Viability of developing cross-platform mobile business applications using a HTML5 Mobile Framework

Joshua Morony

November 13, 2013

Supervisor: Paul Calder

Submitted to the School of Computer Science, Engineering, and Mathematics in the Faculty of Science and Engineering in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Software) at Flinders University — Adelaide Australia

Abstract

Recent and continuing advances in web technologies, including an increasing number of smart phone features being exposed to the web browser through JavaScript APIs, has increased the viability of developing cross-platform mobile applications from a single codebase. This thesis evaluates the current viability and limitations of this approach for business applications through an analysis of current capabilities and a real-life scenario case study.

The investigation shows that, given the current landscape of cross-platform development approaches, a HTML5 approach alone is not viable in general for business applications from a functionality perspective. However, in cases where the required functionality is supported by HTML5, CSS or JavaScript, a HTML5 framework can produce an application of similar quality to a natively coded application. The use of native packagers, such as the Sencha Touch Native Packager, are a plausible way for HTML5 applications to achieve functionality close to that of a native application. Although a natively packaged application could no longer be considered solely a HTML5 application, it requires little extra development effort making it more viable from a business perspective.

In support of this investigation, two mobile applications were built using the Sencha Touch framework; an application to test the defined criteria titled 'ThesisApp' and an application for the Strategiize case study.

I certify that this work does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

Signed: _____ Date: _____

Contents

1 Introduction

There are many approaches to developing cross-platform mobile applications, all of which have the very obvious advantage of only having to be developed once and working across multiple platforms. Developing a mobile application using native code involves developing the application multiple times for each platform – in Objective C for iOS, Java for Android and RIM (BlackBerry), Symbian C++ for Symbian, C# for Windows Phone and so on. Given this advantage cross-platform approaches have over the traditional native method, one might assume this would be the most widely used approach to mobile application development. The various cross-platform approaches are not without their disadvantages however — it is often much more difficult to achieve the native look, functionality and performance using a cross-platform approach.

1.1 Purpose

This thesis aims to investigate the viability of one particular approach to cross-platform mobile application development: the use of web technologies such as HTML, CSS and JavaScript to create a cross-platform mobile application from a single codebase. Viability will be determined by the approaches ability to achieve functionality whilst operating from a completely single code-base containing only HTML, CSS and JavaScript code. This investigation into the viability of the web approach was chosen because of the following reasons:

- The approach allows for a single codebase or a very high percentage of code reuse, and does not require maintaining native SDK's.

- Web technologies are rapidly evolving and an increasing number of smart phone features are being exposed through JavaScript APIs.

- Many developers are familiar with HTML, CSS and JavaScript

- This approach is valuable from a business perspective as it has the potential to save development time and cost

- The common opinion is currently in favour of natively built applications

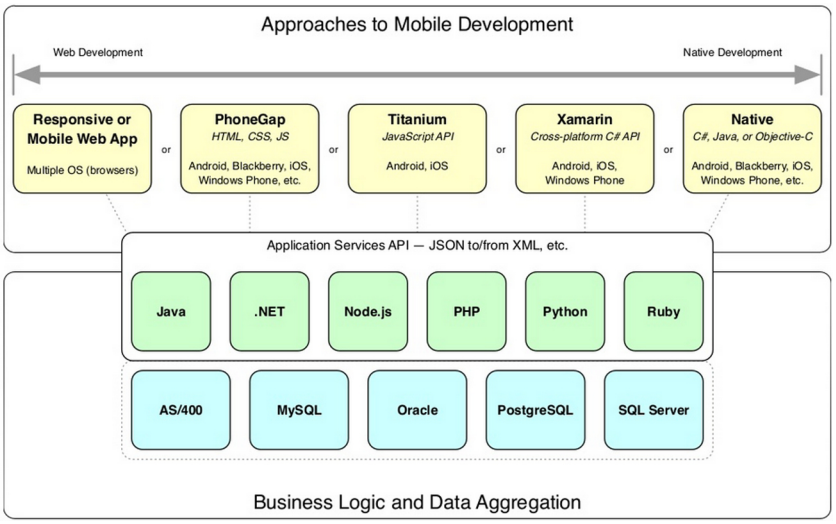

Figure 1: Mobile Application Spectrum [22]

1.2 The Environment: Web, Hybrid and Native Applications

In general, all mobile applications could be classified as being either web, hybrid, or native applications. Web applications utilise purely web code such as HTML, CSS and JavaScript, native applications utilise the native SDK of their specific platform, and hybrid applications are somewhere in between those two. Nathan Smith proposed an approaches to mobile application development spectrum at the Web Afternoon conference in 2013. [22] He arranged different cross-platform development tools on a scale from 'Web Development' to 'Native Development' with mobile web applications and native applications being at the two extremes respectively. This spectrum is illustrated in Figure 1.

To present the situation in an overly simplistic way we could state the following. Applications to the right side of the spectrum, that is native development, have greater functionality and a better performance and user experience, but no cross-platform support. Applications to the left side of the spectrum have poorer functionality, performance and user experience but have excellent cross-platform support. Approaches on the native development side of the spectrum of course include things like the native iOS and Android SDK's. Those in the middle include frameworks such as Xamarin and Titanium that utilise native code, and PhoneGap that creates a bridge between native and JavaScript API's. On the web development side there are HTML5 frameworks such as jQuery Mobile and Sencha Touch.

The native approach is obvious; native SDK's are utilised to access features of the device as intended. Similarly, approaches that use an abstracted language

and get converted into native code also work intuitively. The way in which a mobile web application works is not as intuitive, instead of utilising native SDK's the application makes use of the devices browser. We can display the application directly through the browser and it will behave as any website would, except that we have some JavaScript code powering interactions and mimicking the way a native application looks and works [23]. This approach can also be extended with a technology such as PhoneGap or another native packager to actually store the application on the device just like any other application; theoretically the user would not be able to tell the difference. In this case, a native application is built which launches a browser within the application. It is then able to load the application resources into that browser as it normally would but now we are able to interact with the applications JavaScript interface from native code [5]. This process is explained in more depth later and is illustrated in Figure 4.

When choosing between a web approach and a native approach, the question for a business then becomes: do we want portability or functionality? This thesis aims to challenge that notion, and investigate whether one can achieve both portability and functionality with a web approach that would satisfy a typical business application.

1.3 HTML5, CSS3 and JavaScript

With the release of HTML5 and CSS3, and JavaScript technology becoming the front line for the browser wars between Google, Mozilla, Microsoft and more [4], the web is constantly being pushed to new limits at a very fast rate. Figure 2 shows the increase in JavaScript performance for some of the latest releases of popular browsers run against Google's V8 benchmark suite. With a web approach, JavaScript, an interpreted language, is essentially competing against compiled native code. Generally, compiled code will run much faster, however JavaScript is now able to hold its own against these languages due to all these advances. A testament to this is the fact that HP Palm webOS 2.0 rewrote their Java services layer with node.js, which is built on Google's V8 engine. This resulted in better performance at a lower CPU cost [4].

Developers now have access to a wide range of new features that have expanded the realm of what mobile web can do. HTML5 offers features that make it an alternative to Flash, which has long had a stranglehold in this area, and allow the creation of much more interactive and engaging applications with just HTML alone [17]. CSS3 allows for a great amount of control over the appearance of an application, and is now able to create elements and effects that would have required additional images, JavaScipt or Flash in the past. CSS3 can be utilised to create drop shadows, 3D shapes, animations, opaque elements and a lot more. All these features make the web a potentially powerful tool for creating mobile applications.

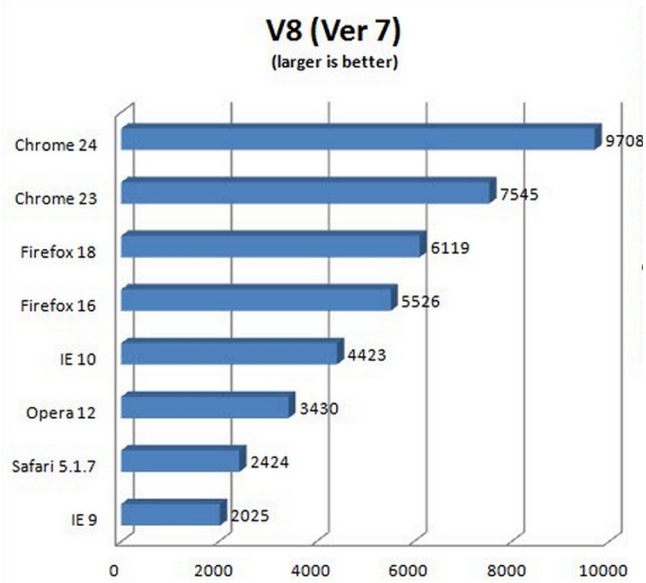

Figure 2: V8 Benchmark Suite 2013 [12]

1.4 Platform Market Share

The obvious benefit of a cross-platform approach is that the application does not need to be re-built for each device platform. An important business consideration then is of course the market share of various platforms.

As of September 2011, there was a noticeable increase in market share for Android, but iOS was still the market leader by quite a bit. The iOS category, which takes into consideration platforms including the iPhone, iPod and iPad, had a 54.6% market share — which was up 12.6% from a year prior. Java ME (Micro Edition) came in at second place with a 18.12% market share — Java ME is typically used for feature phones which is a term used to describe low-end phones that do not have the capability of a smart phone [3]. Android had the other sizeable chunk of the market at 16.26%, a variety of other platforms shared the rest of the market. These statistics are illustrated in Figure 3.

Java ME can safely be ignored for the purpose of this decision, as it is not a targetable platform for these mobile applications. That leaves the two major players, iOS and Android which account for 70.86% of the market. Whilst that is a majority, there is still a sizeable chunk left over which could be targeted by a cross-platform approach, but would be extremely expensive to cater for with a native approach.

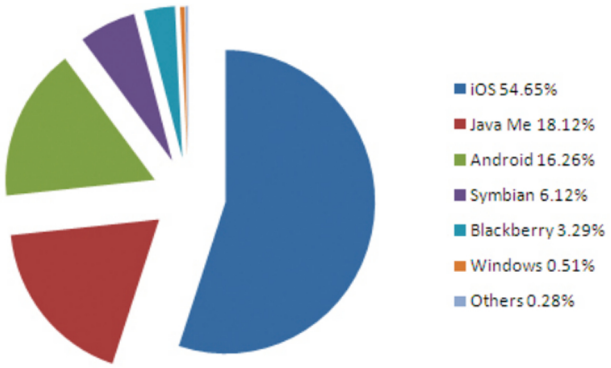

Figure 3: September 2011, Mobile OS Market Share [3]

1.5 Cross-Platform Development in a Desktop Environment

The problem of programming for multiple platforms is certainly not a new one, and has existed for a long time in other areas such as desktop application development. Often targeted desktop platforms include Windows, Linux/Unix and Mac. Similarly to cross platform mobile application development, cross platform desktop applications are generally either compiled for each platform or run on each platform using the same code base. Cross platform desktop applications also faces similar challenges as mobile, including increased difficulty of testing, using a lowest common denominator approach, and difficulty in achieving a native look and feel [6]. A variety of tools are available to create cross platform desktop applications, a few of the more common approaches are summarised below:

1.5.1 Mono

Mono is an open source .NET framework for creating cross platform desktop applications. Mono uses CLR (Common Language Runtime) and allows developers to use multiple different programming languages including C#, F#, Java, Scala, Python, JavaScript and many more. It offers a range of benefits including automatic memory management, threading and access to a comprehensive class library. MonoTouch is also available for developing cross platform mobile applications.

1.5.2 Haxe

Haxe is designed to run on and compile for multiple platforms, and can also be compiled in languages including JavaScript and PHP [16]. It utilises a single cross platform library and multiple API's specific to the platforms targeted, including Windows, Linux and Mac OS X. One specific benefit Haxe offers over some of the other approaches is that it implements a strictly typed language which requires the code to be compiled before testing in a browser and offers helpful debugging information, unlike a JavaScript approach.

1.5.3 Qt

Qt allows developers to create cross platform applications using its C++ application framework. It can be used to deploy applications to Windows, Linux, Unix and Mac OS X and has been used successfully by thousands of commercial applications [7]. As well as allowing the development of desktop applications, Qt also has support for mobile platforms and allows access to native functionality.

1.5.4 wxWidgets

wxWidgets (formerly wxWindows) provides a toolkit for cross-platform GUI programming and allows the native look and feel to be achieved across the Windows, OS X, Linux and UNIX operating systems [14]. wxWidgets differentiating feature to other approaches is that it uses the native API rather than emulating a GUI. Developers can use this approach with several different programming languages including Python, Perl and Ruby.

2 Approaches to Cross-Platform Mobile Development

There are a myriad of cross-platform development tools available for mobile applications, each with their own unique implementations and methods. The way in which the majority of cross-platform approaches achieve functionality could be categorised into one of the following:

• Frameworks that utilise web technologies to render the application through a browser

• Frameworks that interpret code at runtime

• Frameworks that offer an abstracted SDK and convert directly to native code [19]

This section will summarise and discuss some of the more commonly used approaches to developing cross-platform mobile applications.

2.1 HTML5 Mobile Frameworks

HTML5 mobile frameworks provide a way for developers to create mobile applications, similar to native mobile applications, using web technologies. There are many elements of mobile applications that users have come to accept as standard such as scrolling lists, swiping and smooth animations; recreating these to a standard comparable to a native application is complex. HTML5 mobile frameworks make this a lot easier by abstracting away the need to recreate complex features, by doing the majority of the work behind the scenes. In the case of Sencha Touch, a scrolling list can be created as follows [21]:

```
Ext.create('Ext.List', {
    fullscreen: true,
    itemTpl: '{title}',
    data: [
        { title: 'Item 1' },
        { title: 'Item 2' },
        { title: 'Item 3' },
        { title: 'Item 4' }
    ]
});
```

This code will generate a list from some data source that has smooth scrolling animations with acceleration and deceleration just as one would expect from a native application. As well as this, methods to capture events such as tapping or swiping a list item are provided. HTML5 frameworks also often provide the capability to store data from the application offline using the web browsers local storage, as well as limited access to device features such as the camera. A mobile application created this way is run directly through a smart phone's web browser (it also has the capability to be run through any web-kit supported browsers on a computer). The disadvantages to this approach include that it may be more difficult to achieve the look and feel of a native application, functionality is limited, and it is not able to be submitted through the various application stores unless additional packaging tools are used.

As HTML5 frameworks are the focus of this thesis, a range of the more popular frameworks available today will be summarised below. The framework chosen as the test case for this thesis will be elaborated on in a later section.

2.1.1 jQuery Mobile

jQuery Mobile is one of the most popular HTML5 mobile frameworks, due in part to the pervasiveness of jQuery itself — a JavaScript library used widely throughout the web. jQuery Mobile is built on top of jQuery and the jQuery UI framework and is compatible across all major smart phone platforms.

jQuery Mobile has good documentation and due to its popularity there is plenty of community support, it is easily customisable and supports a lot of 3rd party plugins. It can however be slow when compared to Sencha Touch for example, which uses hardware acceleration. It does not support MVC and performs poorly with PhoneGap.

2.1.2 jQT

jQT, formerly jQuery Touch, is a Zepto/jQuery plugin — it can be used with Zepto.js or jQuery Mobile but it is not in itself a framework. It has the advantage of being fast and has great theme support but it only works on web-kit browsers and there is little forward momentum for the plugins development.

2.1.3 Sencha Touch

Similarly to jQuery Mobile, Sencha Touch is also a very popular and widely used HTML5 mobile framework. Unlike most of the other frameworks, Sencha Touch is not based on jQuery. It is one of the fastest frameworks available, supports an MVC architecture and provides its own native app packager. The fact that Sencha Touch is based purely on JavaScript, and not on jQuery, makes the learning curve for the framework quite steep and official support is only available to those with a commercial license. A license is not required to use the framework however.

2.1.4 Intel App Framework

The Intel App Framework, formerly jQ.Mobi, is a framework provided by Intel that is comprised of three parts [11]:

- A query selector library
- A UI/UX library
- Plugins built on top of the framework

It has the advantage of being fast, lightweight, supports MVC and also offers its own native wrapper which could be used in place of something like PhoneGap for providing additional phone functionality. Disadvantages include that the default UI is not great and the documentation is poor.

2.1.5 Kendo UI

Kendo UI is also based on jQuery, and offers a simple programming interface, a MVVM framework and widgets among other things. Compared to most of the other HTML5 frameworks, Kendo UI's default UI is much better and can support a native UI look. It does however require a commercial license to use, and documentation and support for the product is lacking.

2.2 PhoneGap

PhoneGap is a hybrid approach to mobile application development; it is an open source platform that allows the creation of native mobile applications using HTML, CSS and JavaScript [1]. Unlike a web based approach, using PhoneGap requires building the application with the target platforms specific SDKs and tools, and in the case of iOS this requires the use of the XCode IDE [5]. Although there are different structures and set ups for each target platform, the application code itself can be entirely HTML, CSS and JavaScript. The advantage PhoneGap has over a standard web based approach is the additional functionality it provides. Unlike the HTML5 Mobile Frameworks and other approaches, PhoneGap does not provide an easy way to create mobile application features such as lists and animations. PhoneGap is commonly combined with another framework to handle the common user interface elements and other features of mobile applications, however it is possible to define all of this using plain HTML, CSS and JavaScript.

At run-time, the application is displayed to the user through a full-screen Web View element that renders the GUI, and also interprets the JavaScript code

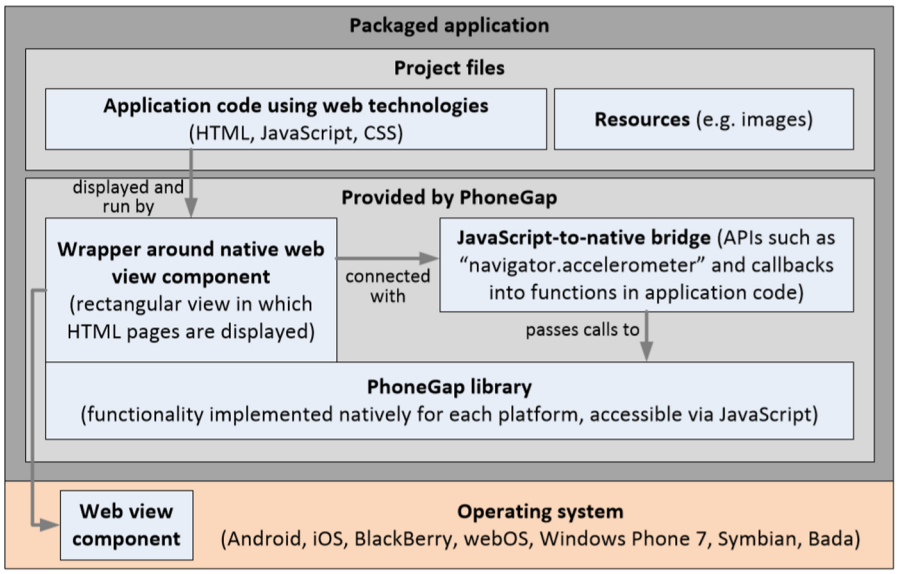

Figure 4: PhoneGap Architecture [24]

that can access the device hardware through an interface provided by PhoneGap [15]. Essentially, PhoneGap creates a bridge between the web application and the platforms native API's. PhoneGap sits in the middle, and interprets JavaScript API's into Native API calls in order to achieve functionality that is not supported by web technologies alone such as the Accelerometer, Bluetooth, Camera and Calendar. A more detailed architecture of a PhoneGap application, created by IBM, is shown in Figure 4.

In some ways, PhoneGap provides a best of both worlds approach. It primarily makes use of web technologies to build the application with the option to include native API's as well [2]. Downsides to this approach are similar to a purely web based approach, aside from the functionality aspect, and that the native SDK's still need to be used. If one wanted to build a PhoneGap application for both iOS and Android, both of the native SDK's would need to be maintained on their machine. This adds some complexity to the management of the development, but also requires the use of a Mac computer and XCode in order to build for iOS.

2.2.1 PhoneGap Build

PhoneGap Build is a cloud service offered by Adobe that simplifies the implementation of PhoneGap by abstracting away the need to maintain multiple SDKs and multiple code bases when developing an application with PhoneGap,

whilst still being able to utilise the functionality made available by PhoneGap to some extent. Users are able to upload HTML, CSS and JavaScript code to the PhoneGap Build server and have compiled application files returned for iPhone, Android, Windows Phone, Blackberry, HP, and Symbian. Compilation and the integration of SDK's is handled on their end, which makes this approach much easier than maintaining a full PhoneGap application [25].

However, in using this approach there is a sacrifice in functionality. The PhoneGap Build service generally implements an older version of PhoneGap, at the time of this writing PhoneGap Build is using version 2.7.0 whilst the latest PhoneGap version is 2.8.1. As well as this, the available plugins are limited and, unlike using the full version of PhoneGap, it is not currently possible to add your own custom plugins. Another drawback of this approach is that in order to test the application, the developer must wait for the application to compile through the PhoneGap Build servers — typically this takes a few minutes for each build. Over time this can dramatically slow down the development process.

2.3 Appcelerator Titanium

Appcelerator Titanium is a development platform for cross-platform mobile applications, and is able to be deployed to iOS, Android, Windows and Blackberry. The Titanium SDK is JavaScript based and allows for 60-90% code re-use and is claimed to be 20% faster to develop with than native code [10].

Titanium is a stand-alone SDK which allows developers to create mobile applications using JavaScript code. Titanium offers a range of API's which allow access to functions of the device as well as allowing the GUI to be defined. It uses a JavaScript interpreter to compile the application into a package which is interpreted at run time [15] as shown in Figure 5.

Benefits of using Titanium include that it uses a native user interface, there is code reuse across platforms and it is entirely JavaScript based. However common disadvantages include that applications built with the platform are often slow, testing and debugging is difficult and the Titanium framework creates a large file size. Whilst a Titanium application is developed in JavaScript, unlike the HTML5 frameworks the user interface is not rendered through the smart phones browser. The defined user interface elements are converted into actual native user interface elements when the application is run on a device [8].

The Titanium platform falls into an interesting place on the mobile application development spectrum mentioned in Section 1.2. What is interesting is that Titanium falls directly in the middle of this spectrum, it is somewhere between a web development approach and a native approach. Titanium attempts to emulate the appearance of a native application and uses native controls, but as Smith mentions it risks falling into the 'uncanny valley' [22].

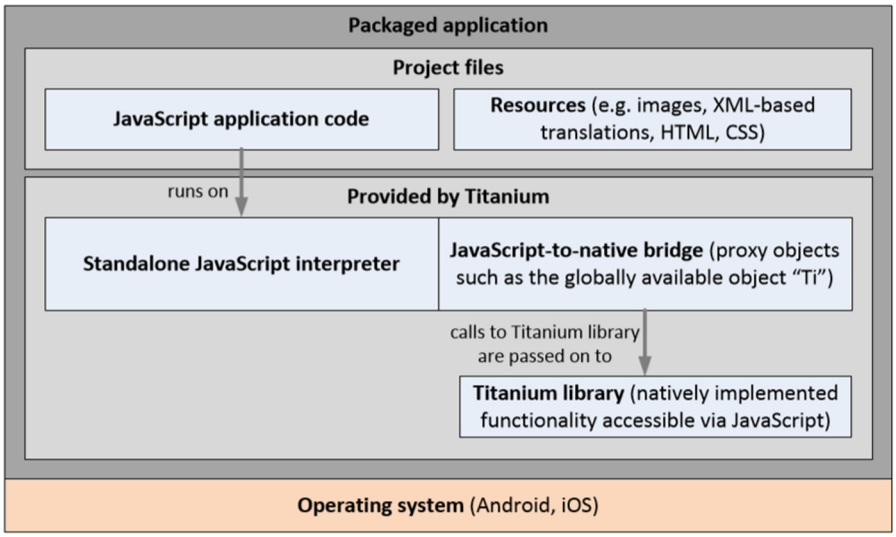

Figure 5: Titanium Architecture [24]

The 'uncanny valley' is a term used in robotics, it describes the eerie effect a robot has on a human as the robot becomes more aesthetically humanlike.As a robot becomes more human in appearance, subtle deviations from normal human behaviour become more noticeable and it has the effect of being eerie [13]. Smith notes that this same principle can be applied to applications. As Titanium is attempting to emulate native applications, these subtle deviations in performance will be noticeable, and it is important that the application performs at, at least, 60 FPS.

2.4 Xamarin

Xamarin allows developers to create iOS and Android applications in C#, as opposed to their native languages, Objective C and Java respectively. Xamarin is also a single code-base solution, as the same C# code is able to be deployed to both platforms and it has complete access to all native APIs.

Xamarin uses MonoTouch to offer a .NET based framework to create cross-platform mobile applications, and provides a one-to-one mapping from C# to the native API. In order to compile the application, due to restrictions imposed by Apple, AOT (Ahead of Time) compiliation is used instead of JIT (Just in Time) compiliation [18].

The major benefit of a framework like Xamarin is that it compiles completely to the native code of the target platform, giving it the same performance as a

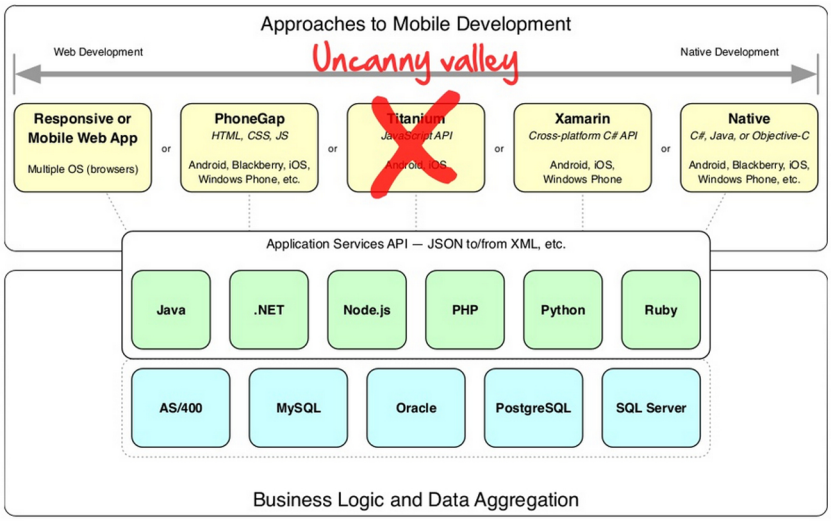

Figure 6: Uncanny Valley [22]

natively coded application. Xamarin is able to utlise all of the native API's and reuse code between platforms. It is however a commercial framework and has licensing fees associated and there is some extra bulk in the file size due to the Mono framework needing to be included [22].

2.5 Corona & Unity 3D

The Corona SDK can also be used as a cross-platform development tool. Corona excels in creating 2D mobile games, however it can also be used to develop business applications. Corona is based on the Lua scripting language and it makes use of HTML5 web views and OpenGL graphics. It's compatible with iOS and Android but also with eBook platforms such as the Kindle Fire and NOOK [20]. Although mobile games are not a consideration for this thesis, it is worth noting that a cross-platform approach for games is also possible.

Similar to Corona, Unity 3D provides and different set of functionality to most other cross-platform approaches and is typically used for 3D games, and allows developers to create applications in C# and JavaScript [20].

3 Methodology

3.1 Scope of Mobile Applications to be Evaluated

The aim of this thesis is to determine the viability of a cross-platform approach using a HTML5 Mobile Framework. The focus will be from a business perspective, which means that the most common requirements of mobile applications will be taken into consideration. In general this will include applications that need to create, read, update and delete data, utilise smart phone functionality such as the camera, GPS, and accelerometer and utilise third party services such as Facebook. The viability of creating games using this approach will not be a focus of this thesis. In the evaluation, capability in terms of functionality will be the primary concern — all other factors such as documentation, developer support and user interface will not be a focus.

This thesis investigates the viability of the use of a completely web approach. Although some HTML5 frameworks do offer their own native packagers, the focus will be on what is possible with web technologies alone and not those that have been compiled into native applications through packagers. The use of native packagers will be included in the analysis however, as this is an important factor that would impact a business decision.

3.2 Approach to Evaluation

In order to determine the viability of using HTML5 frameworks to build cross-platform mobile business applications, one particular framework will be focused on, tested and analysed. Sencha Touch was chosen as it is a good representation of what a HTML5 framework is capable of, which is explained in more detail in the following section.

3.2.1 Sencha Touch 2.2.1

Although there is no clear overall winner between the various HTML5 frameworks, Sencha Touch is a popular framework that is always considered among the best. The latest version of Sencha Touch at the time of the investigation, 2.2.1, was chosen because it:

- Takes advantage of hardware acceleration and is one of the fastest frameworks
- Provides a good UI
- Includes its own native packager
- Uses an MVC approach
- Is being actively developed and supported

Among other reasons, this makes Sencha Touch a good candidate for testing the viability of this approach.

3.2.2 Sencha Touch Native Packager

The Sencha Touch Native Packager is a tool that is included as part of a Sencha Touch installation. Whilst building an application with Sencha Touch allows for a native application like experience delivered through a web browser, the application can not access a lot of smart phone features that a native application can such as the accelerometer or the camera. As well as this, the applications can not be distributed through application stores.

By building the application using the native packager, some of these problems are circumvented. The native packager generates platform specific native code from the single code-base, which allows the application access to some native functionality as well as making it possible to submit the application through application stores and to install it on devices. This is achieved essentially through creating an instance of a native web view, and using that view to display the application as it would normally be displayed through the phone's standard browser. Although this is still running through a web browser, users will be completely unaware of this. Since it is now running as a native application, an interface can be created to start interacting with the native features of the phone.

3.2.3 Platforms

HTML5 Mobile Frameworks generally support at least the five most common smart device platforms: iOS, Android, Windows Phone, Blackberry and Symbian. The two most common smart device platforms by a large margin however are iOS and Android, as such they will be focused on and tested in this thesis.

3.2.4 Test Cases

Sencha Touch will be measured against a set of criteria established in Section 3.3 to determine the viability of the approach. The stated criteria will be executed under two different test cases: whether it is possible using Sencha Touch through a mobile web browser and whether it is possible using the Sencha Touch native packager. The successful execution of the test cases is not the only important factor, but also the means by which they are achieved; is it a viable substitution for the traditional native method? In all of these cases, sample code and a discussion of the result will be provided. In order for the stated criteria to be deemed viable, the criteria must be achieved across both the iOS and Android platforms and the solution must be reasonable to implement. What is reasonable will be a subjective judgment based on the effort required to implement the solution compared to the natively coded solution.

This thesis will also incorporate a case study involving the creation of a real-world mobile business application, Strategiize. Following the methodology of this thesis, requirements for the application will be established and tested against the Sencha Touch framework in the same way.

3.3 Criteria

The following criteria were subjectively established on the basis of considering what functionality is most commonly used by mobile application developers; it is not an exhaustive list.

3.3.1 Picture Capture

The camera is a standard feature on most smart phones, providing users the ability to capture pictures. The range of applications for the camera is almost endless, and thus it's an important feature to include.

Pass Criteria: A user is able to capture a picture using the device's camera and the pictures data is made available to the application.

3.3.2 Video Capture

Further to the cameras ability to capture photos, most devices offer the capability to capture video recordings with the camera. Although not as commonly used as still picture capture, video recordings are still often used — especially socially as internet speeds increase.

Pass Criteria: A user is able to capture a video using the devices camera and the video data is made available to the application.

3.3.3 Audio Capture

All phones of course have the ability to capture a users voice, and the ability to create audio recordings is also a common feature in mobile applications. Audio capture is often used for note taking or hands-free purposes.

Pass Criteria: A user is able to capture audio using the devices microphone and the audio data is made available to the application.

3.3.4 Vibration

Most phones have the ability to vibrate, providing haptic feedback to the user for events such as incoming phone calls and messages. Vibration is also commonly used within applications for similar purposes.

Pass Criteria: The application is able to cause the phone to vibrate.

3.3.5 Contacts

Users are able to store contacts on their device which associate a name, as well as other details, to a phone number. It is often quite valuable for an application to be able to access this list.

Pass Criteria: The application is able to access the phones contact list.

3.3.6 Data Storage

The ability to save data to the users device is critical, as it allows the application to retain its state after it has been closed.

Pass Criteria: The application can permanently store data on the device.

3.3.7 Relational Database

Mobile devices commonly offer access to SQLite which is a simplified version of SQL, and provides a relational database to mobile application.

Pass Criteria: The application has the ability to store data and retrieve data from a relational database using SQL queries.

3.3.8 Photo Access

As well as storing data on the phone, applications commonly need access to other data stored on the phone such as photos.

Pass Criteria: The application can access the data of photos stored on the device, that were not stored through the application itself.

3.3.9 Geolocation

The ability to track a devices location using GPS is a valuable feature that provides a lot of opportunities to mobile application developers.

Pass Criteria: The application can access GPS data from the device.

3.3.10 Acceleration

The accelerometer in some smart phones allows for the detection of acceleration, this is commonly used for utilising motion based gestures such as shaking.

Pass Criteria: The application is able to access data from the devices accelerometer.

3.3.11 Orientation

It is often useful for an application to detect what orientation the phone is being held at.

Pass Criteria: The application is able to detect accurate orientation changes in the device.

3.3.12 Touch

The abiliity to detect a users touch on the screen is one of the core features of a smart phone.

Pass Criteria: The application is able to detect when a user interacts with the screen via touch.

3.3.13 In App Purchases

Apple and Google offer the ability for users to make purchases from within an application using their iTunes or Google Account. This is important from a usability perspective as it allows users to make purchases without having to enter any payment details, thus making the user more likely to make a purchase.

Pass Criteria: The application is able to make a charge to the user using the devices respective in app purchase system.

3.3.14 Push Notifications

Push notifications allow applications to send an alert to a user even when they do not have the application open.

Pass Criteria: The application is able to send a push notification to a user.

3.3.15 Facebook Integration

It is common for applications to utilise 3rd party social networks to offer additional functionality such as authentication and interacting with the social network. Whilst other social networks such as Instagram are also used, Facebook is the most common.

Pass Criteria: The application is able to authenticate users using the Facebook Graph API and post a status to the users wall.

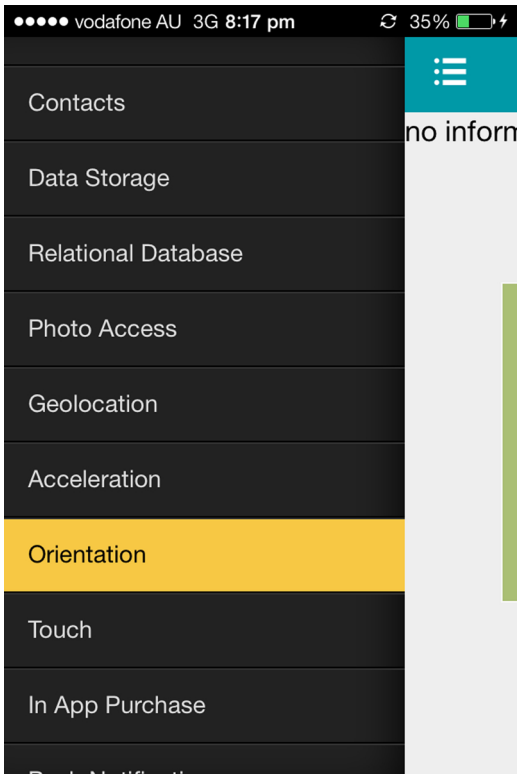

Figure 7: ThesisApp Menu

4 Evaluation of Test Criteria

The test criteria was executed using the Sencha Touch Framework (version 2.2.1) and tested on both an iOS and Android device. An application, 'ThesisApp', was created for the purpose of this investigation and was used to test the criteria. The application contains a menu that links to each individual test case as show in Figure 7. The results have been split into five distinct categories which require similar methods of access, as described below, in order to analyse the results.

4.1 Access to Device Hardware

The following test criteria have been categorised under 'Access to Device Hardware' as they require access to hardware features of the phone such as the camera. Overall, the web approach did poorly in these categories — only a subset of the criteria were executed successfully and this was only with the inclusion of the native packager.

4.1.1 Picture, Video and Audio Capture

Accessing the devices camera and taking a photo is possible with a natively packaged Sencha Touch application, but it is not possible with a web based version. Video and Audio capture is not supported with or without the native packager. We are able to use Ext.device.Camera to access the device's camera, and utilise the data passed back to the success function to handle what needs to be done with the image, this is illustrated in the following code snippet:

```
Ext.device.Camera.capture({
    success: function(image) {
        var imageView = Ext.getCmp('imageContainer')
        imageView.setSrc(image);
    },
    quality: 75,
    width: 300,
    height: 300,
    destination: 'data',
    source: 'camera',
    encoding: 'jpg'
});
```

The experience for the user in taking a photo is identical to that of a native application, as the device's default camera is launched and the photo data is passed back into the requesting application.

4.1.2 Vibration

The application is only able to trigger the users device to vibrate when it has been built with the native packager. Whilst it is possible to implement the vibration code for the web based version, the result is a shaking animation on the screen but the device itself is unaffected. Ext.device.Notification is used in order to access the vibration feature, and is triggered simply with:

```
Ext.device.Notification.vibrate();
```

A limitation of this approach is that, unlike a natively coded application, the length of the vibration can not be controlled.

4.2 Access to Sensors

This category was separated from the other hardware features of the phone as it involves hardware that 'sense' many different types of input available to the phone outside of the typical user input. The web approach performed fairly well in this category, with location services and some orientation and touch data being available but it is still lacking somewhat.

4.2.1 Geolocation

We are able to successfully retrieve the users latitude and longitude through both the web based and natively packaged approaches. Ext.device.Geolocation is utilised to retrieve the data from the device's in built GPS. The following code example demonstrates how one can retrieve the users current position:

```
Ext.device.Geolocation.getCurrentPosition({
    success: function(position) {
        alert("Latitude: " + position.coords.latitude);
        alert("Longitude: " + position.coords.longitude);
    },
    failure: function() {
        alert('Could not get location');
    }
});
```

Sencha touch also provides the ability to use the watchPosition function which periodically triggers updates to the users location.

4.2.2 Acceleration

Sencha Touch does not allow access to accelerometer data in any way.

4.2.3 Orientation

Sencha Touch documents two ways for interacting with the orientation sensor. Ext.Viewport can be utilised to detect 90 degree changes in orientation, to determine how the application should be displayed on the screen — portrait or landscape for example. With the use of the native packager, the documentation states that additional orientation information can be accessed by using Ext.device.Orientation in the following manner:

```
Ext.device.Orientation.on({
    scope: this,
    orientationchange: function(e) {
        console.log('Alpha: ', e.alpha);
        console.log('Beta: ', e.beta);
        console.log('Gamma: ', e.gamma);
    }
});
```

However, even when used under the correct conditions the preceding code fails to trigger a response and display the orientation. Despite a rigorous amount of research into this issue, no working solutions were able to be found.

4.2.4 Touch

A reasonable number of touch events are supported in Sencha Touch, including the following:

- Single Tap
- Double Tap
- Long Press
- Pinch
- Rotate
- Swipe

We can find which ever element we wish to apply the touch event to using the DOM, and then apply a listener to it to wait for the event. The following code retrieves an element with an id of "swipeIt" and sets up a swipe event:

```
var element = Ext.get("swipeIt");

element.on({
    swipe: this.onSwipe,
    scope: this
});
```

Once a user swipes the element the listener was applied to, the onSwipe function is called where we can handle the event as required. Whilst the touch events supplied by Sencha Touch are reasonable for most typical applications, it lacks the advanced gesture recognition that a natively coded application could offer [9].

4.3 Access to External Software

This category is concerned with functions that involve accessing data stored on the phone that is outside the applications own 'sandbox'. Although the test cases performed fairly in this category for the native packager, the results for the web approach were poor.

4.3.1 Contacts

It is only possible to access the devices contact list with the use of the native packager. Ext.device.Contacts is used to access the contact list and its use is officially documented, however it does not work in the way described. The code from the documentation states the list can be accessed using the following code:

```
xtype: 'list',
itemTpl: '{First} {Last}',
store: {
    fields: ['First', 'Last'],
    data: Ext.device.Contacts.getContacts()
}
```

This should supply the list with the data from the device's contact list, it does not however. The only way in which the contacts data could be retrieved was adding a success handler to the function, as demonstrated below:

```
Ext.device.Contacts.getContacts({
    success: function(contacts) {
        alert(contacts[0]);
    },
    failure: function(msg) {
        Ext.Msg.alert('Error', 'Failed to read contacts: ' + msg);
    }
});
```

However, throughout the investigation into using this method no elegant solution was found to make use of the contact data. The only way in which the contacts data could be read properly was through performing complicated procedures on all of the data passed back — to see this code please view the 'ContactsView.js' file in the supporting files.

4.3.2 Photo Access

Access to the photos already stored on the device is also only possible with the incorporation of the native packager. We are able to utilise similar code to that of the photo capture covered earlier, with slightly modified options as shown below:

```
Ext.device.Camera.capture({
    success: function(image) {
        var imageView = Ext.getCmp('imageContainer')
        imageView.setSrc(image);
    },
    destination: 'data'
});
```

The user is able to browse through their photos in the same way that they would in a native application.

4.4 Integration with Supporting Services

This category looks at the support this approach has for integrating with third party services. There are a wide range of these services available but the most commonly used have been focused on. Again, with the inclusion of the native packager there is reasonable support for these features. The web approach did perform considerably worse in this category, however depending on circumstances it still may be a viable approach.

4.4.1 Push Notifications

Whilst push notifications are technically supported they are not feasible to implement in general. The Sencha Touch Framework is restricted to supporting push notifications for iOS only, which of course greatly inhibits the cross platform applicability of this approach. Using this approach will also require the maintenance of the SSL certificate required to use the service on ones own server, as third party push notification services such as Urban Airship and Pushwoosh are not able to be integrated. It is however simple to register the device for push notifications if only iOS is required, illustrated by the following code snippet:

```
Ext.device.Push.register({
    type: Ext.device.Push.ALERT|Ext.device.Push.BADGE
    |Ext.device.Push.SOUND,
    success: function(token) {
        console.log('Push notification registration was successful');
    },
    failure: function(error) {
        console.log('Push notification registration was unsuccessful');
    },
    received: function(notifications) {
        console.log('A push notification was received');
    }
});
```

Most of the work for enabling push notifications is done outside of the application itself, such as acquiring the necessary certificates from the iOS Developer Portal, but the process is no different to the way it is done traditionally with a native approach.

4.4.2 Facebook

Interestingly, Sencha Touch is able to achieve Facebook integration without the native packager, but it no longer becomes possible once it has been compiled with the native packager. The only way to achieve Facebook integration with Sencha Touch alone is to rely on a web based approach using the JavaScript SDK for Facebook. This involves the user being redirected to the Facebook website to enter credentials, and then back to the requesting application. Due to the inability to launch a child browser in Sencha Touch, the ability to redirect to and from the Facebook website is lost once the application is removed from the web and packaged with the native packager. Although it is possible to integrate Facebook with the inclusion of a technology such as PhoneGap, this is not the focus of the thesis. Once the user has been authenticated, interactions with Facebook can be achieved through JavaScript calls as demonstrated by the following code which posts to a users Facebook wall:

```
function facebookWallPost() {

    var params = {
    method: 'feed',
    name: 'Example',
    link: 'http://example.com',
    picture: 'http://example.com/picture.png',
    caption: 'Example caption',
    description: 'Example description'
    };

    FB.ui(params, function(obj) { console.log(obj);});

}
```

The disadvantage of this approach however is that the user is momentarily removed from the application and asked to log in with their Facebook account details. This is an inconvenience to the user, as most native applications will utilise the native Facebook SDK which takes advantage of the Facebook application already installed on most users' phones to automatically authenticate a user.

4.4.3 In App Purchase

Sencha Touch appears to support In App Purchases for natively packaged applications, however only iOS is supported. The service should be able to be interacted with by using code such as the following:

```
loadProducts: function() {
    var list = this.getProductsList();

    Ext.device.Purchases.getProducts({
        scope: this,
        success: function(store) {
            list.setStore(store);
        },
        failure: function() {
            Ext.device.Notification.show({
                title: 'Product',
                message: 'Problem loading products'
            });
        }
```

```
    });
}
```

This code would retrieve all of the products currently set up through iTunes Connect. However, this was unable to be tested during this thesis as a relevant tax and banking information as well as a valid application must be submitted to Apple before testing could take place.

4.5 Data Storage

The final category is concerned with the storage of data on the device. The ability to save data and have it available to the user upon re-entering an application is a critical feature, and this section will focus on determining whether this is viable with the web approach.

4.5.1 Permanent Storage

Most importantly, there needs to be some way in which data can be permanently saved to the device. This particular test was executed to ensure there was some method, no matter how it was done, that this could be achieved. By default, Sencha Touch allows this through the use of the browsers local storage. We can define a model of the data that will be stored using this local storage mechanism with the following code:

```
Ext.define("thesisapp.model.LocalExample", {
    extend: "Ext.data.Model",
    config: {
        idProperty: 'id',
        fields: [
            {name: 'id', type: 'int' },
            {name: 'title', type: 'string'},
            {name: 'dateExample', type: 'date', dateFormat: 'c'}
        ]
    }
});
```

One can then retrieve records from this store through the use of various functions provided by Sencha Touch, for example the title of the first record in storage could be retrieved with the following code:

```
var record = exampleStore.getAt(0);
var title = record.get('title');
```

This method leads to a lack of flexibility in querying data, which is addressed in the following section.

4.5.2 Relational Database

As well as the local storage approach, many native applications make use of an SQLite database which grants the ability to store and query data in a similar way to a typical relational database. Sencha Touch also supports the use of a relational style database through the use of WebSQL, which is essentially a web based version of SQLite. In a similar way to how the default storage mechanism is defined, we can define a WebSQL store through the use of the SQL proxy as follows:

```
Ext.define("thesisapp.model.SQLModel", {
    extend: "Ext.data.Model",
    requires: ['Ext.data.proxy.Sql'],
    config: {
        fields: [
            "title",
            "description",
            {
                name: "dateExample",
                type: "date",
                defaultValue: new Date()
            }
        ],

        proxy: {
            type: 'sql'
        }
    }
});
```

This will have the effect of creating a table with the columns 'title', 'description' and 'dateExample'. WebSQL functions can be used in conjunction with the table now to perform operations. For example one could use the executeSql function to add a row as follows:

```
db.executeSql()"INSERT INTO example (title,description,date)
VALUES ('value', 'value', 'value'));
```

4.6 Summary of Results

The results from the execution of the test cases are summarised in Figure 8. It is obvious from this result that without the use of a native packager, the functionality that can be achieved with a web approach is very limited.

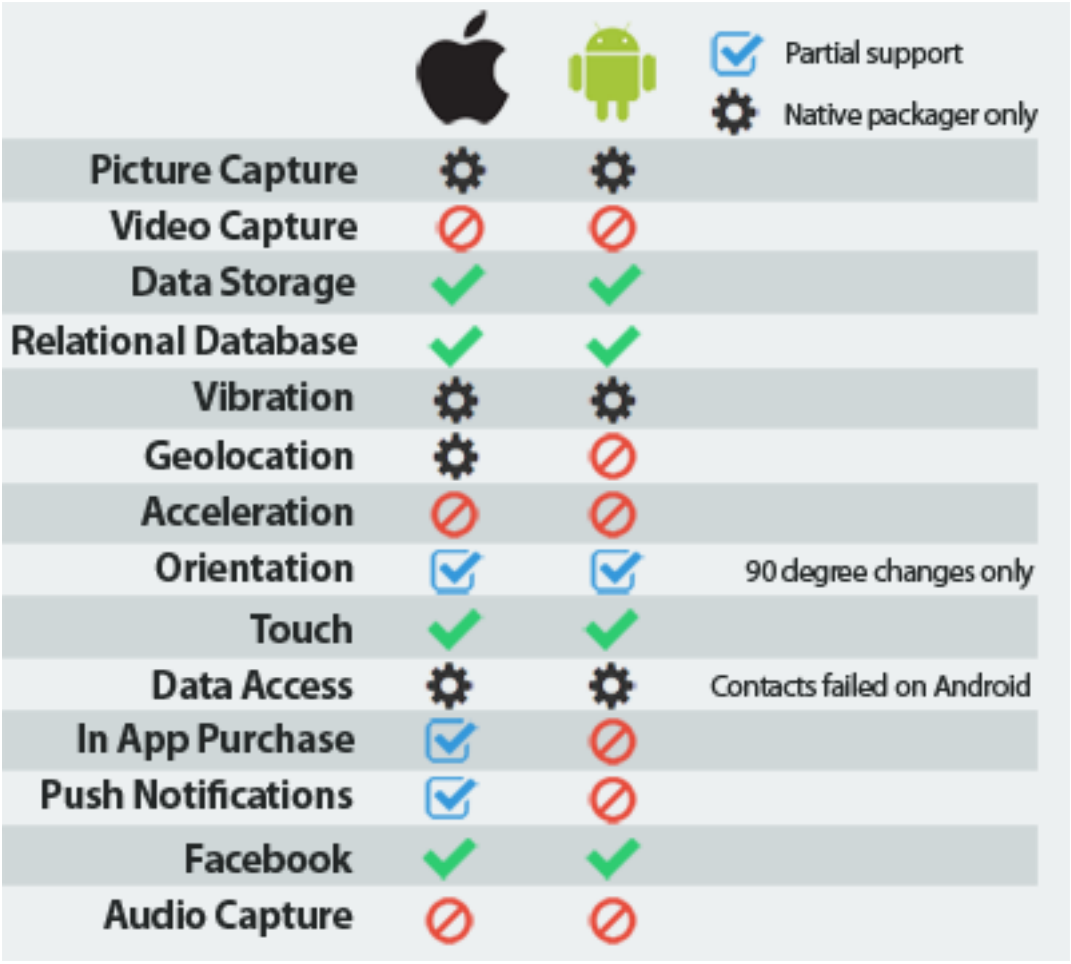

Figure 8: Results Table

If a native packager is used however, although there are still restrictions in functionality, a lot more functionality is unlocked. The restrictions are primarily due to lack of access to the device itself, as the application is sandboxed in the web view. In many cases, businesses may not require these functions but before any meaningful conclusions as to whether it is a suitable substitution for business applications in general can be drawn, more research needs to be done in defining essential criteria.

5 Case Study: Strategiize

5.1 Introduction

The purpose of this case study is to provide a real world example of a business application built using Sencha Touch. This section details the creation of a mobile application that will be used to interface with a ASP.NET web application named Strategiize.

Strategiize is an innovation management tool that aims to help companies disclose innovations and identify projects with commercial potential, as well as guide them along the commercialisation process. Strategiize Pty. Ltd., the company that created Strategiize, wishes to build a mobile application that will interface with the web application. The application is to be primarily used by managers, who are interested in viewing an overview of project details and receiving alerts.

5.2 Requirements

This section will give an overview of the requirements supplied for this project. No limitations on requirements were stipulated, and they were formed entirely around the business goals to be accomplished.

5.2.1 List of Projects

Users should be able to access a list of all projects available within their network. An option to proceed to view more detail about the project should be accessible from this list.

5.2.2 Search

Users should be able to perform a keyword search to sort through the project list.

5.2.3 Project Overview

An overview of a specific project can be displayed, providing the user with the following information for that project:

- Title
- Summary
- Team
- Scores
- Alerts

5.2.4 Project Whiteboard

In addition to the data mentioned previously, which will be displayed in the project overview, users will also be able to view the projects whiteboard. This is a creative space attached to a project that contains automatically generated suggestions and allows the project team to collaborate.

5.2.5 Alerts

A list of alerts should be displayed to the user from a convenient location such as a dashboard. Alerts will be in the form of messages that display information such as when new projects are added.

5.2.6 Style

The application should conform to the style of the web based Strategiize application.

5.3 Design

Illustrated in the following sections are the major screens of the application, along with the rationale for the approach that was taken.

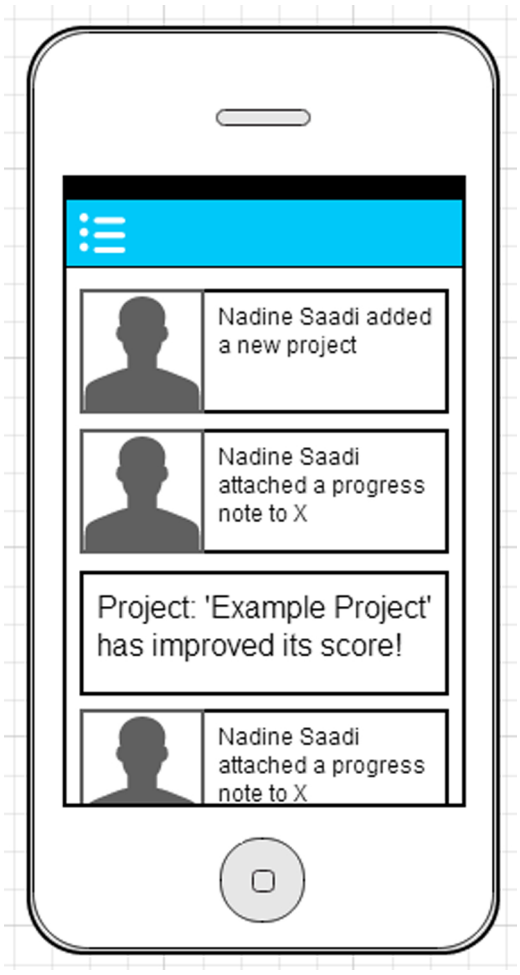

Figure 9: Dashboard

5.3.1 Dashboard

This application was to be designed with managers in mind, who are often time poor and need to absorb information quickly and efficiently. The dashboard, as the first screen that is seen once the application is launched, was used to display the list of alerts to the user. This allows the manager to see an overview of important happenings in projects at a glance, through the use of a Facebook style news feed. A wireframe for this design is shown in Figure 9.

5.3.2 Project List

The number of projects available to the manager within their network is theoretically infinite, thus the navigation needed to account for this. It was decided that a Facebook style pop in menu would be used to display a list of all projects. The user can scroll through these projects and more will be loaded in as the bottom of the list is reached if required. The search bar is also incorporated into the same area as shown in Figure 10.

5.3.3 Project Overview

Only the most immediately relevant information was needed to be shown on the overview page. The design incorporates the use of three gauges that indicate the projects overall health in the areas of market, business and technology. Additional information was to be displayed in expanding lists at the bottom of the screen, and users would also be able to launch the whiteboard from this screen.

5.4 Analysis

All the requirements were satisfied to a reasonable extent in the execution of this case study. Given difficulties in authenticating with the ASP.NET application, the mobile application is loading example data and is not connected with the real application. This is certainly possible to achieve with the framework however, and the application could easily be switched to use a different data source once authentication issues are worked out.

Aside from minor differences, the application performs and feels just like a native application. There are some noticeable differences, although subtle, in screen transitions; however these do not impact the experience in any significant way. The application also has its own custom style to suit that of the Strategiize web application, thus any UI differences are not as noticeable. Figures 12, 13

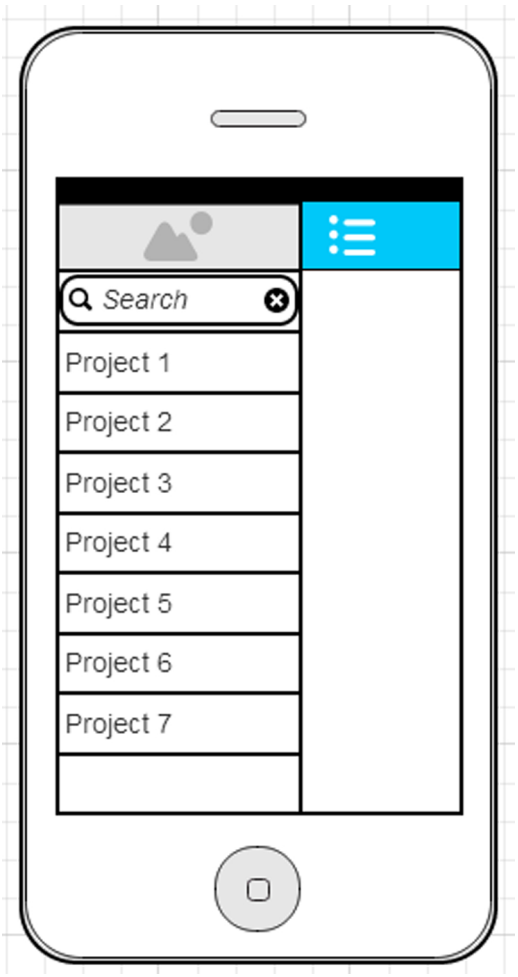

Figure 10: Project List

Figure 11: Overview

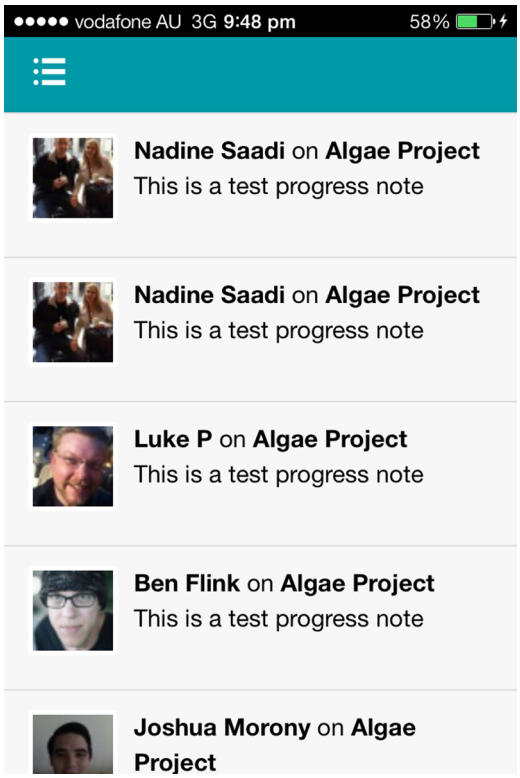

Figure 12: Dashboard

and 14 illustrate the resulting application. It would be an interesting extension to this investigation to have this application implemented both natively and using the Sencha Touch framework. Not only would a side by side comparison be insightful, but it would allow for an experiment involving actual users.

As mentioned previously it is difficult to define exactly what a "typical business application" is. However it is reasonable to assume this case study fits into the realm of business applications. Of course, this only proves the viability for one particular case and not for business applications in general. This particular case study was perhaps not the most interesting example in terms of ambitiousness of the functionality, and the outcomes were not surprising as the same functionality could be achieved in a standard web application. Nonetheless, the results are valid and it is representative of what many business applications would look to achieve.

Figure 13: Project page

6 Conclusion

HTML5 frameworks, as has been shown, are certainly a viable option for creating mobile applications of an acceptable quality. The successful execution of the Strategiize case study and the creation of 'ThesisApp' have shown this. The question posed however is of their viability as a substitution for natively coded business mobile applications. In the general case, it could be concluded that HTML5 frameworks, using the Sencha Touch framework as a representation, are not currently a viable substitution for native applications. Alone, they impose a great restriction on functionality — many features that are commonly used are not possible to implement with Sencha Touch alone. Whether HTML5 frameworks are a viable substitution for native business applications remains to be shown, and depends on the definition of a business application and the functionality it requires.

Although the lack of functionality is a major drawback, and it may not be suitable for the general case, for specific cases where required functionality

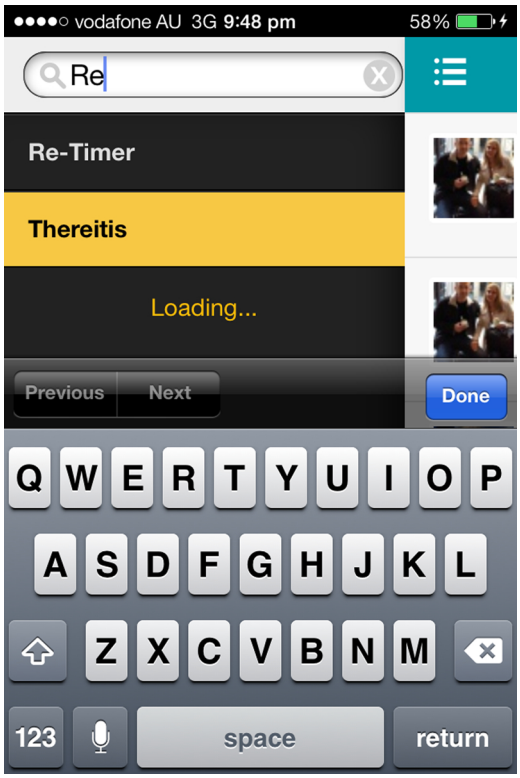

Figure 14: Search function

can be achieved it is an attractive option that offers a lot of value. Due to its use of the browser, the HTML5 approach can achieve near 100% code re-use across all the major platforms with very few drawbacks. It is harder to achieve the native look and feel in Sencha Touch, but it is difficult to notice any difference in feel or performance otherwise. Aside from some minor differences in animations, any difference between the look, feel and performance of HTML5 and native applications is negligible. However, it is important to consider the unique features of a mobile device and the possibilities of an enriched user experience that sensors such as the accelerometer, gyroscope or others could provide. HTML5 can not take full advantage of these sensors, and may provide a lesser experience to the user.

Another important discovery during this investigation was that the code did not work cross-platform 100% of the time. In some cases code executed as expected on iOS but the same code running on Android did not achieve a successful result. This is not to say that these features are not possible on an Android device, but one of the major benefits of a cross-platform approach, HTML5 especially, is that code can be reused across platforms. However, it

has been shown that this is not always the case. As well as the code that was expected to work on Android, there are still features available that are only supported for iOS such as push notifications and in app purchases - these services are offered by the Android platform but can not be taken advantage of with the Sencha Touch framework.

The HTML5 approach has many drawbacks now, but is well positioned for the future. Web technologies are becoming much more capable at a rapid pace. Features that are currenly only supported with the use of the native packager, such as the camera, photo access and vibration, can still be implemented but just retrieve dummy data, or in the case of vibration, a simulation of the function. Should such a time arrive that these features become accessible through a JavaScript API, and the trend is certainly heading in that direction, these features will be taken advantage of immediately.

6.1 Further Research

Although some research was done into what exactly the commonly used features of mobile business applications are, this could be elaborated on further and would help round out the research in this thesis. A more indepth study including a survey of current businesses and their needs would be helpful in deducing the most important functionality. In a similar sense, it would also be beneficial to gather statistics in areas such as the attitudes of developers towards HTML5 frameworks, current market share of both the native and cross-platform approaches and other relevant statistics.

Whilst this investigation focused on a functionality perspective, there remains many issues and considerations in this space - all of which warrant investigation. Issues that were not covered include:

- The availability of developers for cross-platform tools
- Adoption speed of cross-platform frameworks for new features released by the native platform
- Complexities such as debugging due to the extra layer of abstraction and lack of debugging tools
- How will an organisation adapt their processes to incorporate these tools

The use of technologies such as PhoneGap was intentionally ignored in this thesis, due to the fact that the end result is a native application, not a web application. The relationship between HTML5 frameworks and PhoneGap is obvious however, and a similar investigation into the viability of creating mobile business applications, or even applications in general, using a HTML5 framework combined with PhoneGap would be insightful.

At the conclusion of this investigation, there has already been significant advances to the technologies discussed including the release of Sencha Touch 2.3. Due to the rapidly changing nature of web technologies, and of the HTML5 frameworks themselves, the question posed in this thesis will be interesting to revisit in the near future.

References

[1] Adobe. Phonegap documentation, 2013.

[2] Sarah Allen, Vidal Graupera, and Lee Lundrigan. Phonegap. In *Pro Smartphone Cross-Platform Development*, pages 131–152. Apress, 2010.

[3] Ankit Asthana and RGS Asthana. Ios 5, android 4.0 and windows 8–a review. *IEEE Code of Ethics*, 2012.

[4] Andre Charland and Brian Leroux. Mobile application development: web vs. native. *Communications of the ACM*, 54(5):49–53, 2011.

[5] Adam M Christ. Bridging the mobile app gap. *Connectivity and the User Experience*, 11(1):27, 2011.

[6] Michael A. Cusumano and David B. Yoffie. What netscape learned from cross-platform software development. *Commun. ACM*, 42(10):72–78, October 1999.

[7] V.E. Fine. Cross-platform qt-based implementation of low level {GUI} layer of {ROOT}. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 502(23):681 – 683, 2003. ¡ce:title¿Proceedings of the {VIII} International Workshop on Advanced Computing and Analysis Techniques in Physics Research¡/ce:title¿.

[8] Gustavo Hartmann, Geoff Stead, and Asi DeGani. Cross-platform mobile development. *Tribal, Lincoln House, The Paddocks, Tech. Rep*, 2011.

[9] Shah Rukh Humayoun, Stefan Ehrhart, and Achim Ebert. Developing mobile apps using cross-platform frameworks: a case study. In *Human-Computer Interaction. Human-Centred Design Approaches, Methods, Tools, and Environments*, pages 371–380. Springer, 2013.

[10] Appcelerator Inc. Titanium sdk product page. http://www.appcelerator.com/titanium/, 2013.

[11] Intel. App framework guide, 2013.

[12] Adrian Kingsley-Hughes. The big browser benchmark (january 2013 edition), 2013.

[13] Karl F MacDorman. Subjective ratings of robot video clips for human likeness, familiarity, and eeriness: An exploration of the uncanny valley. In *ICCS/CogSci-2006 long symposium: Toward social mechanisms of android science*, pages 26–29, 2006.

[14] Koichi Momma and Fujio Izumi. An integrated three-dimensional visualization system vesta using wxwidgets. *Comm Crystallogr Comput IUCr Newsl*, 7:106–119, 2006.

[15] Julian Ohrt and Volker Turau. Cross-platform development tools for smartphone applications. 2012.

[16] S. Ortiz. Computing trends lead to new programming languages. *Computer*, 45(7):17–20, 2012.

[17] Mark Pilgrim. *Dive Into HTML5*.

[18] Arno Puder and Oren Antebi. Cross-compiling android applications to ios and windows phone 7. *Mobile Networks and Applications*, 18(1):3–21, 2013.

[19] R. Raj and S.B. Tolety. A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In *India Conference (INDICON), 2012 Annual IEEE*, pages 625–629, 2012.

[20] Ramzi N Sansour, Nidal Kafri, and Muath N Sabha. A survey on mobile multimedia application development frameworks.

[21] Sencha. Touch 2.2.1 sencha docs, 2013.

[22] Nathan Smith. Html5 can't do that. In *Web Afternoon Conference*.

[23] P Smutny. Mobile development tools and cross-platform solutions. In *Carpathian Control Conference (ICCC), 2012 13th International*, pages 653–656. IEEE, 2012.

[24] Andreas Sommer. Comparison and evaluation of cross-platform frameworks for the development of mobile business applications.

[25] Sharon Whitfield. Introduction to phonegap: Developing native applications for libraries. *The Reference Librarian*, 53(4):441–447, 2012.