Will Progressive Web Apps make native mobile apps obsolete?

Nicolas Raube

November 30, 2018

Written elaboration of the special learning achievement

Schriftliche Ausarbeitung der besonderen Lernleistung

Contents

1 Introduction

Today, we all are carrying small computers around in our pockets. Thanks to instant messengers, we can communicate for free with people on the other side of the planet. We can look up where to go next to reach our destination. In a city, it has become impossible to leave the house without seeing people staring at their smartphones. Apps have become a massive part of our lives and they, as mentioned, offer tons of useful features. However, the trend with apps seems to cool down.

According to ComScore, 51% of users downloaded not a single app during one month in 2017 [Lella and Lipsman, 2017, p. 12].

People also spent most of their time (53%) with only the top 10 apps. Meanwhile, less popular apps (ranked 501 and higher) got a mere 9% of a user's time. [Lella and Lipsman, 2017, p. 10]

Another critical statistic from ComScore shows that even though apps reached a significantly higher usage time than the mobile web (16 times as high), the mobile web had reached 2.2 times more people [Lella and Lipsman, 2017, p. 9].

There is now a way to combine the long usage times of apps and a bigger audience from the web. This can be achieved with the Progressive Web App (PWA).

In simple terms, a PWA is a website that acts like a native app. It adopts the advantages from the web and supplements the features of a native app. There is serious potential for PWAs to be even better than native apps—for developers and users.

Over recent years, multiple highly reputable companies have developed their own PWA, e.g., Forbes, Alibaba, The Washington Post or Twitter.

This gathered the author's attention who is interested in this topic because he is a software developer. He published multiple apps for Android, and he wants to find out which type of app he should develop in the future.

This paper aims to analyze if and how PWAs can render native apps obsolete, in order to find out if developers should continue developing native apps. Analyzing the features and capabilities of both app types, a conclusion is found. It will be explained what the original advantages of native apps in comparison to websites are. In compliance with that, the features of a PWA are examined and the question of why anyone would want to use them instead will be answered. In the end, it will be evaluated if PWAs will replace native apps.

2 Definition of app types

2.1 What is a native app?

A native app is what is commonly used on a smartphone. It is an application built specifically to run only on one operating system, and it is developed using a programming language that is

executable on this operating system [Rouse, 2013] because the architecture provides a runtime environment for a specific language. For example, apps for Android typically are written in Java or Kotlin [Google, 2018a], iOS apps in Swift [Apple, 2018a]. They first have to be installed on the device before the user can take advantage of them [Rouse, 2013], usually through an app store.

2.2 What is a Progressive Web App?

Basically, a PWA can be imagined as a website pretending to be a native app. When such a website is visited, the browser prompts users to add it to their home screen. The website can now "feel" like a native app. It is a paradigm which was first named so by Google developers. Google defines which criteria a website has to fulfill to qualify as a PWA. There is not much more to it than a website built with Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript. Those requirements can be found in Google's "Progressive Web App Checklist" [Google, 2018f].

First, there needs to be a Web App Manifest which is nothing more than a JavaScript Object Notation (JSON) file defining metadata of a PWA. It tells the browser that the user should be able to add the website to the home screen as a PWA. Developers can use it to define, for example, what the title that is shown on the home screen will be, what the icon on the home and splash screen with the background color will be or if the app should be launched without the browser user interface (UI). [Gaunt and Kinlan, 2018]

All pages should be responsive to the screen size so that it runs on any device with a web browser. Furthermore, the website needs a Service Worker, a script that can manipulate the website's network requests and runs in the background even if all tabs and the browser are closed [Google, 2018d]. As a Service Worker requires the site to be delivered through Hypertext Transfer Protocol Secure (HTTPS), such a more secure connection is another criterion for a PWA. [Google, 2018f]

Every web developer should make sure that the developed website runs on every platform and in every browser. This can be a real challenge if the site should run correctly on even Internet Explorer 6. When this version was last updated, modern technologies such as ECMAScript 6 maybe were not even thought of. The outdated browser versions are still around today, which means that many new technologies are not available to the users of those outdated browsers. This is where the term "progressive" comes in.

It is taken from the concept "progressive enhancement", meaning that developers will check if a particular feature, e.g., the Service Worker, is available in the current browser. If it is, everything is fine, but if it is not, developers should try to use alternatives for these older browsers. In the worst case, the website will remain a basic website that can in no way replace a native app. Progressive

enhancement makes sure all the essential PWA features are available on the browsers that support them, and if the browser does not support all the features, the site remains usable at least. The concept is not a must-have for a PWA, but if it is not respected, the website will work correctly only for the users with up-to-date browsers. [Ater, 2017, p. 21]

3 Analysis of features

Now that both types of applications evaluated in this thesis have been defined, the advantages of native apps over traditional websites need to be worked out to understand why native apps have become popular in the past and what PWAs have to offer to be able to replace native apps.

3.1 Why are native apps used instead of websites?

3.1.1 Installable

Native apps need to be installed on the user's device. That is one of the most important factors why native apps have succeeded because it allows native apps to run offline. A package is downloaded once onto the device, installed and everything the app needs is available on the device, without having to communicate with a server as traditional websites have to do on every reload. An exception to this is a native app that works with dynamically fetched data, e.g., a weather app, but in this case, the native app can at least load the latest fetched data from local storage or show an error message if it has no access to the internet.

3.1.2 Engaging

To open a website on a mobile device, the user opens a web browser, enters a Uniform Resource Locator (URL), or navigates to the page through a search engine. That page is now surrounded by the browser's UI and possibly the operating system's UI like the status bar. This approach is a bit distracting but necessary. A native app does it differently: It has its own dedicated launch icon on the home screen. This allows instant access after unlocking the phone. Furthermore, users are reminded of the app every time they look at their home screen. When opened, there is no UI from the browser, which makes the user experience more immersive than that of a web page. One could argue that a traditional web page can be added to the home screen, but that works more like a bookmark that again opens the page within the browser UI.

Native apps can also continue working in the background and send push notifications. This means a server sends a message to the clients and they display a notification even when the app is closed. This concept is being used in many apps: For example Whatsapp, which notifies users of

incoming messages. Other common use cases include notifying the user of temporary discounts or reminding them of using the app. Websites could not use such a re-engaging feature to their advantage, giving developers another reason to use native apps.

3.1.3 Better Performance

As mentioned before, a native app is compiled and downloaded. It is written in a programming language executable on the operating system, and the performance is better than what can be achieved with a typical website. On the contrary, a website's HTML and CSS code, once downloaded, first needs to be interpreted by the browser and rendered. JavaScript is interpreted and executed line by line, and additional resources such as images or fonts are pulled in. This procedure slows down the whole experience.

3.1.4 Device API Access

Today's popular operating systems expose various application programming interfaces (APIs) allowing native apps to access the device's hardware and integrate with the operating system. For example, the native app could, possibly requiring the correct permission, access the camera, microphone or battery status. Even sensors like the ambient light sensor and more are available.

3.1.5 Easier monetization

On the web, a lot of revenue is made through advertising. The popularity of advertisements may be justified by the relative simplicity of displaying ads by connecting with an advertising network like Google AdSense. If developers planned to accept direct payment by their users, they had to choose or implement their own payment solution and process, which can be quite a challenge. Users then would have to enter their bank information every time, not even being sure if they can trust the site [Ater, 2017, p. 239/240].

Native apps can be monetized more easily because developers are not only able to display advertisements, but they can sell the app itself through an app store, generating a revenue linear to the number of downloads. The app store can save the user's payment information and reuse it, reducing the work required by the user to pay. The popular stores also offer an in-app-purchasing service, that is universal across all apps. This unified payment system is made as easy as possible by the app store, which brings the native app's payment process to a higher level of trust from the user and simplifies receiving money for developers. In return, the store demands a share of the payment. These monetization methods have contributed a lot to the rise of native apps in comparison to websites.

3.1.6 App Store Distribution

App stores are a central place where users can look for native apps. They allow searching for keywords or browsing categories such as “gaming” or “education”. Given that an application is well-developed and has already gained a certain amount of attention, app stores often decide to promote or recommend them to the user. This usually leads to tremendous popularity of the selected app.

In contrast to that, the web has search engines to find websites by keywords, but they do not allow browsing categories of websites and do not promote individual websites.

3.2 Can PWAs keep up with native apps?

In order to to be able to replace native apps, PWAs have to fulfill every criterion of why native apps could emerge and were being used instead of websites. In the following, it will have to be analyzed if and how a PWA can match these requirements.

3.2.1 Offline Functionality

Hearing the words “website” and “offline” invokes a feeling of failure. Users associate them with not being able to access content on the web because offline working websites were not widely spread. Making a website work offline was only possible with the now deprecated Application Cache API, which had its deficits.

Jake Archibald has written a thorough article on these shortages. For example, he states that cached resources would only be checked for newer versions if the manifest defining the resources available for offline use would be updated. Images varying in size could not be cached based on the user’s screen size, which means it would make little sense to serve smaller images for smaller devices because every image (in every size) had to be cached. [Archibald, 2012]

Having a connection to the internet was a precondition most of the time for websites, and offline functionality was the job for a native app. Offline functionality still is not associated with websites. Today, more and more websites work offline, but how do they accomplish the goal of connection-independent availability if not with the Application Cache API?

The web standards are in constant development, further improving the capabilities of the browser and thus the web. One of those improvements was introduced as the Service Worker.

Service Worker

The Service Worker is a JavaScript script that runs on a separate thread in the background just as the Web Worker, of which the Service Worker is a descendant [Google, 2018d]. The difference

between the two is that a Service Worker can control network requests by sitting between the website and the server, passing the requests to each other after processing them [Ater, 2017]. It runs independently of the site that it was created by and still runs if the tab and even if the browser is closed (at least on mobile operating systems) [Google, 2018d].

Service Workers are, amongst other things, responsible for making a website work offline: When catching a network request, it can access the data and, using the Cache API, cache it. If network requests fail (for example, when the user is offline), the Service Worker manipulates the network requests to serve not the web-fetched data (which would be impossible), but the cached data. Also, the developer can clearly decide which resources to cache. Static content will be wanted to be cached more than dynamically loaded resources.

HTTPS is another criterion for a PWA because it is required by Service Workers to work [Google, 2018d]. Imagine a scenario in which a user is waiting in a public space with an open Wi-Fi network. A hacker could, if the connection to any site the user visits is insecure, initiate a "man-in-the-middle" attack [Ater, 2017, p. 21]. This would allow the hacker not only to read the user's data but also to write and send data to the user, possibly injecting JavaScript code. Now the hacker could also install a Service Worker, which can read and alter every request and will continue to run in the background. It could forward the user request to a malicious site or inject JavaScript into a site, even if the hacker finished the attack and the user is connected to a different network on the other side of the earth. HTTPS does not fully protect from "man-in-the-middle" attacks, but it prevents a significant amount of attacks from succeeding.

Therefore, a website, and thus a PWA, can function correctly even when the user is offline using the Service Worker. Though, what happens if the user is online, but the website's server is out of service? In this case, the Service Worker's network request will fail in the same way as if the user would be offline, serving the requested data from the cache. A PWA can fully adopt a native app's offline behavior, except the way it is achieved is different: A native app has to be downloaded and installed, while a PWA is downloaded and possibly served from the cache.

3.2.2 Performance

A website's performance plays a significant and crucial role in the sales of an online business. For example, Google ranks faster pages higher in search results [Dlugos, 2018]. Logically, the faster a website loads and the less any delay is noticeable, the more likely a user will stay on the website and spend money. Fast Company's Kit Eaton states that Amazon calculated that if their page load time were to increase by just one second, they would earn $1.6 billion less in one year [Eaton, 2012]. Every millisecond in load time has enormous consequences for a business, especially if it is so large a player.

Native apps have no performance problems, given a well-implemented app runs on a device with sufficient hardware. It is easy for website developers, though, almost to ignore performance and reach a loading time that is everything but optimal. Especially large software libraries or images can slow down the download process. We already have got to know one tool to render many loading times irrelevant for a PWA: the Service Worker.

If the requested data does not come from the server, but from the cache, the PWA does not need to take a trip to the web for every resource.

Google has found that the cache used by Service Workers enables even faster loading in comparison to the usual HTTP cache [Walton, 2016].

A particular architecture utilizing the cache is the Application Shell Architecture. With this architecture, the fundamental static UI elements that are needed on the pages of the website are cached, e.g., a navigation bar, menus or the logo. Dynamic content is loaded after displaying the cached content. [Ater, 2017, p. 86]

It is also possible to cache resources from pages the user is likely to visit next, so the waiting time is reduced to a minimum even if the user has never before visited the next page [Ater, 2017, p. 84].

In the first place, the page has to be downloaded on the first visit and can only make use of the cache on following visits. This means that if users have a rather slow connection to the internet, as is often the case in developing countries, a PWA may render and run slower than a native app on the first visit would. A native app has to be downloaded and installed before being usable.

Developers should stick to the PRPL pattern to optimize time-to-interactive. It means making use of HTTP/2 Push to send multiple resources along the initial request, rendering as fast as possible, pre-caching routes and lazy-loading [Sheppard, 2017, p. 243/244].

Debatable is if PWAs are able to provide the same run-time performance as native apps. For rendering and animations, the browser can make use of hardware acceleration and the GPU with WebGL [Ater, 2017, p. 242]. WebVR allows to implement virtual reality (VR) applications on the web today [Ater, 2017, p. 245] and animation with 60 frames per second is possible even on the mobile web [Rosário, 2016].

The VR documentary “Bear 71” was being reimplemented as a WebVR app in 2016. They state that a lot of optimization was necessary to “run at good framerates across most devices.” [Dysinski, 2017]

Currently, complex VR apps or games seem to challenge the web’s performance.

JavaScript is doubted to be able to keep up with native apps perceptibly, but browsers optimize JavaScript code to run as efficiently as possible. It depends on the hardware being used if a PWA can perceptibly keep up with a native app. With computers and smartphones speeding up every year, this will become less of a concern in the future.

If developers still need more performance in a PWA, e.g., for complex algorithms or games, they can use WebAssembly, which promises near-native performance [Miller, 2018].

According to Mozilla's David Bryant, this is achieved by parsing and compiling code before being downloaded by the browser. This code can be executed extremely fast. [Bryant, 2017]

An example of a successful PWA from the real world is the e-commerce Jumia, which has ditched its native app for a PWA. They are based in Africa where most users are connected to 2G networks. Nevertheless, they have a 33% higher conversion rate amongst other increasing metrics. [Google, 2017a]

This means PWAs already can be a viable alternative to native apps regardless of network conditions and device capabilities. PWAs can keep up with a native app's performance if adequate caching-strategies and other performance-oriented techniques are being used.

3.2.3 Engagement / Immersive Experience

Immersive

While a traditional website can only be added to the user's home screen like a bookmark, the PWA can be added to the home screen in a more native-like fashion. This can be done by the user in two different ways. For the sake of simplicity, the process will be described as it is found in Chrome. Other browsers might implement slight differences.

The first way is opening the browser menu and selecting "add to home screen." This will open a dialog showing the name of the app and two buttons to cancel or initiate "installation". Once the user clicks on the "add" button, the user can find the PWA on the home screen, just like a native app.

The other way is to use the information bar shown on top of the website by the browser if specific criteria are met. The user can click on "Add app to Home screen," and the final confirmation dialog is shown again.

The required criteria include an engagement threshold that shows the user is interested in the PWA. In Chrome it is currently exceeded if the user stayed on the site for longer than 30 seconds. The site also has to have a Service Worker listening to the "fetch" event. Finally, the PWA needs a manifest including a name, icon, the initial URL to load and the display mode (e.g., fullscreen). [LePage, 2018]

The developer can decide if the PWA launches without the browser UI being visible. This can make the user experience as immersive as a native app's experience because users do not get distracted by the browser UI. It is even possible to display a splash screen, as it is often used in native apps, with a background color and the PWA's icon, mimicking the behavior and appearance of a native app perfectly. The launch behavior can be defined in the Web App Manifest.

Engagement

Native apps use methods for increasing engagement that users do not expect to be used by websites. They can continue running in the background, sync data with a server even if the user closed the app or display push notifications. These seem more like web-incompatible features, but a PWA has something that can make them possible. It is, again, the Service Worker.

Background Sync

If there is no connection and users are able to use a PWA, they might expect to be able to use it as usual. They want to fill out a form or send a message, and after sending the entered data, they close the app [Ater, 2017, p. 135].

This would typically mean the message is lost and the user might not realize it until talking to the targeted recipient in real life. Developers could store the data in a local database. Then they can send it when the user opens the app again, and a connection is established. What happens if the user does not open the app again?

With Background Sync, the Service Worker will receive an event to try to sync and sends the locally stored data from the background once there is a connection, even if the app was closed. If the sync fails, it will be tried again after some time. [Archibald, 2018]

Push Notifications

Many native applications would be much less useful without push notifications. Developers also can at least remind users of the app's existence if they forgot about it and potentially cram out a little more revenue [Ater, 2017, p. 183].

Notifications displayed on a website are already widespread in the web today. The Service Worker now allows for receiving push messages and also displaying them as a notification, of course all from the background.

This sounds as if there is a lot of potential for the developer to exploit the push messaging system by spamming users. Could a hacker send an infinite amount of messages to the clients? Of course, the Push API's design hinders developers or hackers to act in such a way.

A push service chosen by the browser itself [Gaunt, 2018] manages every connection to PWA clients, and much work is lifted off of developers' shoulders. Messages are verified to come from the authorized, correct sender and push messaging spam is prevented by such a service. If the device is currently offline, the push service waits and delivers as soon as the device comes back online. [Ater, 2017, p. 185]

The sent data must be encrypted to prevent the push service from reading along, and the service has to make sure that only authorized requests come through [Ater, 2017, p. 198]. This is done by

using Voluntary Application Server Identification (VAPID) private and public keys that have to be created once for the application [Ater, 2017, p. 199].

The technologies mentioned here, mainly Background Sync and push notifications, are crucial to the PWA's ability to replace the native app. They were, amongst other things, what allowed the native app to compete with the web in the first place. Now the web is catching up, and even though yet unexpected by users, the web and thus PWAs do not have to miss out on engagement opportunities in comparison to native apps.

As an example, BaBe, a news app from Indonesia, had trouble distributing their native app through app stores. They wanted to reach more people, and that is why they developed a PWA. They found the user engagement stayed the same and the PWA even performed faster than their native app. [Google, 2016b]

BaBe's CEO even has switched from using their native app to the PWA:

> "I've started to consume BaBe more through the mobile web than through the native app. It's just faster and better. Why would I bother downloading an app anymore?" [Google, 2016b] (Weihan Liew, BaBe CEO)

3.2.4 Device API Access

What a native app has relatively easy access to are the hardware components of a smartphone. Due to the integration into the operating system and given the user granted permission, native apps can capture media, read any sensors or even modify files saved on the device. Luckily, websites have been able to catch up on those device-accessing features with ever-improving browsers. Allowing websites to access device APIs without giving permission would pose serious security flaws. That is why, of course, there is a permission system for the web, too.

Before accessing any sensitive feature, the PWA must ask for permission and can use it after being accepted by the user.

The website "What Web Can Do Today" offers a broad overview of the web's capabilities to interact with the device. A PWA can get access, for example, to the geolocation, files, camera, microphone and the common sensors like the gyroscope, proximity or ambient light sensor through the browser web APIs. [Bar, 2018]

3.2.5 Monetization

The current approach to handle payments for the web is the Payment Request API, that lets users enter their payment information once and enter it automatically everywhere the API is used. Thanks to this API, PWAs can receive the same amount of trust from a user like a native app because the payment UI will look the same for every website the user visits. [Ater, 2017, p. 240]

It speeds up the payment not only for users but also for developers. Users obviously do not have to enter their information every time they pay on a website and developers can, after specifying metadata, show the payment UI by writing literally one line of code [Ater, 2017, p. 241].

This sounds as if accepting payments on the web is a breeze for developers, but the Payment Request API is exclusively responsible for data collection [Kitamura and Gash, 2018].

Processing the actual payment transaction is up to the developer, but can be tremendously simplified by using a third-party Payment Service Provider. These providers handle all the security measures and of course the transactions. They can enable developers to implement one-time purchases or subscriptions so that developers and users do not have to miss out on anything if developers take the extra step of communicating with a Payment Service Provider.

Although web developers are now dependent on a third-party, it is worth it for small companies that do not have the means to comply with expensive and complicated security standards [Kitamura and Gash, 2018].

It also has to be remembered that developers using the app store payment methods are dependent on the store as well.

A prime example of successful monetization is AliExpress. They were able to increase their conversion rate by 104% in comparison to their previous website after developing a PWA. [Google, 2016a]

3.3 What PWAs offer that native apps do not

If PWAs are able to accomplish all a native app does, why should we even bother replacing native apps with a new standard?

3.3.1 Easier/Faster Development

Comparing the difficulty of native app development to PWA development is complicated because it mostly depends on the developer's personal preferences and experience. There are facts about both that can give developers a good idea of whether a native app or a PWA is more comfortable to implement. If developers already have experience with web development (most essentially HTML, CSS, and JavaScript), it should be reasonably easy to build a PWA because there is not much more required. In the case of Android, building a native app requires not only to know Java but also the Android framework, which includes, e.g., all the classes, different layouts, and components of an app. However, in that case, the developer only has an app for one platform.

Even supposing that the developer knows both—native app and web development—developing a PWA should be more efficient because a PWA still is a website. This means, of course, that it runs on every platform that has a browser, whereas a native app would have to be developed for

every platform once, resulting in much more work and higher costs of development. A PWA can be developed once and will run on virtually every platform, at least as a basic website if PWA features are unavailable. On the other hand, a web developer needs to know how code behaves in different browsers and often even how to support versions of browsers that possibly are many years old.

The development work-flow is entirely different, too. For native app development, it is required to install the necessary SDKs and favorably set up an integrated development environment (IDE) with it. There is an extra build step, in which the programming languages used for native app development are compiled before being able to execute, and the app package is rebuilt. This build step is necessary whenever developers want to test the changes. Afterward, the new package has to be loaded onto a device or emulator. Especially using an emulator will again bring delay in running the app. After all, there is a whole operating system emulated on the test machine. These steps can luckily be simplified by an IDE made for native app development, but there still is a noticeable delay that grows with the size of the app between making code changes and testing them. Over time, this can feel like a burden to developing a native app.

Unfortunately, Apple forces developers to develop apps for iOS on an Apple machine, impeding thus the development of an app for multiple platforms.

A PWA is developed in a different way: A website consisting of HTML, CSS, and JavaScript does not need to be compiled because these languages are interpreted by the browser on run-time. Modifications of the code can be seen in action right away, and all we really need is a text editor and a browser. Those are pre-installed programs on every commonly used operating system. In addition, developers are not forced to take the extra cognitive load of learning to use and dealing with an IDE. Those circumstances lower the entry barriers for web and thus PWA development. Nevertheless, it has to be kept in mind that PWAs are not able to run on outdated browsers and developers should look for alternative implementations for these versions.

3.3.2 Distribution

The first place where users go to download a native app is the primary app store from their platform.

They may enter a search phrase, are shown a result list and then can further inform themselves about the single apps on their respective sites. After deciding which app to use, they must click on the install button, possibly accept the intimidating permissions required by the app to function, wait for the app to finish downloading and wait to finish the installation before they can use the app for the first time. [Dascalescu, 2016]

That is actually a big problem for user acquisition because users have plenty of time to change

their mind and to not install an app. According to Gabor Cselle, every step in the installation process causes ca. 20% of users to turn away [Cselle, 2018].

A PWA has a more direct way of getting from discovery to first start. The ideal case would be a user to hear about a PWA, for example through an advertisement or a recommendation by a friend, and to enter the URL in the browser. The PWA can now be used with all the features, just by entering the URL or clicking on a link. If users are convinced of a PWA, they can easily "install" it (add it to the home screen). Even if there is no direct recommendation, there is only one extra step to the use of the PWA. Users can employ their favorite search engine to search for websites fitting their needs and have to make one click on a link to open the PWA.

Another advantage to PWAs being almost instantly usable is that even the content of PWAs can be crawled through search engines. For example, if a PWA contains multiple topics (e.g., "science" or "technology") of articles to read, the pages of those topics can be discovered and accessed instantly through search engines by users. [Dascalescu, 2016]

Users have the chance of not having to start from the main page of the PWA. They can get straight to the content they need and still choose to "install" the PWA if they want to.

The content of native apps is not crawled by search engines or the app stores [Dascalescu, 2016]. Native apps would have to include the content (for example, topics like "science") in the description of the app to be somewhat discoverable. Even if users find the app by this keyword, they still need to go through all the installation steps and need to click around in the app to get to the relevant content, making PWAs appear like a profoundly revolutionary technology, even though they essentially are just websites.

This signifies that PWAs potentially can acquire more users because users have little time to change their mind or be distracted and they have faster access to the content they need.

3.3.3 Shareable

One criterion for PWAs established by Google is that every page of the PWA must have its own unique URL [Google, 2018f]. This implies every page being accessible by shared links [Dascalescu, 2016], allowing PWAs to make use of another powerful default feature of the web.

While it is possible to deep link into native apps in the meantime, developers are burdened with extra work to make it function, and the user is compelled to have the targeted native app installed. Note that there is already an implemented attempt to deep link into not installed apps for Android with Google Play Instant [Google, 2018c]. This also requires additional work by the developer and the overall market share of Instant Apps is too small to make a difference (0.01% of all apps for Android are Instant Apps [Vogelzang and Maurer, 2018]).

Supposing the PWA launches in standalone or fullscreen mode, the URL bar of the browser

would be missing. So how can a PWA still be shareable?

Any URL on the web can be shared with the help of the Web Share API. It can invoke the natively provided share process, in which the user can decide where to share the URL. [Ater, 2017, p. 245]

3.3.4 Up-To-Date

To deploy changes of a website usually is no big deal and can be plainly done by, e.g., pushing a git commit to an individual remote branch, where continuous deployment could handle the publication, after possibly being tested by continuous integration. In the most basic case, the developer uploads files to a server through the File Transfer Protocol (FTP). After deployment, the changes are visible to the user immediately, provided the browser does not keep loading from HTTP cache. With a Service Worker and the Cache API typically used in a PWA, developers luckily do not depend on the HTTP cache and can follow their own caching strategy. Updating even a cached PWA is less error-prone due to more control through the Cache API.

Updating a native app through the app store is entirely different. After testing the changes and building the app package again, developers need to upload it to the app store. The stores typically need a lot of hours to process and to deliver the update to the users. If this process is not completed automatically, users have to wait for the update to finish after clicking on a notification or checking the app store manually. Regarding Apple, there even is a review process by staff members after every update developers want to publish, which can take several days to be approved. This causes a relatively long delay between deploying changes and users receiving them, so developers cannot make any quick changes or fixes. They have to consider carefully what they want to include in an update for their native app. Having a critical bug in an app can be severely dangerous to a business if users have to wait several days for an update.

PWA developers are more in charge of what they can publish, and users will then receive the changes faster because the app store does not interfere [Dascalescu, 2016], allowing developers to make quick changes and receive faster feedback.

3.3.5 Less Data Usage

Any app occupies a certain share of local storage depending on how much code and assets there are. All this needs to be saved locally on the device. The package, of course, has to be downloaded before installation. Bigger app sizes mean less storage available for the user and obviously longer download times, which can even lead to abandoning the download. Developers therefore seek to optimize every asset and remove unnecessary code to keep the app size as small as possible.

A basic Java app for Android showing a label reading “Hello World!” with nothing more

included than the necessary Android framework and an optional library to maintain backward compatibility reaches a fairly small size of 539 KB. [Majmudar, 2018]

This size is totally acceptable because it can be downloaded fast and will not waste lots of space after installation. However, the usual app will want to do more than just displaying a simple text, so the size of 539 KB is more to be seen as a minimum.

The average package size for iOS apps is 38 MB and 15 MB for Android (February 2017) [Boshell, 2017].

That is significantly more than the previously mentioned minimum size. In comparison, the average 3 MB [Everts, 2017] size of a web page (February 2017) appears to be small. Keep in mind, however, that the average size does not tell us much about the exact common size of a web page without the standard deviation. All web pages could precisely have the size of 3 MB, but there could also be a considerable amount of smaller and a few bigger pages, we do not know.

Like a native app, a PWA does not have to download all its content on subsequent visits. By following the right caching strategy, it is "installed" like a native app. For example, the app shell can be cached and will not be required to be downloaded with the stylesheets and logic on further visits to the same PWA.

It is definitely possible for developers to lower the size of their PWA in comparison to their native app, as, e.g., Twitter did. They obviously earn more money by reaching more users.

They state on Google's PWA case study website that they optimize PWA size, performance and download speed to reach users even in emerging markets with limited data and hardware capabilities. The size of their PWA is 600KB, contrasting their 23.5MB native app for Android. On the first page load, they only load resources visible above-the-fold and the Service Worker caches all resources, such as the shell. They optimize images and partly make use of the PRPL pattern as well, to offer a fast user experience and lightweight app to their users. [Google, 2017b]

It has already been mentioned that a small PWA size is especially needed in developing and low-income countries, where mobile data is costly [Dascalescu, 2016], and storage space may be rather limited.

For example, the average person in Zimbabwe would have to pay not less than 33% of income for one GB of mobile data in 2017. A less extreme example is Kenya, where users have to pay 4% of their income. [World Wide Web Foundation, 2017]

This still is a significant amount.

Downloading a native app in such countries could become costly. A PWA could save lots of data if the overall size is smaller and especially if the user is mainly interested in one page. The users then also do not have to worry too much about their local storage, as also pointed out by Twitter's PWA engineering lead:

> "Twitter Lite is now the fastest, least expensive, and most reliable way to use Twitter. The web app rivals the performance of our native apps but requires less than 3% of the device storage space compared to Twitter for Android." [Google, 2017b]
>
> (Nicolas Gallagher)

3.4 Shortcomings of a PWA

3.4.1 Support

PWAs highly depend on the operating system and the browser, as they run in it. The operating system has to implement PWAs to enable browsers to add a PWA to the home screen. All the features that are important for keeping up with native apps, such as notifications or accessing device APIs, are provided by the browser. This means that the PWA's overall success is strongly tied to the advancement of browsers and the widespread distribution of the newest versions of a browser and operating system. A browser last updated in 2012 will not be able to provide all the modern features.

Browser vendors are not obligated to implement a feature others are promoting, and they furthermore cannot be forced to implement something in a timely manner. The final answer if PWAs will render native apps obsolete has to take into account how many users will get access to the newest browser version and if the browser vendors will adopt modern features.

For example, Apple's iOS did not even support PWAs until they released iOS version 11.3 in March 2018. The implementation does not show a polished appearance, however. To mention some deficits: After a few weeks of no usage, iOS will delete the PWA's cache, so there is no guarantee that the app is further available offline. Background Sync and Web Push APIs are not even supported, and there is no install banner. In addition, if the PWA has no own means of navigating between screens, the user will not be able to do exactly that. [Firtman, 2018]

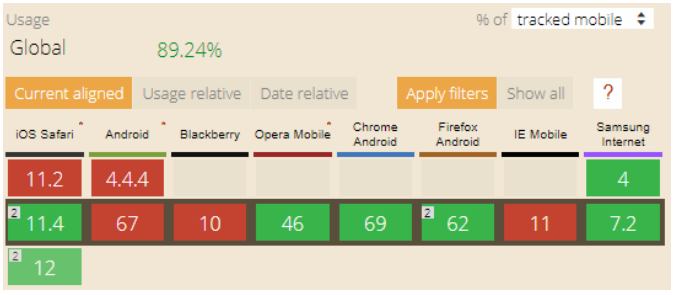

Figure 1: Manifest support on mobile browsers, according to caniuse.com, CC BY 4.0 [Deveria, 2018b]

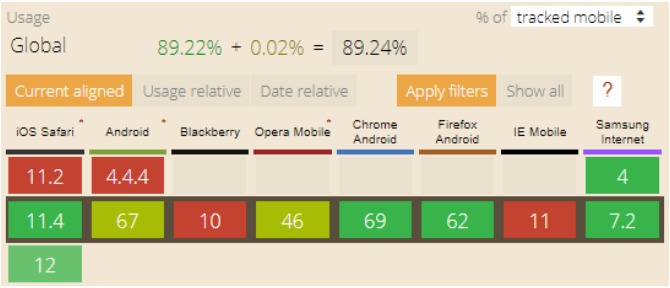

Figure 2: Service Worker support on mobile browsers, according to caniuse.com, CC BY 4.0 [Deveria, 2018a]

The current support for the main technologies a PWA consists of is already given for the majority of mobile users with 90% of users having support for the manifest (figure 1) and Service Worker (figure 2). These APIs are the most crucial ones for a PWA to work.

The problem is that every browser vendor can implement the PWA support in the way they want and whenever they want. It is unrealistic to assume that all users will receive the newest features.

What is more, different operating systems may have different features and different ways of working. If it gets to the point where there are many differences between PWA implementations on all the operating systems, developers would have to bear all those differences in mind and possibly build multiple solutions for the platforms. This would run counter to the aim of the web and would make PWAs less cross-platform again.

There should be a PWA specification by the W3C to give browser vendors a guideline to uniformly implement PWAs. That would encourage the same functionalities and behaviors across all platforms. Even then vendors are not forced to implement anything.

Keep in mind that PWAs combine those features that operating systems have in common.

3.4.2 App Store

PWAs do not depend on an app store. For developers, this means they have to implement their own solution for monetization, which can become more difficult without an app store because an own backend is required. Much workload can be reduced by using a payment service provider like Stripe. Developers could keep more money, as payment service providers charge a smaller fee than the current app stores. Stripe, for example, charges 2.9% of the transaction plus 30 cents [Stripe, 2018]. The Apple App Store [Apple, 2018b] and the Google Play Store [Google, 2018h] both take 30% of sales revenue.

App store promotions for single apps would also disappear. They hugely benefit the promoted

app but are done for already successful apps most of the time and large sums of money may play a big part in being promoted. Smaller developers would not have a disadvantage if app store promotions were missing, which would be fairer for everyone.

Also, statistics, e.g., regarding downloads or errors for native apps are managed by app stores. To achieve the same level of tracking, developers need to implement solutions like Google Analytics, which are subject to more complicated privacy policies and unwanted cookie messages. The past has shown that this should not get in the way of a business being successful.

Are there disadvantages for users if there are no traditional app stores? They would have no big and central place to search for PWAs. However, they can use a search engine, and there are already PWA "stores" in which users can search by keywords and sort by category. Microsoft even includes PWAs in the Microsoft Store. These new platforms could also implement a rating system so that users can evaluate apps as in the traditional stores.

The most significant disadvantage for users is that a PWA cannot be scanned for malware before being published as thoroughly as in an app store.

3.4.3 Security

It sounds dangerous that the Service Worker can manipulate network requests, but it can exclusively do so for the same origin. Service Workers have a scope that is valid for the directory in which its JavaScript file is located. This means it can only manipulate requests coming from the same directory or subdirectories, making it impossible to alter foreign websites. [Ater, 2017, p. 27/28]

A PWA has to be delivered through HTTPS and does not function through HTTP. Otherwise, it would be relatively easy for a hacker to install a malicious Service Worker [Ater, 2017, p. 21].

As PWAs are websites, they offer the same attack surface for operating system exploits. This means the risk of PWAs cannot exceed that of websites. The permission system prevents quiet usage of any feature the user would not want to allow everyone to have access to, and, for example, push notifications are made sure to be as secure as necessary by the browser vendors.

Another security measure are browsers that incorporate various methods of protecting users from malicious websites. The simplest one is a list of websites the browser should block from being accessed [Biersdorfer, 2016].

Google is doing major work in identifying unwanted software in websites and allows everyone to use their technology called "Safe Browsing". They automatically scan and analyze websites. [Mavrommatis, 2015]

That cannot possibly happen instantly for every new website and what if the attackers use not-yet found security breaches? This might not be an ideal solution, but the same problem applies to

native apps.

3.4.4 Deeply Integrated Features

PWAs are not yet able to replace all functionalities of a native app.

Examples of missing features are setting alarms or using traditional means of communication like calls or SMS [Dascalescu, 2016].

Developers have no access to features that reach deeply into the operating system like wake-locks, accessing calendars or changing system settings such as the wallpaper. Widgets are missing as well.

4 Future

4.1 Adoption of lacking features

Google has its "Capabilities" program in order to drive forward the development of the web APIs [Google, 2018b]. Some of the mentioned missing features are already planned to be implemented in the future (for example, a WakeLock API and widgets) [Google, 2018e].

More sensitive features may require a layer of security other than a permission system. "Google Play Protect" scans native apps before being downloaded and does so even with apps not coming from the Play Store [Google, 2018g]. Google should be able to transfer that mechanism to the web. If browsers implemented such a service, it would be possible to reach the same level of security as with native apps. The PWA would have to be scanned every time it is updated.

4.2 The disappearance of app stores

App stores not only provide security, but they are a big source of revenue for the companies running them as well. Apple, for example, has seen their consumers spend approximately 22.6 billion dollars (figure 3) in alone the first half of 2018. That would mean a revenue of 6.78 billion dollars for Apple at a 30% revenue share. What is more, they demand a fee for developers to be able to publish apps.

App stores easily maintain their monopoly because of the network effect [Keese, 2014, p. 155] or, in the case of Apple, because the operating system does not allow installing apps from different sources [Williams, 2018].

PWAs have the potential to be the "disruptive innovation" [Keese, 2014, p. 155] that could bring an end to the current app stores, as they are not distributed through an app store. There are already stores for PWAs, but they cannot take a share of anything and cannot be directly profitable.

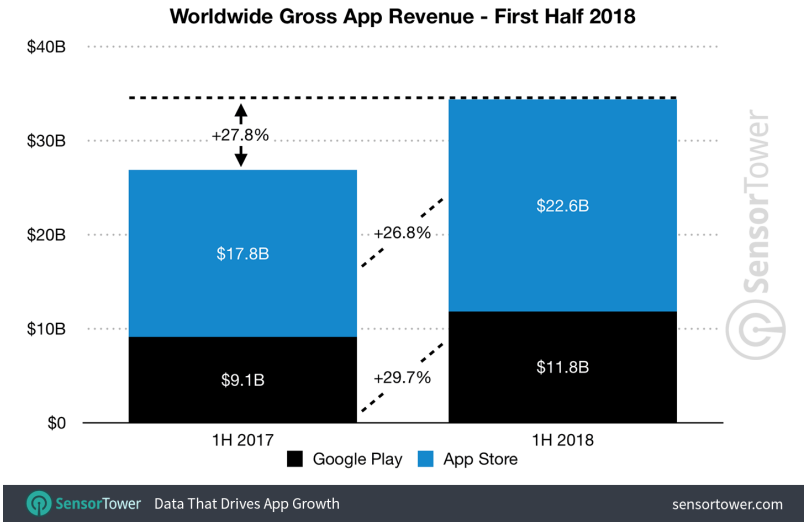

Figure 3: app store revenue according to SensorTower [Nelson, 2018]

The current app stores for native apps will lose potential revenue or may even vanish if PWAs turn out to dominate the market in the future. Apple or Google would probably not want that to happen. Maybe that is the reason for Apple not to put too much effort into the implementation of PWAs. Google, though, is the driving force behind this new technology. How could that be?

Google's roots lie in the search engine, where it is holding a quasi-monopoly with a market share of 92% (October 2018 [StatCounter, 2018]).

They have a collection of all the websites and have acquired what may be the most extensive data collection about people. They know best what people want and where to find it. For Google, it would be relatively easy to develop a second search designated for PWAs or at least highlight them in search results. They can build the best PWA "store" and establish another monopoly, while the traditional app stores and thus big revenue sources for others are gone. Google's business heavily relies on advertisements, and they can possibly compensate for the loss of the Play Store revenue with additional advertisements in the PWA search, better than anyone else.

Google's goal could be to get users back to the web and their search engine to maximize their profit and possibly gain even more power.

4.3 Scenarios

4.3.1 PWAs do not replace native apps and become deprecated

This is the scenario least likely to occur in the future. Some operating system developers could decide to completely abandon PWA implementation by suspecting that their app store will lose revenue or for other reasons. That could lead to a disconcern amongst developers, but with a PWA they still would have a website that works as expected and can be similar to a native app on different platforms. Google and others will further integrate PWAs into their software, so companies not embracing PWAs will quickly look outdated. Website owners and developers have too much of an advantage of PWAs not to use them.

4.3.2 PWAs replace native apps

It will take time before every platform would replace native apps entirely. Once there is an abundance of PWAs, new operating systems can have a chance to emerge by relying on PWAs. At the moment new operating systems find it hard to compete because they do not have an app store that can provide lots of apps. That is why most smartphone manufacturers include a version of Android on their phones. However, operating systems relying on PWAs do not have to establish another app store. They can make use of already existing PWAs.

One operating system is already proving to be successful especially in emerging markets: KaiOS, a descendant of Firefox OS [KaiOS, 2018a].

KaiOS does not exactly use PWAs as their manifest is different [KaiOS, 2018b], but they rely on websites that are similar to PWAs. Developers can make small adjustments to their existing websites (at least defining the manifest), and they have an app running on the basis of this new operating system that "feels" like a native app. This app still has to be submitted to KaiOS' own app store. [KaiOS, 2018c]

Nevertheless, the entry barrier for developers is much lower than that for a new operating system where they would have to develop a new native app.

Google is, by the way, a major investor in KaiOS [Codeville, 2018].

It is likely that more and more apps are developed only as a PWA because the developers are content with the reach of the web and certain operating systems. What may happen is that users are tempted to switch to operating systems fully supporting PWAs and abandoning the currently established ones, including their app stores. If new PWA operating systems are used, and their phones are more attractive than current ones, more users will consider to change their smartphone. This process will also be fueled by new apps only developed as a PWA that will be missed out on by users of operating systems without full PWA support. Google is preparing Android for this

movement away from app stores, and they will likely offer extensive PWA support in their new operating system Fuchsia.

However, the web is made to be accessible on every platform. To replace native apps for all platforms, the web would need to adapt to all the different functionalities and mechanisms unique to every operating system. Different implementations then would be required for a PWA to run correctly on this system. That should not be striven for.

4.3.3 PWAs and native apps coexist

Especially for the near future, it is unlikely that native apps vanish because the operating system and browser support are not controllable and there will always be users using an outdated version of their software. App store owners would lose much of their revenue without native apps, and they might try to hinder PWA adoption. Therefore native apps can definitely not be replaced overnight. It is likely, however, that PWAs will reach a market share that is substantially bigger than that of native apps. PWAs surely have the potential to become the dominantly used type of application because of the cross-platform approach that saves lots of effort for developers.

5 Conclusion

PWAs allow for more efficient and cheaper development, can reach most users of the web, and can acquire users easier. The engagement can be as high as with a native app, and the convenient sharing mechanism of the web also leads to faster growth of the user base. PWAs can even save lots of user's money by operating with less data than a native app, especially in developing countries.

After a PWA is added to the home screen, there are not many features a PWA cannot make use of in comparison to a native app. Developers may not even miss out on monetization. The most important features that are not yet implemented on the web are planned by Google and their "capabilities" program.

Performance-wise it can be said that making use of performance optimizations and caching strategies, PWAs do not have to be noticeably slower than native apps. For demanding mobile games and computationally intensive apps, it still is better to employ a native app. That will change in the future with promising technologies like WebAssembly and our hardware becoming more capable every year. Although, there will be many developers not paying attention to performance. There will be a wide variety of differently performing PWAs, but the potential definitely is there.

As more attention will turn away from app stores, a relevant question is how could a PWA store/search be designed? Can it be profitable?

There is enough evidence that, even today, PWAs are a viable alternative to native apps because

many well known profit-oriented companies have developed PWAs. After replacing their native app, they have observed significant improvements in user-acquisition and sales, even in developing countries.

In regards to security, Google will probably provide a service that is on par with native app security measures. A question for future work is how to design a system that is secure but still lets websites access more sensitive operating system features without enabling misuse.

Sadly, a big obstacle for PWAs is that operating system developers do not want to completely implement PWAs in order not to lose their app store revenue. That is why PWAs unquestionably cannot replace native apps right now, but over time, every company could lose its significance. A giant like Google pushing PWAs can play into that.

Even if the app store providers did not play a role, there would be the problem of features that are unique to an operating system. The browsers cannot provide the necessary APIs only for one platform if special implementations tailored to operating system mechanisms are wanted for an app.

That is why it can be assumed that PWAs will not be able to replace native apps. Instead, there will be a myriad more PWAs than native apps. If developers only need features common with every operating system for their app, they will develop a PWA. Otherwise, it will be a native app.

The broad adoption of PWAs would be a positive change, allowing developers to bring their app on every platform running a browser. The app world would be fairer for developers and more logical for users. Just like there is an app for anything today, there will be a PWA for anything in the future, and a small share of native apps.

The work for this paper was worth it for the author and has allowed him to learn about all the features of PWAs and their implementation. He is now able to implement his own PWA, and he can decide whether to use a native app or a PWA.

Portions of this paper are modifications based on work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License. Attribution is given by using references.

Portions of this paper are modifications based on work created and shared by the Android Open Source Project and used according to terms described in the Creative Commons 2.5 Attribution License. Attribution is given by using references.

References

[Apple, 2018a] Apple (2018a). About Swift. `https://swift.org/about/#platform-support`. Accessed: 2018-08-21. Usage: Determine iOS app development language.

[Apple, 2018b] Apple (2018b). Program Membership Details. `https://developer.apple.com/programs/whats-included/`. Accessed: 2018-10-27. Usage: Determine revenue share.

[Archibald, 2018] Archibald, J. (July 2018). Introducing Background Sync. `https://developers.google.com/web/updates/2015/12/background-sync`. Accessed: 2018-09-12. Licensed under CC BY 3.0 (https://creativecommons.org/licenses/by/3.0/), changes were made. Usage: Explain Background Sync.

[Archibald, 2012] Archibald, J. (May 2012). Application Cache is a Douchebag. `https://alistapart.com/article/application-cache-is-a-douchebag`. Accessed: 2018-10-11. Usage: Explain why old application cache API is deprecated.

[Ater, 2017] Ater, T. (2017). *Building Progressive Web Apps. Bringing the power of native to the browser*. O'Reilly, Sebastopol, first edition.

[Bar, 2018] Bar, A. (2018). What Web Can Do Today. `https://whatwebcando.today/`. Accessed: 2018-08-28. Licensed under CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0/), changes were made. Usage: Determine web capabilities.

[Biersdorfer, 2016] Biersdorfer, J. D. (December 2016). How Does My Browser Recognize Malware? `https://www.nytimes.com/2016/12/22/technology/personaltech/how-does-my-browser-recognize-malware.html`. Accessed: 2018-10-29. Usage: Determine how secure the web is.

[Boshell, 2017] Boshell, B. (February 2017). Average App File Size: Data for Android and iOS Mobile Apps. `https://sweetpricing.com/blog/2017/02/average-app-file-size/`. Accessed: 2018-09-30. Usage: Compare average app to website size.

[Bryant, 2017] Bryant, D. (March 2017). Why WebAssembly is a game changer for the web and a source of pride for Mozilla and Firefox. https://medium.com/mozilla-tech/why-webassembly-is-a-game-changer-for-the-web-and-a-source-of-pride-for-mozilla-and-firefox-dda80e4c43cb. Accessed: 2018-11-02. Usage: Determine why WebAssembly is fast.

[Codeville, 2018] Codeville, S. (June 2018). Google Leads Series A Investment Round in KaiOS to Connect Next Billion Users. https://www.kaiostech.com/google-leads-seriesa-investment-round-kaios-connect-next-billion-users/. Accessed: 2018-10-27. Usage: Mention Google's investment.

[Cselle, 2018] Cselle, G. (July 2018). Every Step Costs You 20% of Users. https://medium.com/gabor/every-step-costs-you-20-of-users-b613a804c329. Accessed: 2018-11-04. Usage: Why native app installation steps are problematic.

[Dascalescu, 2016] Dascalescu, D. (August 2016). Why "Progressive Web Apps vs. native" is the wrong question to ask. https://medium.com/dev-channel/why-progressive-web-apps-vs-native-is-the-wrong-question-to-ask-fb8555addcbb. Accessed: 2018-10-28. Usage: PWA advantages and missing features.

[Deveria, 2018a] Deveria, A. (2018a). Service Workers. https://caniuse.com/#feat=serviceworkers. Accessed: 2018-10-25. Licensed under CC BY 4.0 (https://creativecommons.org/licenses/by/4.0/), no changes. Usage: Determine Service Worker support on the mobile web.

[Deveria, 2018b] Deveria, A. (2018b). Web App Manifest. https://caniuse.com/#feat=web-app-manifest. Accessed: 2018-10-25. Licensed under CC BY 4.0 (https://creativecommons.org/licenses/by/4.0/), no changes. Usage: Determine manifest support on the mobile web.

[Dlugos, 2018] Dlugos, C. (January 2018). Page-Speed: Google macht Ladezeit zum Rankingfaktor in der mobilen Suche. https://t3n.de/news/page-speed-google-macht-ladezeit-914751/. Accessed: 2018-10-29. Usage: State the importance of fast loading.

[Dysinski, 2017] Dysinski, T. (February 2017). Bear 71 and WebVR. https://developers.google.com/web/showcase/2017/bear71. Accessed: 2018-11-22. Licensed under CC BY 3.0 (https://creativecommons.org/licenses/by/3.0/), changes were made. Usage: Example for web performance limits.

[Eaton, 2012] Eaton, K. (March 2012). How One Second Could Cost Amazon $1.6 Billion In Sales. https://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales. Accessed: 2018-10-29. Usage: State the importance of fast loading.

[Everts, 2017] Everts, T. (August 2017). The average web page is 3MB. How much should we care? https://speedcurve.com/blog/web-performance-page-bloat/. Accessed: 2018-09-30. Usage: Compare average app to website size.

[Firtman, 2018] Firtman, M. (March 2018). Progressive Web Apps on iOS are here. https://medium.com/@firt/progressive-web-apps-on-ios-are-here-d00430dee3a7. Accessed: 2018-10-25. Usage: Show deficits of Apple's PWA implementation.

[Gaunt, 2018] Gaunt, M. (September 2018). How Push Works. https://developers.google.com/web/fundamentals/push-notifications/how-push-works. Accessed: 2018-09-15. Licensed under CC BY 3.0 (https://creativecommons.org/licenses/by/3.0/), changes were made. Usage: Determine who chooses a push service.

[Gaunt and Kinlan, 2018] Gaunt, M. and Kinlan, P. (October 2018). The Web App Manifest. https://developers.google.com/web/fundamentals/web-app-manifest/. Accessed: 2018-11-21. Licensed under CC BY 3.0 (https://creativecommons.org/licenses/by/3.0/), changes were made. Usage: State what the manifest is.

[Google, 2018c] Google (2018c). Google Play Instant. https://developer.android.com/topic/google-play-instant/. Accessed: 2018-09-27. Licensed under CC BY 2.5 (http://creativecommons.org/licenses/by/2.5/), changes were made. Usage: Mention Instant Apps.

[Google, 2018e] Google (2018e). Issues - chromium - An open-source project to help move the web forward. https://bugs.chromium.org/p/chromium/issues/list?cursor=chromium%3A257511&q=proj-fugu&sort=m. Accessed: 2018-11-15. Usage: Outlook on the adoption of lacking features.

[Google, 2018g] Google (2018g). Schutz vor schädlichen Apps: Google Play Protect. https://support.google.com/googleplay/answer/2812853. Accessed: 2018-11-09. Usage: State how Google protects from malicious native apps.

[Google, 2018h] Google (2018h). Transaktionsgebühren. https://support.google.com/googleplay/android-developer/answer/112622. Accessed: 2018-10-27. Usage: Determine revenue share.

[Google, 2018f] Google (July 2018f). Progressive Web App Checklist. https://developers.google.com/web/progressive-web-apps/checklist. Accessed: 2018-07-20. Licensed under CC BY 3.0 (https://creativecommons.org/licenses/by/3.0/), changes were made. Usage: Determine PWA requirements.

[Google, 2016a] Google (May 2016a). AliExpress. https://developers.google.com/web/showcase/2016/aliexpress. Accessed: 2018-11-22. Licensed under CC BY 3.0 (https://creativecommons.org/licenses/by/3.0/), changes were made. Usage: Example for monetization.

[Google, 2016b] Google (May 2016b). BaBe. https://developers.google.com/web/showcase/2016/babe. Accessed: 2018-11-22. Licensed under CC BY 3.0 (https://creativecommons.org/licenses/by/3.0/), changes were made. Usage: Example for good PWA engagement.

[Google, 2017a] Google (May 2017a). Jumia sees 33% increase in conversion rate, 12X more users on PWA. https://developers.google.com/web/showcase/2017/jumia. Accessed: 2018-11-02. Licensed under CC BY 3.0 (https://creativecommons.org/licenses/by/3.0/), changes were made. Usage: Example for good performance.

[Google, 2017b] Google (May 2017b). Twitter Lite PWA Significantly Increases Engagement and Reduces Data Usage. https://developers.google.com/web/showcase/2017/twitter. Accessed: 2018-10-04. Licensed under CC BY 3.0 (https://creativecommons.org/licenses/by/3.0/), changes were made. Usage: Example for less data usage.

[Google, 2018b] Google (November 2018b). Capabilities. https://developers.google.com/web/updates/capabilities. Accessed: 2018-11-15. Licensed under CC BY 3.0 (https://creativecommons.org/licenses/by/3.0/), changes were made. Usage: Outlook on the adoption of lacking features.

[Google, 2018a] Google (October 2018a). Application Fundamentals. https://developer.android.com/guide/components/fundamentals. Accessed: 2018-11-27. Licensed under CC BY 2.5 (http://creativecommons.org/licenses/by/2.5/), changes were made. Usage: Determine Android app development language.

[Google, 2018d] Google (September 2018d). Introduction to Service Worker. https://developers.google.com/web/ilt/pwa/introduction-to-service-worker. Accessed: 2018-09-04. Licensed under CC BY 3.0 (https://creativecommons.org/licenses/by/3.0/), changes were made. Usage: Explain Service Worker.

[KaiOS, 2018a] KaiOS (2018a). KaiOS, the emerging OS. https://developer.kaiostech.com/. Accessed: 2018-10-27. Usage: Introduce KaiOS.

[KaiOS, 2018b] KaiOS (2018b). Manifest. https://developer.kaiostech.com/first-app/manifest. Accessed: 2018-11-18. Usage: Determine that KaiOS does not use PWAs.

[KaiOS, 2018c] KaiOS (2018c). Ready to develop for KaiOS? https://developer.kaiostech.com/others. Accessed: 2018-11-18. Usage: Determine that KaiOS' apps have to be submitted to the KaiOS store, and what is required to be approved.

[Keese, 2014] Keese, C. (2014). *Silicon Valley. Was aus dem mächtigsten Tal der Welt auf uns zukommt*. Albrecht Knaus, Munich, eighth edition.

[Kitamura and Gash, 2018] Kitamura, E. and Gash, D. (September 2018). How the payment ecosystem works. https://developers.google.com/web/fundamentals/payments/basics/how-payment-ecosystem-works. Accessed: 2018-09-20. Licensed under CC BY 3.0 (https://creativecommons.org/licenses/by/3.0/), changes were made. Usage: Show usefulness of Payment Request API.

[Lella and Lipsman, 2017] Lella, A. and Lipsman, A. (August 2017). 2017 US Mobile App Report. https://www.comscore.com/Insights/Presentations-and-Whitepapers/2017/The-2017-US-Mobile-App-Report. Accessed: 2018-10-24. Usage: Show declining app trend.

[LePage, 2018] LePage, P. (October 2018). Add to Home Screen. https://developers.google.com/web/fundamentals/app-install-banners/. Accessed: 2018-11-21. Licensed under CC BY 3.0 (https://creativecommons.org/licenses/by/3.0/), changes were made. Usage: When does the browser show the install banner?

[Majmudar, 2018] Majmudar, D. (March 2018). Comparing APK sizes. https://android.jlelse.eu/comparing-apk-sizes-a0eb37bb36f. Accessed: 2018-09-30. Usage: Determine minimum app size.

[Mavrommatis, 2015] Mavrommatis, P. (March 2015). Protecting people across the web with Google Safe Browsing. https://googleblog.blogspot.com/2015/03/protecting-

people-across-web-with.html. Accessed: 2018-10-29. Usage: Determine how secure the web is.

[Miller, 2018] Miller, P. (April 2018). Web apps are only getting better. https://www.theverge.com/circuitbreaker/2018/4/11/17207964/web-apps-quality-pwa-webassembly-houdini. Accessed: 2018-10-28. Usage: High performance possible with WebAssembly.

[Nelson, 2018] Nelson, R. (July 2018). Global App Revenue Reached $34 Billion in the First Half of 2018, Up 28% Year-Over-Year. https://sensortower.com/blog/app-revenue-and-downloads-1h-2018. Accessed: 2018-10-27. Usage: Emphasize relevance of app stores to their owners.

[Rosário, 2016] Rosário, J. (August 2016). Smooth as Butter: Achieving 60 FPS Animations with CSS3. https://medium.com/outsystems-experts/how-to-achieve-60-fps-animations-with-css3-db7b98610108. Accessed: 2018-10-30. Usage: Determine that smooth animation is possible on mobile web.

[Rouse, 2013] Rouse, M. (February 2013). What is native app? https://searchsoftwarequality.techtarget.com/definition/native-application-native-app. Accessed: 2018-08-21. Usage: Define native app.

[Sheppard, 2017] Sheppard, D. (2017). *Beginning Progressive Web App Development. Creating a Native App Experience on the Web*. Apress, New York. Usage: State what the PRPL pattern is.

[StatCounter, 2018] StatCounter (October 2018). Search Engine Market Share Worldwide. http://gs.statcounter.com/search-engine-market-share. Accessed: 2018-10-27. Licensed under CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0/), changes were made (but not to the provided data). Usage: Show Google's prevalence.

[Stripe, 2018] Stripe (2018). Stripe: Pricing and fees. https://stripe.com/us/pricing. Accessed: 2018-10-27. Usage: Determine Stripe's fee to compare it with app stores.

[Vogelzang and Maurer, 2018] Vogelzang, M. and Maurer, U. (November 2018). Android Instant Apps - Android SDK statistics. https://www.appbrain.com/stats/libraries/details/instant-apps/android-instant-apps. Accessed: 2018-11-21. Usage: Determine relevancy of Instant Apps.

[Walton, 2016] Walton, P. (July 2016). Measuring the Real-world Performance Impact of Service Workers. https://developers.google.com/web/showcase/

2016/service-worker-perf. Accessed: 2018-11-22. Licensed under CC BY 3.0 (https://creativecommons.org/licenses/by/3.0/), changes were made. Usage: State that Service Worker cache is faster than HTTP cache.

[Williams, 2018] Williams, P. (November 2018). Supreme Court to rule if customers can sue Apple claiming App Store is illegal monopoly. https://www.nbcnews.com/politics/supreme-court/supreme-court-rule-if-customers-can-sue-apple-claiming-app-n939001. Accessed: 2018-11-27. Usage: Determine that Apple only allows App Store as app source.

[World Wide Web Foundation, 2017] World Wide Web Foundation (2017). Mobile Broadband Data Costs. https://a4ai.org/mobile-broadband-pricing-data/. Accessed: 2018-11-21. Licensed under CC BY 4.0 (https://creativecommons.org/licenses/by/4.0/), changes were made (but not to the provided data). Usage: Point mobile data prices in developing countries out.

Illustration Directory

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die schriftliche Arbeit/schriftliche Ausarbeitung ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benutzt habe.

Berlin, den 30. November 2018

Declaration of Authorship

I hereby declare that I have written the work without the help of others and have used only the sources and aids listed in the bibliography.

Berlin, 30. November 2018