2016

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

EVALUATION OF DEVELOPMENT TOOLS FOR CREATING MODERN MOBILE APPLICATIONS

Billy James Stroud

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

Department of Computer Science
Department of Information Systems and Operations Management

University of North Carolina Wilmington

2016

Abstract

Evaluation of Development Tools for Creating Modern Mobile Applications. Stroud, Billy James, 2016. Capstone Paper, University of North Carolina Wilmington.

Due to the proliferation of mobile devices, primarily those that run iOS and Android, developers are under increasing pressure to develop applications that work on the disparate platforms; these developers are turning to cross-platform tools. However, the choice of which cross-platform tool to utilize is becoming increasingly difficult. With several different types of cross-platform tools, each with a myriad of choices, mobile developers can easily become overwhelmed by the possibilities. The objective of this paper is to investigate the popular tools being used today, to provide an easy guide to solutions and tutorials, to evaluate their effectiveness as a cross-platform development tool, and to provide my experience in developing solutions with these tools.

Table of Contents

Tables

Figures

Chapter 1: Introduction

During a time when the smartphones in our pockets have more processing power than all of the NASA supercomputers that were used to first reach the moon in 1969, there is more need than ever to quickly and easily produce powerful mobile applications. Many developers decide to utilize cross-platform tools to more quickly target both the iOS© and Android© market shares [2]. However, the choice of which cross-platform tool is becoming an increasingly difficult one. With several different types of cross-platform tools, each with a myriad of choices, mobile developers can easily become overwhelmed by the possibilities.

This project delves into the many different types of cross-platform tools and emerge with a guideline for mobile developers to use when choosing a tool. To accomplish this goal, I have researched many types of hybrid mobile-development tools and chose the most feasible types based on their respective advantages and disadvantages against a set of applications. After researching the plethora of multi-platform development environment, I have narrowed down the specific choices for each type to one or two solutions per group based on their respective popularity, learning curve, and features. I have developed two cross-platform mobile applications with the development tools selected for review. Further I have compared and contrasted the observed advantages and disadvantages of each, based upon access to device-hardware features, speed of development, developer environment, documentation, development community, and performance. The mobile applications that were implemented on these platforms have been chosen according to their utilization of common functions found in mobile apps today such as the following: access to hardware features (GPS, Bluetooth©, or camera), third-party APIs (Google™©, Twitter™©, or Facebook™©), and platform-

specific features (iOS© 3D Touch©, Android© Widgets, or iPad© Split-Screen multitasking).

Chapter 2: Literature Review and Analysis

After nearly nine years since Apple™ first introduced the iPhone, the smartphone app industry has finally started to become mature. These nine years have given many others the opportunity to delve deeper into the world of hybrid mobile applications [1]. Most, however, have chosen to focus on comparing these cross-platform apps to their native counterparts. I had instead chose to focus on comparing and contrasting the many different aspects of several hybrid tools. Nevertheless, these related works have provided invaluable third-party insights into my research.

Most research has been focused on comparing native apps to a single type of cross-platform tool. PhoneGap©, the name of a particular cross-platform tool, is all too often the cross-platform tool of choice by researchers. In reality there are several different categories of hybrid tools, each with multiple platforms [3]. Some researchers start out by intending to take these other tools into consideration, but ultimately decide to abandon this work due to time or ability constraints in favor of focusing on web-based hybrid tools such as PhoneGap© [4].

Other related works have focused on user experience (UX) and native-feel of hybrid apps. Surveys and tests have been completed to determine how everyday iPhone© and Android© users feel about using a hybrid application over a native application [5]. These tests have usually been concerned with the efficiency, intuitiveness, and user satisfaction that the users report back [5]. Fewer tests have been performed on resource usage (memory, battery, and CPU usage) and the respective development environment and community [6, 3].

Chapter 3: Tool Types

Hybrid (Web-Views)

One of the most widely-used cross-platform tools falls under the category of hybrid applications. These mobile apps are essentially mobile webpages that have been developed to run on a mobile browser [7]. The developer is given a small pre-written native app that only runs a full-screen implementation of the default web browser on the mobile phone [7]. This web browser implementation is known as a Web View [9]. The developer can then write their entire mobile application in the standard web-based languages (HTML©, CSS©, and JavaScript©) [9]. They can store these files on a remote web server, and simply configure the Web View to connect to that server's IP address upon opening the app.

The following are the two main tools that fall under the category of hybrid apps: PhoneGap© and Ionic©. All of these however utilize the Apache Cordova© framework. Cordova© is the framework that allows these tools to develop mobile apps with standard web technologies. PhoneGap© and Ionic© were built on top of Cordova© to provide more plugins and features to developers.

Hybrid Advantages

- HTML©, CSS©, and JavaScript©
- Low learning curve
- Low development time
- Allows forcing immediate updates to users

Hybrid apps come with many advantages that help make it be so popular among developers. With its use of web languages such as HTML©, CSS©, and JavaScript©, it is usually considered to have a much lower learning curve when compared to other cross-platform and native development methods. Most developers of all kinds tend to have these programming languages in common unlike other more complex languages such as Java© or Objective-C© [9]. This use of standard languages generally allows for lower development time for small applications [9]. Another great advantage of hybrid apps is that because the application files can be stored on a remote server instead of the local device, developers can quickly and easily push out immediate updates to their apps without having to wait for the review process of Apple™'s App Store© and other exclusive app stores.

Hybrid Disadvantages

- Using a Web View to display HTML© has been shown to perform slower and be less efficient than a native solution
- Remote server requires internet connection for the app to function
- Device-specific and complex functionality is often difficult to implement without using a 3rd party plugin or having to develop one on your own

Hybrid apps have their disadvantages as well. Much like their advantages, most of hybrid apps' disadvantages stem from the use of HTML©, CSS©, and JavaScript©. Hybrid apps are generally said to be quite slow [10]. This is most likely due to the use of the standard web languages (HTML©, CSS©, and JavaScript©). Rendering webpages often produces much lower performance than native code [10]. The use of a remote webserver

to store the web files will also limit app speed to the speed of the user's internet connection and will not function at all if used offline. Lastly, more complex functionality is often difficult to implement due to the fact that most web languages are considered high-level languages. To get around this issue, developers are often forced to find and use a third-party plugin, that may or may not be regularly updated, or develop their own plugin [10]. This can lead to security issues in future and will most certainly increase the time needed for development.

Examples

Adobe™ PhoneGap©

Adobe™ PhoneGap© is the most well-known and most used of the hybrid app tools. Like most other hybrid solutions, PhoneGap© is built upon the Apache Cordova© framework [9]. It uses HTML©, CSS©, and JavaScript© files to render the app's UI elements on the embedded Web View [4]. PhoneGap© apps can utilize third-party plugins to help increase the app's functionality [8].

PhoneGap's© Unique Features

- Platform maturity
- Substantial user base

Hybrid solutions often don't have as many flashy features as native and native-like solutions (discussed later). PhoneGap© is probably the most bare-boned example of hybrid platforms [9]. PhoneGap© simply provides a mobile app to stream your app to a mobile device for testing purposes [8]. Almost all hybrid solutions have this capability and can't be used to distinguish PhoneGap© from others. PhoneGap's© greatest unique feature however is probably its maturity. PhoneGap© is by far the most popular hybrid

solution available and has been around for a long time [9]. This makes its large development community its greatest strength. Solutions with large communities are often easier to develop with because developers can more easily find help and third-party plugins for their apps.

Native-Like

The native-like solutions utilize wrapper classes, written in the tool's preferred language, to make platform API calls through the tool's API. This method allows the developer to produce a native-like look and feel on their apps without necessarily having to know the platform-specific language [9]. These apps are usually said to have better performance but often requires more time to develop [13].

There are two main subcategories of native-like cross-platform apps. These include interpreted solutions and cross-compiled solutions. Interpreted solutions are categorized by their use of an interpreter during runtime. Cross-compiled solutions, on the other hand, are compiled before runtime, executed in the native assembly language [3]. Inclusion in these categories often depends on the languages that are used within the cross-platform tool. For example, scripting languages such as JavaScript© or Python© would normally accompany an interpreted solution because they require an interpreter to run; while compiled languages like C#© and C++© would categorize a cross-compiled solution because they use compilers instead of interpreters.

Native-Like Advantages

- Produces a native app
- Doesn't utilize Web Views, which increases performance

These apps are called native-like because they technically produce a native app. This is because they are compiled or interpreted down to binary before they are executed on the device. This helps to create native-looking UI elements such as buttons, tab bars, and menus. A side-by-side comparison of these is shown in Figure 1. By using only compiled and interpreted languages such as C#© and JavaScript©, they don't require a Web View to render HTML© code. This can greatly increase performance over the hybrid apps discussed earlier [13] .

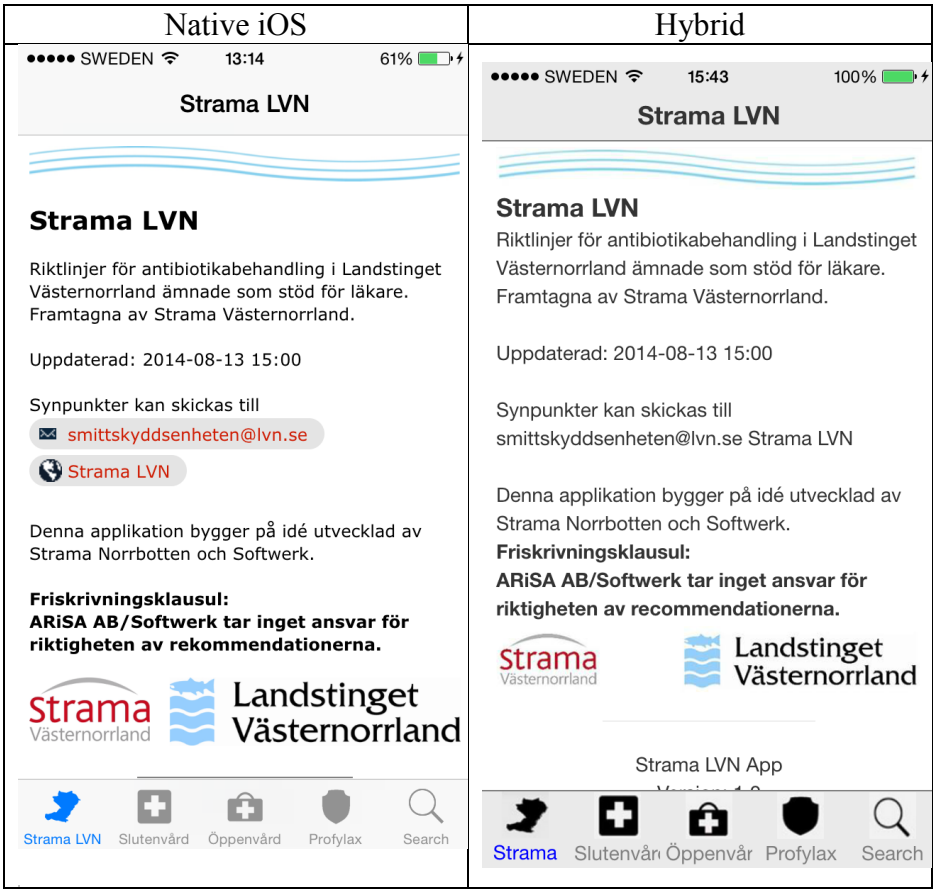

Figure 1: Native vs Hybrid UI

Native-Like Disadvantages

- Usually more complex than hybrid solutions
- Some platform-specific knowledge is needed for complex functionality

Because native-like solutions use API wrapper classes, these solutions are often more complex than their hybrid counterparts [3]. This also causes the developer to learn some knowledge of platform-specific APIs in order to make calls to these with their preferred solution's programming language [14]. These disadvantages together help to make the development time with these solutions longer than with hybrid solutions, but claims to still be better than native, especially after becoming more comfortable with the platform.

Examples

Appcelerator Titanium©

Appcelerator© is a popular tool that tries to bypass the bad performance of Web Views but still allows developers to program in JavaScript© [13]. All of the code when using Appcelerator© is written in JavaScript©, no HTML5© or CSS© is ever used [10]. The JavaScript© is converted into binary (object code) at compilation, thereby allowing the product to be considered a native application [3]. This also makes Appcelerator© perform much better at runtime than other solutions that must rely on a Web View to render the HTML© and CSS© code [13].

Appcelerator's© Unique Features

- Excellent UI creation
- Multiple cloud-based development tools

- Arrow© (API builder)
- Push© (Push notification service)
- Analytics© (Develop custom metrics)
- Dashboard© (Overview of key metrics)

In order to develop an app's UI, Appcelerator© provides an SDK that the developer calls to produce the desired UI [9]. Although limited when attempting to make a very custom UI, because it tries to match UI elements to their native version for each platform, Appcelerator's© UI tools are considered quite powerful. It can be used to create a standard UI so quickly that often some developers use it to make mockups of their apps even if they decide to go with another solution [10]. However, Appcelerator's© SDK isn't perfect for every UI need and also creates another proprietary language for developers to learn [9].

Appcelerator© also provides four cloud-based tools to help developers reduce their development time. These include the following services: Arrow©, Push©, Analytics©, and Dashboard©. Arrow© is a tool that developers can use to create their custom APIs for not only apps but for desktop applications and web apps as well [15]. Modern platforms often require the use of a backend API to send and receive data to and from a backend database. This process usually requires developing PHP© scripts that will connect to a backend database and act as a middleman between the database and any app or web app that uses it. This can easily turn into a very long process often needing to interact with multiple different data sources to get the required functionality. Arrow attempts to simplify this by providing a visual way to develop these APIs. Developers can install prebuilt components called Connectors that will allow Arrow© to connect to

and query from nearly all of the most common databases [15]. These include MySQL©, MS SQL©, MongoDB©, Salesforce©, ArrowDB© (Arrow's© proprietary database), and more [15]. After the installation and setup of a connector, Arrow© will automatically create an API endpoint for each table in the database [15]. These endpoints will automatically have all of the common CRUD (Create, Read, Update, and Delete) uses such as Select All, Find by ID, and more [15]. Users also will have the ability to add or remove their own API calls without any code. Lastly Arrow© even allows developers to create endpoints utilizing joins between tables from different sources [15]. For example, a single API call can join three tables (one from MySQL©, one from MongoDB©, and one from ArrowDB©) without needing any handwritten code [15].

Appcelerator's Push© service provides developers with tools to create their own custom push notifications to their apps [15]. Specific options can be configured to customize the notifications to meet the needs of the developer. This includes things such as which users to notify (all users, specific users, or users in a specific geographic location), sound options (no sound, platform's default alert, or custom sound), and other options [15]. Each notification can be configured with prebuilt analytics to keep track of which notifications entice users to use the app more [15].

Lastly, Appcelerator© provides two other services that go hand-in-hand to help developers measure how their users use their apps. With Appcelerator's Analytics© tool, developers can create custom analytics to view real-time metrics such as number of installs, active sessions, average session lengths, crash rates, and more [15]. Analytics© come with some prebuilt metrics, but developers can create their own custom ones to measure whatever they like [15]. All of these measurements can be seen on their

Dashboard© tool. Dashboard© shows real-time data from all of the active metrics and even a map to show geographic location information for active app sessions [15].

Table 1: Summary of Solutions

Name	Type	Languages	Important Features/Facts
PhoneGap©	Hybrid	HTML©, CSS©, JavaScript©	Large Community
Appcelerator	Native-Like	JavaScript©	Great UI Creation Dev Tools

Table 1: Summary of Solutions

Project Proposal

The purpose of this project was to determine which cross-platform mobile app solutions should be recommended to developers based on the desired app and their skills. In order to achieve this goal, I have developed two cross-platform mobile apps utilizing two of the solutions mentioned in Chapter 3. I have documented my experiences learning and using these tools for the first time so that I may compare and contrast these solutions.

Tool Selection

My selection of cross-platform tools was based upon the following three main elements: their respective popularity, learning curve, and unique features. After researching some of these solutions, I chose to use PhoneGap© and Appcelerator© to develop the two apps. PhoneGap© was chosen over other hybrid solutions due mainly to its maturity and large developer community. Most other hybrid solutions have very few

differences, but don't have nearly as large developer communities and respect in the hybrid category. Appcelerator Titanium© was also chosen for its maturity and large developer community but also for its unique plethora of development features such as its great UI creation, Arrow© API service, and Analytics© service.

App Selection

The two mobile apps that were developed with these solutions were chosen based upon their utilization of common functions found in mobile apps today such as the following: access to hardware features (GPS, Bluetooth©, or camera), third-party APIs (Google™©, Twitter™©, or Facebook™©), and platform-specific features (iOS© 3D Touch©, Android© Widgets, or iPad© Split-Screen multitasking). It is important to note that not every app utilized each of these three features, however each feature was used by each cross-platform solution at least once between the two mobile apps.

App 1: Photo Upload to Face Aging Group

The first app that was developed allowed users to upload a photo to the Face Aging Group's database. The user can choose to either take a new photo or upload a previously taken photo to the Face Aging Group for later processing. This app required access to the camera and photo gallery in order to choose a photo. A small custom API was needed to be written in PHP© to allow the app to send the photo to the Face Aging Group's database. After successful upload, the app asked the user if they want to share to Twitter™© to get their friends to upload photos as well. Finally, an iOS© 3D Touch© Quick Action was implemented as a platform-specific function for quick access to upload the last photo taken or to take a new photo.

App 2: Add URL download to Dropbox©

The second app that was developed allowed users to send URL file downloads straight to Dropbox© without first having to store the file on the mobile device. This allowed users to quickly send a file that they want to keep in their Dropbox© without needing access to their computer and without taking the time to download the file straight to their mobile device before uploading to their cloud storage. This app required a small custom API written in PHP© to download and store the linked file and then upload it to Dropbox©. This was needed because a suitable third-party plugin couldn't be found that had this functionality. Finally, the app was intended to feature an Android© widget that will allow users to quickly submit a URL to the service without ever opening the application; however it was found that this couldn't be accomplished without adding native code the the app.

Development Platform Evaluation

After completion of development of the previous two apps, each cross-platform solution was evaluated based on six elements. These elements included 1) access to hardware features, 2) speed of development, 3) developer environment, 4) documentation, 5) developer community, and 6) performance. These six elements are the most important aspects for developers to consider when choosing a new development platform. Development platforms need to provide the versatility to develop complex functionality while still providing a helpful development environment. The platforms that balance these two things will prove to be the best choices.

Chapter 4: Project Findings

PhoneGap

Installation & Setup

PhoneGap© provided the following two installation options: install a GUI-based desktop app or their command line interface (CLI) tool. The desktop app, while more user-friendly was still in beta and didn't yet provide full functionality. Until the desktop app comes out of beta, the command line interface is the preferred option for most developers. The desktop app was a downloadable disk image or executable installer for Mac and Windows users respectively. By simply following the install wizard, the installer handled everything that was needed.

The CLI tool required a bit more preparation before installation. First, one must have Node.js© installed. This was easily accomplished by downloading and running an installer from the Node.js© website. Secondly, the user must have Git© installed on their machines. Most Macs have this preinstalled unlike Windows. Users can easily test for a successful Git© installation by opening either Terminal or Command Prompt and typing "git" and pressing enter. If the Git is installed, then the user will see a lot of Git usage information printed to the console. If one doesn't see this, then Git can be installed from their website via a downloadable installer. Lastly, to finish the PhoneGap© CLI installation, I simply ran the following command inside of the command line: "npm install -g phonegap@latest". Typed "phonegap" and pressed enter after installation to test if everything was installed correctly. During my experiences developing with PhoneGap©, I first attempted to only work with the desktop app but eventually fell back onto the CLI tool only when it became necessary.

After installation was complete, a PhoneGap© project could be created. From the desktop app, I created a project by clicking on the plus sign (+) and entering a name for the app, a local path to store the app, and an app ID. If using the CLI, I simply navigated to the local path where I wanted the app stored and typed "phonegap create [name of app]". Either way, the app was created with all the files needed to start developing.

Development Environment

PhoneGap© doesn't attempt to force developers into any one environment. Instead, the developer is free to choose any tools they like to start development. All that was needed to start editing code was some sort of text editor. Most developers will already have a personal favorite that they will want to use. My choice was Sublime Text. Sublime Text allowed me to open an entire folder of files at once so that I could quickly and easily switch back and forth between files. Other good choices are Notepad++, TextWrangler, and NetBeans. However, even barebones text editors like Notepad and Text Edit will work.

The PhoneGap© project had multiple folders that were created. The "www" folder was where all coding was done. This folder contained several subfolders and files including a: css folder, img folder, js folder, and index.html. The directory setup greatly resembled a website root directory on a web server. The css folder contained index.css, where all UI styling was entered. The img folder held all images that was used within the app, and the js folder contained index.js, where all of the JavaScript logic was stored. Lastly, the index.html file contained the HTML code that was rendered when the app launched. I also decided to create another folder in this location named "web_plugins". This was used to store special web design plugins such as the Bootstrap framework and

the jQuery source file. While not necessary, this became helpful by keeping all of my original work separate from third-party files that shouldn't be edited.

Before editing, I decided that it was best to view what the default app looked like first. The easiest way to do this was to serve the project over a wireless network to the PhoneGap© companion app on my phone. By searching the app store for PhoneGap© Developer, I downloaded the free companion app to my iOS and Android devices. To view the app on the device, I could either click the arrow button beside the app's name in the desktop app or type "phonegap serve" from the project's root folder in the command line. This returned back an IP address and port number. I then made sure my phone was connected to the same Wi-Fi network as my computer and opened the PhoneGap© companion app that had just been downloaded. The app opened up with a textbox where I could enter the same IP address and port number that was reported on the computer. I needed to type it in with this format, "IP address:port number". In my case it was "10.0.0.12:3000". A screenshot of this can be seen in Figure 3. After typing in the address, I clicked the connect button and was able to see my app running on the phone. Any changes I made in the files was sent over to the app every time I saved my file changes. From my experience, this worked well, however occasionally the app would freeze and you would need to restart it to get it working again. I could tap the screen with four fingers to refresh the app or with three fingers to go back to the server address textbox.

Figure 2: Screenshot of PhoneGap© Developer companion app

The companion app worked well in my experiences with PhoneGap© for testing UI changes such as adding HTML tags and CSS styling. When it came to testing hardware and operating system changes, however, it failed to be very useful. The companion app often would fail to perform functionality such as getting images from the camera or media gallery. It also provided no way of logging debug information. After researching for better testing methods, I finally found that installing the app on a device or emulator to be the best method. By installing the app on a device or emulator, it allowed me to print out error messages to the consoles of native IDEs such as Xcode and Android Studio. This helped greatly when attempting to debug errors. Without it I was essentially debugging blindly.

Unfortunately, installing the apps onto real devices and emulators brought along its own problems. The companion app didn't actually require me to compile and install any native SDKs to run the app. This luxury wasn't given when attempting to actually install apps. To remedy this issue, PhoneGap© pushes you to upload your app's files to their PhoneGap© Build (www.build.phonegap.com) website for compilation. There, the app could be compiled into a native project that could then be imported into Xcode and

Android Studio for installation. This worked great for Android compilation, however, for iOS, it required you to be a part of the Apple™ Developer program before compiling iOS code. In order to become a part of the Apple™ Developer program, you must pay a yearly fee of $99. This is necessary for selling apps on Apple™'s App Store, but if you just wanted to practice development, you probably don't want to pay this much up-front cost.

In order to circumvent this, the app can be compiled from your local machine (if using a Mac) with the CLI tool. The desktop app did not yet (and may never) provide a way to compile apps. To compile the app from the CLI, I needed to navigate to the app's root directory and type "phonegap run [platform]" where platform is the OS for which I was compiling. If the command executed without errors, I could then look inside of the platforms folder located in the app's root directory and find a folder named iOS, Android, or whatever platform I chose (in this case, iOS). Opening that folder would reveal a file with the extension ".xcodeproj". By double clicking this file, I was taken to the Xcode IDE, where I was able to install the app onto the device or emulator. This is accomplished by choosing the desired device or simulator in the devices dropdown box inside of Xcode and clicking the run button. This can be seen in Figure 4. Unfortunately, this would either give an error or install and run but fail to perform some functions such as camera features. This was because the app was missing the necessary plugins needed to perform this functions. These plugins weren't needed when using the companion app or when building from the PhoneGap© Build website, but when building from a local machine, these were needed in order to call functions from plugins.

To add the necessary plugins, it required me to go back to the CLI and type "phonegap plugins add cordova-plugin-[plugin-name]" where the plugin-name is the

name of the desired plugin. For this example, the plugin name is “file-transfer”. This added the plugin to the project’s code. Some third-party plugins found online may have different installation instructions. After installing all of the needed plugins, the app needed to be compiled from the command line again. Then the project could finally be opened in Xcode and installed once more. The app would then function correctly assuming there were no errors. For testing purposes however, I realized that I needed to install the “console” plugin before running the app. This allowed me to use “console.log(‘error message’)” with JavaScript syntax to show debug statements inside of the consoles of IDEs such as Xcode and Android Studio. Lastly, unlike the companion app, the project needed to be compiled and installed again after each change or plugin addition. This wasn’t as slow of a process as it sounds because the IDE’s installation time dramatically decreases after the first install.

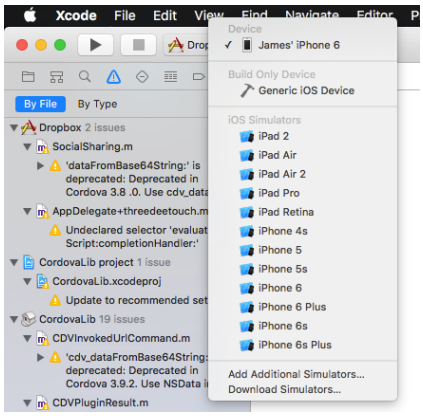

Figure 3: Device dropdown list and Run button in Xcode

Development Process and Experience

App 1 – Face-Aging Uploader

I started my PhoneGap© development experience with App 1 (Face-Aging Uploader). In total, this took approximately 14 hours of development time. This doesn't include about an hour of download, installation, and environment setup needed to start developing on the PhoneGap© platform. It also doesn't include the time needed to get Android Studio installed on a Windows machine (discussed later). The installation process was fairly simple with both the CLI and desktop versions, however the CLI version did require some very basic knowledge of terminal commands.

I immediately downloaded the PhoneGap© app to my iPhone to view the current iteration of the app. From there, I wrote some basic HTML, CSS, and JavaScript in order to get a basic UI created for the app. During the creation of the UI, I ran into my first difficulty. The app that is created at the beginning is given a div tag with the id of "deviceready." This div block is used by the PhoneGap© system code to accept the "deviceready" event, which is fired-off when the app has been rendered and is ready for custom code to be executed. Before knowing this, I immediately deleted this block to free up the view for my custom UI code. This caused the app to fail to accept the "deviceready" event, and therefore the custom JavaScript code fails as well. I found that the easiest way to correct this issue is by simply setting the div's display attribute to "none". This allowed the app to accept the "deviceready" event without needing to actually show the div block on the app's view.

After creating a basic UI for the app to function, I then needed to add hardware functions like accessing the camera and gallery. Not knowing how to do this, I was able to quickly find a forty-minute YouTube video providing a guide on how to utilize basic hardware functions such as these. By following this video, I was able to quickly setup camera and gallery functions for the app. I was able to change the logo image on the view

to the image that was returned by the functions. I then did some research on how to send the image to a web server. I was able to quickly find a way to do this with a downloadable third-party plugin named File-Transfer. However, when I attempted to run this on the PhoneGap© live preview app, this failed to work. I originally thought that I may have had an error in my code to cause it to fail; so I spent a long time attempting to debug the app with no progress. I finally decided that I needed a way to print out some information to a console in order to help with the debugging process. So I tried to find a better way to execute the app. After doing some research, I came across a Google™ Chrome extension, named Ripple Emulator. This extensions attempts to emulate a device running your app and contains an output console to help with debugging. I was never able to get the Ripple Emulator to fire-off the "deviceready" event that was mentioned earlier. Due to this, I eventually decided to install the app on an actual device. This allowed me to write output to the consoles found inside of Xcode and Android Studio. In order to run the app via Xcode or Android Studio, I needed to first build the app into an executable package. As was mentioned earlier in the development environment subsection, there are two ways to build the app packages – from the command line and from the PhoneGap© Build website. Due to the need of an Apple™ Developer subscription, I eventually built the iOS app using the CLI tools. This created an Xcode project to be ran and installed via the Xcode IDE. During the first run, I received an error saying that the plugins weren't present in the project. After doing some research, I found that when not building the app through the website, the plugins aren't automatically installed. I then installed the plugins via the CLI with the "phonegap plugins add" command. This allowed the app to function, however, I still wasn't able to write information to the output console with JavaScript's "console.log" function. I was able to remedy this by installing the Console plugin. After

doing all of this, I found that there never was a bug in my code. The only issue was from the fact that the PhoneGap© preview app is incompatible with third-party plugins.

After getting the app to run on an actual device and basic debugging capabilities to work, I was able to add plugins for dialog alert boxes and the Twitter™ integration. I also was able to add time tracking via JavaScript and storing the times into a MySQL database on my AWS web server. Storing the times in this way allowed me to discretely record the data and later process these from the database.

After completely creating the app for iOS, I tried to get the app to run on Android. I purchased a used LG Logos Android phone to test the Android apps. The Android phone has Android Jellybean 5.0.2 installed. This phone was chosen due to this particular version of Android. Android 5.0.2 is the second-newest major Android release, and due to Android's fragmentation issues, is found on more devices than the newest version of the operating system.

The Android versions of these apps created a lot of issues for me during the development process. The first issue was the incompatibility of the LG Logos and the Mac computer, that was being used to develop. Mac OS X was unable to recognize the USB connected phone. To get around this issue, I needed to first install a Windows virtual machine on my Mac computer. I installed the Windows 10 operating system on this virtual machine. The Windows OS was able to recognize the connected phone and allowed the phone to install the necessary drivers to move data to and from the computer. Secondly, I needed to download and install the PhoneGap© to the Windows machine. As mentioned earlier, PhoneGap© requires node.js and Git to be installed. I installed node.js and Git and then installed PhoneGap. After installing these, I installed Android Studio in order to install the app. Android studio required that I install Java and the Android SDK.

While trying to do this, the Android SDK filled up the SDD on my computer and didn't give an error, so it just ran the download function for a long time before I realized the issue. I was able to make room on the SSD and install the SDK without issue. After attempting to run the Android Studio for the first time, it gave an error about several Java and SDK path environment variables. After awhile of research into this, I was able to finally get the IDE to open. Finally, the app just needed to be built with the CLI tools. Unfortunately, after changing the environment variables, the PhoneGap© and Git commands wasn't recognized by the command prompt even though it was working before installing the IDE. I tried to remedy this by restarting the machine several times and reinstalling node.js, Git, and PhoneGap© again. This didn't fix the issue. I was finally able to install the Android SDK on my Mac, build the app via the CLI, and then copy the app's files over to the Windows drive. This allowed me to open the app in Android Studio and install it onto the LG smartphone.

After finally getting the app installed, I ran into a couple of Android specific issues. The image chosen from the gallery and camera became rotated when placed on the app's image placeholder. To fix the image rotation issue, I used an if-statement to check if the app is running on an Android device and then applied a CSS3 transform attribute on the image to rotate it 90 degrees to the left. Screenshots of App 1 (built using PhoneGap) running on iOS can be found in Appendix A and the same app running on Android in Appendix B.

App 2 – Dropbox URL

The development of App 2 – Dropbox URL – went much faster due to learning a lot of the quirks and being able to copy and paste a lot of code from the previous

PhoneGap© app. App 2 took approximately 6 hours of development time. In order to save time, I decided to use App 1's files as a starting place for App 2. The following files were copied over: index.html, index.css, and index.js. I also saved time by installing the app on actual devices for viewing and debugging instead of trying to run it via the PhoneGap© preview app or the Ripple Emulator extension. All of these things helped to shave off a lot of development time for App 2.

Most of the UI for App 2 is very similar to the Face-Aging app's UI. So the main thing that needed to be done was to add the Dropbox uploading functionality. Dropbox doesn't provide an official SDK for PhoneGap, and after searching for a third-party plugin, I was unable to find one that had the needed functionality and had been updated for the current Dropbox API. To get around this issue, I downloaded the official Dropbox SDK for PHP. This allowed me to create a script on the web server and simply send it the URL that the user enters. This is also a better option than a PhoneGap© only solution because the web server can handle the uploading of large files while allowing the user to continue to use their phone as usual. Lastly, I added the same time tracking code that was implemented to track the render times of the app. I then sent that data to the same database with a different source ID.

After getting the Android issues (mentioned in the App 1 subsection) figured out, I was able to fairly easily get the Dropbox URL app installed on the Android device. Android didn't cause any runtime issues during App 2's execution, unlike when running the Face-Aging app for the first time. Screenshots of App 2 (built using PhoneGap) running on iOS can be found in Appendix C and the same app running on Android in Appendix D.

Performance Measurements

Several different measurements were taken to measure performance. The load times were tracked for both the Face-Aging and Dropbox apps. This tracks how long it takes for the device to render the view. Additionally, the time that it takes to open the camera and gallery functions were also tracked on the Face-Aging app. The average measurements of these metrics can be seen in the following table along with the number of measurements from which the metrics were derived.

There is a clearly visible pattern in these metrics where the Android device was substantially slower than the iPhone. It is a bit unclear as to whether this is caused by the hardware of the devices or the operating systems themselves. The iPhone that was used for this test was an iPhone 6, equipped with Apple™'s A8 processor. The Apple™ A8 is a 1.4 Ghz dual-core 64-bit processor based on the ARM architecture, with a semiconductor size of 20 nanometers and 2 memory channels. The Android phone that was used during the tests was a LG Logos. The LG Logos is equipped with a Qualcomm Snapdragon 410. The Snapdragon 410 is a 1.2 Ghz quad-core 64-bit processor with a semiconductor size of 28 nanometers and only 1 memory channel.

Metric Type	Device	App Number	Average (ms)	Count
Load	iPhone	App 1	682.55	20
Load	Android	App 1	908.15	20
Open Camera	iPhone	App 1	3.25	20
Open Camera	Android	App 1	64.6	20
Open Gallery	iPhone	App 1	1.35	20

Open Gallery	Android	App 1	20.8	20
Load	iPhone	App 2	356.65	20
Load	Android	App 2	857.3	20

Table 2: PhoneGap© Performance

Access to Hardware Features

A qualitative measure of 1 to 3 was given to the PhoneGap© platform to show the tool's ability to implement some important hardware and software features. A measure of 1 was given if PhoneGap© can't currently implement the feature, 2 if it can be implemented, and a 3 was given if it was easy to implement the feature with PhoneGap. Easy to implement can be described as having readily available documentation and requires little coding to implement. The following features have been evaluated: camera, gallery, 3D Touch, and Android Widgets.

Feature	Measure
Camera	3
Gallery	3
3D Touch	2
Android Widgets	1

Table 3: PhoneGap© Capabilities

Developer Community and Documentation

A search for "PhoneGap" on StackOverflow.com (one of the most popular development help websites), produced over 46,200 different posts. A search for

"PhoneGap©tutorial" on YouTube yielded about 64,000 tutorial videos. PhoneGap© provides very few documentation articles and tutorials. Most of the useful documentation has to be provided by the developers of the plugins you choose. Most plugins were found on GitHub, and the relevant documentation was located in the README.md file found in the plugin's repository. Some GitHub developers never even add documentation to their repo. This prevents developers from having a single location to find all of their needed documentation, if they can find any at all. Fortunately, I was able to find sufficient documentation to complete both apps.

Appcelerator Titanium

Installation & Setup

Appcelerator© provides an online portal, called Dashboard, where users will be able to access their account and download Appcelerator© Studio, their proprietary IDE. To get started, you must create a free account on their website. Developers can also choose to login with a GitHub account if they prefer. After logging into my account, the Dashboard view was presented. From here I could access everything needed to build mobile apps. This includes an overview of key app metrics, IDE download, documentation repository, and more. A screenshot of Dashboard can be seen in Figure 5. The first thing needed was the Appcelerator© Studio. This can be downloaded by clicking the "Studio" option in the top menu. Appcelerator© also provided a CLI tool that can be used if preferred, however Appcelerator© Studio provided all of the necessary functionality. The download included an installer package that simply needed to be ran. After installation, important SDKs needed to be installed for each desired platform. When

opened for the first time, the IDE allows users to download any SDKs that may be needed. This can be limited however due to your computer's operating system. For example, only Macs can download and use the iOS SDKs, while only computers running Windows may develop for Windows Phone. Both operating systems can develop for Android devices. These SDKs can be installed via the built-in installation wizard.

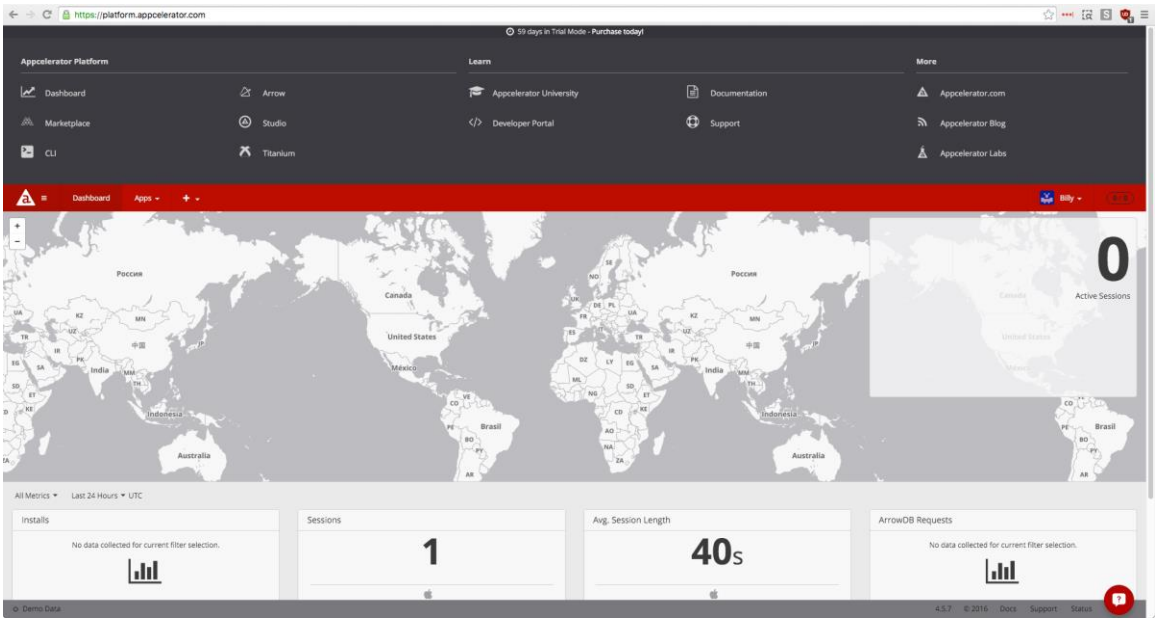

Figure 4: Appcelerator Dashboard

Development Environment

Appcelerator© Studio was the only tool that was needed during my experience with the project. It provided a fairly standard interface with most of the same components that are common among all IDEs. This includes a project file explorer, build and run interface, and a console to view output. An image of Appcelerator© Studio can be seen in Figure 6.

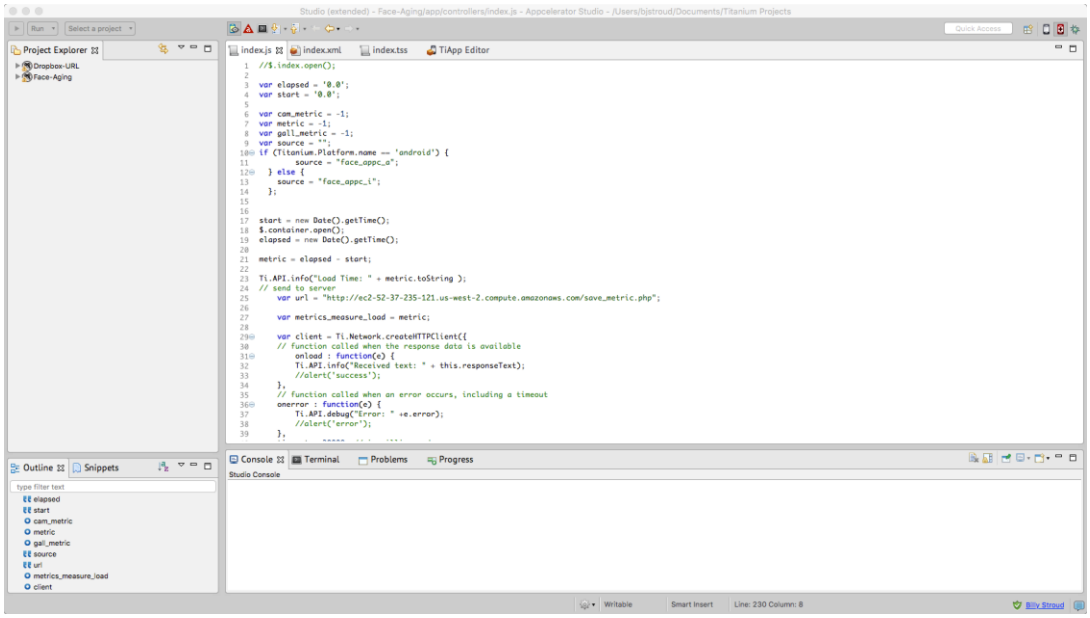

Figure 5: Appcelerator Studio

The project's root directory provided several subfolders, however most of the work was done from within the "app" directory. This folder was already generated with a starting template that uses the model-view-controller (MVC) framework. Each view in the app is created with its own model, view, and controller which is stored in the respective subfolder. For example, the index.xml file is the view file of the starting screen. This XML file contains Alloy (Appcelerator's GUI SDK) nodes that gets processed to show UI elements such as textboxes, images, and buttons. This works in much the same way as HTML files in PhoneGap. The index.js file, found within the controller folder, is where all of the app logic was placed for the view file of the same name. UI elements can also be created from within the controller files. This allows the developer to programmatically create UI elements dynamically. The model folder was created to store all files that help to build and store data. This includes JavaScript helper classes, database files, or any other necessary files used to model the apps data. The final

two folders, assets and styles, were used to store media and UI styles. The styles folder had two files, app.tss and index.tss. This allows developers to create styles in the app.tss file, which gets shared across all views, and the index.tss for specific views.

Unlike PhoneGap, Appcelerator© didn't provide a downloadable app to view a live preview of your application. Instead, Appcelerator© Studio allows developers to use its Live View feature to install and run the app on your device. An image of the Live View option can be seen in Figure 7. The Live View feature ensured that each time I saved my changes inside of the IDE, the app refreshes to show the changes that was just made. However, when installing with the Live View option enabled, the device must be connected to the same Wi-Fi network as the computer and connected via a USB cable. The Live View option also prevented the device from being able to run the app after disconnecting from the computer. In order to continue to use the app after disconnecting, I needed to disable Live View and reinstall the app to the device.

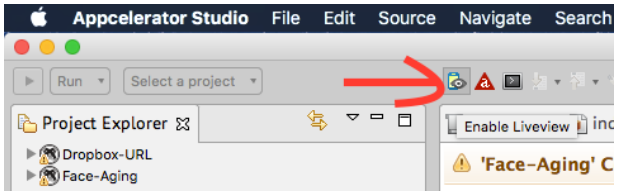

Figure 6: Appcelerator Studio's LiveView option

While PhoneGap© required plugins to implement most features, Appcelerator© has packaged nearly everything that was needed inside of its Titanium SDK. Appcelerator© can still accept plugins, although the process of installing these aren't nearly as simple as the CLI command used with PhoneGap©. There appeared to be multiple ways to install plugins, and the way that is required is based on how the developer chose to implement the plugin. One way that this is implemented is by

downloading a zip file of the plugin from GitHub, copying it to the "/Library/Application Support/Titanium" directory, and then adding it to tiapp.xml file with the module tags ("<module version="0.1">com.coronalabs.coronacards</module>" is an example of this). Another installation method is by choosing the "install new software" option under the "help" menu item. From here, I needed to enter a URL to the plugin's update site. Often I was never able to find this URL (GitHub's repo .gist URL doesn't work for this). Usually downloading a zip file from GitHub and attempting to use the first method proved most effective. I installed a couple of plugins during my experience with Appcelerator, however I later ended up removing them and working solely from the Titanium API.

Development Process and Experience

App 1 – Face-Aging Uploader

I started my Appcelerator© development experience with App 1 (Face-Aging Uploader). In total, the Face-Aging app took approximately 12 hours to complete compared to 14.5 hours using PhoneGap©. To begin, I started by downloading and installing the Appcelerator© Studio IDE. In order to download the IDE, I first had to sign up for a free account on Appcelerator© website. The installation was fairly easy, however installing the Android SDK, which is necessary for Android development, took quite awhile to download due to its large size.

After getting Appcelerator© Studio set up, I started developing the GUI for the application. By running the app with Appcelerator©'s Live View option enabled, I was able to quickly see the changes made while editing the UI. Appcelerator© provided three different layout styles to choose. These included the following: composite, vertical, and

horizontal. Composite, also known as absolute, is the default layout style within Titanium and is based entirely on five positioning attributes – top, bottom, left, right, and center. These work in a very similar way to the same respective attributes found in CSS absolute positioning. By giving these attributes a value (number of pixels), developers can accurately place the elements exactly where they want them. Vertical positioning lays out the content based on the order that they appear in the XML file. The first element is placed at the top and each subsequent element is placed below it. Horizontal layout, like Vertical layout, is based on the order of elements. It works similarly to adding elements to a webpage. The first element is placed in the top left corner and subsequent elements are placed to the right of it until that row is filled and the elements drop to the next row below the original.

The XML tags that produce UI elements are all of the proprietary Titanium syntax. For example, unlike in PhoneGap© where textboxes are denoted by "input" tags with the "type" attribute set to "text", Appcelerator© required the simpler "textbox" tag. During my experience with Appcelerator©, these different XML tags were often simpler and easier to remember than the standard HTML tags that are used within PhoneGap. However, while the tags were often simpler, their functional and styling attributes were often a bit more difficult to remember than that of HTML. I often found myself needing to look up the correct attribute that corresponds to the HTML attribute that I was trying to add. For example, to add the HTML "placeholder" equivalent to a textbox, users must instead use the "hintText" attribute.

I soon ventured into the app logic part of my application. I found that some standard JavaScript would still work within Appcelerator©, while other things had to be called with a Titanium equivalent. Examples of these are the JavaScript alert() and

console.log() functions. Alert() has been linked within the Titanium SDK to bring up a standard alert box. Console.log(), on the other hand, isn't automatically linked. It instead required me to use the Titanium.API.log() function or the Titanium.API.info() function.

I was also able get the app installed on my Android device. Appcelerator© Studio required some setup such as giving it a pointer to the installed Java JRE. This however required an older version of the JRE than the one officially distributed by Oracle. I had to download the required JRE from a third-party site in order to get it to work correctly with the Android SDK. The IDE also required that I install the Android 6.0 SDK despite the fact that my device runs Android 5.0.2. I first tried to run the app with Appcelerator's built-in Android emulator. During my tests, this emulator would often take a very long time to open and start-up. It would always take so long that the IDE would timeout the install request and cause the build to fail. Installing apps on the actual device worked much more reliably.

The differences between iOS and Android caused several unique issues for this app. The first issue occurred with the Twitter™ integration. Appcelerator© provides built-in Twitter™ integration for iOS. However, unlike the the PhoneGap© version, this method required setting up a developers account with Twitter™ and asking the user for permissions to tweet from their account. To attempt to find a better way to handle this feature, I started looking for a way to tweet in the same fashion as the PhoneGap© app. I was only able to find one plugin to provide this functionality, however it had not been updated in at least two years. Upon installing it, it threw many build errors. This seems to have been due to a change in the either the current Appcelerator© or iOS version. Due to this, the built-in method was chosen for the app. This built-in Twitter™ sharing that Appcelerator© provides for iOS doesn't support Android. After doing more research

however, I was able to find a way to share content to Twitter™ with Appcelerator's built-in "Titanium.Android.Intent" module. Intents allow Appcelerator© to call Android's native share menu. The share menu shows all ways that the user can share their content. These options are based on the apps and services they have installed on their phone. An image of the share menu can be seen in Figure 8. This approach allows developers to harness the sharing power of the Android OS without needing to setup any kind of special permissions with Twitter™ for that user's account. This worked very similarly to the way sharing was implemented in the iOS version of the same app on PhoneGap.

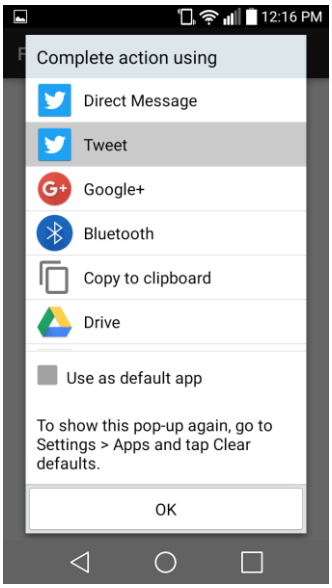

Figure 7: Android Intent Share Menu

Lastly, Appcelerator© had the same issue as PhoneGap© did with rotating images ninety-degrees to right when displaying them on the current view. In order to fix this, I was able to apply a transformation on the image. "ImageView.transform = Ti.UI.create2DMatrix({rotate:-90})" rotated the image to left ninety degrees. This caused the image to not be center. I was never able to center the image after performing the rotation, but the image was still in view. Screenshots of App 1 (built using

Appcelerator) running on iOS can be found in Appendix E and the same app running on Android in Appendix F.

App 2 – Dropbox URL

App 2 – Dropbox URL – went much more smoothly than the Face-Aging app in Appcelerator. This app only took approximately 7 hours to develop compared to the 6 hours needed for PhoneGap. This was mostly due to the simpler app logic and fewer iOS versus Android issues. Again, I started with getting most of the UI elements on the screen. This was quite easy due to being able to copy and paste some of UI code from App 1. Only a few aesthetic changes were needed.

When developing the app logic for this app, I started by researching Dropbox SDKs and plugins for sending the file to the Dropbox servers. Dropbox doesn't provide any SDKs for the Appcelerator© platform. I eventually decided to use the HTTPClient module to send the URL that the user enters to the same script on my AWS webserver that the PhoneGap©version used. Again, this was the preferred method because it freed up the phone to do other things and not use large amounts of cellular data.

Unlike App 2, the Dropbox URL app caused little to no problems from the differences between iOS and Android. This was probably due to a less complex user interface than App 1's interface. Screenshots of App 2 (built using Appcelerator) running on iOS can be found in Appendix G and the same app running on Android in Appendix H.

Performance Measurements

Several different measurements were taken to measure performance. On both the Face-Aging and Dropbox apps, the load times were tracked. This tracked how long it took for the device to render the view. Additionally, the time that it takes to open the camera and gallery functions were also tracked on the Face-Aging app. The average measurements of these metrics can seen in the following table along with the number of measurements that the metrics were derived.

The same pattern seen with the PhoneGap© platform, where Android is slower than the iPhone, can be seen in Appcelerator's metrics as well. This pattern breaks only on the load metric for app 2. This seemed to be due to an outlier in the data for this metric for the iPhone. If the count was increased to a much larger number of records, this metric would probably match the pattern that has been seen with all of the other metrics.

Metric Type	Device	App Number	Average (ms)	Count
Load	iPhone	App 1	25.75	20
Load	Android	App 1	42.3	20
Open Camera	iPhone	App 1	4.2	20
Open Camera	Android	App 1	84.75	20
Open Gallery	iPhone	App 1	3.87	22
Open Gallery	Android	App 1	26.35	20
Load	iPhone	App 2	26.85	20
Load	Android	App 2	21.4	20

Table 4: Appcelerator Performance

Access to Hardware Features

A qualitative measure of 1 to 3 was given to the Appcelerator© platform to show the tool's ability to implement some important hardware and software features. A measure of 1 was given if Appcelerator© can't currently implement the feature, 2 if it can be implemented, and a 3 was given if it is easy to implement the feature with Appcelerator. Easy to implement can be described as having readily available documentation and requires little coding to implement. The following features have been evaluated: camera, gallery, 3D Touch, and Android Widgets.

Feature	Measure
Camera	3
Gallery	3
3D Touch	2
Android Widgets	1

Table 5: Appcelerator Capabilities

Developer Community and Documentation

A search for "Appcelerator" and other related terms on StackOverflow.com produced a little over 25,200 different posts. A search for "Appcelerator© tutorial" and other related terms on YouTube yielded only about 8,000 tutorial videos. Appcelerator©, unlike PhoneGap, provided a lot of documentation articles and tutorials on its official documentation website. Most functionality was built into the Titanium SDK and didn't require plugins. However, if third-party plugins are needed, you have to rely on developers writing their own documentation on GitHub like PhoneGap© plugins.

Chapter 5: Conclusion

UI Comparison – iOS

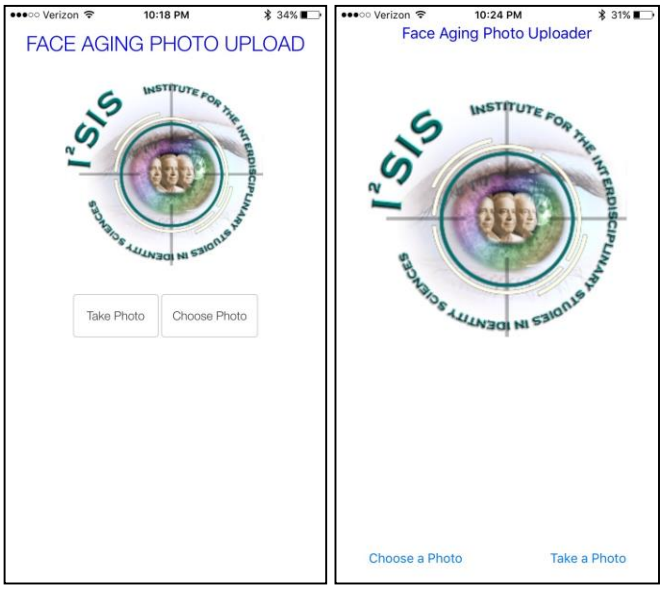

Figure 8: App 1 – Face-Aging:
Left: PhoneGap Right: Appcelerator

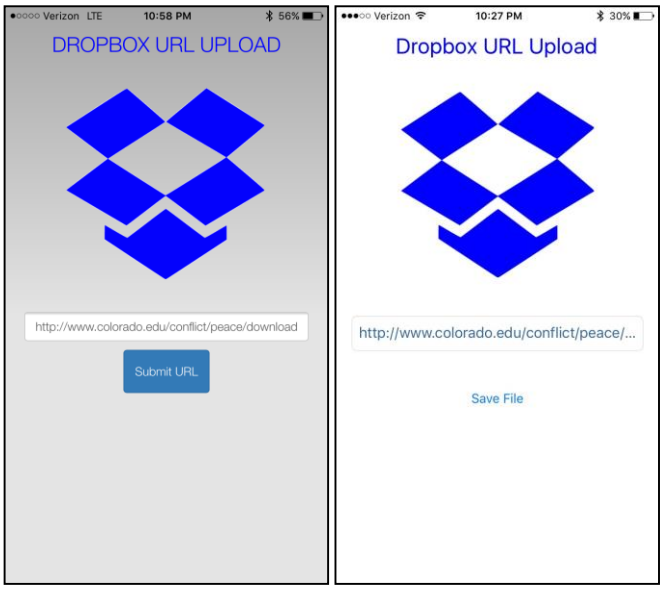

Figure 9: App 1 – Face-Aging:
Left: PhoneGap Right: Appcelerator

UI Comparison – Android

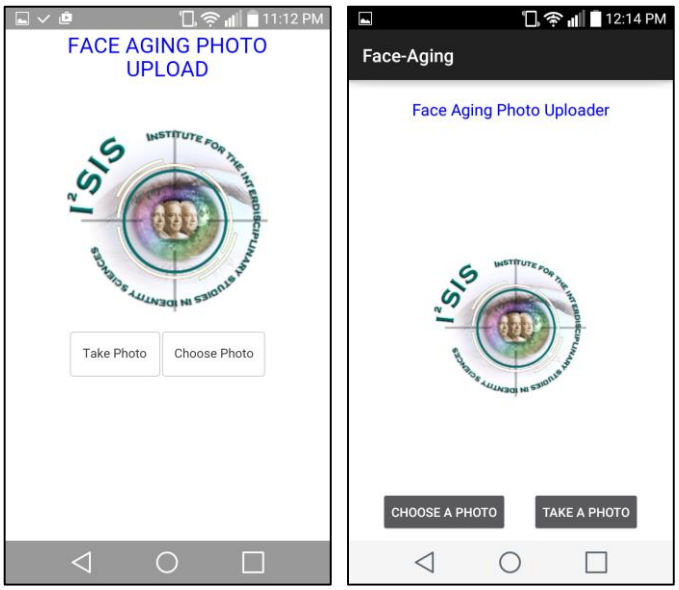

Figure 10: App 1 – Face-Aging:
Left: PhoneGap Right: Appcelerator

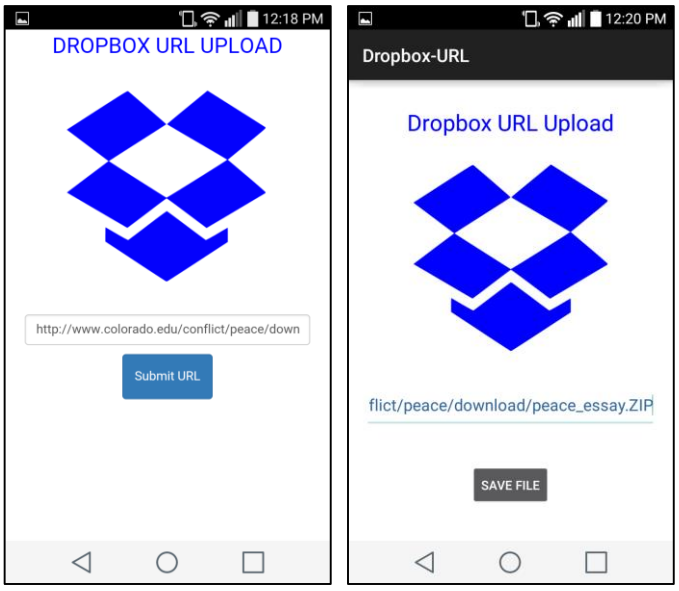

Figure 11: App 1 – Face-Aging:
Left: PhoneGap Right: Appcelerator

PhoneGap

Overall, PhoneGap© was quite easy to work with and is a great way for new mobile developers to enter the world of mobile development. PhoneGap© has several qualities that I liked and thought were quite useful for new mobile developers. These qualities are listed here:

- HTML, CSS, & JavaScript syntax is used throughout
- Large assortment of third-party plugins
- Ability to use web design plugins
- Large and knowledgeable developer community

The use of HTML, CSS, and JavaScript syntax over proprietary syntax made PhoneGap© extremely easy to start developing with as a new mobile app developer. Anyone who has any knowledge whatsoever of web development, will find PhoneGap© very familiar and comfortable to use. Even if the developer doesn't know anything about web development, great resources that are already available for web developers make for great documentation for PhoneGap© developers as well. Secondly, due to its popularity and maturity, there were a large number third-party plugins available for use. However, some of these plugins were out-of-date or didn't provide good documentation. In addition, a lot of UI features and functionality could be implemented with the help of web development plugins, due to their similarity. Lastly, PhoneGap's© popularity has blessed it with a large developer community on the web. The sites StackOverflow and YouTube is filled with great help and tutorials for the PhoneGap© platform. These came in very handy when experiencing trouble.

Based on this study, PhoneGap© also had a number of flaws that users might run into when working with the platform. Some of these are listed here:

- Lack of official documentation
- Plugins were often outdated or undocumented
- Lack of a powerful IDE
    - Inability to easily install on devices
    - Complete lack of any debugging features

As mentioned before, most functionality for PhoneGap© was provided by third-party plugins. Due to this, PhoneGap© provided very little documentation other than what is needed for installation and startup. This means that nearly all documentation is written by the third-party developers. This often meant that the provided documentation is either unhelpful, sparse, or simply nonexistent. Lastly, PhoneGap© didn't provide any type of code editor or IDE. Two of biggest issues that I faced while using PhoneGap©, installing apps and debugging, could be greatly alleviated by providing a powerful IDE similar to Appcelerator© Studio. An IDE could allow for installing apps on iPhone and Android devices straight from itself and bypass the idiosyncrasies of installing them through Xcode and Android Studio. An IDE would also be able to add some debugging features or at the very least a console where developers can print out information to help with finding bugs.

Appcelerator

Overall, Appcelerator© was fairly easy to work with and, like PhoneGap©, is also a good solution for new mobile developers. Appcelerator© had a few qualities that may set it apart from PhoneGap©. These qualities are listed here:

- Provides a powerful IDE for development
- Automatically sets UI elements to their native design
- Renders UI faster than PhoneGap

One of the best things about Appcelerator© was its IDE. Appcelerator© Studio provided all of the necessary components to develop and test apps on both iOS and Android devices. Although the Android SDK and iOS SDK must be installed to build apps and install them on devices, all of this could be done without ever opening Xcode or Android Studio. This made running apps significantly easier than with PhoneGap©. It also provided a debugger mode and output console to make it easier to find hidden bugs. Secondly, the Titanium SDK did a fantastic job of converting most UI elements to their native counterparts to provide some homogeny to help with blending the app in with native apps on users' devices. Lastly, as originally expected and later proved with my tests, Appcelerator© can render app UI faster than PhoneGap's© built-in web browser. If creating an app with a lot of UI elements, Appcelerator© could noticeably outperform PhoneGap©.

I also came across several things that I didn't like about Appcelerator. A list of these can be found here:

- Proprietary Titanium syntax
- Smaller community than PhoneGap©

- Buggy IDE

One of the biggest flaws of Appcelerator© was the necessity of using the proprietary Titanium SDK syntax. I had to look for documentation quite a bit more with Appcelerator© just to determine the correct syntax for the desired functions. I often found myself thinking, "If I have to learn a lot of new syntax, I may as well be learning the native syntax in order to make better apps in the future." Although the syntax was fairly easy to learn, it still added another layer of information to learn before getting a fully built app. Another flaw of Appcelerator© was its smaller (compared to PhoneGap) and declining community. As the smartphone industry has matured, the Appcelerator© community has been slowly declining while interest in Apache Cordova products like PhoneGap© has skyrocketed and now plateaued. This means that the number of quality third-party plugins and community members are much lower compared to their PhoneGap© counterparts. Lastly, one of Appcelerator©'s greatest strengths, Appcelerator© Studio, was also one of its weaknesses. I found from my experience with the IDE, that although powerful, it was also quite buggy. Often when opening the program, it would ask for my password. After entering the password, it would continue to ask for at least 3 times despite entering in the correct password each time. It also would occasionally get stuck giving errors during the build process when no errors are in the app. A simple restart of the IDE would fix this issue. The most common issue with Appcelerator© Studio, however, was its tendency to freeze randomly. When this happened, I had to force the program to quit and then restart it.

PhoneGap vs. Appcelerator Summary

Performance

Metric Type	Device	App Number	PhoneGap Average (ms)	Appcelerator Average (ms)
Load	iPhone	App 1	682.55	25.75
Load	Android	App 1	908.15	42.3
Open Camera	iPhone	App 1	3.25	4.2
Open Camera	Android	App 1	64.6	84.75
Open Gallery	iPhone	App 1	1.35	3.87
Open Gallery	Android	App 1	20.8	26.35
Load	iPhone	App 2	356.65	26.85
Load	Android	App 2	857.3	21.4

Access to Hardware Features

Feature	PhoneGap Measure	Appcelerator Measure
Camera	3	3
Gallery	3	3
3D Touch	2	2
Android Widgets	1	1

Developer Community

Website	PhoneGap Measure	Appcelerator Measure
StackOverflow	46,200 posts	25,200 posts
YouTube	64,000 videos	8,000 videos

Conclusion and Recommendations

After developing two apps on both the PhoneGap© and Appcelerator© Titanium platforms, I have learned a lot of the advantages, disadvantages, and quirks that they exhibit. Before starting this project, I had only knowledge of standard programming and basic web development experience. My goal was to recommend the better solution to other hopeful mobile programmers with similar knowledge. Both Appcelerator© and PhoneGap© have their merits and are both great choices for new developers. However, my recommendation to nearly all new mobile developers would be to choose Adobe™ PhoneGap© or some other Apache Cordova framework.

PhoneGap©, with its use of HTML, CSS, and JavaScript, was the easier solution with which to develop. I was able to almost instantly start developing after setup without needing to look up documentation on how to create my desired UI. Most developers will feel much more comfortable developing on the PhoneGap© platform. Online resources, although not officially from Adobe™, were readily available, and the large number of plugins made finding necessary plugins easier despite some being outdated and lacking

great documentation. Even though Appcelerator© renders UI faster than PhoneGap©, this is rarely going to be an issue for new mobile developers unless they plan on building a very complex user interface. Developers who want to develop apps that require a lot of processing or graphical power, like mobile games or photo editors, should choose Appcelerator© or native. PhoneGap© doesn't provide the necessary power and speed for apps in these categories. Lastly, the lack of an IDE to help with installation and debugging was still a big flaw, however this can be alleviated greatly with the help of Xcode and Android Studio.

Opinion on the Future of Cross-Platform Technologies

After performing a Google™ Trends comparison on both PhoneGap© and Appcelerator, it was fairly easy to see that the popularity of these platforms has been dropping significantly in the past few years [19]. I also have been keeping up with mobile technology news in the past few years as well. In 2014, Apple™ introduced a new programming language, called Swift, that will eventually completely replace Objective-C as the primary programming languages for all Apple™ devices. Swift was based on the Objective-C language but was designed to be much easier to learn and to work with by Apple™'s developers by using less syntax. In addition, it is actually faster than its Objective-C predecessor. Apple™ has stated that executing depth-first search on a graph with 10,000 integers yields results 2.6 times faster than Objective-C [17]. This is quite impressive because languages that run at a higher level than C-based languages are often much slower than the lower-level C-based languages from which they were derived. After a year of gaining huge popularity amongst Apple™'s developers, Apple™ decided

to make Swift open-sourced. This allows non-Apple™ developers to contribute to Swift and to utilize it across the industry.

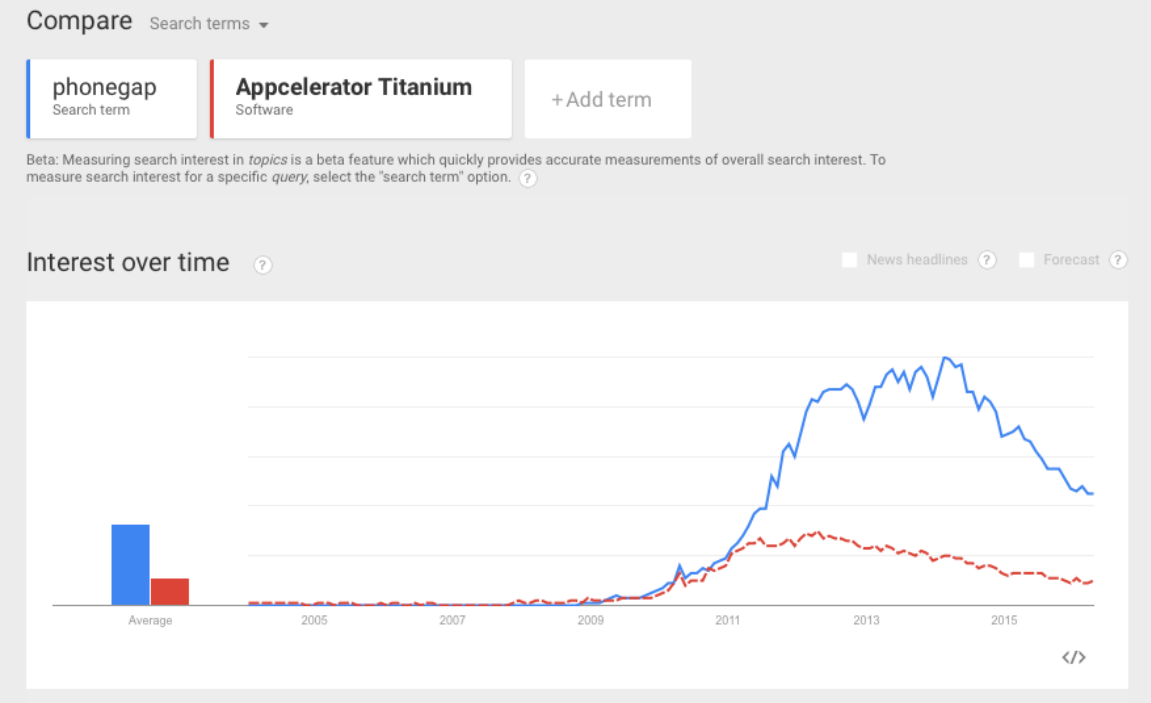

Figure 12: Google™ Trends Results (PhoneGap & Appcelerator) [19]

Due to Apple™'s new focus on Swift, this has left Google™ with trying to provide Android developers with a better programming language as well. It has even been reported recently that Google™ has looked into utilizing Swift as a possible replacement for Java for Android development [18]. If the native mobile development industry continues with this push towards simpler languages for their developers, it seems likely that many of the new developers, who would otherwise choose cross-platform over native, will instead choose to go with native development as a way to utilize native's better performance and long-term benefits. While it is unlikely that cross-platform tools will ever disappear completely, it seems that the downward trend of their popularity will continue.

References

[1] iMore, "History of iPhone: Apple™ reinvents the phone", 2015. [Online]. Available: http://www.imore.com/history-iphone-original. [Accessed: 08-Dec-2015].

[2] T. Puiu, "Your smartphone is millions of times more powerful that all of NASA's combined computing in 1969", ZME Science, 2015. [Online]. Available: http://www.zmescience.com/research/technology/smartphone-power-compared-to-apollo-432/. [Accessed: 24-Nov-2015].

[3] J. Danielsson, "Comparison Study of Cross-Platform Developing Tools for iPhone Devices", 2014.

[4] C. Mazarico and M. Carrera, "Comparison between Native and Cross-Platform Apps", 2015.

[5] F. Nayebi, J.-M. Desharnais, and A. Abran, "The state of the art of mobile application usability evaluation," 2012 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE).

[6] I. Dalmasso, S. K. Datta, C. Bonnet, and N. Nikaein, "Survey, comparison and evaluation of cross platform mobile application development tools," 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC).

[7] E. Angulo and X. Ferre, "A Case Study on Cross-Platform Development Frameworks for Mobile Applications and UX," Proceedings of the XV International Conference on Human Computer Interaction - Interacción '14.

[8] "Easily create apps using the web technologies you know and love: HTML, CSS, and JavaScript," PhoneGap. [Online]. Available at: http://www.phonegap.com/. [Accessed: 4-Jan-2016].

[9] J. Cowart, "Pros and Cons of the Top 5 Cross-Platform Tools - Developer Economics," Developer Economics, Dec-2013. [Online]. Available at: http://www.developereconomics.com/pros-cons-top-5-cross-platform-tools/. [Accessed: 10-Jan-2016].

[10] R. Gravelle, "Pros and Cons of Cross-platform Mobile Development Frameworks," Pros and Cons of Cross-platform Mobile Development Frameworks. [Online]. Available at: http://www.htmlgoodies.com/beyond/mobile/pros-and-cons-of-cross-platform-mobile-development-frameworks.html. [Accessed: 11-Jan-2016].

[11] A. Chalkley, “Why Ionic is Reigniting the Native vs HTML5 Debate - Treehouse Blog,” Treehouse Blog, Feb-2015. [Online]. Available at: http://blog.teamtreehouse.com/ionic-reigniting-native-vs-html5-debate. [Accessed: 11-Jan-2016].

[12] “Create incredible apps.,” Ionic: Advanced HTML5 Hybrid Mobile App Framework. [Online]. Available at: http://ionicframework.com/. [Accessed: 11-Jan-2016].

[13] “Cross-Platform Framework Comparison: Xamarin vs Titanium vs PhoneGap - Optimus Information Inc,” Optimus Information Inc, 2015. [Online]. Available at: http://www.optimusinfo.com/blog/cross-platform-framework-comparison-xamarin-vs-titanium-vs-phonegap/. [Accessed: 11-Jan-2016].

[14] N. Haberl, “Cross Platform Development Possibilities and drawbacks of the Xamarin platform,” 2015.

[15] “Mobile App Development & MBaaS Products | Appcelerator,” Appcelerator Inc Products, 2012. [Online]. Available at: http://www.appcelerator.com/mobile-app-development-products/. [Accessed: 11-Jan-2016].

[16] “Xamarin Get the Xamarin logo,” Mobile Application Development to Build Apps in C#. [Online]. Available at: https://xamarin.com/platform. [Accessed: 11-Jan-2016].

[17] Apple™, "Swift", 2015. [Online]. Available: http://www.Apple™.com/swift/. [Accessed: 24-Apr-2016].

[18] The Next Web, "Google™ is said to be considering Swift as a ‘first class’ language for Android", 07-Apr-2016. [Online]. Available: http://thenextweb.com/dd/2016/04/07/Google™-facebook-uber-swift/#gref. [Accessed: 24-Apr-2016].

[19] Google™ Trends, "Google™ Trends: PhoneGap and Appcelerator Titanium", 24-Apr-2016. [Online]. Available: https://www.Google™.com/trends/explore#q=phonegap%2C%20%2Fm%2F09g71rc&cmpt=q&tz=Etc%2FGMT%2B4. [Accessed: 24-Apr-2016].

Appendix A
App 1 – Face-Aging – PhoneGap - iOS

Appendix A

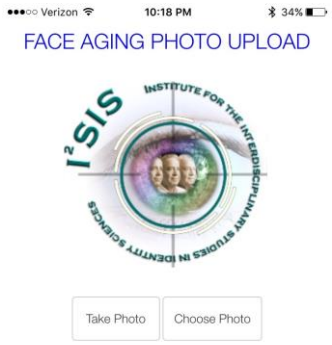

Figure 13: App 1 - PhoneGap - iOS - Start Screen

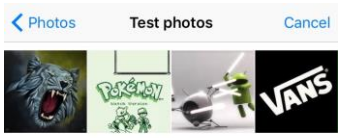

Figure 14: App 1 - PhoneGap - iOS – Gallery

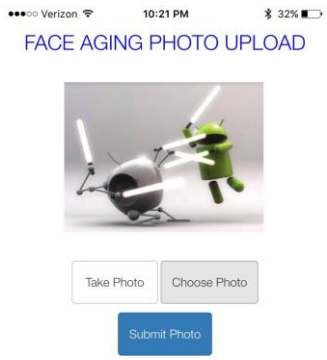

Figure 15: App 1 - PhoneGap - iOS - Image Chosen

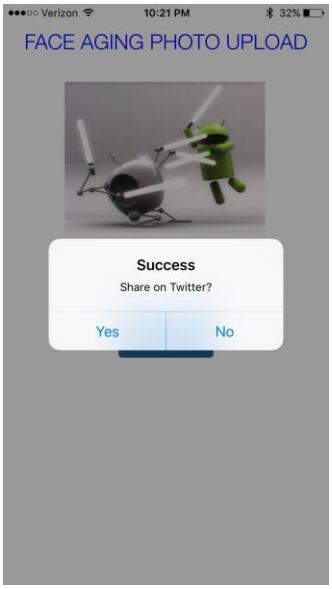

Figure 16: App 1 - PhoneGap - iOS – Submitted

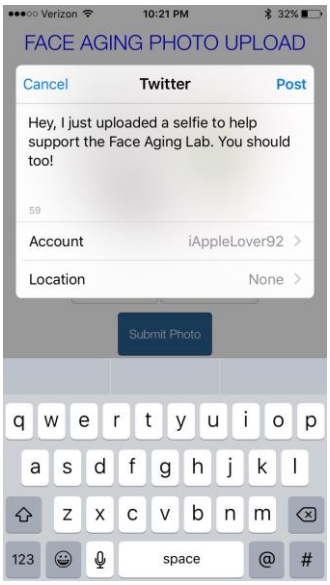

Figure 17: App 1 - PhoneGap - iOS – Tweet

Appendix B

App 1 – Face-Aging – PhoneGap – Android

Figure 18: App 1 - PhoneGap – Android – Start Screen

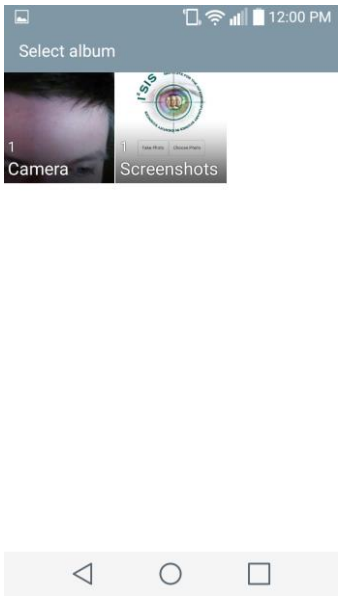

Figure 19: App 1 - PhoneGap - Android – Gallery

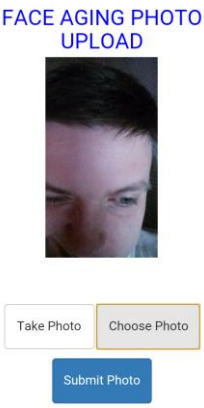

Figure 20: App 1 - PhoneGap - Android - Chosen Image

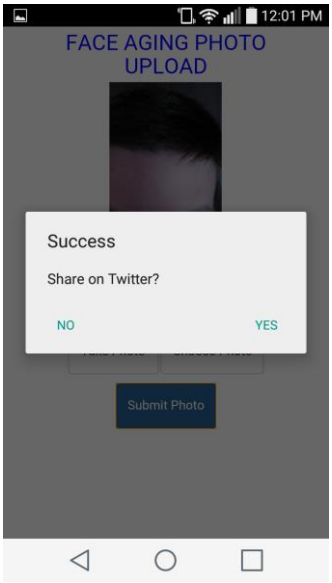

Figure 21: App 1 - PhoneGap - Android – Submitted

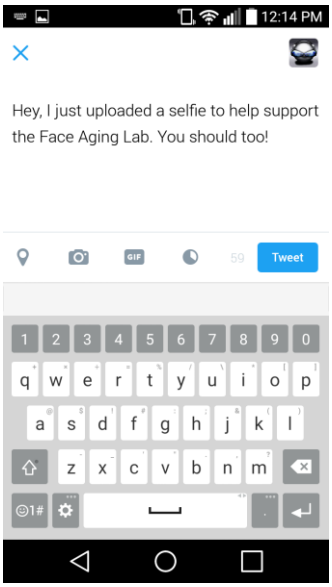

Figure 22: App 1 - PhoneGap - Android - Tweet

Appendix C

App 2 – Dropbox URL – PhoneGap – iOS

Figure 23: App 2 - PhoneGap - iOS - Start Screen

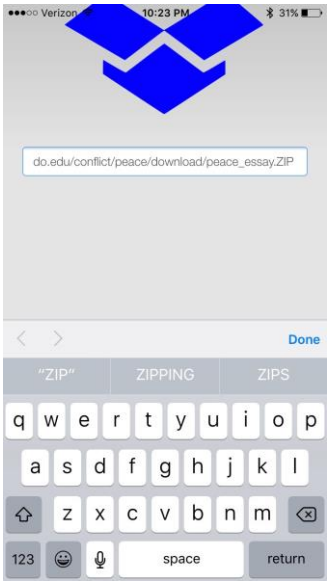

Figure 24: App 2 - PhoneGap - iOS - Edit Textbox

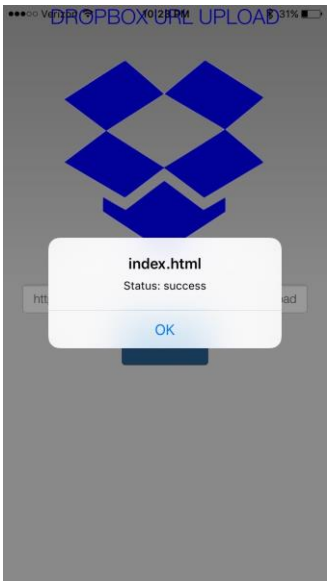

Figure 25: App 2 - PhoneGap - iOS – Submitted

Appendix D

App 2 – Dropbox – PhoneGap – Android

Figure 26: App 2 - PhoneGap - Android - Start Screen

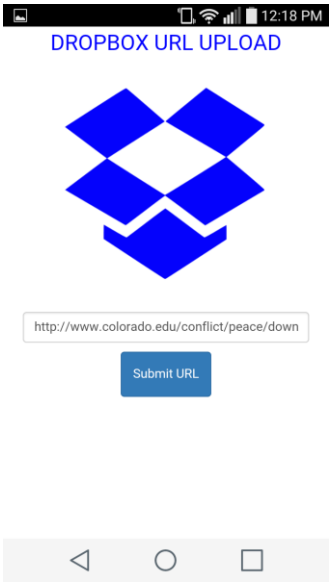

Figure 27: App 2 - PhoneGap - Android - Edit Textbox

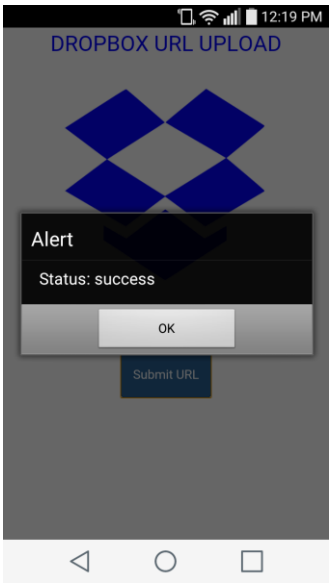

Figure 28: App 2 - PhoneGap - Android - Submitted

Appendix E

App 1 – Face-Aging – Appcelerator – iOS

Figure 29: App 1 - Appcelerator - iOS - Start Screen

Figure 30: App 1 - Appcelerator - iOS – Camera

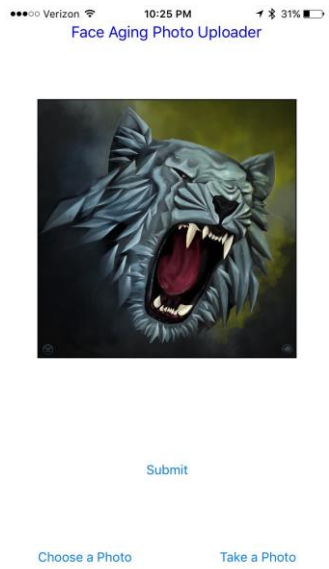

Figure 31: App 1 - Appcelerator - iOS - Chosen Image

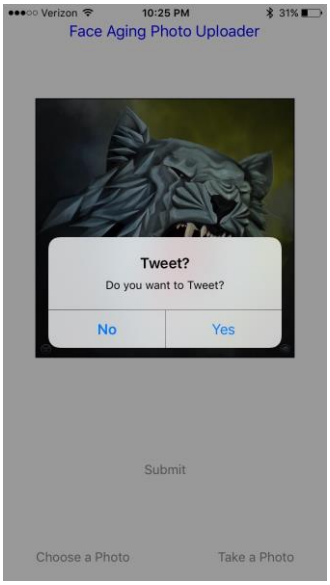

Figure 32: App 1 - Appcelerator - iOS – Submitted

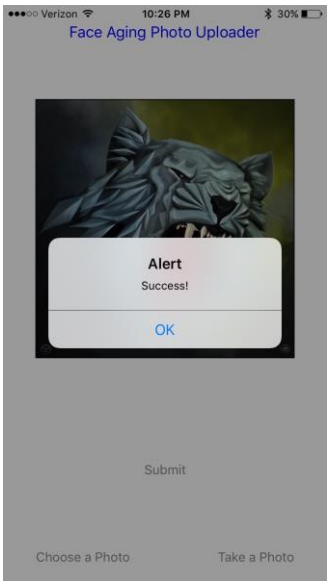

Figure 33: App 1 - Appcelerator - iOS – Tweet

Appendix F

App 1 – Face-Aging – Appcelerator – Android

Figure 34: App 1 - Appcelerator - Android - Start Screen

Figure 35: App 1 - Appcelerator - Android – Camera

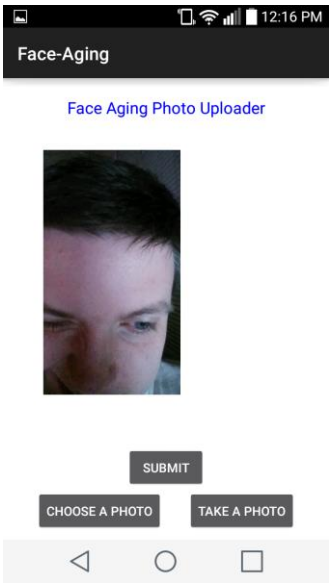

Figure 36: App 1 - Appcelerator - Android - Chosen Image

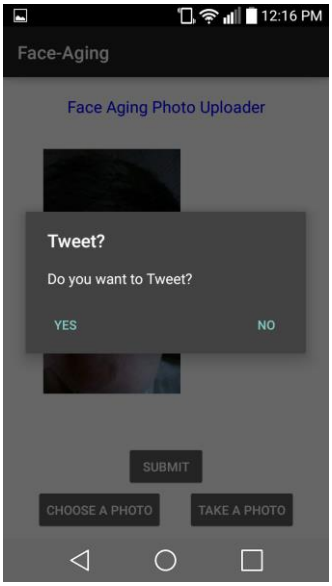

Figure 37: App 1 - Appcelerator - Android – Submitted

Figure 38: App 1 - Appcelerator - Android – Tweet

Appendix G

App 2 – Dropbox – Appcelerator – iOS

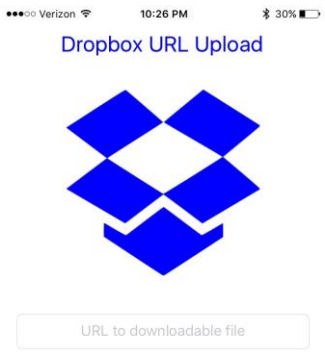

Figure 39: App 2 - Appcelerator - iOS - Start Screen

Figure 40: App 2 - Appcelerator - iOS - Edit Textbox

Figure 41: App 2 - Appcelerator - iOS - Submit Button

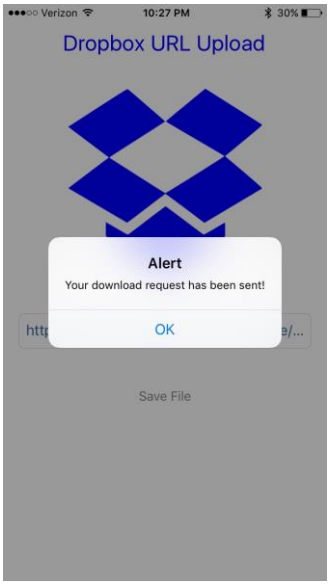

Figure 42: App 2 - Appcelerator - iOS – Submitted

Appendix H

App 2 – Dropbox – Appcelerator – Android

Figure 43: App 2 - Appcelerator - Android - Start Screen

Figure 44: App 2 - Appcelerator - Android - Edit Textbox

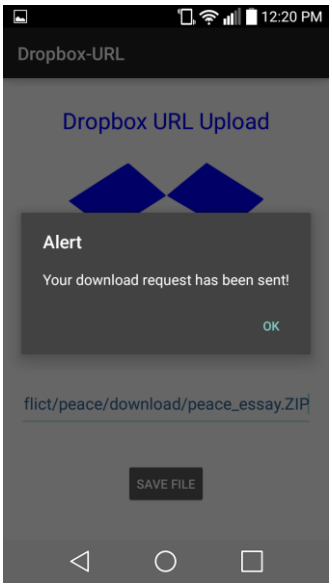

Figure 45: App 2 - Appcelerator - Android - Submitted