Are Web Apps a Feasible Alternative to Native Apps in the Mobile Environment?

Fergus Kenny

A research Paper submitted to the University of Dublin,
in partial fulfilment of the requirements for the degree of
Master of Science Interactive Digital Media

2014

Declaration

I declare that the work described in this research paper is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed: _____

Fergus Kenny

28/02/2014

Permission to lend and/or copy

I agree that Trinity College Library may lend or copy this research Paper upon request.

Signed: _____

Fergus Kenny

28/02/2014

Acknowledgements

I would like to thank Glenn Strong and Finola O'Shea who have been very patient and supportive throughout the researching and writing of this paper.

Abstract

Within a very short timeframe, mobile devices have become an intrinsic part of everyday life. The versatility of the devices is driven by a process which allows users to run mobile apps to perform tasks. The majority of mobile content today is consumed through custom built native apps. Web apps, although popular in the desktop environment, have never enjoyed the same success in the mobile environment. Recent updates to web technologies have reignited the question of the viability of web apps on mobile devices. Understanding the differences, advantages and disadvantages of each paradigm may help in deciding if the option of web or native is the best fit for differing circumstances. The goal of this paper is to investigate if web apps are a feasible alternative to native apps in the mobile environment.

Table of Contents

Introduction

Since the introduction of the first iPhone in 2007, mobile device sales have overtaken PC sales. Gartner (2013) reports that sales of mobile devices, be that smartphone or tablet, are twice that of PC's and growing considerably every year. Tablet sales alone are expected to exceed PC sales by 2015. Although the term "Post-PC era" was used by Steve Jobs in an interview in 2007 (All Things D, 2007), this comment would appear to be very premature as StatCounter (2013) reports over 75% of internet usage is still PC based. However the growth in mobile internet usage cannot be ignored. With the continued growth in sales of mobile devices we can expect that a larger proportion of the future internet will be mobile based.

Mobile devices have two primary methods of connecting with the internet. These devices can use an internet browser similar to any PC. They also have the option to use native applications, shortened to apps. Although native apps were only introduced to users for the first time after the launch of the first iPhone, native app usage far exceeds browser usage on mobile devices. In fact, Khalaf (2013) reports that most mobile device users spend only 20% of their time on the device in the web browser.

HTML is the language used for structuring and presenting content on the World Wide Web. A new version, HTML5, is the latest standard. (W3C, 2014) HTML5 was designed to deliver everything from animation to graphics, music to movies, and can also, importantly, be used to build complicated web applications. With the introduction of HTML5 and the continued advances made in native app development, there is now a debate over which method of connecting a mobile device to the internet is best. This paper aims to address this debate, asking if web apps are now a feasible alternative to native apps in the mobile environment?

The first section of this paper addresses the state of the art in the mobile environment. It highlights the fragmentation of the mobile market and the dominance of both the Apple and Google platforms. It looks at the emergence of the app as a new breed of software application dedicated to mobile devices. Differences between platforms are highlighted, which in turn create difficulties for developing apps. It introduces the two competing app paradigms, native apps and web apps. New considerations for developing for mobile devices are also addressed. Issues such as battery, screen and processing limitations are introduced.

The second section of the paper analyses specific differences between web and native apps. Topics for consideration will include technical issues such as performance, device hardware and platform fragmentation. It looks at the distribution models and considers the engineering challenges of developing both native and web apps.

The third section of the paper will discuss the results of the analysis, highlighting specific circumstances where web apps may be a viable alternative to native apps. Instances where knowledge of native app development is low and where high performance, discoverability and access to device hardware are not important issues prove to be situations where a web app is a feasible alternative. Hybrid apps are discussed as an option to augment web apps with additional functionality without the need to convert to native app. Hybrid frameworks give web app developers access to certain device hardware. Hybrid also packages the app in a native container, allowing for distribution on the app stores. Finally instances where web apps are not a feasible alternative are discussed. Instances where high performance is key, such as graphics intensive games, and where device hardware access is essential are situations where web apps are not a feasible alternative to native apps.

The State of the Art

Background

The introduction of Apple's iPhone, in 2007, heralded a significant shift in mobile technology. (Apple, 2007) Although the term 'smartphone' dates back to the late nineteen nineties, earlier iterations of smart devices did not captivate mainstream consumers to the same extent. Companies such as RIM enjoyed moderate success with their Blackberry devices which were quite popular with businesses as they provided secure email connections and instant messaging services for employees on the move. The iPhone, however, with its multi touch interface, large touchscreen and direct finger input captivated consumers globally. Where previous smartphones required the use of buttons, keypads or stylus for input, the fluid intuitive touch interactions of the iPhone proved an instant hit.

A year later, in 2008, Google released their first Android smartphone. Android devices provided a similar interface to the iPhone, with large touchscreen interfaces and a similar look and feel to the Apple devices. The market for mobile devices has expanded further with the introduction of tablet devices, the first Android based tablets were released in 2008, with Apple following suit in 2010 with their first iPad.

Since 2007, the global market for smartphones has expanded considerably. Gartner (2013) estimate the global smartphone sales for quarter three, 2013 was just over 250 million units. Extrapolating this figure to an annual amount means that global smartphone sales annually are approximately 1 billion units. Gartner again predict that the global sales for tablet devices in 2014 will be over 263 million units. Most analysts predict that tablet sales will overtake PC sales this year, with some forecasting over 400 million tablet sales per year by 2016.

The growth in sales of mobile devices has resulted in considerable demand for programs that run on these mobile devices. When sales of mobile devices are exceeding one billion units per year, there are a lot of owners of these devices that wish to have programs available to run on their mobile devices.

The term 'app' is used to describe these mobile programs which was derived from a shortening of the term 'software application'. Apps were originally intended to provide for general productivity and informational purposes. The original iPhone also included an iPod app to allow users to play music. Preinstalled Apps on most platforms included for example a web browser, email client, calendar, contacts book and weather information.

With the initial release of the iPhone, Apple championed Web 2.0 and AJAX as the solution to the creation of apps for the iPhone. (Bell, 2011) There was no option to develop native apps available to the development community. Steve Jobs himself highlighted the ease of web development, which did not require a SDK, the security, the ease of updates and the existing knowledge within the development community as the key advantages of web apps. Developers were left with little option as Apple seemed firmly intent on not granting open access to the development of native apps.

Within a short few months Apple had backtracked on this plan. (Ritchie, 2010) There was a backlash against Apple from the development community in relation to their stance on native apps. Added to this, less than a month after the release of the iPhone, it had been jailbroken and the means of jailbreaking the device had been published online. Jailbreaking is a method of exploiting the hardware or software of a device in order to gain privileged access to the operating system beyond what the platform developer had originally intended. Developers flocked to develop native apps on jailbroken devices, all of which were unauthorised by Apple. Apple were now in a situation where they were facing a backlash from the development community for not officially allowing native apps along with a scenario where other developers were proceeding to develop natively regardless of Apple's approval. The combination of these

events led to Apple changing their stance with regards to native apps and introducing a SDK to allow for native app development.

Upon the release of the SDK, which provided tools for developing and testing native apps, developers started looking at innovative ways to develop new apps for smartphone users. This, along with increasing consumer demand, led to a massive escalation in the market for apps. New app markets emerged in many areas including mobile games, location based services using the device's GPS functionality and instant messaging tools. The SDK changed the market significantly as it went from a situation where web apps were the only option to native apps proceeding to become the dominant force in mobile apps right up to today.

Mobile Platforms

The two most common mobile platforms are Google's Android platform and Apple's iOS platform. As of the start of 2013, Android dominate the mobile market with approximately 79% of the market. Following this is iOS with 14% of the market, Microsoft Windows with over 3%, Blackberry with under 3% and the remaining platforms getting just 1% of the market. (Gartner, 2013) These remaining platforms include Bada/Tizen, Sailfish OS, Firefox OS and Symbian amongst others.

The Android platform is free and open source. (Android, 2014) The source code is released by Google under the Apache license which allows the the software platform to be freely modified and distributed by device manufacturers and the open source community. The platform is based on the Linux operating system. Apple's iOS platform is a proprietary system built on UNIX operating system specifications. (Apple, 2014) This means that Apple have complete control over the operating system platform.

As each new mobile platform comes to market, developers are also introduced with a new platform to develop apps for. Each of the major platforms differ from one another, so knowledge of one platform does not necessarily transfer to any other platform. Obviously basic programming knowledge transfers but beyond that each platform presented developers with an entirely different method of building apps. These different methods included differing languages, interfaces and development tools. When one considers how much the market for apps has expanded and how the platform market has evolved, this has presented developers with a number of challenges. Initially developers were presented with the Apple platform, and because this was the only player in the app market at the time, developers proceeded to learn how to build apps on this platform. Then Android was introduced and proceeded to overtake Apple,

developers were then rushing to learn this new platform as well. It is important to note that these two platforms have only been around since 2007 and 2008 so these are relatively new platforms which developers were expected to learn in quite a short period of time. Microsoft Windows phones (Windows Phone, 2014) adds a third layer of required knowledge to the app domain, as their platform also differs from the previous two. Added to this was the explosion in demand for apps on both platforms due to the massive growth in sales of mobile devices. Therefore it is in developers best interest to work as fast as possible to keep up with the demand for apps on multiple platforms.

This leaves developers in a difficult situation. They are presented with a market divided into multiple platforms. Each platform differs from one another in terms of knowledge required to develop native apps. Compounding these issues is the insatiable demand for apps from a rapidly growing user market.

This is where the renewed question of web apps arises. Web apps are presented as a solution to the platform divide, providing a method of supporting each platform without the need to develop specifically for each platform. Web apps also build on the pre-existing knowledge developers have of developing for the World Wide Web. Web apps are presented as an alternative to native but as of yet, we do not know if they are a viable alternative.

Web Apps

In order to understand web apps, we need to understand the web. The World Wide Web was proposed and developed by Tim Berners-Lee at CERN between 1989 and 1991. It is a system of interlinked hypertext documents accessed via the internet. These documents are formatted in a markup language called HTML (HyperText Markup Language). HTML provides a means of structuring the content of web pages, allowing for text, images, embedded objects and interactive forms. (Web Foundation, 2014)

A vitally important piece of software is required to use the World Wide Web, the first of which was again created by Berners-Lee. A web browser is a software application which reads and interprets HTML documents. The first browser had the ability to interpret eighteen HTML tags, providing a very simple web compared to today. Early standardisation of HTML involved the adoption of new tags which proved successful in browser prototypes. As competing interests became involved it was necessary to set up a group, the World Wide Web Consortium (W3C), to maintain and develop HTML standards.

An important concept regarding the web was that Berners-Lee made it royalty free. He did not patent the idea, nor demand any royalties from the use of the idea. No approvals or licenses are required. The web became a truly open platform, available to be easily adopted by anyone. There is no centralised control mechanism governing any set of rules or restrictions, only a few minimal standards that adopters were required to follow.

As the web evolved, new standards were introduced. JavaScript, introduced by the Netscape browser, is a client-side lightweight scripting language which runs in the browser. JavaScript provides a standard mechanism for performing some computational work on the client, or actual device, the browser is running on. It allows for greater user interaction, asynchronous communications with the web server, and the ability to manipulate the Document Object Model (DOM) altering the contents of the document displayed. Cascading Style Sheets (CSS) also became a standard. CSS is a style and presentation language. It can be used for controlling the font, colour and layout of web documents. CSS was envisioned to provide for a separation between the content of the web page, which is contained in the HTML, and the presentation. It also allows for multiple pages to share the same presentation formatting, reducing repetition.

With these new standards and technologies the web began to change. Prior to the year 2000, most of the web was static. Information was presented to the user. If the user wanted more information, they could follow links, if they had enough they could stop reading. Web 2.0 was a term introduced by Darcy DeNucci in 1999. (DeNucci, 1999) This does not mean we have a second 'version' of the web but it heralded a change in how we used the web. The web had moved on from being a passive medium to being an interactive medium. More users now had the ability to create content on the web. Web 2.0 gave us social networking sites, blogs, wikis, video sharing sites, podcasts and web apps.

Web 2.0 also brought us into the era of cloud computing and Software as a Service (SaaS). AJAX was developed, allowing for asynchronous communications and http requests to be performed in the background. This meant that the client browser could now communicate with the web server without the browser having to reload the entire page. Users could continue to interact with the page while new data is being retrieved and loaded. The page performed better for the user and it made the page seem like it functioned the same way as a desktop application. Google Maps is an example of this process whereby the user can navigate the map by panning and zooming in or out and the maps gets loaded as the user interacts. The Google Maps web page does not need

to refresh to respond to the users inputs, the response and the new data is automatically loaded into the page. The word processor in Google docs is another example of this mechanism. Once the page loads, it performs in a similar manner to any equivalent desktop word processor. Functionality is limited by comparison for now, but is improving all the time. Google docs is an example of SaaS. The software and all the associated data is stored in the cloud. This software is then accessed through a web browser. As a result of the improvements of Web 2.0, SaaS is rapidly becoming a popular method of software delivery for many business functions such as Human Resources, Enterprise Resource Planning, Customer Relationship Management and Productivity Tools such as Google Docs. This increase in server processing has led to the creation of enormous server farms where thousands of servers are required to maintain online services and provide the massive processing capacity in order to run these services.

As a result of the heavy emphasis on server side processing the web browser can be viewed as a 'thin-client'. This term is derived from older mainframe systems where a central server provided some service to multiple terminals, where the server provided all the processing and computational functionality with the terminals merely presenting the information. However due to the increased interactivity and reliance on JavaScript the browser as a client can be described as being a 'fatter' client, due to the amount of processing that is taking place within the browser as well as on the server side.

At the moment, we are seeing the implementation of a new HTML standard, HTML5. (W3C, 2014) This is the first major new version of HTML since HTML 4.01 in 1999. This fifth version aims to increase multimedia support with the addition of support for video and audio playback. It also brings improved semantics and more powerful APIs allowing for richer more complex applications. For this reason HTML5 is the driving force behind any move towards web apps. It is important to note that the HTML5 standard is still a work in progress, with a stable 5.0 recommendation due by the end of 2014.

So, what are web apps? To surmise, web apps are applications that run in a browser and originate from a web server. Web apps are created in browser supported languages such as HTML, JavaScript and CSS.

Native Apps

A native app is a software program designed to perform some function on a given operating system platform. The native application is built specifically for that platform alone and will not function on any other platform. The different apps are written and compiled into the machine language of each specific platform. Each platform has their

own programming languages and development environments. The programming language used to develop native apps for iOS devices is the Objective-C language which is compiled into a .ipa for installation on an iOS device. Objective C is the standard Apple development language as it is shared between their mobile platform iOS and their PC platform OSX. For the Android platform the programming language is Java which incorporates many specialised classes for dealing specifically with the Android platform. Android apps are packaged into a .apk collection for running on Android devices.

A native app is also installed directly onto the device hardware. It may access online content but the primary functionality of the app is being run from code in the device memory. It therefore needs to be installed on the device before it can be used. Most apps are relatively lightweight pieces of software but install time can vary depending on the size of the download, the speed of the connection and the speed of the device.

Android Apps

Android is a Linux based operating system developed by Google and is part of the Open Handset Alliance which promotes open standards for mobile devices. (Android, 2014) Android is currently the world's number one platform in terms of users and devices. The Google app store, called Google Play, now has over 1 millions apps available for download and has over 50 billion app downloads since it was launched in 2008. Android, being part of the Open Handset Alliance is an open source format. The platform itself is however developed in private by Google. Upon release of the new version developed in private, the source code is then made freely available to the wider community. This has resulted in a large developer community that enhance and extend the functionality of the platform.

Android runs on a 32-bit ARMv7 hardware architecture platform. Android is Linux based as it runs on a Linux kernel. On top of the Linux kernel runs middleware, libraries and API's primarily the C programming language with the application software running on Java compatible libraries. Therefore, Android applications are written using the Java programming language. This code is developed and compiled using the Android SDK (Software Development Kit). This is a piece of software released for free by Google which provides a set of development tools for creating Android apps. These tools include, tutorials, libraries, documentation, debugging and an emulator used to recreate usage of the app on different devices. The Android SDK is available for Windows, Mac and Linux machines. In order to restrict spamming apps and increase the quality of the Android apps available in the Google Play store, Google impose a one-time $25 fee to register as a developer to submit apps to the store. Android phones also allow for

installation of non market apps, so developers have the option to not use the market as a method of distributing their app, if they wish to do so.

Apple iOS apps

iOS is a Unix based operating system developed by Apple. (Apple, 2014) Originally developed specifically for the iPhone device, it has since been extended to support other Apple mobile devices such as the iPod and iPad. The iOS app store also has over 1 million apps available for download and has over 60 billion app downloads since it launched. In contrast to Google's open source policy, the iOS platform is a closed source proprietary software, that restricts anyone from modifying or distributing the software. Apple also does not license iOS on any non Apple hardware.

iOS runs on 64 and 32 bit ARM architectures. iOS contains four abstraction layers. The Core OS layer contains low level features and the kernel. The Core Services layer provide fundamental services for all apps. The media layer provides multimedia functions such as graphics, audio and video. The cocoa touch layer is the top layer and provides multitasking and gesture recognition services. The cocoa touch layer is written in the Objective-C language, which means that iOS apps also need to be developed in the Objective-C language. Apple also provide an iOS SDK to developers. Similar to the Android SDK, the iOS SDK also provides all of the development tools necessary to create iOS apps. The iOS SDK is also freely available for download. Apple however limit the SDK to only run on OSX, Apple's Macintosh PC's operating system. This means that those running Windows or Linux are excluded from running the iOS SDK. There are some workarounds for users running Windows or Linux to develop for iOS or alternatively they can use virtualisation software to run a version of OSX. Apple, like Android, also charge a fee to become part of the iPhone Developer Program. The fee currently stands at $99 per year. Without paying the fee, developers can use the SDK but they cannot submit any apps to the app store. Also, unlike Android devices, Apple devices do not have the option to allow the installation of non app store apps. Developers can apply to Apple to allow the transfer of apps to specific devices purely for testing purposes but the general distribution of non app store apps in not possible.

Mobile Device Considerations

Most application development up to the introduction of the first iPhone in 2007 had not taken place in the domain of mobile devices. The primary market for applications were desktop and laptop PCs. Some basic considerations had to take place before this knowledge could be successfully transferred over to development for mobile devices. In

reality, although the technology on mobile devices has improved exponentially in recent years, there are some very important limitations requiring detailed consideration before proceeding to develop apps for mobile devices.

The primary limitation is that one cannot guarantee that the device will always have an available connection to the internet or that the connection may be considerably slower than anticipated. In the vast majority of instances a mobile internet connection is slower than a fixed line connection. A connection in a rural location may revert to EDGE or even GPRS, which may only provide download speeds of 20 kb/s. This would differ drastically from the 3G and 4G connections available in built up areas. The non mobile developer would rarely have needed to consider the amount of data that was required (within reason), as modern fixed line connections provide a high enough bandwidth. This means that a developer would now have to carefully consider the amount of data they are transferring to the mobile device, and that depending on where the user is, the connection maybe much slower than expected. In many cases users also have limited data capacity on their mobile device contract with their service provider and going over this data cap can result in quite high charges. This means developers must be conscious of the amount of data being transferred. On top of this, the developer would also have to consider the loss of connectivity. Mobile networks frequently drop signal, be it the weather, a building in the way, a tunnel or a deep valley, connections can be lost. The developer will have to make considerations for both the loss of connectivity and potentially slower than anticipated connections.

Another vitally important limitation on any mobile device is battery usage. As mobile devices are inherently mobile, batteries are required to power the device. In order to maintain usability of mobile devices important size and weight considerations mean that the size of the battery on any given device is extremely limited. Given that most mobile devices incorporate large touchscreens and power hungry processors the battery consumption become a very important factor as the devices must provide a basic level of functioning time. This means that developers must consider processing time when writing apps as a processor heavy function could deplete the battery in an unreasonable amount of time, rendering the device useless.

Another limiting factor on mobile devices for development consideration is the limited screen size. Whereas on PC computers most screens had been getting bigger up to that point, now developers were presented with a situation where screen space was severely restricted. Compounding the screen space restriction was the touch interface of the

mobile devices which required touchable icons and buttons to be large enough to be selected, without causing the user input problems.

Finally, the developer is also presented with new user interfaces on the device. Touchscreen technology had been not widely used prior to modern smartphones. Some of the early smartphones used a stylus but a keypad had generally been the primary input method. PC developers had a background in keyboard and mouse interactions but now would have to accommodate this new input technology. Although more intuitive for users, touchscreens provided for no right click or hover functionality which a mouse previously would have offered. Developers had to become more creative with the methods in which to present these tools and options to the mobile device user.

Conclusions

This section introduced the importance of the rapidly expanding mobile device market. This expansion is driving demand for apps to run on mobile devices. Native platform specific apps are currently the dominant force in the app market. The mobile platform market is however divided between two main mobile platforms, Apple's iOS and Google's Android. Native apps are platform specific, therefore web apps are presented as a possible solution to this divide.

Developers are also faced with a number of considerations when developing for mobile devices such as connectivity, battery power, screen size and user input which may impact their choice of app.

The next section will look in more detail at the two app paradigms to see where their strengths and weaknesses lie.

Analysis of Apps

In 2007, Apple's about turn regarding web apps drastically changed the direction of mobile apps. (Ritchie, 2010) The market changed from a situation where web apps were the only option available to native apps proceeding to become the dominant force in mobile apps right up to today. The backlash from developers in 2007 regarding web apps being the only option available implies that the development community did not think that web apps were a feasible alternative to native apps. Up until their u-turn, Apple contended that web apps had a number of advantages and were a perfectly acceptable method of producing apps. Today, we are seeing a possible re-emergence of web apps as an alternative to native apps. In this section, in order to analyse the question of the viability of web apps, we will consider and contrast the advantages and disadvantages of the two app paradigms within the confines of technical, distribution and engineering considerations.

Technical Considerations

Performance

"Performance: The Hobgoblin of Software Development"
Charland and LeRoux (2011)

"0.1 second is about the limit for having the user feel that the system is reacting instantaneously, meaning that no special feedback is necessary except to display the result."
Nielsen (1993)

Vision Mobile's survey of 6,000 developers in 2013 (Vision Mobile, 2013) states that 45% of the respondents cited performance as the biggest impediment to mobile developers using web apps. Zakas (2013) defines latency as "the delay experienced between request and response". As highlighted by Nielsen, a delay of longer than 0.1 of a second will be noticeable to the user. Latency on mobile devices can arise in two different instances. Latency can be caused by network latency and execution time.

The defining factor influencing the network latency of mobile devices is the bandwidth available to the device. Although all connections suffer some latency, mobile connections suffer more than wired connections as they have more barriers influencing the connection. Zakas (2013) points to a test where the presence of a running microwave oven severely impacted latency on a network. We have also raised other

network considerations in the previous chapter regarding environmental situations such as rural locations and tunnels as circumstances which can impact networks. Countering this argument would point to the fact that mobile connectivity has advanced a huge amount since the inception of the first iPhone, which only connected to 2G GPRS and EDGE networks. These second generation mobile networks are quite slow in comparison to the networks being introduced today. 2G networks supported speeds up to 250 kbit/s in theory but practically delivered speeds closer to 150 kbit/s. Today we are seeing the introduction of 4G networks which are capable of speeds up to 25 Mbit/s. This means that users should suffer less from network latency as the networks are faster than ever.

Both native and web apps on mobile devices can be impacted by the network latency. Both types of apps can need to access information from a remote server. Web apps can suffer more from network latency in certain circumstances. Web apps could be developed in such a way as that no content of the app is stored locally on the device. This would result in the need for a permanent connection in order to access the app at all. This situation could be compounded by the web app requiring a large amount of assets to be downloaded each time it is accessed.

The second aspect of latency in an app is execution time. Execution time can impact the time it takes an app to respond to a user command. Charland and LeRoux (2011) describe execution time "as a key facet of performance". Execution time also has consequences for mobile devices as it makes demands of the device processing power and battery life. As discussed in the previous section, both processing power and battery life are quite restricted on mobile devices, especially compared to the desktop computer. Increased execution time means that the limited processing power of the device is in use for more time, which in turn uses more of the battery power. The ideal therefore, is to reduce execution time to a minimum in order to extend battery life.

A marked issue for web apps in the area of execution time is, as Crawford (2013) states, "the fact that JavaScript is slower than native code". Crawford tries to quantify the differing performance of web and native apps and concludes that native code runs approximately five times faster than JavaScript. Juntunen et al. (2013) and Corral et al. (2012) back up this assertion. Corral et al. performed a detailed study to investigate the differing performance between web and native apps. They conclude that "in 7 out of 8 routines, web-based implementation was slower than the native one".

JavaScript suffers from a performance perspective in comparison to native code as JavaScript is an interpreted programming language. JavaScript needs to be both

interpreted and then executed at runtime. This means that JavaScript runs slower than native code as it takes more processing to both interpret and execute. Juntunen et al. (2013) states that "the browser itself adds another layer of complexity between the application and the hardware". This extra level of abstraction also impacts the execution time of the JavaScript. Native apps, as we discussed in the previous section, run the same programming language as the platform they are residing in, Objective C for iOS and Java for Android, and hence do not suffer any of these interpretation delays.

Zakas (2013) reveals one solution is to "avoid using JavaScript" as much as possible and to only "use JavaScript as is absolutely necessary to accomplish the goal at hand". For rich and interactive web apps this option is not always viable. In order to improve JavaScript, the browser developers are constantly upgrading the JavaScript engines in their browsers to improve the performance of the language. (Heath, 2014)(Mathews, 2013) Researchers such as Lee et al. (2010) and Swiech and Dinda (2013) are also looking at methods to make JavaScript more efficient and less power hungry by reducing the lines of code and throttling JavaScript interpretation speeds.

Access to Hardware & Device Features

"If you can't think of a way to improve your web app using Android SDK features…..you're doing it wrong."
Reito Meier, (Meier and Mahemoff, 2011)

Meier, an Android developer at Google, is of the view that native apps are more powerful than web apps. Going beyond the performance differences raised in the previous section, he cites access to the hardware of the device as a major advantage of developing native apps over web apps. This view is backed up by Lionbridge (2012). Meier also cites the velocity at which new hardware is becoming available on mobile devices. Multi-touch, accelerometers, GPS and compass, cameras both rear and front facing, Bluetooth, gyroscopes & NFC are all features of mobile devices that have come into being in a very short space of time. With native development one can take advantage of these innovations immediately using the native development APIs. Meier highlights this as a major advantage of native apps, allowing native app developers to become market leaders by being the first to incorporate these new hardware features into innovation functionality in native apps.

This ability to immediately use new hardware features in native apps can be contrasted with the HTML standards process which web apps are bound by. With this process new functionality gets implemented into HTML standards first, followed by the browsers

implementing these standards before the functionality becomes available to web apps. This seems a reasonable argument as we are seeing new HTML5 standards being implemented for geolocation, multitouch, device orientation and camera and microphone interactions. These standards are only in the process of being implemented now and not all are supported by all browsers to date, so the standards process is delaying innovation in the web app environment compared to the native environment.

Beyond the advantage native apps have regarding the faster access to new hardware features, they also offer greater tie in with the other software and other native apps available on the device. Native apps have the ability to run in the background, listen for events and react to them, run concurrently with other apps and have the ability to call other apps as necessary.

Examples of this would be a music player app, Spotify for instance. (Spotify, 2010) The Spotify native app can be minimized and run in the background while it continues to play music from the playlist selected. The user can proceed to interact with any other app, or turn off the screen of the device and the Spotify native app will continue in the background. Grooveshark offer a similar streaming music service to Spotify with a HTML5 based browser app. Grooveshark currently have an issue (Grooveshark, 2014) with their web app as when it is running in the background it finishes playing the current track but does not proceed to play the next track on the playlist. Native apps have the ability to force the OS to keep the app running in the background whereas web apps do not have this capability. Another example of native apps interacting with other apps is the instant messaging service WhatsApp. (WhatsApp, 2014) This native app can access the contacts list on the phone and uses this list to identify other WhatsApp users which appear on the list.

Web apps by comparison have very little access or interaction with any apps, other than the browser, on the device. Most browsers offer web apps some basic interactions such as calling the dialler, email and sms apps. Browsers will also respond to URLs for sites such as YouTube, Google Maps and app stores by opening the appropriate native app. (Apple, 2014)

Meier (Meier and Mahemoff, 2011) and Dalmasso et al. (2013) also refer to offline functionality as a limiting factor of web apps. Meier refers to a situation where his native email client allows him to check and reply to his mails while on The Underground and offline and when the connection is restored, the native app has the ability to synch with the mail server and send the mails he had written while offline. Offline support,

however, is now something that is supported by HTML. Recent HTML5 updates include functionality to allow local client side storage, including database storage, application cache which allows web apps to function offline. Online & offline events allow the browser to check if the device is online or offline. This caching and offline functionality also aids performance of the app as it reduces the need for the app to be permanently online and the need to download all of the app assets each time the app loads. These issues were raised in respect of the network latency problems addressed earlier.

Fragmentation

Native app development is made more difficult due to the fragmentation of platforms. We have seen the overall market divide between the differing iOS and Android platforms. Joorabchi et al. (2013) makes specific reference to the inability of developers within their survey to reuse and or port code across platforms. Even where the language is the same, the different platforms just don't work the same way. The overarching belief from the survey is that it is best to write native apps for differing platforms from scratch.

Joorabchi et al. (2013), also highlight an increasing fragmentation within platforms. This most notably occurs on the Android platform where there is a high degree of variance between the different devices available on the platform. Within the Android platform there are numerous competing manufacturers such as Samsung, HTC, LG, Sony and Asus to name but a few. Each of these manufacturers release devices with many differing properties, from memory to CPU speed, to screen size and resolution. iOS does not suffer as significantly as there are far fewer devices and all devices on the iOS platform are produced by Apple themselves. A greater fragmentation of the version of Android running on each device is also becoming more of a problem as low end and older devices can only support older versions of Android, limiting access to newer features. At the time of writing, Google seemingly intent on solving the issue of Android fragmentation may force all new devices to support the most recent version of Android based on a leaked Google memo. (Smith, 2014)

This fragmentation has an impact on both native and web apps. Native apps are impacted as developers must now consider many differing screen sizes. The native platforms do automatically scales and resizes apps for the different screens but developers realistically need to optimize their app code as automatic scaling can lead to a bad UI experience. Respondents to Joorabchi et al. (2013) also cite the deprecation or removal of certain functionality between versions of the platform as a major issue. This is driving developers to a situation where, not only is their code base fragmented between two platforms but it is also fragmented within one of those platforms as well.

This fragmentation also has an impact on the web app. From Charland and LeRoux (2011) to Meier and Mahemoff (2011) to Lionbridge (2012), a common thread cites cross platform compatibility as a major advantage of web over native apps. As Mahemoff states, "One could not imagine a mobile device without a browser". All mobile platforms currently provide a browser app as part of the platform. An important consideration of the browser is the level of HTML supported.

Mesbah and Prasad (2011) underscore a very basic issue with browsers, cross browser compatibility. This difference arises from the differing Rendering Engines which each browser uses to read web pages and render them on screen for the user. Mesbah and Prasad (2011) use a worst case example where important functionality in the form of widget options are displayed to the user in version 8 of Internet Explorer but the same page displayed in Firefox version 3.5 fails to include the widget options thereby excluding these users for important functionality that the page provides. Other differences between browsers, such as visual differences, are less impacting on user functionality but can alter how the page appears to the user. These include how the different engines render fonts, certain widths, padding and margins.

The W3C was introduced to control the standards of HTML. The HTML standards created by the W3C are implemented as recommendations. Recommendations however by their nature are suggestions or proposals as to how best to implement HTML. Although the W3C works with all the major browser developers to develop the new standards, the browser developers can have differing timelines and implementation periods for incorporating these new functions into the browsers.

These cross browser compatibility issues are becoming more pronounced with the introduction of HTML5. HTML5 is still not at a final recommendation stage as of today. The W3C intend to have a final recommendation by the end of 2014. This means that the standards of HTML5 have not been fully finalised. They will not be finalised until "two 100% complete and fully interoperable implementations exist". The browser developers are at very different levels of implementation of HTML5, with desktop and mobile versions of the browsers also differing. HTML5test (2014), provides up to date statistics on how much of the HTML5 standard different browsers have implemented to date. They test browsers out of a score of 555 points and grade different features on its importance to web developers and how difficult they feel it is to implement these features. They state that they use the grading system so that a browser that includes a large number of easy to implement features does not rank higher than a browser that

has included much more important and harder to implement features. They acknowledge that the grading system is a personal preference highlighting the lack of a truly objective alternative. The results of this test on mobile browsers show a wide variation in scores, with no browser achieving more than 500 points in the results. The highest rank mobile browser is the Blackberry browser with a score of 491. The lowest ranked mobile browser is the Windows phone browser with a score of 332.

This test does show that the implementation of HTML5 across mobile browsers is at very different stages. This means that the portability of a web app declines with the use of HTML5 features, as a developer cannot be sure if the functionality included in the app will be available to all users of the app. Another important consideration is that users may not always have the most up to date version of their browser. This means that a browser may have implemented some new features of HTML5 but the user has yet to update or notably may not be able to update to the latest version of the browser. This is where the fragmentation of the platform impact on web apps. A situation is arising whereby older and low-end devices are outdated and unable to update to the most recent version of the browser resulting in the inability use certain functionality on web apps.

HTML and web, however have a long history of dealing with different screen sizes and resolutions. Web developers also have a history of dealing with the differing browsers and how they have implemented the HTML standards at different times. Advocates of HTML5, such as Mahemoff (Meier and Mahemoff, 2011), argue that HTML's flexibility allows for graceful degradation, whereby these differences between the newer browser versions and the older ones can be overcome by the browsers falling back and ignoring those functions it does not understand without crashing completely or presenting the user with an error message. The problem with graceful degradation being that support for older browsers is not a top priority in web development today, with the starting point to many designs concentrating on the most up to date features of the browser.

Alternatively the developer could opt for a progressive enhancement approach as advocated for by Wells and Draganova (2007) and Desruelle et al. (2011). With progressive enhancement, the opposite approach to graceful degradation is used, whereby the least capable devices are the first devices taken into account. The document is divided into layers; context, presentation and behaviour. The context layer provides the most basic markup possible in order to be useable to the most basic of browsers. The presentation and behaviour layers then enhance the page using technologies such as CSS or JavaScript. These enhancements are externally linked and

therefore do not get downloaded to the device unless they are useable to the browser. Progressive enhancement recognised a situation brought about by the introduction of mobile browsers, whereby devices were no longer getting faster and more powerful as they were in the desktop environment. Devices were now restricted in terms of speed and bandwidth with low functionality browsers.

The fragmentation of the market impacts on both web and native apps. It forces developers to work harder to optimise apps for differing devices. Web apps are not impacted by cross platform fragmentation as they are supported by all platforms but they do need a certain level of optimisation in order to make up for differing browser support. Native apps suffer considerably from both platform fragmentation as developers need to support both the Android and iOS platforms. Within the platforms, especially in Android, native apps also require anything from optimisation up to a different code base in order to overcome the fragmentation.

Distribution Considerations

"Application stores have played a crucial role in the proliferation of applications for smartphones and other mobile devices."
Juntunen et al. (2013)

One of the most important aspects of Apple's decision to open the iPhone to native app development was the creation of the App Store. The App Store proved a massive success growing rapidly from 500 apps at launch, with 10 million downloads in the first weekend, to over one million apps available today. Google followed Apple's lead introducing their own app store, Android Market, which has since been renamed Google Play. Juntunen et al. (2013) describes the app stores as "the primary way in which users on these platforms find, download, and update their applications".

Discoverability is one of the major benefits of the app stores. It provides an easy mechanism for users to browse and search for apps. App developers are presented with a very easy method of distributing and marketing their apps. Developers are also benefitting as Apple and Google are handling all the fees and charges that the developers may wish to bill the user for using their app. Apple were perfectly primed for this billing procedure as many Apple users already had an iTunes account setup which included billing information. This billing can be up front, paying to install the app, or in app purchases, or both. Commission is charged by both Apple and Google at a rate of 30%. We have already seen that both also charge developer registration fees in order to reduce spamming and increase app quality. As previously mentioned, the only method

of distributing apps for Apple's iOS platform is through the App Store. Android on the other hand, does allow non app store apps to be installed if the user so wishes. There are other Android app stores besides the Google Play Store, the Amazon Appstore which targets Amazon's Kindle line of tablet is a notable alternative. Other options include GetJar and Slide ME.

The discoverability previously mentioned as a benefit of the app stores is now coming into question. Haselmayr (2013) and Kingsley-Hughes (2012) both raise concerns regarding the actual discoverability the app stores are offering. Both Apple's App Store and Google Play each have over one million apps available for download. Haselmayr points to a ineffective search mechanism, general categories and a questionable ratings mechanism which lead to many apps never getting discovered. Tyson (2012) highlights a report which states that 60% of App Store apps never get a downloaded a significant number of times. Without a considerable marketing campaign or somehow managing to get into the top of the charts lists on the app stores, many apps simply never get noticed.

Web apps, by contrast, have no equivalent one-stop-shop for discovering and installing apps. Web apps, which are built in HTML, are inherently searchable. As long as this HTML is available on a URL, the search engines will find it and index it. A search engine listing nonetheless does not compare with a dedicated location to search for web apps. This puts web apps at a disadvantage to native apps in terms of discoverability. Web app developers also do not have the luxury of being able to automatically tie in with a payments processing system like the native apps. In order for developers to charge for their web apps, they would need to implement a payment system in app, which is not user friendly and certainly not as seamless as the native app procedure for payments.

Web Apps do however avoid certain rules and regulations set down, by Apple in particular, regarding distribution of apps in the app stores. The most obvious is the previously mentioned commission rates of 30% on all revenue which the app stores charge. Certain organisations such as Amazon (Vaughn, 2011) and The Financial Times (Roberts, 2013) have turned to the web app as a method of avoiding this commission. Both companies specifically highlighted the 30% commission rate as the primary reason for this change of direction. Amazon were under pressure from Apple to pay the 30% commission on all e-books sold through their app so they moved this portion of their business off the Apple platform.

Another important consideration with the app stores are app approval procedures which the app store owner impose. Obviously a web app is controlled by the developers themselves so no rules govern what functionality they decide to provide. Google have a relatively lax app approval system which essentially accepts any app without prior approval and will investigate reported breaches of policy, and remove the offending app. (Mahoney, 2008) Apple take a more proactive approach to the approval process by pre approving all apps before they are published to the app store. (Duryee, 2012) This pre approval procedure has proved controversial as it can take anything up to a number of weeks to get an app through the vetting process.

Apple, in particular, are notoriously controlling in deciding which apps they want to approve for their App Store. Previously their approval process included a non-disclosure agreement whereby the reasons that an app may have failed the approval process could not be distributed or published for other developers to see. (Chen, 2008) This meant that the very important development tool of information sharing was lost to the community. Apple have since dropped this policy due to the burden it was placing on development. Apple, however, do continue impose quite rigorous tests before approving an app for publication.

Both Apple and Google could also be accused of censorship within their app stores when it comes to certain apps. Google have frequently removed ad blocking apps from the Google Play Store. (Wauters, 2013) These apps would obviously have impacted Google's primary revenue source. Apple similarly have blocked apps which compete with the Apple's own apps such as email apps, SMS apps and Music player apps. Apple cite duplication of existing app with no added functionality for rejecting such apps. Apple also frequently refuse to publish apps which they declare to be either offensive or obscene. (Price, 2013)

There does not appear to be any set list of explicit rules governing Apple's approval process. Rice (2012) lists some tips for improving chances of getting approved such as, following Apple's HCI guidelines, robust testing and a unique icon design. Other instances, such as Wang et al. (2013), prove that the approval process may not be as robust as some suggest. Wang et al. successful submitted an app to the App Store which contained malware. Nevertheless, there seems to be no guarantees of approval, with the only way to be sure being to submit the app and wait for a response.

Lionbridge (2012) uses the term "front of mind penetration" in relation to an advantage of native apps. After installing a native app from the app store, an icon is placed on the

app list on the device. This icon serves as a visible reminder to the user that the app is available for use. The app store alerting users to the availability of updates for the app also acts as a visible reminder to the user about the app which they have installed. While web apps do not provide alerts regarding updates, these updates are picked up automatically the first time the user loads the app after the update on the server has occured. Web apps now do have the ability to be presented as an icon on the app list on the device. Sin et al. (2012) provide a detailed description of how a web app can be produced to have an icon and splash screen similar to those of native apps, along with a method of producing the app in fullscreen mode without any of the browser address or toolbars being visible. These features give the web app a much more native feel as it appears on the app list the same as any native app and loads in full screen mode like any native app. Obviously considerations need to be made to ensure no loss of functionality in fullscreen mode but these can be handled in the design of the app. Both Safari and Chrome beta currently provide this functionality to web apps.

Engineering Considerations

Programming Language and Development Tools

Whereas all web apps are developed using a combination of HTML, CSS and JavaScript, the most fundamental issue for native app development is the differing programming languages necessary to develop apps for the different platforms. The two main languages required, Java and Objective C are different, although they have some similarities in terms of object models and syntax. Goadrich and Rogers (2011), state that more computer science students are familiar with the Java language in combination with the Software Development Kit (SDK) Eclipse than are familiar with Objective C and the iOS SDK, XCode. This results in a "modest upfront cost" to choose iOS development.

Where the platforms and languages differ, the Software Development Kits (SDKs) also differ between the native platforms. For iOS development the standard Integrated Development Environment (IDE) is XCode which incorporates the iOS SDK and iOS Simulator. There are other options for iOS app development such as Appcode. Goadrich and Rogers (2011) point to an important consideration with regards to the XCode IDE for iOS development. They say, "hardware may be one of the most irksome impediments to iOS development". iOS development requires the Mac OSX operating system as XCode will only run on this platform. Mac OSX is only licensed to run on Apple Macintosh PCs. Therefore a prerequisite for iOS development in most cases is the purchase of Apple Macintosh hardware.

For Android development the standard bundle contains the Eclipse IDE along with the Android SDK. There are many alternative Java IDEs available to use instead of Eclipse. The Android SDK, in contrast to iOS, supports all three of the major desktop operating systems, OSX, Windows and Linux. This means that whichever of the three operating systems the developer is currently working on, they can proceed to develop Android apps without the need to buy specific hardware in order to run the SDK.

Each SDK is quite a complex and heavyweight tool for development. As they each have their own features, interfaces and require knowledge of differing API's, there is a significant learning curve involved in the use of these SDKs.

The differences between the languages and which language is considered better or worse than the other does not concern this discussion. Nor does the differences between the SDKs. Each have their own advocates who will argue the benefits of one language and SDK over the other. Farina (2011) and Green (2009) are two such examples. The primary objective here is to point to the fact that native developers will need to be knowledgeable in both languages and both SDKs in order to be able to develop native apps for the two major platforms. Developers for iOS will also have to consider the potential cost of any hardware necessary to run the XCode IDE.

Testing

Testing is obviously a major part of the app development process. Joorabchi et al. (2013) talks about the different aspects of the native app testing procedure produced by their survey. This highlights the prevalence of manual testing over automated testing, the separation of platform testing and the problems of GUI (Graphical User Interface) testing. Although automated testing of web apps is quite prevalent in the desktop environment, the automation of mobile web apps, like the automation of native apps, is an emerging market. Quinn (2013) indicates that the market for automated app testing software is growing rapidly but underscores the need for research into the options before committing to any one tool. Huggins (2012) states that, "there is no consensus on the right tool for testing anything on mobile", and that while the market is maturing most app developers have remained performing manual tests until this consensus on automation is reached.

A major stumbling block for testing are the emulators and simulators available to test apps. The Emulator is the tool used in Android development to test and debug apps, the Simulator is the equivalent tool in iOS. Both tools allow developers to test their how their apps function on a range of different devices and hardware. Joorabchi et al. (2013)

specifically highlight the inability to mimic real world situations regarding items such as network latency, sensor readings, Bluetooth and GPS from within the Emulator and Simulator, thus not allowing the developer to test specific test cases that are very important to some apps.

Performance of these tools is also noted as a key factor that continuously gets mentioned by survey participants of Joorabchi et al. (2013). Farina (2011), also points to flaws in both the Emulator and Simulator. The Simulator is so called, as it simulates the iOS app running on the iOS platform. While it appears to run as a mobile app, it is in fact running as a desktop application and simulating the iOS environment. This means that the app does not exactly replicate the iOS environment so errors may occur when the app is eventually moved to the device.

The Android emulator by comparison runs a full version of the Android OS in a virtual machine on the desktop. This sounds ideal in theory as the issues raised by the Apple's Simulator would not occur in this instance. The major trade off is the performance of the Emulator. Both Joorabchi et al. (2013) and Farina (2011) directly highlight this issue. Farina (2011) states that it takes the Emulator two minutes to boot up and over thirty seconds to reload after each edit. He contrasts this with the approximate reload time of five seconds on Apple's simulator. The best alternative to the Android Emulator is to use an actual Android device for testing.

Goadrich and Rogers (2011) also envision a situation whereby in order to perform realistic usability tests of the apps, actual mobile devices are needed, as opposed to testing the app on a desktop machine using an emulator or simulator. This also overcomes the difficulties mentioned previously such as network speeds and GPS functionality. Testing on devices also overcomes any difficulties arising from flaws in the Emulator and Simulator. The main issue that arises here is the number of devices required. We discussed platform fragmentation earlier, which means testers need at least a tablet and smartphone on both the Android and iOS platforms. The issue of fragmentation within platforms also arose, thus testers need multiple Android devices with differing versions, screen sizes and resolutions.

The difficulty related to testing applies across all apps, both native and web. In order to successfully test any app, we need to either make use of the Emulator and Simulator or, in order to avoid their faults, test directly on the devices. Neither scenario is ideal, as the test tools have flaws and to have sufficient devices available to test all mobile scenarios can prove very difficult.

User Experience

Wasserman (2010) and Charland and LeRoux (2011) both underline the importance of the user experience with regards to apps. Wasserman outlines that mobile platforms "include their own UI libraries and guidelines, so native applications for a device will share a common look and feel". Native apps are expected to adhere to standards set down by the platform owners to ensure that users have a standard experience and can utilise standard gestures on the platform. Users are familiar with standard menu bar positions, go back options and swipe gestures and expect that these familiarities will persist between apps on the same platform. Web apps on the other hand do not have the native interface libraries available to them. As Charland and LeRoux (2011) point out, web apps must also contend with conflicting user familiarities such as the positioning of menus and tabs as users may expect them to be in certain locations based on the native standard. A further conflict can appear where certain devices have physical buttons that other devices do not. The most obvious example of this is the physical back button available on all Android devices that does not exist on an iOS devices. Charland and LeRoux (2011) point to a number of frameworks for use with web apps which give a native look and feel to web apps but some features such as smooth transitions and bouncy scrolling are proving difficult to achieve.

Conclusions

This section has outlined the differing aspects of the mobile app paradigms. We have looked at technical considerations such as performance, access to hardware and device features and fragmentation. We have looked at the considerations regarding the distribution and install of apps, highlighting some of the benefits and drawbacks of the app stores. We have also looked at some of the engineering considerations with regards to the development of apps such as the programming language and development tools, testing and user experience.

Both native and web apps have different advantages and disadvantages, each lending themselves to different circumstances. The next section will discuss these differing circumstances this in terms of web apps potential to being a viable alternative to native apps.

Results of Analysis

We apps have been presented with many favourable attributes, which may appeal to certain segments of the mobile app market. Web apps have also been presented with some drawbacks which may be a cause for concern within other segments of the mobile

app market. We have currently seeing a lot of discussion regarding the viability of web apps, particularly where major companies are concerned. We have seen both Facebook (Constine, 2012) and LinkedIn (O'Dell, 2013) move from web based apps back to native apps. On the other hand we have seen The Financial Times (Roberts, 2013) and Amazon (Vaughn, 2011) leave the native app platform in favour of web apps. This section of the paper will outline, based on the analysis of the previous section, where web apps may be appropriate for use on mobile devices and other instances where web apps may not be appropriate. There is no clear cut answer, an outright statement declaring that web apps are or are not feasible is not possible as differing app requirements can change the answer.

Viable Web Apps

Web apps have come to us from the World Wide Web, which has proved an open and accessible medium. Web apps are also built using the same technologies as websites, HTML, CSS and JavaScript. This is probably the key benefit of web apps. Web apps provide developers with a method of producing mobile apps without the need to learn new languages and frameworks. If one is not familiar with the native development languages, development tools and APIs there is a significant learning curve to become proficient. Compounding this learning curve is the fact that in order to get the majority of the market a developer needs to be proficient in two native platforms. Therefore we can say that if a developer or a company is considering mobile app development, and they currently have no experience in native development, web apps should at least be a consideration. The costs in this situation of either hiring outside developers or training in house developers is a massive consideration for anyone considering native apps. This forces a situation where web apps maybe a viable alternative to native, purely based on the cost of development.

Costs need to be controlled by the app will also need to function as required. If a web app is to be a viable alternative it must provide all the functionality that the users may require. Significant drawbacks of web apps, as previously encountered, are performance, access to hardware features and distribution. The viability of a web app declines the greater the need for each of these aspects of the app. We have seen The Financial Times and Amazon move their mobile apps from native to web. Roberts (2013), interviews FT.com's Managing Director who states that the discoverability of the app store is not a major concern for well established brands. The Financial Times and Amazon are very well recognised brands, with loyal users. Brands such as these are not impacted by the discoverability of the app store. Apps such as these do not suffer from

performance issues as they are not processor intensive and also do not need access to specific hardware features of the device, making them quite suitable for web apps. One difficulty highlighted is the payments procedure, as web apps do not have a convenient payment processing system that native apps provide. Roberts (2013) alludes to this question to The Financial Times who acknowledge the issue but point to options such as PayPal as alternatives. Amazon do not have this problem as they already have a payment procedure in place for anyone registered with an Amazon account.

If an app does not need high performance, does not need discoverability and does not need access to device hardware features, then it is a good candidate for being produced as a web app. Clearly the discoverability issue is contingent on the a number of issues such as the user following and brand loyalty. This is not a concern for big companies but may be the deciding factor with smaller market players. If the app is free, then the payment system does not factor but if it does there are options available such as PayPal. If the web app procedure is functioning successfully for The Financial Times, then there is no reason why it would not function for other newspaper apps. Newspapers would not suffer from discoverability issues as their sites are frequently accessed and apps would not need high performance or access to device hardware. Another possible candidate would be an app such as eBay. eBay would not suffer from the lack of discoverability, and does not need a high performance app with access to hardware features. eBay accounts are also already tied in with PayPal accounts so the payments procedure would not be an issue.

If the discoverability issue were to be solved, it appears that web apps may become available to more than just the big companies with brand loyalty. One solution would be for the current app stores to accept web apps. This may counter any of the benefits web apps currently have, like being outside the control of the app stores. It appears there may be a market for an enterprising individual to create a purely web app based store.

Augmented Web Apps

There are methods to mitigate against some of the gaps in the web app environment. Consider an app that is a perfect candidate as a web app, but it needs to access one hardware feature which is not available from the browser. Or, it meets all the requirements technically, but needs the discoverability or payment system of an app store.

Hybrid app frameworks can provide solutions to these issues. These frameworks provide an additional abstraction layer around a web app which allow for access to some device features. The framework also packages the app in a native container, allowing for the app to be distributed on the native app stores. This could solve the problems mentioned above where web apps provide the majority of the solution but fall short on one or two features which are unavailable.

The obvious disadvantage is that in order to use the frameworks, additional knowledge of the API's within the framework is required . The major advantage of web apps is the pre existing knowledge of the technologies involved. The introduction of new frameworks can negate this benefit. Whether the benefits gained of using a hybrid app outweigh the costs associated with the extra work involved in development will depend on each individual apps circumstances.

Heikötter et al. (2013) and Ronkainen et al. (2013) provide detailed analysis of the different Hybrid app frameworks available. In the context of our discussion it is important to recognise that the Hybrid option provides methods of negating some of the drawbacks of pure web app development.

Native Apps

The primary downfall of web apps in comparison to native apps appears to be the performance issues. JavaScript simply cannot compete with the native code in terms of processing speed and ability. This means that there are times where web apps are just not an option. The use of Hybrid frameworks does not improve performance either so where Hybrid provided solutions in other instances, it cannot make up for the gap in performance. Examples of such high performance apps are graphics intensive games, animations, photo and video editing apps. Games are a significant portion of the native app market, 33% of downloads and 66% of revenues. Games apps need to be native as they need the performance and availability of the app stores makes them easier to monetise.

The question of where the performance cut off is, is also a difficult question to answer. It will very much depend again on each individual situation, and what the developer may or may not deem an acceptable delay. A stuttering game will not enhance the users experience whereas a seconds delay with an appropriate message may be deemed perfectly acceptable in other instances.

Other instances where native apps are the only option is where access to certain device hardware is neither available to the browser, nor available to the hybrid frameworks. Again, if this hardware is an essential part of the app, then native is the only solution. This may be highly relevant where the app developer is fighting to be first to market with a new innovative app. Similarly, if having a consistent UI with the platform the app is sitting in is an essential requirement of the app, native seems to be the only option. Web UI frameworks have come close to native UI's but subtle differences still remain.

Conclusions

This paper has examined the question if web apps are a feasible alternative to native apps in the mobile environment. The current state of the mobile environment, with its rapidly expanding market driving demand for apps, has been highlighted as an important market. The market for apps is currently dominated by native apps which run on two main platforms, Apple's iOS and Google's Android. With a fragmented market and platform specific apps, web apps are presented as a potential solution to this divide.

The paper has analysed both native and web apps. Advantages and disadvantages of both paradigms have been presented. Web apps show promise in the areas of programming languages and cross platform support. Native apps prove better under performance and access to device hardware. Arguments have been presented showing that distribution methods such as app stores can have positive and negative impacts, some apps needing them for discoverability while other abandon them due to restrictions and controls. Both native and web apps have different advantages and disadvantages, each lending themselves to different circumstances.

The paper contends that in certain circumstances web apps are a feasible alternative to native apps. These circumstances include instances where developer knowledge of the complex native app development process is low and where high performance, discoverability and access to device hardware are not important to the developer. Some of these tradeoffs may not be acceptable to certain developers. Hybrid apps are presented as a solution to some of the failings of web apps, such as access to device hardware and discoverability on the app stores. Hybrid apps bring with them a need for increased knowledge of the Hybrid frameworks. Finally instances where web apps are not a feasible alternative are discussed. Instances where high performance is key, such as graphics intensive games, and where device hardware access is essential are situations where web apps are not a feasible alternative to native apps.

Bibliography

All Things D, 2007. Bill Gates and Steve Jobs at D5 - TRANSCRIPT [WWW Document]. All Things D. URL http://allthingsd.com/20070531/d5-gates-jobs-transcript/ (accessed 2.24.14).

Android, 2014. Android [WWW Document], URL https://www.android.com (accessed 2.14.14).

Apple, 2007. - Press Info - Apple Reinvents the Phone with iPhone [WWW Document], URL https://www.apple.com/pr/library/2007/01/09Apple-Reinvents-the-Phone-with-iPhone.html (accessed 2.27.14).

Apple, 2014. Develop for iOS - Apple Developer [WWW Document], URL https://www.developer.appple.com/technologies/ios/ (accessed 2.27.14).

Apple, 2014. Apple URL Scheme Refernece: About Apple URL Schemes [WWW Document]. URL https://developer.apple.com/library/ios/featuredarticles/iPhoneURLScheme_Reference/Introduction/Introduction.html (accessed 2.27.14).

Bell, K., 2011. Steve Jobs Was Originally Dead Set Against Third-Party Apps for the iPhone | Cult of Mac [WWW Document]. URL http://www.cultofmac.com/125180/steve-jobs-was-originally-dead-set-against-third-party-apps-for-the-iphone/ (accessed 2.20.14).

Charland, A., Leroux, B., 2011. Mobile application development: web vs. native. Communications of the ACM 54, 49.

Chen, B.X., 2008. Apple Imposes NDA For App Store Rejections | Gadget Lab | Wired.com [WWW Document]. Gadget Lab. URL http://www.wired.com/gadgetlab/2008/09/apple-imposes-n/ (accessed 2.28.14).

Corral, L., Sillitti, A., Succi, G., 2012. Mobile Multiplatform Development: An Experiment for Performance Analysis. Procedia Computer Science 10, 736–743.

Constine, J., 2012. Facebook Speeds Up Android App By Ditching HTML5 And Rebuilding It Natively Just Like The iOS Version. [WWW Document], TechCrunch. URL http://techcrunch.com/2012/12/13/facebook-android-faster/ (accessed 2.20.14).

Crawford, D., 2013. Why mobile web apps are slow | Sealed Abstract [WWW Document]. URL http://sealedabstract.com/rants/why-mobile-web-apps-are-slow/ (accessed 2.20.14).

Dalmasso, I., Datta, S.K., Bonnet, C., Nikaein, N., 2013. Survey, comparison and evaluation of cross platform mobile application development tools, in: Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International. Presented at the Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International, pp. 323–328.

DeNucci, D. 1999. Fragmented Future [WWW Document], URL http://darcyd.com/fragmented_future.pdf (accessed 2.10.14).

Desruelle, H., Blomme, D., Gielen, F., 2011. Adaptive Mobile Web Applications Through Fine-Grained Progressive Enhancement. Presented at the ADAPTIVE 2011, The Third International Conference on Adaptive and Self-Adaptive Systems and Applications, pp. 51–56.

Duryee, T., 2012. The Latest Long Apple Line: Developers Waiting for App Approval. [WWW Document]. URL http://allthingsd.com/20121108/the-latest-long-apple-line-developers-waiting-for-app-approval/ (accessed 2.10.14).

Farina, N., 2011. An iOS Developer Takes on Android [WWW Document]. URL http://nfarina.com/post/8239634061/ios-to-android (accessed 2.17.14).

Gartner, 2013. Gartner Says Worldwide PC, Tablet and Mobile Phone Shipments to Grow 4.5 Percent in 2013 as Lower-Priced Devices Drive Growth [WWW Document]. URL http://www.gartner.com/newsroom/id/2610015 (accessed 1.25.14b).

Gartner, 2013. Gartner Says Worldwide Traditional PC, Tablet, Ultramobile and Mobile Phone Shipments On Pace to Grow 7.6 Percent in 2014 [WWW Document]. URL http://www.gartner.com/newsroom/id/2645115 (accessed 1.25.14c).

Gartner, 2013. Gartner Says Smartphone Sales Accounted for 55 Percent of Overall Mobile Phone Sales in Third Quarter of 2013 [WWW Document]. URL http://www.gartner.com/newsroom/id/2623415 (accessed 1.25.14c).

Goadrich, M.H., Rogers, M.P., 2011. Smart Smartphone Development: IOS Versus Android, in: Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education, SIGCSE '11. ACM, New York, NY, USA, pp. 607–612.

Green, D., 2009. Green's Opinion: Android versus iPhone Development: A Comparison. [WWW Document]. URL http://greensopinion.blogspot.ie/2009/07/android-versus-iphone-development.html (accessed 2.17.14)

Grooveshark, 2014. Grooveshark Help - Devices & Apps [WWW Document]. URL http://help.grooveshark.com/customer/portal/topics/290-devices-apps/articles (accessed 2.26.14).

Haselmayr, M., 2013. App Discovery: Why Can't Anyone Figure This Out Yet? - Forbes [WWW Document]. URL http://www.forbes.com/sites/allbusiness/2013/08/15/app-discovery-why-cant-anyone-figure-this-out-yet/ (accessed 2.23.14).

Heath, N., 2014. Google revs Chrome's V8 JavaScript engine to drive high-performance web apps [WWW Document]. ZDNet. URL http://www.zdnet.com/google-revs-chromes-v8-javascript-engine-to-drive-high-performance-web-apps-7000026409/ (accessed 2.20.14).

Heitkötter, H., Hanschke, S., Majchrzak, T.A., 2013. Evaluating Cross-Platform Development Approaches for Mobile Applications, in: Cordeiro, J., Krempels, K.-H. (Eds.), Web Information Systems and Technologies, Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, pp. 120–138.

HTML5test, 2014. HTML5test - How well does your browser support HTML5? [WWW Document]. URL http://html5test.com/ (accessed 2.10.14)

Huggins, J., 2012. Automated Testing for Mobile Apps | Selenium Testing? Do Cross Browser Testing with Sauce Labs. [WWW Document]. URL http://sauceio.com/index.php/2012/11/musings-mobile-app/

Haselmayr, M., 2013. App Discovery: Why Can't Anyone Figure This Out Yet? - Forbes [WWW Document]. URL http://www.forbes.com/sites/allbusiness/2013/08/15/app-discovery-why-cant-anyone-figure-this-out-yet/ (accessed 2.23.14).

Joorabchi, M.E., Mesbah, A., Kruchten, P., 2013. Real Challenges in Mobile App Development, in: 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement. Presented at the 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 15–24.

Juntunen, A., Jalonen, E., Luukkainen, S., 2013. HTML 5 in Mobile Devices – Drivers and Restraints, in: 2013 46th Hawaii International Conference on System Sciences (HICSS). Presented at the 2013 46th Hawaii International Conference on System Sciences (HICSS), pp. 1053–1062.

Khalaf, S., 2013, Flurry Five-Year Report: It's an App World. The Web Just Lives in It [WWW Document]. URL http://blog.flurry.com/bid/95723/Flurry-Five-Year-Report-It-s-an-App-World-The-Web-Just-Lives-in-It (accessed 2.24.14).

Kingsley-Hughes, A., 2012 6, 600,000 apps in Apple's App Store, yet I can't find anything I want [WWW Document]. ZDNet. URL http://www.zdnet.com/blog/hardware/600000-apps-in-apples-app-store-yet-i-cant-find-anything-i-want/19549 (accessed 2.26.14).

Lee, S.-W., Moon, S.-M., Jung, W.-K., Oh, J.-S., Oh, H.-S., 2010. Code Size and Performance Optimization for Mobile JavaScript Just-in-time Compiler, in: Proceedings of the 2010 Workshop on Interaction Between Compilers and Computer Architecture, INTERACT-14. ACM, New York, NY, USA, pp. 6:1–6:7.

Lionbridge, 2012. Mobile Web Apps vs. Mobile Native Apps: How to Make the Right Choice. [WWW Document]. URL http://www.lionbridge.com/files/2012/11/Lionbridge-WP_MobileApps2.pdf (accessed 2.20.14)

Mahoney, J., 2008. Android Market, Google's App Store, Will Not Require Approval For Applications [WWW Document] Gizmodo. URL http://gizmodo.com/5043178/android-market-googles-app-store-will-not-require-approval-for-applications (accessed 2.28.14).

Mathews, L., 2013. OdinMonkey adds a turbocharger to Firefox's JavaScript engine | Apps and Software | Geek.com. [WWW Document]. URL http://www.geek.com/apps/odinmonkey-adds-a-turbocharger-to-firefoxs-javascript-engine-1543735/ (accessed 2.20.14)

Meier, R., Mahemoff, M., 2011. HTML5 versus Android: Apps or Web for Mobile Development? [WWW Document]. URL https://www.google.com/events/io/2011/sessions/html5-versus-android-apps-or-web-for-mobile-development.html (accessed 2.20.14)

Mesbah, A., Prasad, M.R., 2011. Automated cross-browser compatibility testing. Presented at the Proceedings of the 33rd International Conference on Software Engineering, ACM, pp. 561–570.

Nielsen, J., 1993. Response Time Limits [WWW Document]. URL http://www.nngroup.com/articles/response-times-3-important-limits/ (accessed 2.23.14).

O'Dell, J., 2013. Why LinkedIn dumped HTML5 & went native for its mobile apps. [WWW Document]. URL http://venturebeat.com/2013/04/17/linkedin-mobile-web-breakup/ (accessed 2.23.14).

Price, D., 2013. The chilling effect of Apple's App Store censorship - News [WWW Document], Macworld UK. URL http://www.macworld.co.uk/news/apple/chilling-effect-apples-app-store-censorship-3442095/ (accessed 2.28.14).

Quinn, B., 2013. Mobile App Test Automation—The Options [WWW Document]. TechWell. URL http://www.techwell.com/2013/01/mobile-app-test-automation-options (accessed 2.18.14).

Rice, K., 2012. 6 Ways to Ensure App Store Approval | Kinvey Backend as a Service Blog [WWW Document], URL http://www.kinvey.com/blog/1600/6-ways-to-ensure-app-store-approval (accessed 2.26.14).

Ritchie, R., 2010. App Store Year Zero: How unsweetened web apps and unsigned code drove the iPhone to an SDK | iMore [WWW Document]. URL http://www.imore.com/history-app-store-year-zero (accessed 2.20.14).

Roberts, J.J., 2013. FT launches "second generation" web app, says online payments will soon be much easier. [WWW Document]. URL http://paidcontent.org/2013/04/03/ft-launches-second-generation-web-app-says-online-payments-will-soon-be-much-easier/ (accessed 2.27.14).

Ronkainen, J., Eskeli, J., Urhemaa, T., Koskela-Huotari, K., 2013. Experiences on Mobile Cross-Platform Application Development Using PhoneGap. Presented at the ICSEA 2013, The Eighth International Conference on Software Engineering Advances, pp. 146–151.

Sin, D., Lawson, E., Kannoorpatti, K., 2012. Mobile Web Apps - The Non-programmer's Alternative to Native Applications, in: 2012 5th International Conference on Human System Interactions (HSI). Presented at the 2012 5th International Conference on Human System Interactions (HSI), pp. 8–15.

Smith, C., 2014. All new Android devices must run KitKat, says alleged leaked Google memo. [WWW Document]. URL http://www.techradar.com/news/phone-and-communications/mobile-phones/all-new-android-devices-must-run-kitkat-says-alleged-leaked-google-memo-1225553 (accessed 2.17.14)

Spotify, 2010. iPhone app updated - background listening arrives! [WWW Document]. URL http://news.spotify.com/ie/2010/07/02/background-listening/ (accessed 2.16.14)

StatCounter, 2013. New StatCounter data finds that tablet internet usage is less than 5% globally | StatCounter Global Stats [WWW Document], URL http://gs.statcounter.com/press/new-statcounter-data-finds-that-tablet-internet-usage-is-less-than-5-percent-globally (accessed 2.24.14).

Swiech, M., Dinda, P., 2013. Making JavaScript better by making it even slower. Presented at the Proceedings of the 2013 IEEE 21st International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems, IEEE Computer Society, pp. 70–79.

Tyson, M., 2012. Sixty per cent of App Store apps have never been downloaded? - Apple - News - HEXUS.net [WWW Document]. URL http://hexus.net/mobile/news/apple/43285-sixty-per-cent-app-store-apps-never-downloaded/ (accessed 2.26.14).

Vaughn, A., 2011. Amazon Goes The WebApp Way — Launches iOS-Friendly Web-Based Kindle Cloud Reader [WWW Document]. URL http://appadvice.com/appnn/2011/08/amazon-goes-the-webapp-way-launches-ios-friendly-web-based-kindle-cloud-reader (accessed 2.27.14).
Vision Mobile, 2013. How can HTML5 compete with Native? [WWW Document], URL http:// http://www.visionmobile.com/product/how-can-html5-compete-with-native/ (accessed 1.28.14).

Wang, T., Lu, K., Lu, L., Chung, S., Lee, W., 2013. Jekyll on iOS: when benign apps become evil. Presented at the Presented as part of the 22nd USENIX Security Symposium}, USENIX}, pp. 559–572.

Wasserman, T., 2010. Software Engineering Issues for Mobile Application Development. Proc. of the FSE/SDP workshop on Future of software engineering research, FOSER 2010, IEEE Comp. Soc. Press, pp. 397-400

Wauters, R., 2013. Adblock Plus Launches New Android App, Snubbing Google [WWW Document], The Next Web. URL http://thenextweb.com/apps/2013/03/20/after-getting-booted-from-the-google-play-store-adblock-plus-releases-new-android-app/ (accessed 2.28.14).

Web Foundation, 2014. History of the Web [WWW Document], URL http://www.webfoundation.org/vision/history-of-the-web/ (accessed 2.20.14).

Wells, J., Draganova, C., 2007. Progressive enhancement in the real World. Presented at the Proceedings of the eighteenth conference on Hypertext and hypermedia, ACM, pp. 55–56.

WhatsApp, 2014. WhatsApp :: Home [WWW Document]. URL http://www.whatsapp.com (accessed 2.20.14)

Windows Phone, 2014. Windows Phone [WWW Document], URL http://www.windowsphone.com (accessed 2.27.14).

W3C, 2014. HTML5 [WWW Document], URL http://www.w3.org/TR/html5/ (accessed 2.27.14).

Zakas, N.C., 2013. The Evolution of Web Development for Mobile Devices. ACMQueue 11, 30:30–30:39.