Progressive Web Apps

Mohd Aamir

College of Computing Sciences and Information Technology,
Teerthanker Mahaveer University, Moradabad, India
aamir2expert@hotmail.com

I. Introduction

Progressive web applications (PWAs) are web applications that load like regular web pages or websites but can offer the user functionality such as working offline, push notifications, and device hardware access traditionally available only to native applications. PWAs combine the flexibility of the web with the experience of a native application.

II. Popularity

Progressive Web Apps are user experiences that have the reach of the web, and are:

Reliable - Load instantly, even in uncertain network conditions.

Fast - Respond quickly to user interactions with silky smooth animations and no janky scrolling.

Engaging - Feel like a natural app on the device, with an immersive user experience.

This new level of quality allows Progressive Web Apps to earn a place on the user's home screen.

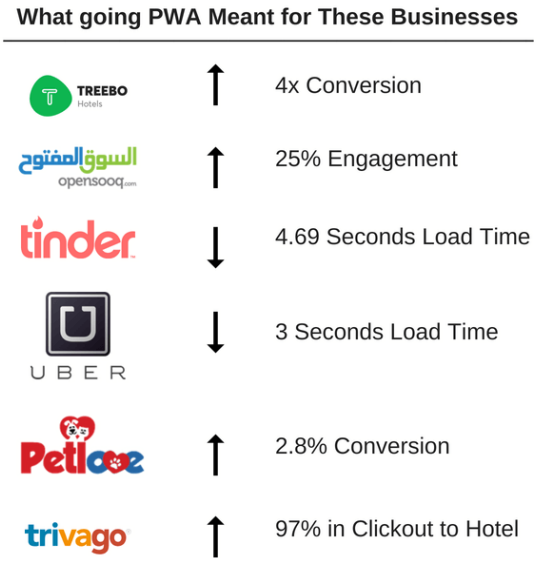

Fig. 1 Positive response of PWAs to Businesses

III. History

In 2015, designer Frances Berriman and Google Chrome engineer Alex Russell coined the term "progressive web apps" to describe apps taking advantage of new features supported by modern browsers, including service workers and web app manifests, that let users upgrade web apps to progressive web applications in their native operating system (OS).

IV. Technologies behind

A. Manifest

The web app manifest is a W3C specification defining a JSON-based manifest to provide developers a centralized place to put metadata associated with a web application including:

1. The name of the web application
2. Links to the web app icons or image objects
3. The preferred URL to launch or open the web app
4. The web app configuration data for a number of characteristics
5. Declaration for default orientation of the web app
6. Enables to set the display mode e.g. full screen

This metadata is crucial for an app to be added to a home screen or otherwise listed alongside native apps.

B. AppCache (obsolete)

An earlier technology to support offline use of the web. It works adequately for the use case it was designed for (single-page application), but fails in problematic ways for wikis and other multi-page apps. Currently supported by major browsers and in

use for years by some sites, but will eventually be removed.

C. Service workers

A service worker is an event-driven worker registered against an origin and a path. It takes the form of a JavaScript file that can control the web-page/site that it is associated with, intercepting and modifying navigation and resource requests, and caching resources in a very granular fashion to give you complete control over how your app behaves in certain situations (the most obvious one being when the network is not available).

Service workers essentially act as proxy servers that sit between web applications, the browser, and the network (when available). They are intended, among other things, to enable the creation of effective offline experiences, intercept network requests and take appropriate action based on whether the network is available, and update assets residing on the server. They will also allow access to push notifications and background sync APIs.

A service worker is run in a worker context: it therefore has no DOM access, and runs on a different thread to the main JavaScript that powers your app, so it is not blocking. It is designed to be fully async; as a consequence, APIs such as synchronous XHR and localStorage can't be used inside a service worker.

Service workers only run over HTTPS, for security reasons. Having modified network requests, wide open to man in the middle attacks would be really bad.

Technically, service workers provide a scriptable network proxy in the web browser to manage the web/HTTP requests programmatically. The service workers lie between the network and device to supply the content. They are capable of using the cache mechanisms efficiently and allow error-free behaviour during offline periods.

D. Web Workers

Allows a web app to run multiple threads of (JavaScript) code simultaneously. Thus, long activities can be moved off the user-interface thread, keeping responses snappy. They have a close relationship with Service Workers, but are more widely supported.

E. WebAssembly

Allows precompiled code to run in a web browser, at near-native speed. Thus, libraries written in languages such as C can be added to web apps. Due to the cost of passing data from JavaScript to WebAssembly, near-term uses will be mainly number-crunching (such as voice recognition and computer vision), rather than whole applications.

F. Indexed Database API

Indexed Database API is a W3C standard API for interacting with a NoSQL database. The API is supported by modern browsers and enables storage of JSON objects and any structures representable as a string.

G. Web Storage

Web Storage is a W3C standard API that enables key-value storage in modern browsers. The API consists of two objects, sessionStorage (that enables session-only storage that gets wiped upon browser session end) and localStorage (that enables storage that persists across sessions).

H. Application shell architecture

Some progressive web apps use an architectural approach called the App Shell Model. For rapid loading, service workers store the Basic User Interface or "shell" of the responsive web design web application. This shell provides an initial static frame, a layout or architecture into which content can be loaded progressively as well as dynamically, allowing users to engage with the app despite varying degrees of web connectivity. The shell can

be stored locally in the browser cache of the mobile device.

V. Abilities of PWAs

Progressive — Work for every user, regardless of browser choice because they're built with progressive enhancement as a core tenet.

Responsive — Fit any form factor: desktop, mobile, tablet, or forms yet to emerge.

Connectivity independent — Service workers allow work offline, or on low quality networks.

App-like — Feel like an app to the user with app-style interactions and navigation.

Fresh — Always up-to-date thanks to the service worker update process.

Safe — Served via HTTPS to prevent snooping and ensure content hasn't been tampered with.

Discoverable — Are identifiable as "applications" thanks to W3C manifests and service worker registration scope allowing search engines to find them.

Re-engageable — Make re-engagement easy through features like push notifications.

Installable — Allow users to "keep" apps they find most useful on their home screen without the hassle of an app store.

Linkable — Easily shared via a URL and do not require complex installation.

VI. PWAs versus Native apps

Native mobile apps deliver rich experiences and high performance, purchased at the expense of storage space, lack of real-time updates, and low search engine visibility. Traditional web apps suffer from the inverse set of factors: lack of a native compiled executable, along with dependence on unreliable and potentially slow web connectivity. Service workers are used in an attempt to give progressive web apps the best of both these worlds.

VII. Conclusion

The research shows that the increasing popularity of PWAs comes from the platform independence that makes it assessible the same with all features on all platforms. A PWA can run on almost every platform with a browser capable of executing HTML, CSS and JavaScript and can provide seem less functionality like a native app. The ability to execute without having to be loaded or installed on the device makes it even more useful. This also reduces the gap between different OS based apps and leads to unification of the world.

VIII. Acknowledgment

This research paper is possible because of the constant effort and guidance from Mr. Shambuj Bhardwaj and Dr. Danish Ather, their consistent motivation make me able to explore the possibilities and abilities of this new emerging technology Progressive Web Apps.

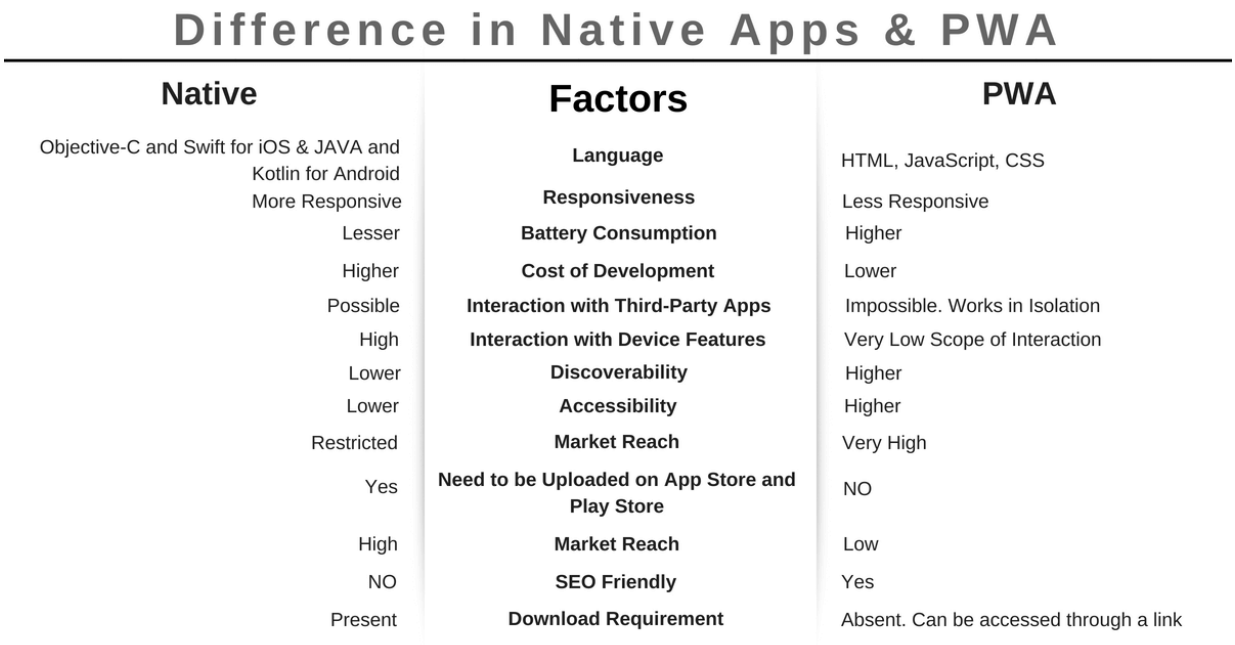

Fig.2 Differences between PWAs and Native apps

IX. REFERENCES

[1] Homepage of App inventive [online] https://appinventiv.com/blog/native-vs-progressive-web-apps

[2] Difference between PWA and Native [online] https://blog.magestore.com/pwa-vs-native-app/

[3] Google Developers Website [online] https://developers.google.com/web/progressive-web-apps/

[4] Wikipedia.org [online] https://en.wikipedia.org/wiki/Progressive_Web_Apps

[5] PWA rocks homepage [online] https://pwa.rocks/