What is in a Web View? An Analysis of Progressive Web App Features When the Means of Web Access is not a Web Browser

Thomas Steiner
Google Germany GmbH
20354 Hamburg, Germany
tomac@google.com

Michael Yeung
Google Hong Kong
Causeway Bay, Hong Kong
micyeung@google.com

ABSTRACT

Progressive Web Apps (PWA) are a new class of Web applications, enabled for the most part by the Service Workers APIs. Service Workers allow apps to *work offline* by intercepting network requests to deliver programmatic or cached responses, they can receive *push notifications* and *synchronize* data in the background even when the app is not running, and—together with Web App Manifests—allow users to *install PWAs* to their devices' home screens. Service Workers being a Web standard, support has landed in several stand-alone Web browsers—among them (but not limited to) Chrome and its open-source foundation Chromium, Firefox, Edge, Opera, UC Browser, Samsung Internet, as well as preview versions of Safari. In this paper, we examine the PWA feature support situation in *Web Views*, that is, *in-app Web experiences* that are *not* stand-alone browsers. Such in-app browsers can commonly be encountered in chat applications like WeChat or WhatsApp, online social networks like Facebook or Twitter, but also email clients like Gmail. We have developed an open-source application called *PWA Feature Detector* that allows for easily testing in-app browsers (and obviously stand-alone browsers on top) and check for the available PWA features. The results help developers make educated choices when it comes to determining whether a PWA is the right approach given their users' target means of Web access.

KEYWORDS

Progressive Web Apps, Service Workers, Web Views

ACM Reference Format:
Thomas Steiner and Michael Yeung. 1997. What is in a Web View? An Analysis of Progressive Web App Features When the Means of Web Access is not a Web Browser. In *Proceedings of ACM Woodstock conference (WOODSTOCK'97).* ACM, New York, NY, USA, Article 4, 3 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

In recent years, there has been a paradigm shift from browser to native apps and back to browser again. The Web currently is undergoing a silent revolution with Web apps, more descriptively *Progressive Web Apps*, or for short just PWAs. How did we get there?

WOODSTOCK'97, July 1997, El Paso, Texas USA

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

1.1 History of Progressive Web Apps

Since around 2005, Web development has moved from static multi-page *documents* to single-page *applications*, heavily enabled by the `XMLHttpRequest` API, a process that eventually led Garrett to coin the term *Ajax* (Asynchronous JavaScript and XML [7]) to describe this shift. Despite an early push for Web-based apps on devices such as the 2007 iPhone, attempts at Web apps mostly failed by comparison to native apps that were distributed through app stores rather than the Web. Native apps not only had direct hardware access to, *e.g.*, camera and microphone, to various sensors like accelerometer or geolocation, but also just in general provided a better user experience and booted faster compared to having to load in a browser at runtime. Additionally, advanced offline support and push notifications were simply unthinkable for Web applications at the time, and Web app icons that already could be added to devices' home screens were mostly just bookmarks with—apart from full screen mode—no special behavior. While straightforward offline scenarios could be realized with `AppCache` [13], more complex offline scenarios were error-prone and hard to get right [2].

As the Web platform matured and more and more hardware-related APIs were implemented in browsers, in the end it was the addition of Service Workers [12] to the Chromium browser in 2014 [5] that started to unlock a new class of Web apps that finally could *work offline*, receive *push notifications* and *synchronize* data in the background even when the app was not running, and—together with Web App Manifests [3]—allowed users to actually *install PWAs* to their devices' home screens with proper operating system integration [10]. Other browsers like Mozilla Firefox, Microsoft Edge, Opera, UC Browser, Samsung Internet, Apple Safari Technology Preview, and several browsers more followed in implementing Service Workers. Now, even multinational companies like Twitter or trivago bet on PWA [6, 15], as well as giant national players like Tencent News or Sina Weibo in China [17].

1.2 Research Question and Paper Structure

In this paper, we look at a special means for accessing PWAs, namely accessing them *not* through stand-alone browsers like the ones listed above, but through *in-app browsers* that render Web content in the context of native applications. Examples of such applications with in-app browsers are chat apps like WeChat (Wēixìn) or WhatsApp, online social networks like Facebook or Twitter, but also email clients like Gmail. The technology that these applications leverage internally are so-called *Web Views*. In order to understand why this presents an interesting research problem, one needs to first understand the role that applications like WeChat play in markets like China. Chan writes in an article [4] for the venture capital firm

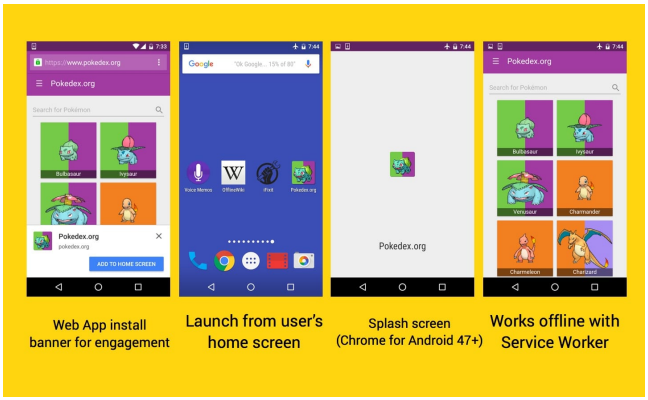

Figure 1: Screenshots showing some PWA features (http://developers.google.com/web/updates/2015/12/getting-started-pwa)

Andreessen Horowitz: "Millions (note, not just thousands) of lightweight apps live inside WeChat, much like webpages live on the internet. *This makes WeChat more like a browser for mobile websites*, or, arguably, a mobile operating system—complete with its own proprietary app store. The lightweight apps on WeChat are called 'official accounts'. Approved by WeChat after a brief application process, there are well over 10 million of these official accounts on the platform—ranging from celebrities, banks, media outlets, and fashion brands to hospitals, drug stores, car manufacturers, internet startups, personal blogs, and more". Chan goes on: "WeChat focuses on taking care of the plumbing—overseeing the integration of such pre-existing services into its portal—by simply linking users from the wallet menu to webpages from within the app. It's yet another way in which *WeChat becomes an integrated browser for the mobile (and web) world*". It is to be noted that this development comes to the detriment of the so-called open Web. As Yang and Yang write in the Financial Times [16]: "[WeChat's] news feed and search tools pull content only from within WeChat's walls rather than from the open web, including updates posted by individual users called moments, corporate accounts and an immense collection of WeChat accounts which are used by newspapers and independent bloggers". While personally we are advocates of the open Web, we therefore examine the implications of an in-app closed Web experience and what this means for PWA.

In the remainder of this paper, we first look at the technical background of Web Views and describe the PWA features and their underlying APIs in section 2, introduce our application *PWA Feature Detector* in section 3, and present and discuss our results in section 4. We close the paper with an outlook on future work in section 5 and draw some conclusions in section 6.

2 BACKGROUND ON WEB VIEWS

There are many different ways to integrate Web content in native applications, each having their own benefits and drawbacks. In the following, we describe the options on the two popular mobile operating system Android and iOS. At time of writing, Safari on iOS does not support Service Workers yet. For the sake of completeness, we nevertheless describe the Web View situation there as well.

2.1 The situation on Android

Android Web Views with `WebView`. In the Android operating system, a `WebView` [1] is a subclass of a `View` that displays Web pages. This class is the basis upon which developers can create their own Web browser or simply display some online content in their apps. It does not include any features of a fully developed Web browser, such as navigation controls or an address bar. All that `WebView` does, by default, is show a Web page. Therefore, it uses the system browser's rendering engine to display Web pages and includes methods to navigate forward and backward through a history, zoom in and out, perform text searches and more. Looper describes [11] the development of the component as follows: "Whereas earlier versions of the Android OS relied on the WebKit rendering engine to power its `WebView`, as of Android 4.4, various versions of Chromium are implemented. Typically, with each consecutive update of Android's OS, a new version of Chromium would also be included, thereby giving access to the new rendering engine's capability. This causes issues in backward compatibility for developers who must support earlier versions of Android. To combat this particular problem, as of Android 5.0, the concept of the auto-updating `WebView` has been introduced. Instead of the `WebView` version and capabilities depending on Android OS' update cycle, the Android 5.0 `WebView` is a system-level `.apk` file available in Google Play that can update itself in the background".

Chrome Custom Tab with `CustomTabsIntent`. While `WebViews` are completely isolated from the user's regular browsing activities, Chrome Custom Tabs [9], available since Chrome 45 (September 2015) and instantiatable as `CustomTabsIntent`, provide a way for an application to customize and interact with a Chrome Activity on Android. This makes the Web content feel like being a part of the application, while retaining the full functionality and performance of a complete Web browser through a shared cookie jar and permissions model, so users do not have to log in to sites they are already connected to, or re-grant permissions they have already granted.

Trusted Web Activity with `TwaSessionHelper`. Chrome Custom Tabs solved many issues of Android Web Views, however, had the drawback of not being available in a fullscreen variant like Web Views. As of October 2017, Trusted Web Activities [8] are a new way to integrate Web app content such as PWAs with Android apps. They can be instantiated with a `TwaSessionHelper` and use a protocol based on Chrome Custom Tabs. Content in a Trusted Web Activity is trusted—the app and the site it opens are expected to come from the same developer, this is verified using Digital Asset Links.[1] The host app does not have direct access to Web content in a Trusted Web Activity or any other kind of Web state. Transitions between Web and native content are between activities. Each activity (*i.e.*, screen) of an app is either completely provided by the Web, or by an Android activity. While not enforced at time of writing, Trusted Web Activities will ultimately need to meet content requirements similar to the "improved add to home screen" flow [10], which is designed to be a baseline of interactivity and performance.

[1] Digital Asset Links: https://developers.google.com/digital-asset-links/

2.2 The situation on ios

ios Web Views with `UIWebView`. Similar to Android, on ios as well Web content could be embedded with a simple system-level Web View called `UIWebView`.[2] With the release of ios 4.3 in early 2011, Apple introduced Nitro, a faster, just-in-time (JIT) JavaScript engine for Safari that considerably sped up the browser's performance in loading complex Web pages. Nitro was exclusive to Safari: third-party developers could not benefit from the faster performance in their Web views based on `UIWebView`, which was widely considered a calculated move to encourage usage of Safari over Web Views and Web apps saved to the iPhone's home screen [14].

ios Web Views with `WKWebView`. In June 2014, Apple announced `WKWebView`,[3] a new API that would allow developers to display Web content in custom Web Views with the same performance benefits of Safari. Designed with security in mind, `WKWebView` featured the same Nitro engine of Safari, while still allowing developers to customize the experience with their own user interface and features.

ios Web Views with `SFSafariViewController`. In September 2015 with the release of ios 9, Apple introduced a new Web View called `SFSafariViewController`[4], which enables apps to delegate the responsibility of showing web content to Safari itself, avoiding the need to write custom code for built-in browsers. Safari View Controller shares cookies and website data with Safari, which means that if a user is already logged into a specific website in Safari and a link to that website is opened in Safari View Controller, the user will already be logged in.

3 PWA FEATURE DETECTOR

3.1 Progressive Web App Features

3.2 Feature-Detecting Various Features

3.3 Implementation Details

4 RESULTS AND DISCUSSION

5 FUTURE WORK

6 CONCLUSIONS

REFERENCES

[1] Android Developers. 2018. Building Web Apps in WebView. https://developer.android.com/guide/webapps/webview.html. (2018).

[2] Jake Archibald. 2012. Application Cache is a Douchebag. http://alistapart.com/article/application-cache-is-a-douchebag. (2012).

[3] Marcos Caceres, Kenneth Rohde Christiansen, Mounir Lamouri, Anssi Kostiainen, and Rob Dolin. 2017. *Web App Manifest—Living Document.* W3C Working Draft 29 November 2017. W3C.

[4] Connie Chan. 2015. When One App Rules Them All: The Case of WeChat and Mobile in China. https://a16z.com/2015/08/06/wechat-china-mobile-first/. (2015).

[5] Dominic Cooney and Joshua Bell. 2014. Chrome 40 Beta: Powerful Offline and Lightspeed Loading with Service Workers. https://blog.chromium.org/2014/12/chrome-40-beta-powerful-offline-and.html. (2014).

[6] Nicolas Gallagher. 2017. How we built Twitter Lite. https://blog.twitter.com/engineering/en_us/topics/open-source/2017/how-we-built-twitter-lite.html. (2017).

[7] Jesse James Garrett. 2005. Ajax: A New Approach to Web Applications. http://adaptivepath.org/ideas/ajax-new-approach-web-applications/. (2005).

[8] Google Developers. 2017. Using Trusted Web Activity. https://developers.google.com/web/updates/2017/10/using-twa. (2017).

[9] Paul Kinlan. 2016. Chrome Custom Tabs. https://developer.chrome.com/multidevice/android/customtabs. (2016).

[10] Paul Kinlan. 2017. The New and Improved Add to Home Screen. https://developers.google.com/web/updates/2017/02/improved-add-to-home-screen. (2017).

[11] Jen Looper. 2015. What is a WebView? https://developer.telerik.com/featured/what-is-a-webview/. (2015).

[12] Alex Russell, Jungkee Song, Jake Archibald, and Marijn Kruisselbrink. 2017. *Service Workers 1.* Editor's Draft, 22 December 2017. W3C.

[13] Anne van Kesteren and Ian Hickson. 2008. *Offline Web Applications.* W3C Working Group Note 30 May 2008. W3C.

[14] Federico Viticci. 2015. ios 9 and Safari View Controller: The Future of Web Views. https://www.macstories.net/stories/ios-9-and-safari-view-controller-the-future-of-web-views/. (2015).

[15] Think with Google. 2017. The Next Billion Users: trivago Embrace Progressive Web Apps as the Future of Mobile. https://www.thinkwithgoogle.com/intl/en-gb/consumer-insights/trivago-embrace-progressive-web-apps-as-the-future-of-mobile/. (2017).

[16] Yuan Yang and Yingzhi Yang. 2017. Tencent pushes into news feed and search in challenge to Baidu. https://www.ft.com/content/59ca05e8-3ba6-11e7-821a-6027b8a20f23. (2017).

[17] Shunhao Zhu and Michael Yeung. 2017. PWA and AMP ♡ China (GDD China '17). https://www.youtube.com/watch?v=JCTjQx56-NY. (2017).

[2] `UIWebView`: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIWebView_Class/

[3] `WKWebView`: https://developer.apple.com/library/ios/documentation/WebKit/Reference/WKWebView_Ref/

[4] `SFSafariViewController`: https://developer.apple.com/documentation/safariservices/sfsafariviewcontroller