Recommendations for Webview Based Mobile Applications on Android

Pinku Hazarika[1] ,Rahul Raj CP[2] ,Seshubabu Tolety[3]
[1]Associate Software Engineer, [2]Assistant Consultant ,[3]Consultant
[1,2,3]Siemens Technology and Services,Bangalore, India
[1]pinku.hazarika@siemens.com,[2]c.raj@siemens.com,[3]seshubabu.tolety@siemens.com

***Abstract*—Today's handy mobiles are an essential part of our daily life. Mobile applications are the lifeline of these silicon devices. The present fragmented mobile landscape, with a handful of popular mobile platforms, has fuelled the trend of developing platform neutral applications. Mobile web has since grown in popularity. But mobile web is not a standalone application by itself; rather it runs on a web browser. The limited real estate area of a web browser, with browser controls covering a lot of the display area is a usability issue. In a step ahead of mobile web, a new mobile application type called webview application brings new possibilities like a native launchpad for web content, real estate area entirely for webpage display, etc. Many third party services, like PhoneGap, etc have engineered this technology, with web to native mechanisms to enable developers to author powerful mobile applications, which is otherwise limited with mobile web. This paper discusses the mobile web and webview based applications and its increasing relevance in today's mobile landscape. The paper also states a set of recommendations while coding the android webview.**

Keywords—Mobile Web; Webview Apps; Native Apps; Android Webview; Hybrid Apps; Cross Platform.

I. INTRODUCTION

Smartphone's continue to change lives across the globe. The ever-growing adoption of Smartphone's has made a huge surge in mobile applications. Mobile applications have become an everyday need and the user appreciates the benefits. The current mobile ecosystem is highly fragmented with a number of mobile platforms like Android, iOS, Windows Phone, Blackberry, etc fighting it out in a hugely competitive market.

Every mobile platform has its own native development ecosystem. A native application is authored using the native stack, where most part of the application is developed using native technologies. However the variety of platforms available demands a more strategic approach towards application design and development and a close evaluation of the different application types against a product's features, target audience, development team, time to market and budget. Code portability across platforms is a disadvantage for an application developed using native technologies. With the emergence of HTML5 for the universal web platform, mobile web has become a preferred area of interest for mobile application development. But, mobile web requires a browser application to run on and has limitations for accessing platform features and in appearance. Both web and native applications has its own advantages and limitations. Webview applications filled this void. A webview application is a native application with a native launcher which hosts web resources [1]. All major mobile platforms, provides a browser library to embed web content as part of a client application which is known as webview. This webview object is responsible for handling the web pages. This useful tool ships with the platform SDK. Webview applications are driven by web technologies like HTML, JavaScript and content is often hosted online against offline device storage.

Developing platform neutral applications is the current trend in mobile application development. Mobile web and webview application types are the preferred choice for cross platform mobile application development.

II. MOBILE WEB APPLICATION

A mobile web application breathes inside a browser application. These applications make use of standard web technologies like HTML, CSS, JavaScript and other scripting languages. Web applications are platform neutral and can run across devices. Application updates and distribution is all hassle-free for a web application. HTML5 capabilities are tailor made for mobile application development. In a leap ahead from HTML4 [2], HTML5 supports geo-location, web applications that can work offline, background processing of computationally intensive tasks without interrupting the user interface, communication over TCP, drag and drop file uploads, 2D graphics drawing using JavaScript, audio/video multimedia elements, local storage, new form types, and more.

Mobile web is not without its share of limitations. User experience is not at par with native standards, and leveraging of device capabilities is limited, which is its biggest drawback. A web application lifespan is dependent on the browser existence. Native style background processing, secured offline storage, etc are quite the limitations for mobile web applications.

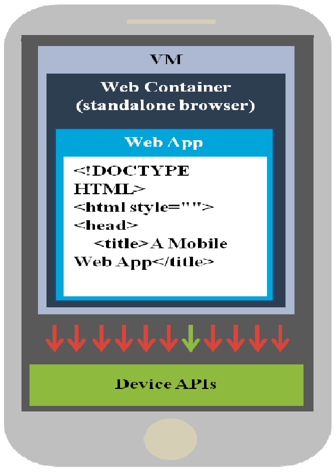
VM
Web Container (standalone browser)
Web App
<!DOCTYPE HTML>
<html style="">
<head>
<title>A Mobile Web App</title>
Device APIs

Fig. 1. Mobile web application

A. *Advantages*

- Lowest development & maintenance cost
- Maximum code reusability - Code reusability across mobile & browser platforms
- Same development team for different platforms
- No marketplace distribution hassels
- Instant updates for clients on newer feature release
- Lower time to market
- Leveraging existing web competence

B. *Disadvantages*

- Dependent on a browser application
- Higher loading time on server fetch
- Limited application management, like notification for updates, etc
- Limited access to device OS API's
- Marketplace monetization not available
- Inconsistent HTML5 support across browsers
- Native look and feel replication is difficult

III. Webview Application

Webview applications are a powerful combination of native and web, deriving the benefits of both the worlds. In simple terms, a webview application is a mobile web application wrapped in a native container. This development approach takes a mobile web application and compensates its limitations by injecting code that talks native internally. The wrapper native application uses the platform OS API's to create a HTML rendering engine which provides the bridge between the browser and device API's.

This native bridge is the key to the prowess of webview applications, enabling easy access to device features. Both ways communication between the hosted mobile web application and the native wrapper is by JavaScript through custom built API's [3]. These API's talk internally to the custom native components made possible by the native bridge for the platform. Mobile developers have the luxury to develop their own native bridge or even proceed to use some of the existing cross-platform offerings in the market.

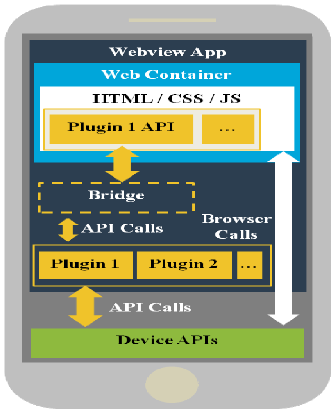

Fig. 2. Webview application

A. *Advantages*

- Best of both native and web worlds
- UI flexibility of HTMl5 with native feature richness
- Independent of a browser application
- Marketplace monetization is possible
- Leveraging existing web competence
- Access to entire device sensor array through native bridge
- Good hybrid solutions readily avilable for use in market
- Platform neutral applications
- Lower time to market

B. *Disadvantages*

- User experience is not at par with native
- Limited native skills required
- Native skills must for feature extention across different platforms
- Challenge to work around HTML5 fragmentation across devices
- Setting up the development environment for different platforms
- Access to advanced device features normally limited

IV. Mobile Web Vs Webview Applications

In today's mobile landscape, mobile strategy for an application is of more priority than implementation of the same. Choosing the right application type for a specific use-case can be a tricky job for an architect. This is clearly because of above discussed pros and cons of mobile web and webview applications.

Choosing the correct application type can be at best consolidated by evaluating each application type against a set of criteria points. The factors that are currently under consideration are performance, UI richness, complexity of development, adherence to timeframe, marketplace distribution and application security. Table I provides rating for mobile web and webview applications against the criteria points. The rating shall be interpreted as follows.

- 1 – Not preferred
- 2 – Moderate
- 3 – Perfect

TABLE I. PREFERRED SELECTION CRITERIA FOR MOBILE WEB AND WEBVIEW

	Mobile Web	Webview
Performance	1	2
UI Richness	2	2
Development Cost	3	2
Time To Market	3	2
Gallery Engagement	1	3
Application Security	1	2

V. WEBVIEW APPLICATIONS – THE FUTURE

Webview applications hold the key to future mobile applications. With the changing business needs, hybrid methodology [4] is fast emerging as the forerunner for mobile application development. Market studies are in favor of hybrid applications to take the big leap. Gartner says by 2016, more than 50 percent of the mobile apps deployed will be hybrid [5]. With more and more enterprises adopting the [bring your own device] BYOD trend, the need for applications to be multi-platform is growing. To address this need, establishments are already adopting webview based applications. Table II provides a list of some of the popular hybrid frameworks and complementing JavaScript libraries available in the market.

TABLE II. HYBRID FRAMEWORKS AND COMPLEMENTING LIBRARIES

Hybrid Frameworks	PhoneGap, Trigger.io, MoSync
Complementing Libraries	Sencha Touch, jQuery Mobile, Backbone.js, Bootstrap, Zepto.js etc

VI. ANDROID WEBVIEW RECOMMENDATIONS

A webview is feature wise, a subset of a web browser. A webview can display a webpage and is a useful tool to display HTML content within a native application, especially for information retrieved often over the network. Modern web browsers like Chrome, Firefox, are continuously updating their runtime with modern feature set [6], but webview's are not keeping pace with the modern web standards, which make webview programming a tricky task. Developers need to be aware of the fact that writing application for webview is different from writing for mobile web browsers. These differences need to be considered and properly addressed while developing webview based applications. Following sections of this article provides an overview of the points to be considered for application development for webview.

A. *Open in New Tab/New Window feature*

A webview doesn't support open-in-new-tab feature. HTML request to open a new window is by default ignored by webview [7]. This includes request from JavaScript or from a HTML link with a target attribute. Easiest way to mitigate this hurdle is to avoid any such feature in the web application. If the application demands such as feature then to support multiple windows, the web container component has to explicitly create a new webview object and attach it to the view system and subsequently handle closing this window.

B. *Navigation – No refresh/back button of browser*

Any web browser comes with a refresh, backward and forward buttons. But the webview does not provide these controls. The web container itself needs to address these situations. Custom controls needs to be provided for handing backward and forward navigations. In the case of a web application, if certain web pages are not available, the user has the option to go back to the previous page by using the back button. Page unavailable scenarios needs to be handled explicitly in the case of a webview based application, as the unavailability of back button could lead a situation where the user cannot navigate further. Similar case applies for a refresh button. The Web container needs to address these scenarios.

C. *Handling the cache of web pages*

Each application is allocated its own cache, cookie store, etc and the cache of a web application is typically handled by the browser. On the other hand the cache of a webview application needs to be explicitly handled by the developer. The web application executed in the web container might enable HTML 5 caching. The cache storage, cache policy and the corresponding settings need to be handled by the web container component.

D. *Handling the device back button*

Pressing the device back button exits a webview application by default, irrespective of multiple web pages being navigated. To handle the back button gracefully, the web container application needs to explicitly check if there are any history items to navigate back, if so, then the application can be loaded with the previous page and if not, we can leave it to the android system to handle.

E. *Handling the web settings – Such as enabling JavaScript*

The web settings object dictates the behavior of the webview. This object is loaded with default settings on webview creation, and thereafter can be configured for custom behavior. By default, JavaScript, database storage API, DOM storage API, application caches API, use of built in zoom mechanisms are disabled. The developer has to explicitly enable these for use in application. If zoom is enabled, controls with either width or height set to its content, can behave unexpectedly and should be avoided [7].

F. *Handling HTTP(s) – Security handling*

When loading a HTTP(s) URL in a webview, one might come across a blank page. Loading the same URL in a web browser, will however present a typical warning dialog, and allows one to accept the certificate and proceed further. To handle it inside a webview, we need to add a certificate of website to the android local keystore. As a workaround, we can program the Web container component to ignore an SSL error, which is not advised. We can even program the web container component to intercept requests beginning with https, and forward it to a web browser to handle, if required.

G. *Maintaining the state of the pages*

Maintaining the page state is one another challenge for webview application. During runtime configuration changes such as screen orientation, etc, an android activity that hosts webview, gets destroyed and recreated [7]. This means that the current web page will be reloaded. Standard implementation to preserve the state of the webview, only partially restores the state, like URL of the page and browsing history. To maintain full state, we need to prevent the activity from destructions.

H. *Handling HTML Local storage*

The local storage of web application and its limit is handled by the browser. In the case of a webview based application this is not enabled by default. The web container component should enable the local storage, storage path and handle the corresponding settings. For database storage, it should be noted that the settings needs be set prior to web page load; setting it at a later point might not work as expected.

I. *Working with plug-in*

The mobile application might deal with different native applications; such as file viewers, email clients, play store applications, video players etc. Mobile browser facilitates this for a web application. For a webview based application, this has to be taken care by the Web container component. The web container needs to have a policy on how to allow invocation of native applications from web space. The web application needs to adhere to this policy and request for the invocation. For example web page loaded in a webview might require a PDF reader application to be invoked in order to display the files. The web container needs to track of such requests and invoke the native applications accordingly.

J. *Consistency*

Consistency of web standards for webview across android devices is one area which needs attention. Android devices by different vendors have different web standards support for webview. The support offered by webview on different devices needs to be well analyzed and the corresponding application requirements need to be positioned accordingly.

K. *Security*

While adopting the current crop of hybrid frameworks for webview applications, one must understand that these frameworks implement a web to native bridge for platform feature access which breaches the browser sandbox [8]. This enables malicious JavaScript code to easily access critical information residing on the device. To handle such scenarios, the application must use secure connections with end to end encryption, as well follow a strict policy for file access.

CONCLUSION

Developing webview applications certainly comes with nominal performance and usability hindrance. However it remains a useful alternative to native, if the client use-case is low on features with device specific hardware. A native launch pad and cross platform compatibility at large remains the biggest benefits of webview applications. Increased time to market and easy access of web resource pool also plays an important factor in its favor. However the decision of adopting webview applications is best left to the application need and business need.

REFERENCES

[1] A. Puder, N. Tillmann, and M. Moskal, “Exposing Native Device APIs to Web Apps” [online]. Available: http://research.microsoft.com/pubs/210175/paper.pdf

[2] “Differences from HTML4”, [online]. Available: http://www.w3.org/TR/html5-diff

[3] “Best Of Both Worlds: Mixing HTML5 And Native Code”, [online]. Available: http://mobile.smashingmagazine.com/2013/10/17/best-of-both-worlds-mixing-html5-native-code

[4] “Hybrid Mobile Application Development Approaches”, [online]. Available: http://www.tcs.com/resources/white_papers/Pages/Hybrid_mobile_application_development_approaches.aspx

[5] “Gartner Says by 2016, More Than 50 Percent of Mobile Apps Deployed Will be Hybrid”, [online]. Available: http://www.gartner.com/newsroom/id/2324917

[6] “HTML5 vs Native: The Mobile App Debate”, [online]. Available: http://www.html5rocks.com/en/mobile/nativedebate

[7] "WebView | Android Developers", [online], Available : http://developer.android.com/reference/android/webkit/WebView.html

[8] M. Neugschwandtner, M. Lindorfer and C. Platzer, “A View To A Kill: WebView Exploitation” [online]. Available: https://www.iseclab.org/papers/webview_leet13.pdf