# Progressive Web Apps: The Possible Web-native Unifier for Mobile Development

Andreas Biørn-Hansen[1], Tim A. Majchrzak[2] and Tor-Morten Grønli[1]
[1]*Faculty of Technology, Westerdals Oslo ACT, Christian Krohgs gate 32, 0186, Oslo, Norway*
[2]*ERCIS, University of Agder, Gimlemoen 25, 4630, Kristiansand, Norway*

Keywords: Progressive Web Apps, Service Workers, Cross-platform, Cross-platform Development, Mobile Web.

Abstract: A recent advancement of the mobile web has enabled features previously only found in natively developed apps. Thus, arduous development for several platforms or using cross-platform approaches was required. The novel approach, coined Progressive Web Apps, can be implemented through a set of concepts and technologies on any web site that meets certain requirements. In this paper, we argue for progressive web apps as a possibly unifying technology for web apps and native apps. After an introduction of features, we scrutinize the performance. Two cross-platform mobile apps and one Progressive Web App have been developed for comparison purposes, and provided in an open source repository for results' validity verification. We aim to spark interest in the academic community, as a lack of academic involvement was identified as part of the literature search.

## 1 INTRODUCTION

In traditional mobile application development, reusability of code between native apps, the web, and mobile platforms has been inherently non-existent. This is due to native apps' non-interoperable code bases, resulting in separate projects and developer environments when support for multiple mobile platforms and operating systems is desired (Heitkötter et al., 2013; Perchat et al., 2013; Majchrzak and Heitkötter, 2014).

For companies without the resources or the willingness to employ specialized mobile developers for each targeted platform, cross-platform development has become a popular alternative (Malavolta et al., 2015b). It typically not only reduces the development effort but also enables a quicker time-to-market (Rahul and Tolety, 2012). While its application and reputation varies in academia and industry communities (Puvvala et al., 2016; Mercado et al., 2016), the idea of a single code base – which can be deployed to multiple platforms – is seen as appealing for a variety of reasons. These include, but are not limited to, budget, human resources, and existing knowledge (Corral et al., 2012).

There are multiple approaches to cross-platform development (Heitkötter et al., 2012; Rieger and Majchrzak, 2016). In addition, a plethora of technical frameworks to support such development are either freely available under open source licenses, or as paid proprietary products. Examples of popular frameworks include Ionic Framework, PhoneGap, React Native and Xamarin (cf. Majchrzak et al. (2017)). Coincidentally these frameworks also represent three technologically distinctive approaches (Majchrzak et al., 2017).

Ionic Framework and PhoneGap both belong to the web- and Cordova-based *hybrid* approach, where user interface components are solely structured and styled using web technologies including HTML and CSS. React Native, of the interpreted approach, does not depend on a web-view as it instead leverages an on-device JavaScript interpreter along with native bridges, resulting in native interface components instead of web-based HTML components. Xamarin's approach is commonly referred to as *cross-compilation* as it compiles C# into native binaries for each supported platform. This results in native apps that are not dependant on interpreters or web-views (Latif et al., 2016). This can also be referred to as truly native (cf. also Heitkötter et al. (2013)).

Until recently, the web platform lagged behind on mobile-centred innovation in terms of being capable of competing with native or cross-platform apps (Puder et al., 2014). A new set of standards advocated by the Google Web Fundamentals group seeks to bridge that gap by introducing features such as offline support, background synchronisation, and home-screen installation to the web. The approach is known as Progressive

Biørn-Hansen, A., Majchrzak, T. and Grønli, T-M.
Progressive Web Apps: The Possible Web-native Unifier for Mobile Development.
DOI: 10.5220/0006353703440351
In *Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST 2017)*, pages 344-351
ISBN: 978-989-758-246-2

Web Apps (PWA), a term coined by Russel and Berriman (2015) in a blog post covering initial design ideas. PWAs are defined by a set of concepts and keywords including progressive, responsive, connectivity independent, app-like, fresh, safe, discoverable, re-engageable, installable, and linkable (Osmani, 2015). These are PWA's contributions to the unification of the mobile experience, where web apps can be installed and distributed without app marketplaces, work without Internet connectivity, receive push notifications and look like regular apps. This is possible due to the `Service Worker` API which empowers developers through the use of a background-executed script acting as a network and device proxy.

While the industry shows investment and interest in progressive web apps, a lack of academic research on the topic was identified as part of the theory search. Consequently, there is a substantial research potential, further discussed in section 6.2. This article aim to provide an introduction to the concepts and technologies behind progressive web apps. Moreover, it seeks to showcase features and to provide a technical comparison. These steps should help demystify and scrutinize the above compiled list of *buzz terms* for PWAs. Three technical artefacts in the form of mobile apps were developed using the Hybrid, Interpreted and PWA approaches. A comparison of start-up speeds, app sizes and render speeds is also provided as part of this article.

This position paper is structured as follows. Section 2 introduces our research methods. Related work is then compiled in Section 3. The results of our work with PWAs are compiled in Section 4. In Section 5 we discuss our findings. Eventually we draw a conclusion and point out to future work in Section 6.

## 2 RESEARCH METHODS

### 2.1 Literature Search

Initial searches for academic involvement in progressive web apps returned a limited amount of results per January 2017. A search for `Progressive Web Apps` on Google Scholar resulted in two theses, a keynote paper for the *Mobile!* conference and an IEEE paper on web app launch times.

Using the same search query, the Taylor & Francis Online article database returned zero results; as did ScienceDirect. The ACM Digital Library returned the aforementioned keynote paper. IEEE Xplore contained the aforementioned launch time paper, which merely mentions PWA (Gudla et al., 2016).

A ResearchGate search covers both publications and questions on their website, but it rendered no relevant results. Academic contributions are, per January 2017, virtually non-existent or not visible nor findable in the search engines explored.

An unknown base of research was identified for *Service Workers*. Searches for `Service Workers` among the aforementioned databases returned various results not constrained to computer science, but also within child care and the public sector. Other combined phrases, such as {`Service Workers`, `web`, `API`} did manage to filter the outcome to a certain degree, still polluted with irrelevant results.

### 2.2 Design Implementation

To gain better understanding of the possibilities of progressive web apps, three technical artefacts were developed for comparison. An *hybrid* app was developed using the Ionic Framework. An *interpreted* app was developed using React Native. Lastly, a *progressive* Web App was developed using React.js. As our main aim is to introduce PWA as a concept together with some technical and higher-level comparisons, we excluded the aforementioned cross-compilation approach from our study.

The applications employ a master-detail navigation pattern, where the master view presents a list of clickable items fetched from an API. Upon list item click, the app navigates to the detail view, displaying the item image together with its author and title. The data is fetched from `www.reddit.com/r/Art/.json`.

All three technical artefacts have been open-sourced[1] to allow verification of the results. The artefacts are depicted in Figures 1, 2 and 3 (see p. 3).

### 2.3 Performance Testing

Three variables were included as part of the performance test. The tests were conducted on a Google Nexus 5X device running Android version 7.0. The aim was to gather initial data on differences in performance between the implementations discussed in Subsection 2.2.

**Size of installation.** Information regarding app size was found at the following path on the Android device: `Settings – Apps – [`*app name*`]`.

**Activity launch time.** This was measured using the *Android Debug Bridge* and its `am_activity_launch_time` command. This measures the time it takes to launch the first activity of an application. The test was conducted ten times for each

[1] https://github.com/andreasbhansen/pwa-paper

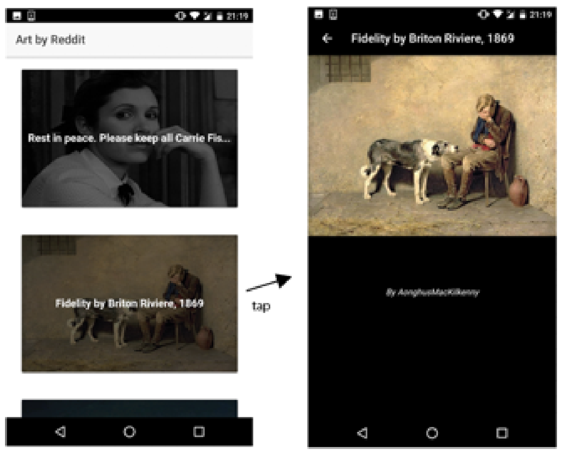

Art by Reddit
Rest in peace. Please keep all Carrie Fis...
Fidelity by Briton Riviere, 1869
tap
Fidelity by Briton Riviere, 1869
By AonghusMacKilkenny

Figure 1: The Ionic Framework Hybrid app.

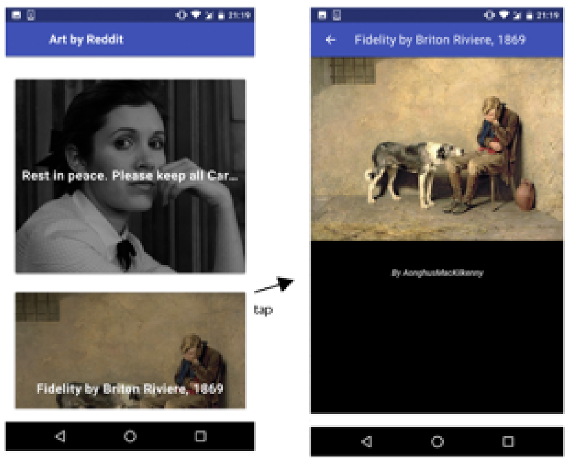

Art by Reddit
Rest in peace. Please keep all Car...
Fidelity by Briton Riviere, 1869
tap
Fidelity by Briton Riviere, 1869
By AonghusMacKilkenny

Figure 3: The Progressive Web App.

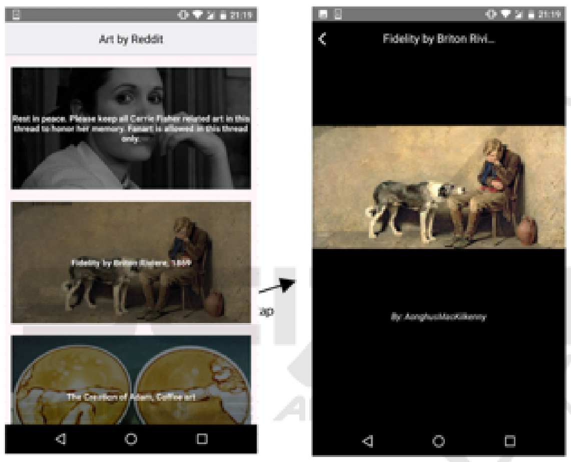


Figure 2: The React Native Interpreted app.

app as the results rendered different at each run. The average time is presented as result.

**Time From App-icon Tap to Toolbar Render.** As no suitable method for measuring elapsed time from *app-icon tap* to user interface *toolbar render* was identified, an online stop watch was used. The toolbar, titled ”Art by Reddit”, is displayed in Figures 1-3. In our applications, this element is the first to be rendered to the screen, thus we used it as the basis for our render-time measurement. As the measurement relied on human reactions, and is thus inherently error-prone, the test was conducted ten times for each app, and the average time is presented as result.

## 3 RELATED WORK

While the lack of academic involvement and research on progressive web apps is evident from the literature search, an established industry-oriented body of knowledge was identified. The *Google Web Fundamentals* group acted as the driving force for the creation and publishing of tutorials and blog posts. Until a solid academic knowledge-base is developed, the Web Fundamentals web site should, thus, be the foundation for upcoming research.

Other than Google-created content, two early-access books were identified (Ater, 2017; Hume, 2017) together with an extensive pool of industry articles. Examples of topics covered are fundamental concepts (Edwards, 2016), challenges and concerns (Mahemoff, 2016), and thoughts on the impact of web innovation (Rinaldi et al., 2016). A notably larger academic foundation was found for the other app development approaches discussed. Numerous papers on cross-platform app development and underlying approaches have previously been identified. Papers found to often recur in related research includes Heitkötter et al. (2012), Xanthopoulos and Xinogalos (2013), Dalmasso et al. (2013), and Palmieri et al. (2012). They make up the theoretical foundation and early research, which newer papers draw from. Their topics range from comparisons of technical frameworks and approaches, to classifications and performance measurements.

Recent research identified includes, but is not limited to, topics such as requirements for cross-platform tooling (Gaouar et al., 2016), development approach evaluation frameworks (Rieger and Majchrzak, 2016), energy consumption comparisons (Ciman and Gaggi, 2016), and end-user perception of cross-platform apps (Mercado et al., 2016; Malavolta et al., 2015a). Their contributions should be acknowledged as important foundation for future research on progressive web apps.

Due to the lack of academic contributions regarding PWA, the keen industry interest should act as a catalyst for further research and more academic involvement. Research suggestions are presented in Section 6 in an attempt to spark interest in relevant research communities.

## 4 RESULTS

### 4.1 Feature Comparison

This section seeks to provide insights into differences between interpreted apps, progressive web apps, hybrid apps and native apps by comparing a set of features and concepts. It also presents insights on such as technical frameworks and experience-unification for end-users.

Table 1 provides a non-exhaustive list of feature available in PWAs as of January 2017, along with their compatibility. Remarks follow subsequently.

Table 1: Feature-comparison of approaches.

| Feature | Interpreted | PWA | Hybrid | Native |
|---|---|---|---|---|
| Installable | Yes | Yes[a] | Yes | Yes |
| Offline capable | Yes | Yes | Yes | Yes |
| Testable before installation | No | Yes | No | No |
| App marketplace availability | Yes | Yes[b] | Yes | Yes |
| Push notifications | Yes | Yes[c] | Yes | Yes |
| Cross-platform availability | Yes | Limited[d] | Yes | No |
| Hardware and Platform API access | Yes | Limited[e] | Yes[f] | Yes |
| Background synchronisation | Yes | Yes | Yes | Yes |

(a) The `Enable improved add to Home screen` developer flag in Chrome Canary for Android can be enabled in order for PWAs to be installed like normal apps (Joreteg, 2016).

(b) PWAs will be made searchable from the Windows 10 app marketplace, thus becoming “first-class citizens” of their app ecosystem (Rossi, 2016).

(c) Push notification support through the `Push` API[2] is available, but limited to certain browsers.

(d) As Apple’s Safari browser does not yet support the Service Workers API, the iOS platform is yet to fully realise and leverage the potential of the technological advancement.

(e) A PWA can use HTML5-based APIs for hardware and platform access in addition to features and functionality made possible by Service Workers.

(f) Hardware and platform access for Hybrid apps is usually provided by Cordova, a library for handling the bridging between a native app’s web-view component and the device’s APIs.

### 4.2 Technologies and Concepts

#### 4.2.1 Framework Agnostic

A plethora of frameworks for web development exists (Smeets and Aerts, 2016). The Web Fundamentals group demonstrated framework agnosticity by implementing PWAs in three different frameworks (Osmani, 2015).

#### 4.2.2 Service Workers

The Service Worker[3] is responsible for most of the core features associated with progressive web apps (Gaunt, 2016). A PWA cannot properly work in browsers without Service Worker support. The worker is registered on a user’s first page visit. It consists of a JavaScript file embodying lifecycle hooks for business logic and cache control. It can be used to handle tasks such as background synchronisation (Archibald, 2016), caching mechanisms for data and application shell, as well as interception of network requests (Osmani and Gaunt, 2017).

#### 4.2.3 Application Shell

The application shell is defined by the Google Web Fundamentals group as “[...] the minimal HTML, CSS, and JavaScript powering a user interface.” Osmani and Gaunt (2017). They list three criteria for the shell: fast loading time, cached, and displaying dynamic content. Data is pulled from external APIs.

#### 4.2.4 Web App Manifest

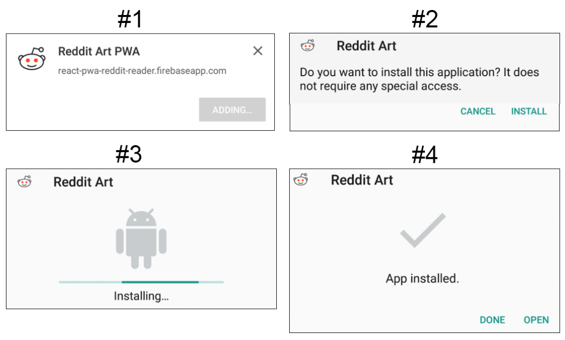
The purpose of the manifest file is to expose certain modifiable settings to app developers. These settings include such as logo image path, app name, splash screen and more. In short, the manifest can be used to modify behaviour and style of PWA applications.

#### 4.2.5 Security through HTTPS

For security reasons, HTTPS is required for a Service Worker to register in the browser and accordingly act on events (Gaunt, 2016). The reason for enforced security is described by Gaunt (2016), as using the “[...] service worker you can hijack connections, fabricate, and filter responses”.

[2]https://www.w3.org/TR/push-api/

[3]https://www.w3.org/TR/service-workers/



Do you want to install this application? It does not require any special access.

Figure 4: PWA installation flow in Chrome Canary.

### 4.3 Unification of Mobile App and Web Experiences

An evident difference between web apps and regular mobile apps is their *explorability*. A regular app requires search and installation via an app marketplace. Progressive web apps enable the best of both approaches, where end-users can easily experience an application through their web browser, then choose to install it via an "Add to Home screen" banner prompted (see Figure 4). The banner will only be prompted to the user if certain qualifications are met (Pedersen, 2016). Figure 4 illustrates the installation process for the PWA designed and implemented for this paper (cropped screenshots).

The illustrated experience is per January 2017 only achievable by enabling the `Improved PWA Installation` flag in the Chrome Canary for Android settings. Without that flag enabled, the experience is comparable to bookmarking a site to the home screen.

Together with the full-screen experience of progressive web apps, the installation prompt can be considered an advance in unification of end-user experience and mobile web perception. Instead of forcing users to download an app from a marketplace, they can experience the product in their web browser as a regular website before installing it via the banner. As we discuss in our list of suggestions for future work, research on potential privacy and security concerns in this regard should be of interest to all parties.

### 4.4 Measurement Comparison between PWA, Interpreted and Hybrid Apps

Table 2 presents a comparison of three different measurements: installation size, launch time and render time. The installation size of the progressive web app is about 157 times smaller than the React Native-based interpreted app, and about 43 times smaller than the Ionic Framework-based hybrid app. Where the hybrid app used more than 9 seconds to render the app's toolbar, the interpreted app used around 860ms on the same task. The PWA rendered different results when (a) Chrome Canary did not run in the background, and (b) it did run in the background, regardless of open website. This is due to PWA's browser dependency.

Table 2: Measurement-comparison of approaches.

| Measure | Hybrid | Interpreted | PWA |
|---|---|---|---|
| Size of installation | 4.53MB | 16.39MB | 104KB |
| Launch time | 860ms | 246ms | 230ms |
| Time from app-icon tap to toolbar render | 9242.1ms | 862ms | (a) 3152ms<br>(b) 1319ms |

## 5 DISCUSSION

### 5.1 Basics

From the perspective of web-native unification, there are certain major limitations to the PWA approach compared to hybrid, interpreted and native apps. We use this section to spark interest by elaborating on a set of these limitations based on our findings, and suggest that future research dives deeper into this regard.

As discussed by Malavolta (2016), a PWA cannot access hardware- and platform-level features not supported by the respective browser. Examples of such non-included features are native calendar and contact list access. However, an increasing pool of platform APIs are becoming available in newer browsers.

For progressive web apps to have the same potential as cross-platform app development, support for `Service Workers` in Apple's iOS Safari browser is a requirement. Without such support, a PWA-enabled website would not deliver a consistent experience across browsers and platforms.

### 5.2 Feature Comparison

Table 1 highlighted differences in feature compatibility between approaches for app development. There are certain profound differences between them, some being inherent characteristics.

PWAs are the only option among the listed approaches that naturally enables testing of an app before installation, due to being accessible in web browsers. If app marketplace presence is required, the three other approaches fully supports such, with a potential PWA entry into the Microsoft app store.

Offline capabilities, push notifications and background synchronisation are available regardless of approach. Cross-platform compatibility is found in the interpreted, hybrid and PWA approaches, the latter with some limitations. As mentioned, this is one of the main reasons why cross-platform approaches have become popular alternatives to native development.

For apps relying on features not found in or requiring performance not achievable in a current browser, the alternative approaches are more suitable. The three artefacts developed as part of this paper are limited in terms of device-platform communication requirements, as they do not require any APIs other than those commonly found in browsers. Progressive web apps are constrained to platform APIs made available through the browser, thus relying on W3C and browser vendors (Puder et al., 2014).

### 5.3 Measurement Comparison

Table 2 presented the results from three measurements conducted for preliminary data gathering. It thereby also provides an anchor for further research.

The results render the presence of certain trade-offs. While the interpreted app installation size is 157 times larger, it is also more than 3.5 times faster to render meaningful content compared to the progressive web app when launching the latter without having Chrome running in the background. If launched with Chrome already running, the gap between the two apps was down to 457ms, still favouring the interpreted one.

The differences are more remarkable when comparing the interpreted app to the hybrid one, with the latter using 10 times longer to render the toolbar. It is worth noting that the hybrid app framework, Ionic 2, was in beta release at the time of implementation, thus, the render-time could be affected by this. However, the code-base was built using production settings, as recommended by the documentation. Thus, we made efforts to mitigate and understand the render-time.

The presented results are limited in that the apps were only tested on one specific device, the Google Nexus 5X, running Android 7.0. The conducted tests do not build on previous research or established methods for measuring performance. The purpose of the tests was to gather and present preliminary results to spark interest for further work. The named limitations are owed to the very early stage of work on the topic; in fact, our mitigation of limitations is even beyond a typical position paper already.

## 6 CONCLUSION AND FURTHER WORK

### 6.1 Conclusion

The industry is investing resources into progressive web apps (PWA) and the development of learning material. The lack of academic involvement denotes a significant knowledge gap but at the same time provides research potential. This paper is an effort to raise awareness of PWA in the academic community by contributing with an introduction to the concepts and technologies, comparison of feature and measurement against established approaches, open source artefacts for result verification, and suggestions for further research.

Per January 2017, the Google Web Fundamentals group is one of the leading driving forces behind advocacy of PWAs. They could be considered as the main publisher of learning material.

The current state of progressive web apps involves a lack of certain hardware and platform APIs and features that only (certain) cross-platform and native apps can access. Recent browser advancements have been forces of unification for the end-user app experience, including, but not limited to, installable and native-looking web apps through PWAs. While Chrome is leading the way for PWA browser support, Apple's iOS Safari is yet to support the necessary Service Worker API.

We find that there is much potential for PWAs to become a unifier for web-native development without the use of cross-platform frameworks. As an end-user, the PWA installation process becomes more similar to regular apps through new advancements in user experience aspects. Web apps can look, feel and act similar to native, hybrid and interpreted apps. While there are hardware and platform API limitations to PWAs not found in the other approaches, product requirements and specifications will in the end be the deciding factor for choice of approach. We would like to conclude with an encouraging note, a quote by Archibald (2016) from the 2016 *Google I/O Conference*: "We want everything that ends up on the home screen to be competitive with native apps. We want to make the web a first-class part of the operating system in the user's mind".

### 6.2 Suggestions for Further Work

One can possibly apply research questions from cross-platform, native, and mobile web app development to research on progressive web apps. This results in a vast and existing knowledge base that can be used as the foundation for PWA and next-generation web research.

The list below provides suggestions for further research of technical, economical, and sociological nature:

- Continue the development of technical implementations for comparison and evaluation purposes against established development approaches.
- In terms of social and economic aspects, look into challenges such as mobile data fees in developing countries and associated costs in downloading marketplace apps compared to progressive web apps.
- Study the impact of "Add to Home screen" versus installation of apps through the app marketplace. Are users more willing to install a PWA as they can test it beforehand? Will ad-hoc usage and installation eventually merge?
- App marketplaces are constrained and deliver a certain degree of security. Does the potential lack of such constraints impact the user's choice of installing a PWA?
- Are there security and privacy risks newly introduced with PWAs?
- Look into possible sales models for progressive web apps, which currently disturbs the established app marketplaces.

## REFERENCES

Archibald, J. (2016). Instant loading: Building offline-first progressive web apps.

Ater, T. (2017). *Building Progressive Web Apps: Bringing the Power of Native to the Browser*. O'Reilly.

Ciman, M. and Gaggi, O. (2016). An empirical analysis of energy consumption of cross-platform frameworks for mobile development. *Pervasive and Mobile Computing*.

Corral, L., Janes, A., and Remencius, T. (2012). Potential advantages and disadvantages of multiplatform development Frameworks–A vision on mobile environments. In *Procedia Computer Science*, volume 10, pages 1202–1207. SciVerse ScienceDirect.

Dalmasso, I., Datta, S. K., Bonnet, C., and Nikaein, N. (2013). Survey, comparison and evaluation of cross platform mobile application development tools. In *Proc. 9th (IWCMC)*, pages 323–328.

Edwards, A. R. (2016). The building blocks of progressive web apps – smashing magazine.

Gaouar, L., Benamar, A., and Bendimerad, F. T. (2016). Desirable requirements of cross platform mobile development tools. *Electronic Devices*, 5:14–22.

Gaunt, M. (2016). Service Workers: an introduction.

Gudla, S. K., Sahoo, J. K., Singh, A., Bose, J., and Ahamed, N. (2016). Framework to improve the web application launch time. In *Proc. 2016 IEEE Int. Conf. on Mobile Services (MS)*, pages 73–78. IEEE Press.

Heitkötter, H., Hanschke, S., and Majchrzak, T. A. (2012). Evaluating Cross-Platform development approaches for mobile applications. In *Web Information Systems and Technologies*, Lecture Notes in Business Information Processing, pages 120–138. Springer.

Heitkötter, H., Majchrzak, T. A., and Kuchen, H. (2013). Cross-Platform Model-Driven Development of Mobile Applications with MD$^2$. In *Proc. 28th ACM SAC*, pages 526–533. ACM.

Hume, D. A. (2017). *Progressive Web Apps*. Manning.

Joreteg, H. (2016). Installing web apps on phones (for real).

Latif, M., Lakhrissi, Y., Nfaoui, E. H., and Es-Sbai, N. (2016). Cross platform approach for mobile application development: A survey. In *2016 Int. Conf. on Information Technology for Organizations Development (IT4OD)*, pages 1–5. IEEE.

Mahemoff, M. (2016). Progressive web apps have leapfrogged the native install model ... but challenges remain.

Majchrzak, T. A., Biørn-Hansen, A., and Grønli, T.-M. (2017). Comprehensive analysis of innovative cross-platform app development frameworks. In *Proc. 49th HICSS*. IEEE Computer Society.

Majchrzak, T. A. and Heitkötter, H. (2014). Status Quo and Best Practices of App Development in Regional Companies. In Krempels, K. and Stocker, A., editors, *Revised Selected Papers WEBIST 2013*, volume 189 of *LNBIP*, pages 189–206. Springer.

Malavolta, I. (2016). Beyond native apps: Web technologies to the rescue! (keynote). Pro. 1st Int. Workshop on Mobile Development. ACM.

Malavolta, I., Ruberto, S., Soru, T., and Terragni, V. (2015a). End users' perception of hybrid mobile apps in the google play store. In *2015 IEEE Int. Conf. on Mobile Services*, pages 25–32. IEEE.

Malavolta, I., Ruberto, S., Soru, T., and Terragni, V. (2015b). Hybrid mobile apps in the google play store: An exploratory investigation. In *Proc. 2nd ACM Int. Conf. on Mobile Software Engineering and Systems*, MOBILESoft '15, pages 56–59. IEEE Press.

Mercado, I. T., Munaiah, N., and Meneely, A. (2016). The impact of cross-platform development approaches for mobile applications from the user's perspective. In *Proc. Int. Workshop on App Market Analytics*, WAMA 2016, pages 43–49. ACM.

Osmani, A. (2015). Getting started with progressive web apps.

Osmani, A. and Gaunt, M. (2017). Instant loading web apps with an application shell architecture.

Palmieri, M., Singh, I., and Cicchetti, A. (2012). Comparison of cross-platform mobile development tools. In *Proc. 16th Int. Conf. on Intelligence in Next Generation Networks*, pages 179–186. IEEE.

Pedersen, M. (2016). Progressive web apps: Bridging the gap between web and mobile.

Perchat, J., Desertot, M., and Lecomte, S. (2013). Component based framework to create mobile cross-platform applications. In *Procedia Computer Science*, volume 19, pages 1004–1011. ScienceDirect.

Puder, A., Tillmann, N., and Moskal, M. (2014). Exposing native device APIs to web apps. In *Proc. 1st Int. Conf. on Mobile Software Engineering and Systems*, pages 18–26. ACM.

Puvvala, A., Dutta, A., Roy, R., and Seetharaman, P. (2016). Mobile application developers' platform choice model. In *Proc. 49th HICSS*, pages 5721–5730. IEEE.

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

Rieger, C. and Majchrzak, T. A. (2016). Weighted evaluation framework for Cross-Platform app development approaches. In Wrycza, S., editor, *Information Systems: Development, Research, Applications, Education*, Lecture Notes in Business Information Processing, pages 18–39. Springer.

Rinaldi, B., Holland, B., Looper, J., Motto, T., and VanToll, T. J. (2016). Are progressive web apps the future?

Rossi, J. (2016). The progress of web apps - microsoft edge dev blog.

Russel, A. and Berriman, F. (2015). Progressive web apps: Escaping tabs without losing our soul.

Smeets, R. and Aerts, K. (2016). Trends in web based cross platform technologies. *International Journal of Computer Science and Mobile Computing*, 5(6):190–199.

Xanthopoulos, S. and Xinogalos, S. (2013). A comparative analysis of cross-platform development approaches for mobile applications. In *Proc. 6th Balkan Conf. in Informatics*, BCI '13, pages 213–220. ACM.