. People reported it’s sluggish. The latter version iUI switches to employ CSS3’s transform properties. If the loading time is the critical issue, one might consider using a smaller library. As shown in Table 5.3, iUI is very small. The total size of the JavaScript file and CSS file is 20KB. JQTouch, on another hand, is 89KB, due to the fact that it uses jQuery. If an application requires varieties of animations, JQTouch might be the one to be considered.

Table 5.3. Size of Source Code for iUI, iWebKit, and JQTouch

<table>
<thead>
<tr>
<th></th>
<th>iUI</th>
<th>iWebKit</th>
<th>JQTouch (v 1.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(v 0.40)</td>
<td>(v 4.6.2)</td>
<td>(jquery.js version 1.3.2)</td>
</tr>
</tbody>
</table>
| Number of lines of JavaScript source code | 663 | 220 | 634 (jqtouch.js)
|                        |      |        | 4377 (jquery.js) |
| Size of the compressed file version | 12KB | 8KB | 12 KB (jqtouch.js)
|                        |      |        | 57 KB (jquery.js) |
| Number of lines of CSS file(s) | 400 | 818 | 372 (jqtouch.css)
|                        |      |        | 678 (apple theme.css) |
| Size of the compressed css file(s) | 8KB | 16 KB | 8 KB (jqtouch.css)
|                        |      |        | 12 KB (apple theme.css) |
| Total of approximately download compressed files | 20 KB | 24 KB | 89 KB |

As for Expense app, I would choose to use either iUI or iWebKit, since the app requires a lot of customization on the layout part. Modifying in JQTouch is more work, since its default behavior added for the beginning users. Although changes needed, JavaScript code
in iUI seems to be a bit convenient to work with Expense. However, whichever framework I choose, it still requires me to do a lot of modification for implementing Expense app.
CHAPTER 6

PERFORMANCE TEST

In the previous chapter we have already seen that a web app can mimic a native app look and feel very well. As I have shown on examples of building Expense app using three different frameworks, all of them successfully created a copycat version of Expense app. Since web applications can be built nearly identical to the native ones, the next important issue is its performance compared to the native app. In this chapter I will focus on performance testing. I will do some experiments to find out the performance on a web app compared to a native one.

iPhone users tend to use applications in only a short period of time and switch to different application often. Therefore, response time is an essential matter for user interface design, especially on mobile applications. For example, if an application takes too long to launch after users tap on an icon, users may not use the application or they may switch to different apps. According to Jakob Nielsen, 0.1 seconds response time makes user feel that it responses instantaneously [26]. Between 0.2-1 seconds, users notice delay but their attentions still stay focus. In addition, an iPhone user interface guideline suggests that applications should launch as quick as they can, so users can interact with them right away [28]. Therefore, it recommends applications to display a launch image that will be shown when applications are launching, so users see response back when opening the application instead of the blank screen.

There will be three sets of tests. The first test is for measuring the application launch time to compare the time the native app and the web app take to launch. The second one is execution time when switching segment control to find the different of execution time between a native app and a web app. The segment switching will involve database access and redrawing the table on the screen. The last test will be on web app data access time. The test is to measure data access time when the application is on three different situations. One is when data is access from online server. The second one is when it runs totally offline. The last one is the same as the first one, but adding the script to save data to the local database.
The purpose of this set of tests is to find the different of remote data access time on different networks, and between remote and local database, as well as, the time that it may take for download data from a remote server and save to local database.

To do the first two tests, I need to implement the identical applications on native app and web app that doing the same task and having the same user interface. Like in the previous chapter, the app that I will be using for testing is the Expense app. For the second test, the applications will be modified slightly to execute the test. The last test will use a different script that is only read on the database and capture the process time.

For the hardware portion, the experiment will be run on two devices: iPhone 3GS and iPod Touch 2nd Generation. In this chapter, iPhone 3GS is referred as just iPhone and iPod Touch 2nd Generation is referred as iPod or iPod Touch. Both devices run on iPhone OS 3.1.3. Table 6.1 lists specification for both devices [1,2]. The iPhone’s processor clock speed is 600 MHz but iPod touch is 533 MHz. The level 1 cache size on iPhone’s CPU is twice the size of the one on iPod, and iPhone has 256 KB of L2 cache, whereas iPod has none. The memory on iPhone is twice as big as the memory on iPod touch. Wi-Fi is the only option for Internet access on iPod. Meanwhile, iPhone users can connect to Internet on Wi-Fi, AT&T 3G network, or Enhanced Data rates for GSM Evolution (EDGE). Therefore, all online web app tests will be testing with all three networks on iPhone and only Wi-Fi on iPod. The 3G network has a maximum download rate of 1.4Mbps [1]. The average of EDGE data speed is between 75-135Kbps [27]. From this specification, one can reasonably assume that applications running on iPhone will be faster ones running on iPod.

### 6.1 LAUNCH TIME

This test is to measure application launch time of the native app and the web app. HTML 5 overcomes the limitation of a web app that requires Internet connection all time. With HTML5, now a web app can run totally offline without a network connection. The native app that will be running the experiment is the Expense app that reads data from local database and shows the list of expenses users enter in the table. The web app version was implemented using iUI framework. It takes advantage of cache manifest feature that is available in HTML5 to make it capable of running even when the phone is in airplane mode. The web app is also exploited client database, a new feature supported by Safari iPhone.
Table 6.1. Hardware Specification for iPod Touch and iPhone

<table>
<thead>
<tr>
<th></th>
<th>iPod Touch 2nd Generation</th>
<th>iPhone 3GS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>ARM11 620 MHz</td>
<td>Samsung S5PC100 ARM Cortex-A8</td>
</tr>
<tr>
<td></td>
<td>(underclocked to 533 MHz)</td>
<td>833 MHz underclocked to 600 MHz</td>
</tr>
<tr>
<td><strong>Cache</strong></td>
<td>L1: 32KB data cache /</td>
<td>L1: 16KB data cache / 16 KB instruction cache</td>
</tr>
<tr>
<td></td>
<td>32KB data cache</td>
<td></td>
</tr>
<tr>
<td></td>
<td>L2: 256 KB</td>
<td></td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>128 MB DRAM</td>
<td>256 MB eDRAM</td>
</tr>
<tr>
<td><strong>Internet Connection</strong></td>
<td>Wi-Fi (802.11b/g)</td>
<td>Wi-Fi (802.11b/g), 3GS, EDGE</td>
</tr>
<tr>
<td><strong>Operating System</strong></td>
<td>iPhone OS version 3.1.3</td>
<td>iPhone OS version 3.1.3</td>
</tr>
</tbody>
</table>

browser. Combining these 2 features of HTML5, a web app can run totally offline, with correct data showing in the list just as a native app does. Users still have full access to applications when the device is in offline mode. Because a web app can be nearly identical to a native app, performance becomes the real factor for developers to decide in which direction they would like to build their applications.

Application launch time is the period of time when users tap on an application icon on the home screen until the application finishes launching. To do the experiment of measuring the launch time, the test application is designed to be just like any regular iPhone app. It is simple and is assumed that the consumption of data storage is light as it is only for one user. The Expense is suitable for an experiment since it obtains features that most applications do. It contains a number of widgets and controls (see Figure 6.1). The summary area below the tool bar displays amount of expense, budget, and balance in the month, as well as the selected date, the start date and end date of the selected week, or the month selected based on what segment it is on. Texts displayed in this area are all put in labels. Below the summary area is the segment control. The segment control allows users to access different list of data in daily, weekly, or on a monthly basis. Next is a table that its content is
all expenses of particular day, week, or month that users select. The header has three icons user can navigate to different date, week, or month. Users allow selecting items in the list, which will then open the new view that will display a detail of selected item. Using navigation view controller does the trick of sliding a new view in. The bottom of the screen is a tab bar that let users to switch to different views including listing, new, search, and report. Each view is independent from each other.

Moreover, the Expense application stores users’ data in the local database internally, just like most typical apps would. The database contains 9 tables. Table 6.2 is the list of the size of all tables. For this test, I needed to dummy some data. For realistic usage, I decided that ten records for user expense each day is reasonable. The data are only added for one-month period with a total of 31 days. Therefore, the total number of rows of “expense_detail” table is 310. The database file that already has data preset for the test is 66 KB. The same set of data and database structure are also used for Expense web application.

The expense web app, on the other hand, uses iUI as a base framework. The web version of the Expense is only implemented the first tab, listing view, for the test. The user interface of the listing view is identical to the native one. The total size of the web app folder is 1.2 MB including images, and iUI’s themes. The manifest tells the browser to cache 42 files. After the application is cached, it can be run totally offline. However, before that
Table 6.2. Size of Tables in Expense Database

<table>
<thead>
<tr>
<th>Table name</th>
<th>Number of Columns</th>
<th>Number of Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>Credit_card</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Currency</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>Expense_detail</td>
<td>14</td>
<td>310</td>
</tr>
<tr>
<td>Payment_type</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Subcategory</td>
<td>3</td>
<td>47</td>
</tr>
<tr>
<td>Type</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>User_detail</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Vendor</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>Total</td>
<td>50</td>
<td>413</td>
</tr>
</tbody>
</table>

happens, it needs to access to the server online to cache all the files from the server. Therefore, I need to launch a web app online on the device before executing the test. In addition, another script is needed to execute before the test to insert preset data into database. In order to find the launch time, the icon for the web app need to add to the home screen. This feature is implemented by adding the Meta tag in the header of the HTML file. Once users tap on the icon, the browser is launched and executes the requested application.

During the test, to measure application launch time, I used a camera to record videos while I launch applications. Each application was captured launching process three times. The test was executed on both iPhone 3GS device and iPod device. After that the videos were imported into iMovie. iMovie allows me find the duration between I tapped on the application icon and the time application finish launching. However, iMovie time smallest unit is limited to 1/10 seconds.

The result of the test on iPhone and iPod are as expected. The launch time on the web app is longer than the native one. The average of launching time of the native app on iPhone is 1.07 seconds, which is close to an acceptable user response time. On the other hand, the web app took almost twice as long as the native one. It approximately took 1.87 seconds to launch the web app (see Figure 6.2). The average launching time on an iPod is going in the same direction as the iPhone as the web app took longer to finish launching. The average is
1.73 seconds for the native app, and 3.13 seconds for the web app (see Figure 6.3). This result shows that the improvement of hardware on iPhone 3GS helps elevates its performance. Web app launch time on iPhone, however, still need improvement. The facts that the web app needs to first launch the browser before being executed is probably the reason it takes longer to launch compared to the native application.

However, the native apps took longer than 1 second (see Table 6.3). If developers follow the iPhone User Interface Guideline by displaying the launch image, they could improve the user experience, since users think that it responds faster, even though it is just an image. For the web app, I noticed that the browser displayed the image of the last view almost right away when the browser is starting up. Users will see the picture of the page while the browser is rendering the page, but users cannot do anything to the application yet.
Table 6.3. Average Launch Time of Apps on iPhone and iPod Touch

<table>
<thead>
<tr>
<th></th>
<th>Native app</th>
<th>Web app</th>
<th>Web App/Native app</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone3GS</td>
<td>1.07 seconds</td>
<td>1.87 seconds</td>
<td>1.75 seconds</td>
</tr>
<tr>
<td>iPod 2nd Generation</td>
<td>1.73 seconds</td>
<td>3.13 seconds</td>
<td>1.81 seconds</td>
</tr>
<tr>
<td>iPod/iPhone</td>
<td>1.62</td>
<td>1.67</td>
<td></td>
</tr>
</tbody>
</table>

6.2 Execution Time

The main task of the Expense application is relied on accessing the database and printing the results into the table. Therefore, to measure the execution time, it is reasonable to find the time when users changing the data context, like in case of Expense, from the list of expenses in weekly to the list of expenses in monthly. The process is taking place when users click on the segment control, the list in the table will display data of the week or month user selected. Therefore, to execute this test, I modified the code to capture the time when switching segment control from weekly to monthly. When switching segments to monthly, the application will read data that only matches the particular month from database and put them in the table view. I chose monthly view because it has the maximum number of rows that is read from the database. The user interface for a native app needs to modify to add a button that will trigger the test. The test will switch back and forth between weekly and monthly 30 times, captures the time the segment is switched and ends the time data are reloaded in the table, calculates the average execution time, and prints the result in console. The number of tests in a native app is limited because the iPhone OS would terminate an application if it takes too long to execute. Also with a web app, the script to force segment switching and capture the execution time is added and is triggered when the page is loaded. The test iterates for 200 times and the result will display on the new page. The results from the tests are displayed in graphs. Figure 6.4 is the result from the execution time test on the Expense native app on iPhone device. Figure 6.5 is the result from the execution time test on the Expense web app on the same device. The results from the tests on iPod device are shown in Figure 6.6 for the native app and Figure 6.7 for the web app. The summary of all execution time tests is listed in Table 6.4. Figure 6.8 shows the graph of the results from execution time tests on both a web app and a native app on both devices.
Figure 6.4. Execution time of switching segment from weekly to monthly on Expense native app on iPhone.

Figure 6.5. Execution time of switching segment from weekly to monthly on Expense web app on iPhone.

Figure 6.6. Execution time of switching segment from weekly to monthly on Expense native app on iPod.
Figure 6.7. Execution time of switching segment from weekly to monthly on Expense web app on iPod.

Table 6.4. Summary of Execution Time Test

<table>
<thead>
<tr>
<th></th>
<th>Native app</th>
<th>Web app</th>
<th>Web App/Native app</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone3GS</td>
<td>82 ms</td>
<td>87.61 ms</td>
<td>1.07</td>
</tr>
<tr>
<td>iPod 2nd Generation</td>
<td>240 ms</td>
<td>268.08 ms</td>
<td>1.12</td>
</tr>
<tr>
<td>iPod/iPhone</td>
<td>2.93</td>
<td>3.05</td>
<td></td>
</tr>
</tbody>
</table>

Figure 6.8. Average of execution time of switching segment from weekly to monthly on Expense native app and web app on iPhone and iPod.
The execution time from the test on the web app is slightly longer than the native app one on both devices as shown in Figure 6.8. On iPhone, its average is 82 milliseconds for native app and 87.61 seconds on web app (Table 6.4). It was only 5 milliseconds different. Users would not able to see the difference between the two. On iPod, the average is 240 milliseconds for a native app, and 268.08 milliseconds for the web app (Table 6.4).

The result of tests when testing a segment switching on the web app on both devices have some strange outliers, but not many on the native app (as shown in Figure 6.5 and 6.7). On the iPod, one blip is over 1000 milliseconds, when most frequency results are less than 200 milliseconds (Figure 6.7). The same result applies to iPhone, where the high blips are around 5 times longer than the normal ones. In fact, the web app for this test was an offline web app, so an Internet connection should not be matter with the results. This could be the result of garbage collection since the test app manipulated with DOM elements when displaying results of the query into the table.

Execution time of both applications on the iPod were around three times slower than execution time on iPhone when launch time on the iPod is only less than two times that of the iPhone (see Figure 6.8 and Table 6.4). This performance issue could be because the iPhone cache size is double of the iPod cache size. The different between the ratio of the native app and the web app on these two tests, launch time and execution time tests, are very close, 0.6 and 0.5 respectively.

### 6.3 DATABASE ACCESS TIME

This test is to measure time that web app spends on accessing database. This test will only execute on the web app that will read data from the database base either from the remote server or the local database. The script will only access database, and it will not touch the user interface. There will be only one seven-column table in the database. The test is divided into three cases: data access online test, data access offline test, and data access online and save local database test.

#### 6.3.1 Online Data Access Time

This test will access database from a remote server that runs Apache 2.0.52. The database is MySQL 3.23.49. The JavaScript code uses AJAX to connect to the server to fetch the data. The server side scripting is using PHP version 4.3.10. The data that retrieved from
the database is limited to 200 rows. The result returns in JSON format. The time is captured from it starts until the Ajax call back function, which will be called once it gets response back from the server is executed. The test ran on three different networks on an iPhone: Wi-Fi, 3G, and EDGE. The graphs of the results are shown in Figure 6.9, 6.10, and 6.11 respectively. Since iPod only can access Wi-Fi network. The test ran only on Wi-Fi and the graph of the result is shown in Figure 6.12.

![Online database access time from iPhone using Wi-Fi network.](image1)

![Online database access time from iPhone using 3G network.](image2)

The results from Wi-Fi on both devices were rather stable. It took approximately 383.81 milliseconds on iPhone and 678.45 milliseconds on iPod. It took about 630.95 milliseconds on 3G network, and over 2 seconds on EDGE network. It is interesting that the
result from iPhone on 3G network overturned the time executed on iPod Wi-Fi network. As one would understand that Wi-Fi would be faster than 3G because the test was more likely to spend most of its time to connect to network to fetch data. My first assumption was that Wi-Fi would be definitely faster than 3G. It seemed like the hardware has more effect than the Internet connection in this case, because when running on the same Wi-Fi, the test application ran far faster on iPhone than iPod. The EDGE network was very slow when comparing to the others.
6.3.2 Offline Data Access Time

This test is to measure time when a web application accesses a local database. Prior to the test, data needs to be preset to the database. In order to do this, I first ran the online version script, which also inserted the result data fetching from the database server into the browser’s database. Then, I executed the offline data access test.

The difficulty when testing database access offline is the necessity of making sure that all the test suits have the same amount of number of rows in the local database. I first tried to take the test with a thousand rows of data. It turned out that the process took too long and it would time out before ever finish saving all the data. This resulted in the inconsistent number of rows in the client database and on different devices. Therefore, I set the limit of data to 200 rows to ensure that both devices have the same set of preset data before I can actually execute the test.

The offline database access on the iPhone was twice as fast than the time on the iPod. It was around 157.5 milliseconds on the iPhone and 319.58 milliseconds on the iPod. If we look at previous test when testing the online database access, the test result when running on the iPhone via Wi-Fi is also twice as fast as when running on the iPod. The results of this set of test are shown in Figure 6.13 for iPhone, and Figure 6.14 for iPod.

![Offline database access from iPhone](image)

**Figure 6.13. Offline database access time from iPhone.**

The graph in Figure 6.15 illustrates all average values of the results from data access time tests. The offline database access time was more than two times faster than the online database access on both devices. Therefore, if web applications can be designed to limit the
Figure 6.14. Offline database access time from iPod.

Figure 6.15. Average of all database access times on iPhone and iPod.

amount of time to access data online and actually save data locally, it would certainly improve its performance and gain user experience.

6.3.3 Online Data Access and Save to Local Database

This test measures the time it takes to download data from the remote server and save data to a local database. The test app extends the online test app to save data to the local database once it retrieve the response from the server. The number of rows are still limited to 200 rows due to some difficulties. Timing the access of online database and to save data to the local database did have some difficulties. Browser executes database transaction asynchronously so that accessing database would not block user from interact with the web page. The iPhone limits the JavaScript execution time to 10 seconds so the main thread
would not be blocked for a long time from the users [29]. Therefore, the script to capture data access time cannot be directly capture at the end of last command call. Since database access will take longer time than the main thread. To capture the time correctly it needs to do it at the database transaction handler function that is executed once the query is completely executed either successful or error. Because this test is to save data to local database and it iterately inserts 200 rows, therefore, the finish time would be when the last row is inserted. To accomplish this, I check for the last row and only capture the time at after finishing inserting the last row to the local database and print the result out. The delay time needs to be added between each iteration to ensure that the data has been saved and time is captured before starting the new one. In that case, I added 1 second before running a next iteration. Same as the previous set of tests, this tests ran on 3 networks on iPhone and Wi-Fi on iPod. The results of the tests on iPhone device are shown in Figure 6.16 for the test on Wi-Fi Network, Figure 6.17 on 3G network, and Figure 6.18 on EDGE network. The result of the test that ran on iPod device connecting via Wi-Fi network is shown in Figure 6.19.

![Graph showing time to access database remotely and save to local database from iPhone using Wi-Fi network.](image)

**Figure 6.16. Time to access database remotely and save to local database from iPhone using Wi-Fi network.**

With the same set of date as the online database access, for extending saving local database test took very long time to download from a remote server and save to a local database. The fastest one on iPhone Wi-Fi was over 1 second. The slowest one was on EDGE network, which took about 5 seconds. For the worst case, users may not wait for an application to download before they switch to different applications. It was three times slower than just downloading data when tested on iPhone, Wi-Fi network and seven times
Figure 6.17. Time to access database remotely and save to local database from iPhone using 3G network.

Figure 6.18. Time to access database remotely and save to local database from iPhone, EDGE network.

Figure 6.19. Time to access database remotely and save to local database from iPod using Wi-Fi network.
slower than the previous test on iPod device. This asserts that developers should design an app to limit times inserting data into a database.
CHAPTER 7

SUMMARY

As the goal of this research is to learn whether a web app can replace a native app. With the improvement of rendering engine of Safari browser and the new HTML specification, HTML5 that most browsers start to support more, web apps can do many things it could not before and things once only possible on native apps. There are three major technical aspects that we are concerned with when deciding to build a web app instead of a native one. First aspect is the hardware and functional accessibilities, whether a web app can take advantage of embedded hardware and functionality the OS provides like a native app. The second aspect is the user interface, whether a web app can have the same user interface as a native one. The last aspect is to look at a web app’s performance when compared to a native app.

In chapter 4, I have pointed out some popular hardware and functionalities that are available on the iPhone. Some are accessible by a web app, for example, a device orientation, the current location information, database, graphic, etc. Some are impossible such as, a proximity censor, camera, video, address book, photo library, etc. Google Latitude, the app that tracks the location of people in users’ contact list, was restricted on accessing user location. Latitude seems to be working well at some points, but it lacks the automatically updated locations feature. Users need to get in the app for their contacts to see their recent logged-in location. Google voice, an app that combines all your phone numbers to one, is also unable to access device’s address book. It finds a way around this by using its own address book.

In chapter 5, we learned that most native apps could be rebuilt as a web app with identical features and user interfaces. There are various tools and frameworks available to help developers. Also, HTML 5 has played an important role to make this possible. As a result, performance becomes a factor that developers should consider before making a decision to whether develop an application as a native one or a web app one.
According to the results from the tests in the previous chapter, the launch time on a native app is almost twice as fast as a web app on the slower processor, in the test case, iPod (see Table 6.3). However, the difference between the two is not so great with a faster processor and more memory. This indicates that in the future when hardware is improved the web app could possibly launch in an acceptable response time. However, the test application has not done any optimization to ensure it launches fastest. The application startup time is the combination of the time to download scripts from the server, the JavaScript parsing time, data access time, and homepage execution time [30]. Instead of waiting for hardware to be improved, developers can optimize their application to make it launch faster and gain user experience. Developers can tune their homepage; for instance, ones can divide a code into modules, group related modules together, and design for them to be loaded later if they are not needed at the startup time. Therefore, the smaller time needs for homepage execution. Ones can also optimize database access, reducing the number of database hits. It would be more effective if a manifest file only contains files that need and less number of files is better.

The execution time based on the second test result is only slightly better on a native app compared to a web app. One possible reason is that SquirrelFish of the Webkit, Safari’s JavaScript engine was rewritten to compile JavaScript code into machine code [31]. It has enhanced JavaScript execution performance since the iPhone browser no longer needs to parse JavaScript code at execution time. If one only considers execution time, a web app is an acceptable choice. Both applications when executed on the faster device, in this case iPhone 3GS, its performance increases tremendously, which is around three times faster. The difference is around 180 milliseconds on the web app (see Table 6.4). The 268-millisecond execution time on the iPod becomes noticeable when comparing it to the iPhone time of 87-milliseconds. Based on these results, one might need to consider the blips in the graph. Many are more than twice as long as the normal range ones and it periodically happened. Although it happens occasionally, the long execution time could result in an unsatisfactory user experience.

The last test on database access considers the difference between online and offline data access, with the reduction number of network connections, the offline version can reduce the time consumption more than twice as much as the online one. Therefore,
developers should consider using offline database in order to enhance the performance. However, realistically, a web app would need to synchronize data with the remote server, since the main purpose of a web app is to be able to see the same set of data from any client. Therefore, one would need to design an app so that it will fetch data from a remote server, and save locally for later access, and synchronize periodically when it wouldn’t affect user experience, or one might design an app to only download a fair amount of data, just like how Gmail app does. In Google code blog [32], it suggested that an app should only access database when needed and combined multiple queries into one, will reduce amount of database hits which speed faster than execute smaller queries separately.

Developers cannot expect users to have Wi-Fi or 3G networks available at all time. In some situations users may need to get Internet access thru EDGE network, which the download speed is only around 75-135Kbps. From the previous chapter, EDGE network was defeated in all online access test cases. Still, one should design an app to make its response time in an acceptable range on all available network connections.

To sum up, web applications cannot replace all native applications yet. However, it is possible in many cases. For example, applications that do not need to access hardware functionalities or applications where speed is not a crucial issue; a web app is a possible choice. There are also some benefits on choosing a web app over a native app. For example, it is much easier to port to different mobile platforms like Android, Palm Pre, Blackberry, etc. If one needs rapid application development, a web app is perfectly suited. In some cases, to avoid Apple’s injection, one can always develop an application on a web base. Google is a good example in this case. Google Latitude and Google voice are web apps that were first intended to be a native app, but were rejected by Apple. If a reason one selects a web app is the short period of software development, one can also consider applications like PhoneGap that can covert a web application into a native app, and it also open up access to hardware functionalities.
BIBLIOGRAPHY


