Portability Study of Android and iOS

Brandon Stewart

Problem Report submitted to the Benjamin M. Statler College of Engineering and Mineral Resources
at West Virginia University in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

James Mooney, Ph.D., Chair
Katerina Goseva-Popstojanova, Ph.D.
Frances Van Scoy, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2012

Keywords: Portability, Mobile Applications, Android, iOS

ABSTRACT

Portability Study of Android and iOS

Brandon Stewart

The Android and iOS platforms have emerged as the top platforms in the growing smartphone market. This makes them the top targets for developers looking to deliver mobile applications to consumers. However, development for each offers its own challenges. Not only do developers have to support a wide variety of hardware configurations, but each platform has its own distinct software development kit employing different programming languages. This creates a challenge for developers wishing to port an application from one platform to the other, or develop an application for both simultaneously. Developers are seemingly faced with either developing two distinct codebases or being locked in to a third party framework that targets multiple platforms.

This problem report addresses this challenge. A study is conducted to investigate the possibility of developing a highly portable codebase for the Android and iOS platforms. A mobile application that serves as a client to a web service was designed and implemented twice on each platform. The first implementation was a purely native application developed using only each platform's language of choice. The second implementation was developed with portability as a goal. This was met by leveraging each platform's ability to run web applications. The resulting applications show that by using standard HTML and JavaScript, native mobile applications can be developed for each platform with a large amount of shared code.

Acknowledgements

I would like to thank each member of my committee. This problem report draws inspiration from classes I took with each.

Dr. Jim Mooney's interest in software portability and the corresponding class provided the initial inspiration for the project. Additionally his help as my advisor over the semesters has been invaluable.

Dr. Katerina Goseva-Popstojanova's classes have provided numerous helpful attempts at the art of reading and writing academic papers. Her empirical methods class also helped provide the structure used in this problem report's study design.

Finally, Dr. Van Scoy's enthusiasm for languages undoubtedly inspired me to create a project involving numerous programming languages. She is most likely the subtle reason that Python was chosen in implementing part of the study.

Table of Contents

ABSTRACT ii

Acknowledgements iii

Table of Contents iv

List of Figures v

List of Tables v

1. Introduction 1

2. Problem Description 2

3. Study Design 4

3.1. Web Service Requirements 4

3.2. Mobile Application Requirements 7

4. Application Implementations 10

4.1. Notes Web Service 10

4.2. Native Android Application 12

4.3. Native iOS Application 13

4.4. Portable Codebase 14

4.4.1. Android Application Modifications 15

4.4.2. iOS Application Modifications 17

5. Portability Analysis 20

6. Future Research 21

7. Conclusion 22

8. References 23

List of Figures

Figure 1 – Notes system overview 4
Figure 2 – Notes Web Service – Database Model 5
Figure 3 – Generic Success XML Message 5
Figure 4 – Generic Error XML Message 5
Figure 5 – Note List XML 7
Figure 6 – Notes Web Service – Note List method 11
Figure 7 – Notes Web Service – Error XML template 11
Figure 8 – Notes Web Service – Note list template 12
Figure 9 – Native Android Application – Login, Register, List, and Edit screens 13
Figure 10 – Native iOS Application – Login, Register, List, and Edit screens 14
Figure 11 – Portable Android Application Main Activity Code 16
Figure 12 – Portable Android Application – Login, Register, List, and Edit screens 16
Figure 13 – Portable iOS AppDelegate implementation 18
Figure 14 – Portable iOS Application – Login, Register, List, and Edit screens 19

List of Tables

Table 1 – Lines of code in each application 20

1. Introduction

In the growing market of smartphones, the Android and iOS platforms have emerged as the platforms of choice by a large margin. Recent sales figures indicate that these two platforms account for approximately 90% of all smartphone sales in the fourth quarter of 2011 [1]. This indicates extensive growth in a market in which they already have a combined majority share [2]. As of January 2012, the Android operating system accounts for 49% of all current smartphone subscriptions with iOS having an additional 30%.

The fact that these two platforms alone account for nearly 80% of the entire smartphone market makes them the obvious targets for developers wishing to tap into the mobile application market. However, developers wishing to target both platforms face portability issues that can significantly increase total development costs. These specific challenges are addressed in Section 2.

This problem report seeks to identify a possible strategy to greatly decrease total development costs by increasing the portability of each application's codebase. This is addressed through a formal study involving the development of multiple versions of the same mobile application on both the Android and iOS platforms. These applications will serve as clients for a shared web service. The design of this study and the software requirements for the web service and the mobile applications are detailed in Section 3. Detailed information on the implementations of these applications to satisfy the study requirements are in Section 4. These mobile applications are then analyzed in Section 5.

The rest of the problem report is structured as follows: Section 6 identifies areas for potential future research, Section 7 offers conclusions, and references are listed in Section 8.

2. Problem Description

Developers wishing to simultaneously target both Android and iOS face numerous challenges in the area of code portability. For starters, the developer has to account for a wide variety of physical devices each with varying capabilities. This is less of an issue with iOS given all of the supported devices are manufactured by Apple, and hardware configuration changes are limited over the course of years. On Android this is a much bigger challenge given the wide range of hardware manufacturers that supply Android-powered devices.

However, the most glaring challenge is that each platform has its own unique software development kit that uses its own programming language. Android is powered by the Dalvik virtual machine [3]. Programs are written in the Java language. Development is usually done using the cross-platform Eclipse IDE, but it is not required. This is in complete contrast to iOS applications which are developed using Objective C [4]. Additionally, Apple requires the use of the Mac OS X-exclusive XCode IDE.

Given these challenges, many third party solutions have been created with the promise of allowing native cross-platform development with a shared codebase [5][6][7][8]. These solutions generally require constant updating to keep up with nuances introduced in each revision of the target platform. Developers may be hesitant to be locked in to a third party toolkit that could potentially be abandoned, leaving their previously developed applications possibly incompatible with these updates. Additionally, many of these solutions have restrictive licenses or high monetary costs that may be prohibitive to small developers.

For example, Marmalade [6] allows a developer to build applications for both platforms with a single build process. This allows a developer to generate iOS applications without having Mac OS X. However, it uses C++ and development is done entirely within the confines of the framework's libraries. This means if a developer wishes to no longer use the framework, the code would need to be rewritten. Additionally, there is currently a $149 per-seat annual license fee if the developer is targeting Android and iOS. Targeting additional platforms and removing the Marmalade splash screen results in a $499 per-seat annual fee.

Titanium [8] applications are developed in a custom IDE using JavaScript. However, building an iOS application still requires Mac OS X. Despite being written in JavaScript, the code still

uses the libraries that are exclusive to this framework. This results in the same lock in problem that exists with the other third party solutions. Should development on the framework cease, or a developer simply wants to stop using the framework, the code would have to be almost entirely rewritten.

Ideally, a developer would like to be able to develop native mobile applications for these platforms with a large amount of the total code being shared between both versions of the application. This problem report presents a study examining a potential strategy to achieve this very goal. The study design is described in the next section.

3. Study Design

To study our goal of analyzing portability strategies across the platforms, a study has been designed. The study created duplicate versions of the same mobile application for the different platforms using different development strategies. These mobile applications serve as clients for a shared web service. This allows a common set of requirements that all of the implemented mobile applications will have to fulfill. This is also reflective of the nature of many modern mobile applications which simply serve as interfaces to a pre-existing service such as social networking, media streaming, or online banking. In this study, a simple text note synchronization service is used.

In this study four separate mobile applications have been developed. The Android and iOS operating systems are targeted. Two of these applications were developed with completely separate codebases using the official SDKs for each platform. The other two applications were developed with the goal of sharing as much of a codebase as possible, thereby maximizing portability between these two distinct platforms.

A high level diagram of how data travels between the web service and a mobile client is shown in Figure 1. The requirements for both the web service and the mobile applications are detailed in the following sections.

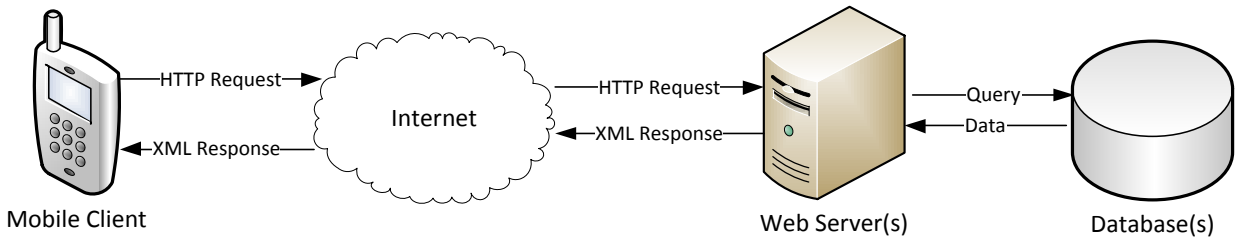

Figure 1 – Notes system overview

3.1. Web Service Requirements

The web service provides the capability of managing a list of notes for multiple users. This is similar to note list applications present on current mobile devices, but in this case the application will interact with the web service to synchronize the list of notes for each user. This allows the user to login on another device to retrieve and edit their same list of notes.

Each note has a unique ID number, a short name, and a text body. Each note and each user is stored in a relational database. The database model used by the web service is shown in Figure 2. This design allows multiple owners for each note, but this is not in the requirements of the mobile application detailed in the next section.

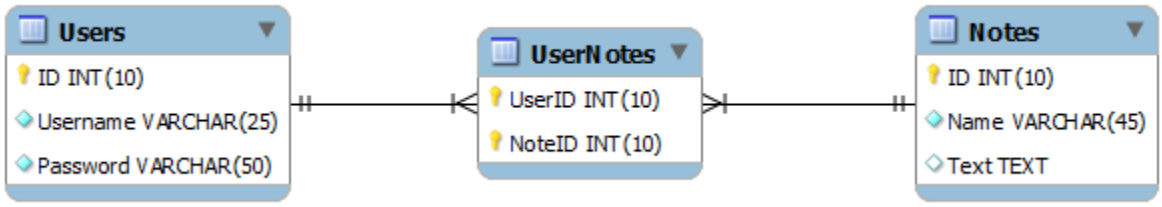

Figure 2 – Notes Web Service – Database Model

The service handles incoming requests through regular HTTP GET or POST methods. The exact method depends on the action being handled. The service must handle the request and response with a properly formatted XML response. The content of this response depends on the action being performed, and may vary in some actions depending on whether the operation was successful. It is up to the client application to handle all possible responses. Examples of generic success and error messages are shown in Figure 3and Figure 4, respectively.

```
<?xml version="1.0"?>
<response type="success">
   <success>
      <description>The operation was successful.</description>
   </success>
</response>
```

Figure 3 – Generic Success XML Message

```
<?xml version="1.0"?>
<response type="error">
   <error>
      <description>An unspecified error occurred.</description>
   </error>
</response>
```

Figure 4 – Generic Error XML Message

The web service is responsible for handling the following actions. The actions are listed as well as the type of HTTP request required (GET or POST) and the relative URL that will be mapped to each action. For this listing, a base URL to the service is assumed and only the end of the

URL relevant to the listed action is presented. For example, the login action's full URL could be "http://my.ser.ver/NotesService/login" but only "(/login)" is listed in the description.

- **Login** – POST – (/login): This handles an incoming request containing a username and password combination. The password will be encrypted by the service and checked against the existing username and password combinations in the database. If the combination is found, the service establishes a new session locally for the authenticated user and returns a success message in XML. If the combination is not found, an error message is returned.
- **Logout** – GET – (/logout): This destroys the session for the currently authenticated user. This means any subsequent request from the same client will not be properly authenticated with logging in again. A success or error message is returned.
- **Register User** – POST – (/register): This handles registering new users for the service. A desired username and password combination is supplied from the client. The service checks if the username is already in use and returns an error message if this is the case. Otherwise, it creates a new user account with the supplied password. A session is then created for this new user so they will already be authenticated without having to issue a login request. A success message is then returned.
- **Get Note** – GET – (/notes/get/<id>): A given note ID number is supplied. The service checks if the current user is authenticated, and then if they are the owner of the note with the given ID. If the user is the owner, then the note is returned. This format of this message is the same as the "List Notes" action and is shown in Figure 5, but it will only contain one note. Otherwise, an error message is returned.
- **List Notes** – GET – (/notes/list): The service checks if the user is authenticated. If so a list of all of the notes this user owns is returned. The XML message for this note list is shown in Figure 5. Otherwise, an error message is returned.

```
<?xml version="1.0"?>
<notes>
   <note id="1" name="my first note">
      <text><![CDATA[This is the full text of my first note.]]></text>
   </note>
   <note id="2" name="my second note">
      <text><![CDATA[]></text>
   </note>
   <note id="3" name="my third note">
      <text><![CDATA[Full text of third note.]></text>
   </note>
</notes>
```

Figure 5 – Note List XML

- **Add Note** – POST – (/notes/add): A note name and body text are supplied. The service checks if the user is properly authenticated and then creates a new database record containing this name and text. It then uses the generated ID of the new note to assign the current user as its owner. If these operations are completed, a success message is returned. Otherwise, an error message is returned.
- **Update Note** – POST – (/notes/update): A note ID, note name, and note body text are supplied. The service checks if the user is authenticated and if they are the owner of the note with the given ID. If so, the note is updated in the database to use the new name and text values that were supplied. A success message is returned if the operation completed, otherwise an error message is returned.
- **Delete Note –** GET – (/delete/<id>): A note ID is supplied. The service checks if the current user is authenticated and if they own the note with the given ID. If they are the owner, the note is deleted from the database. If the operation is a success, a success message is returned, otherwise an error message is returned.

3.2. Mobile Application Requirements

To handle all of the capabilities of the web service, each application has a set of four common forms. This keeps the overall design of the applications as close as possible to more accurately track the differences in developing multiple versions of what is essentially the same application. These common forms will now be described. Implementations of each form along with screenshots are in Section 4.

The first form a user will see is the ***login form***. This form asks for a username and password combination. The user can attempt to log in to the notes service from this screen. This screen handles sending the login request to the web service, reading the response, and acting appropriately. If the web service responds with a success message (which indicates the username and password combination was found), the user is sent to the notes list screen. Otherwise, an error message is displayed and the user remains on the login form.

The login form will also present the user with an option to save their username and password. If this option is selected, the username and password will be saved to the local device. If the user launches the application at a later time and these credentials are present, the login form will be bypassed and the user will be logged in with the stored username and password. When the user logs out, these credentials will be deleted so that a new username and password can be entered.

From the login form the user is also presented with the option to create a new user account. If the user chooses to do this, they will be sent to the ***registration form***. This form is similar to the login form. The user is asked to enter a desired username and password. They are also asked to confirm their password choice. The user can then submit this request for a new account using this form. The form should confirm that none of the fields are blank and that the password and password confirmation strings match. If not, the user should be presented with an error before a notes service request is ever issued. If the fields are correct, then a request to the notes web service is created. If the username is already in use, the web service returns an error, and the user is kept on the registration form. If the registration was a success, the user is moved to the notes list screen.

The ***notes list screen*** shows the user a list of their notes. This list is restricted to only show the notes that belong to the authenticated user. From this screen the user can choose to create a new note, or select an existing note to edit. With either of these actions, the user will move to the edit note screen. This list screen also presents the user with the option of logging out. Choosing this action will result in a logout request being sent to the web service, and the user will be returned to the login screen.

The ***edit note screen*** allows a user to create a new note or edit an existing note. This screen has input fields that allow the user to enter a note name and text. If the action is editing an existing

note, then the form fields will be populated with the currently selected note name and text. When the user chooses to save the note, the application will send a request to the web service. The exact request depends on if the user is editing an existing note, or adding a new one.

The application also must allow the user to delete existing notes. The design decision for how to handle this is left up to each implementing application since the manner in which this operation is typically handled varies by platform.

4. Application Implementations

The following sections detail the implementations of each part of the study: the shared notes web service, the two fully native mobile applications, and the two native mobile applications developed with portability in mind. More emphasis is placed on the portable implementations since this is the primary focus of the study.

4.1. Notes Web Service

This web service is powered by the web.py framework [9]. This is a minimalist Python framework that directly handles URL mapping and maps requests to Python classes based on the URL and the type of HTTP request. Additionally, the framework provides a flexible template system along with database connectivity libraries to handle transparent database access.

The MySQL database system was used to store the database tables used in the notes service. However, the web.py framework allows transparent access to different database software (such as PostgreSQL), so this could be changed depending on the installation target. The framework provides methods for sanitizing input used in database queries to prevent injection attacks. An example of this is shown in Figure 6

The URL mapping is rather straightforward. We define each possible URL path is defined to use a certain class, and then the exact method used within that class depends on the type of HTTP request. For example, a URL path of "/notes/list" is mapped to the class Notes.List (that is, the class List defined within the class Notes – this is a common convention used with this framework). Since this is a GET request type, it is mapped to a method named GET in the List class. The code for this action is shown in Figure 6. This enables us to easily handle all of the required actions for the web service.

```
class List:
        def GET(self):
            ''' Returns all notes in the database and returns
            an XML document listing them. '''

            web.header('Content-Type', 'text/xml')

            ''' User ID and name from session information '''
            (userid, username) = getUser()

            if userid is None:
                ''' This indicates that the userid has not yet been set.
                So, the user has not logged in yet. '''
                return renderxml.error('User is not logged in.')

            notes = db.query("""SELECT ID, Name, Text
                FROM Notes N
                LEFT JOIN UserNotes UN ON N.ID=UN.NoteID
                WHERE UserID=$id""", vars={'id':userid});

            return renderxml.notes(notes)
```

Figure 6 – Notes Web Service – Note List method

Generating XML output is also simplified by using web.py. The system provides simple constructs for handling the variables and iteration necessary for the notes web service. The template used to generate error messages is shown in Figure 7. This shows the use of a variable that already has a default value. In the List method in Figure 6, if the user is not logged in, this error description variable is set to a more useful error description, otherwise the default unspecified message would be shown. The note list template is shown in Figure 8. This shows iteration over the array named "notes." Each element in the array has object properties named ID, Name, and Text.

```
$def with (description="An unspecified error occurred.")
<?xml version="1.0" encoding="UTF-8"?>
<response type="error">
      <error>
            <description>$description</description>
      </error>
</response>
```

Figure 7 – Notes Web Service – Error XML template

```
$def with (notes)
<?xml version="1.0" encoding="UTF-8"?>
<notes>
$for note in notes:
    <note id="$note.ID" name="$note.Name">
        <text><![CDATA[$note.Text]]></text>
    </note>
</notes>
```

Figure 8 – Notes Web Service – Note list template

4.2. Native Android Application

The native Android application was able to be fully developed using only standard libraries present in the regular Android SDK. No external libraries were used.

Network communication is handled through the Apache HTTP component Java libraries that are natively supported on the Android platform. This supports the GET and POST request methods to interact with the notes web service. The resulting XML string is parsed by the W3C DOM libraries that are also supported natively on Android.

The action of deleting notes is handled through a long press in the note list view. The user performs a long press on a note in the list, and they are presented with two options in a contextual menu. This context menu has delete and edit options. The standard menu button on Android devices is also wired up to display a menu on the notes list screen.

The application is shown running in the Android 2.3 emulator in Figure 9.

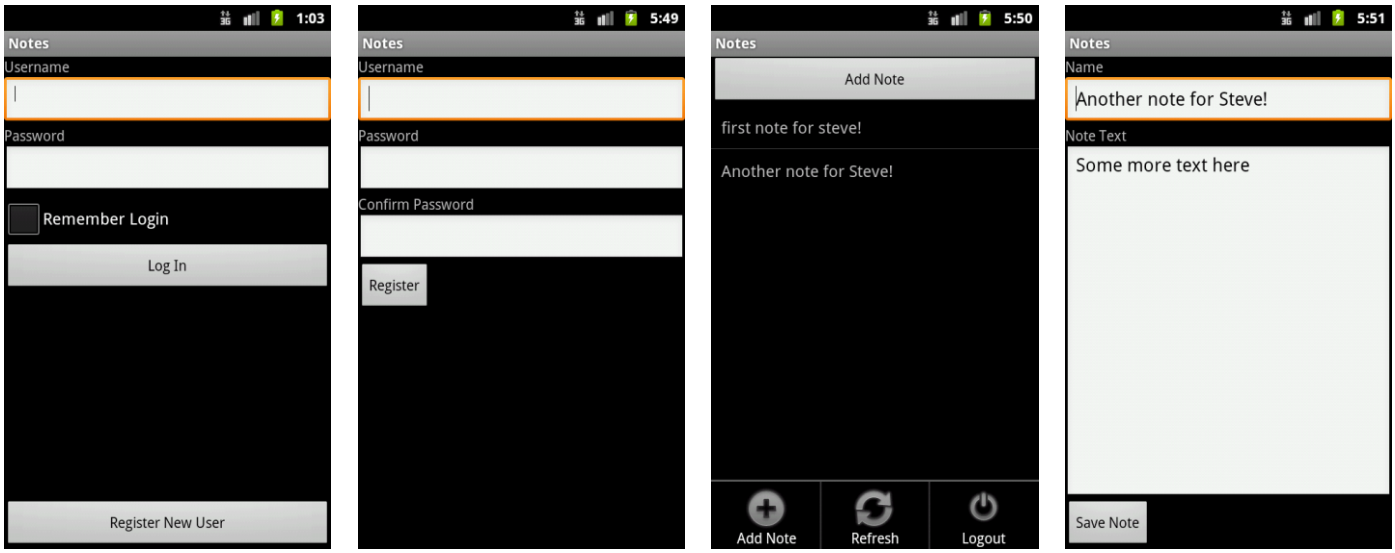

Figure 9 – Native Android Application – Login, Register, List, and Edit screens

4.3. Native iOS Application

As with the Android version, the native iOS application was developed without using any external libraries.

The native NSURLConnection class is used to handle HTTP GET and POST requests to the notes web server. The resulting XML string from the web service is parsed by a custom class using the standard NSXMLParser class.

Deleting a note is handled through a horizontal swipe motion over an existing note in the note list. The standard iOS option of presenting a delete button in the list is shown. If the user presses this button, the note is deleted. Also in keeping with general iOS design standards, controls related to saving data are presented in the navigation bar near the top of the screen.

The application is shown running in the iPhone simulator in Figure 10

Figure 10 – Native iOS Application – Login, Register, List, and Edit screens

4.4. Portable Codebase

The goal of the two previously described applications was to provide a purely native baseline when targeting the Android and iOS platforms. The next two applications were developed with the intention of having as much shared code as possible, thus achieving a high degree of portability.

To achieve this goal, the applications take advantage of the fact that both of the targeted platforms have built in web browsers. More specifically, they have controls that can be embedded into native applications that provide this web browser functionality without having to separately launch the platform's web browser application. This allows us to still have a native application that resides on the user's mobile device, but is essentially only acting as a single window to our forms. These forms can then be reused across both of our targeted platforms.

Using these controls, we are able to embed HTML pages with CSS styling to provide the forms that we need in our notes client applications. To provide the actual application logic necessary to handle functions such as sending requests to the notes web service, we are able to use JavaScript. The jQuery JavaScript toolkit was used in the interest of total development time, but it is by no means a requirement.

The web controls on both platforms allow files to be loaded either locally or remotely from a web server. In the interest of performance, the applications in this study bundle all of the necessary files locally with the application. This prevents having to load the files from a remote web server each time the application is run. Since these are mobile devices, network connectivity may often be limited or unavailable.

The following sections describe the modifications that were necessary for each platform to provide the required functionality that was already present in our fully native applications. These are the modifications are the additions to a "bare bones" project generated by each platform's SDK.

4.4.1. Android Application Modifications

Only one Android activity has to be created to achieve our goal of use the shared HTML and JavaScript approach. This full class definition is shown in Figure 11.

This is all that is necessary to add to the code that is already generated by a new project using the Android SDK in the Eclipse IDE. The common HTML, CSS, and JavaScript files are copied to the application's assets folder within the Eclipse project. This application is shown running on the Android 2.3 emulator in Figure 12.

```
public class NotesWebActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Create the WebView
        WebView webview = new WebView(this);
        setContentView(webview);

        // Puts the scrollbar on top of the WebView
        // so it will fill the entire screen.
        webview.setScrollBarStyle(View.SCROLLBARS_INSIDE_OVERLAY);

        // Enable JavaScript and DOM storage in the WebView
        webview.getSettings().setJavaScriptEnabled(true);
        webview.getSettings().setDomStorageEnabled(true);

        // Prevent caching so our sessions will work correctly.
        webview.getSettings().setCacheMode(WebSettings.LOAD_NO_CACHE);

        // Handle JavaScript alert() functions with an Android message box.
        webview.setWebChromeClient(new WebChromeClient() {
            @Override
            public boolean onJsAlert(WebView view, String url,
                                     String message, JsResult result) {
                return super.onJsAlert(view, url, message, result);
            }
        });

        // Load the login HTML form from the bundled assets.
        webview.loadUrl("file:///android_asset/html/login.html");

    }
}
```

Figure 11 – Portable Android Application Main Activity Code

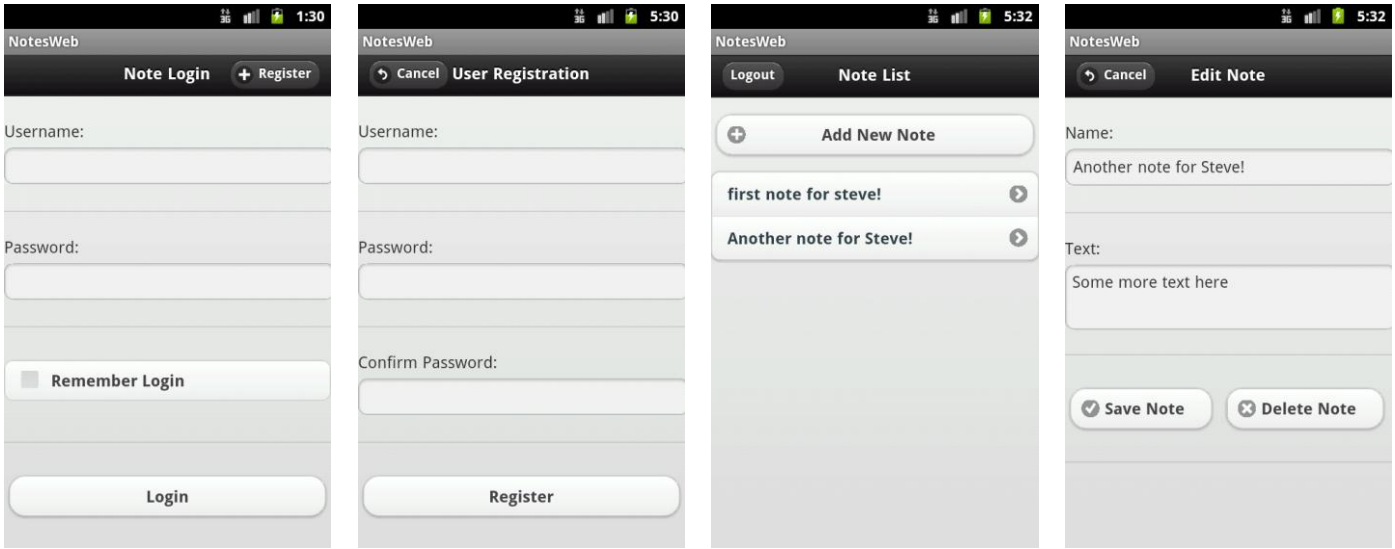

Figure 12 – Portable Android Application – Login, Register, List, and Edit screens

4.4.2. iOS Application Modifications

As with the Android application, the modifications needed for the portable iOS version of the application are rather minimal. The main AppDelegate class needs to be edited to create the UIWebView and load the local login form. This code is listed in Figure 13.

Additionally, during initialize of the class, the application is directed to enable cookie sharing between sessions. This would not necessary if we were loading the files from a remote web server in the UIWebView. However, since we are mixing requests of locally bundled files with JavaScript code that calls a remote web service, we need this cookie sharing enabled. Otherwise, our session would not persist with each subsequent remote request. This leads to a situation where the user is logged in through the login form, then when the note list screen is loaded, they will receive an error message saying they are not logged in despite being authenticated during the login request. The local client in this situation is not aware of the remove session key that had been established. This also causes the web service to continue creating new sessions for each request, even if they are originating from the same client.

```
- (id) init
{
    // Allow cookies to be shared between requests.
    // This is needed to use local html files with remote sessions.
    NSHTTPCookieStorage *cookieStorage = [NSHTTPCookieStorage
sharedHTTPCookieStorage];
    [cookieStorage setCookieAcceptPolicy:NSHTTPCookieAcceptPolicyAlways];


    return [super init];
}

- (BOOL) application:(UIApplication*)application
didFinishLaunchingWithOptions:(NSDictionary*)launchOptions
{

    // Create UIWebView control with app's screen bounds
    CGRect viewBounds = [[UIScreen mainScreen] applicationFrame];
    self.window = [[UIWindow alloc]
                   initWithFrame:[[UIScreen mainScreen] bounds]];
    UIWebView *webView = [[UIWebView alloc] initWithFrame:viewBounds];

    // Generate a request from the local file path
    NSString *filePath = [[NSBundle mainBundle] pathForResource:@"login"
                          ofType:@"html" inDirectory:@"www"];
    NSURL *url = [NSURL fileURLWithPath:filePath];
    NSURLRequest *request = [NSURLRequest requestWithURL:url];


    // Load the request in the WebView and make it visible.
    [webView loadRequest:request];

    [self.window addSubview:webView];
    [self.window makeKeyAndVisible];

    return YES;
}
```

Figure 13 – Portable iOS AppDelegate implementation

This application is shown running in the iPhone simulator in Figure 14.

Figure 14 – Portable iOS Application – Login, Register, List, and Edit screens

5. Portability Analysis

The results of the study are very promising. With very little extra effort, a large portion of the codebase was able to be shared between both platforms. Table 1 shows each implemented mobile application's total lines of code. This line count includes all code source files and form layout files. It also shows the number of lines of code shared in the portable version along with how much of the total codebase this shared code encompasses.

Table 1 – Lines of code in each application

	Notes	**NotesWeb**			
	Total	Total	Unique	Shared	% Shared
Android	1,587	824	119	705	85.56%
iOS	2,108	878	173	705	80.30%

As the results show, the vast majority of the code created in the portable NotesWeb applications is shared. This shows that it is entirely possible to create a highly portable codebase for our example application. It is also interesting to note that the total lines of code written are significantly lower with the portable version.

This analysis shows that using standard web development technologies is a highly appealing strategy to increase overall portability. All of the requirements detailed in Section 3.2 were able to be implemented using primarily the shared codebase containing only HTML and JavaScript. The application's requirements include common requirements in many mobile applications. These include network connectivity and handling input and output with an existing web service.

The biggest downside to this portable solution is in the area of performance. Using JavaScript in an embedded web browser is noticeably slower than using native code for each platform. With this particular application, the difference is not significant. More computationally intense operations would obviously benefit from remaining native. However, both platforms offer means to mix both approaches. That is, native code can be written to directly interact with what is being presented in the web view. This means that this portable approach does not have to be completely abandoned if these more intense operations only make up a few use cases in the application.

6. Future Research

Further study could be done by porting the mobile applications to additional platforms such as Blackberry and Windows Phone. While these platforms do not enjoy the same large market share as Android and iOS, investigating how easily the shared codebase could be integrated with these platforms may serve to strengthen the argument found in this study that standard web development technologies offer a solid strategy for developing highly portable mobile applications.

There are additional features of HTML5 that could potentially be used to enhance portability of more complicated mobile applications. Some of these features could include local storage of data and using location data (such as capturing GPS coordinates on the mobile device).

Along this same line, it could also be beneficial to study portability issues that exist in specialized areas such as game development. OpenGL ES is supported on both platforms. Furthermore, it is possible to use C++ code in iOS applications out of the box, and it can also be used on the Android platform by using the Android NDK. This could result in some amount of shared code between platforms without having to resort to a third party framework. Additionally, WebGL currently has varied amounts of support on both platforms, but it could still be worth researching as a potential future alternative to fully native implementations.

Comparisons to existing third party solutions could be evaluated. This could help show how cost-effective in terms of both monetary cost as well as development effort some of these solutions may be when compared to developing the applications using the standard software development kits.

7. Conclusion

This problem report has shown that it is entirely possible to develop a highly portable mobile application codebase targeting the Android and iOS platforms without resorting to third party tools.

A study was designed using a mobile application design that acted as a client for a web service. This application was implemented for the Android and iOS platforms with two applications for each. The first application was developed using a purely native approach with the respective software development kits for each platform. The second application was developed with the goal of achieving a high degree of portability by employing standard HTML and JavaScript.

The resulting applications were native applications that ran directly on mobile devices. The portable version of the application resulted in at least 80% of each application's total lines of code being shared between both the Android and iOS versions compared to the purely native implementations that shared no code.

8. References

[1] “The NPD Group: Apple Leads Mobile Handsets in Q4 2011, But Android Attracts More First-Time Smartphone Buyers.” [Online]. Available: https://www.npd.com/wps/portal/npd/us/news/pressreleases/pr_120206.

[2] “comScore Reports January 2012 U.S. Mobile Subscriber Market Share - comScore, Inc.” [Online]. Available: http://www.comscore.com/Press_Events/Press_Releases/2012/3/comScore_Reports_January_2012_U.S._Mobile_Subscriber_Market_Share.

[3] “Android SDK | Android Developers.” [Online]. Available: http://developer.android.com/sdk/index.html.

[4] “iOS Dev Center - Apple Developer.” [Online]. Available: https://developer.apple.com/devcenter/ios/index.action.

[5] “Ansca Mobile’s cross-platform mobile app development tool.” [Online]. Available: http://www.anscamobile.com/corona/.

[6] “Mobile Applications Development, iPhone & Android App Development.” [Online]. Available: http://www.madewithmarmalade.com/.

[7] “PhoneGap.” [Online]. Available: http://phonegap.com/.

[8] “Titanium Mobile Platform | Appcelerator.” [Online]. Available: http://www.appcelerator.com/products/titanium-mobile-application-development/.

[9] “Web.py framework.” [Online]. Available: http://webpy.org/.