PROGRESSIVE WEB APPS: FACTORS FOR CONSIDERATION IN DEVELOPMENT

Faculty of Information Technology and Communication Sciences
B. Sc. Thesis
May 2021
Web technologies have become more capable and feature-packed, with new browser APIs constantly being introduced and developed. Consequently, progressive web apps and their development have risen in popularity. However, progressive web apps are still relatively new, while also containing many crucial factors to be considered when developing them. This is especially the case when the goal is to create the best possible native-like experience to the user.

This thesis is a literature study, in which academic research articles, real-world case studies as well as online documentations have been gathered to create a cohesive and up-to-date guide on progressive web apps. The study explains what progressive web apps are, what are the benefits and potential drawbacks of progressive web apps, as well as what other factors should be considered when developing progressive web apps. The goal of this study is to provide a comprehensive overview of progressive web apps as well as become an asset for developers by introducing important factors related to progressive web app development.

The definition of a progressive web app includes identifying and explaining its core components, including service workers, the HTTPS protocol, as well as web app manifests. The benefits of developing progressive web apps proposed in this study include application reach alongside capabilities, cross-platform software with a single codebase, and flexibility. The potential drawbacks mentioned in this study include web security flaws and poor web performance. The other crucial factors of progressive web app development include web performance and optimization techniques, energy efficiency, and user experience of progressive web apps, which are gathered from case-studies and academic research articles.

Keywords: progressive web app, PWA, web performance, service worker, web development, cross-platform development

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.
Contents

1 Introduction ......................................................................................................................... 1

2 Progressive Web Apps ........................................................................................................ 1

   2.1 Service worker ............................................................................................................ 3

   2.2 HTTPS ....................................................................................................................... 4

   2.3 Web app manifest ...................................................................................................... 4

3 Benefits and Drawbacks of Developing Progressive Web Apps ............................... 5

   3.1 Benefits of progressive web apps ............................................................................... 5
       3.1.1 Reach alongside capabilities ............................................................................ 5
       3.1.2 Single codebase for cross-platform software .................................................. 6
       3.1.3 Flexibility .......................................................................................................... 6

   3.2 Possible drawbacks when utilizing progressive web apps ......................................... 7
       3.2.1 Web security flaws .............................................................................................. 7
       3.2.2 Poor web performance ....................................................................................... 7

4 Performance of Progressive Web Apps ........................................................................ 8

   4.1 Web performance measuring tools ............................................................................ 8

   4.2 Common web performance metrics ........................................................................... 9

   4.3 Web performance optimization techniques ............................................................... 9
       4.3.1 Bundling ............................................................................................................. 10
       4.3.2 Code-splitting ................................................................................................. 10
       4.3.3 Caching and pre-caching ................................................................................. 11
       4.3.4 Preloading resources ....................................................................................... 11

5 Other Studies Regarding Progressive Web Apps ......................................................... 11

   5.1 Energy efficiency ....................................................................................................... 12

   5.2 User experience ........................................................................................................ 12

6 Conclusions ....................................................................................................................... 13

References.......................................................................................................................... 14
1 Introduction

Progressive web apps were first introduced in 2015 by designer Frances Berriman and Google Chrome engineer Alex Russel. As modern browsers were constantly introduced with new features and capabilities, progressive web apps were introduced as a way to create web applications that are able to achieve the performance, capabilities and user experience of native applications running on desktop and mobile environments. As web technologies have developed and matured over time, progressive web apps have also seen a steady increase in popularity, even among large companies such as Google and Microsoft (Gavin, 2020; Microsoft, 2021).

This thesis is a literature study, aiming to answer the following key questions:
- What is a progressive web app?
- Why should developers create progressive web apps?
- What factors should be considered when developing progressive web apps?

Most importantly, this study aims to provide a better understanding of progressive web apps and their applications. This study also introduces previously conducted studies and research regarding progressive web apps, as well as case-studies of real progressive web apps.

Each chapter in this study focuses on a specific aspect of progressive web apps. Chapter two introduces progressive web apps, as well as the fundamental components that make a web application progressive. Chapter three introduces the benefits and potential drawbacks for developers building progressive web apps. Chapter four introduces performance measurement tools, metrics as well as optimization techniques for progressive web apps. Chapter five introduces two other studies regarding progressive web apps. One is about progressive web app energy efficiency, while the other is about the user experience of progressive web apps. Finally, chapter six concludes this study as well as the topics that have been introduced in the prior chapters.

2 Progressive Web Apps

A progressive web app, also known as a PWA, is a web application that utilizes emerging web browser APIs and features to create native-like experiences for end-users across different platforms (MDN Web Docs, 2021f). Web applications are most commonly built on technologies such as HTML, CSS, and JavaScript. This is also applicable to progressive web apps, which makes development and maintenance very similar to traditional
web development. According to MDN Web Docs (MDN Web Docs, 2021d), the key principles that define a progressive web app include being:

- **Discoverable**, so the contents can be found by using search engines.
- **Installable**, so it can be found on a device’s home screen or applications.
- **Linkable**, so it is shareable by URL.
- **Network** independent, so it can be used offline or with poor network connections.
- **Progressive**, so it is usable on older browsers, but fully functional on the latest ones.
- **Re-engageable**, so it can send notifications whenever there is new content available.
- **Responsive**, so it is usable on any type of device with a screen and a browser.
- **Safe**, so the connections between the user, the app and the server are secured.

Google developers Richard and LePage (2020b) list similar principles that make a good progressive web app, while also including optimal additions such as accessibility and best practices for healthy code. It is important to note that most of the principles mentioned, such as responsivity and safety, can be applicable to any type of web application, not only progressive web apps.

Progressive web apps are required to meet certain installation criteria before the browser would prompt an install promotion (see Figure 1 and 2).

![Figure 1. Install prompt on a desktop browser (LePage, 2021).](image1)

![Figure 2. Install prompt on a mobile browser (LePage, 2021).](image2)

The installation criteria include technologies such as *service workers*, *HTTPS*, and a *web app manifest* in order for a web application to become an installable progressive web app (LePage, 2021; MDN Web Docs, 2021e). The following sections briefly introduce these technologies and explain their purpose in progressive web apps.
2.1 Service worker

A service worker is a type of web worker or script that the browser runs in the background, most commonly written in JavaScript (Gaunt, 2020; MDN Web Docs, 2021c). In essence, it acts as a proxy server that sits between the web application, the browser, and the network. This allows for certain features to be implemented on web applications, such as push notifications or background synchronization to name a few, that would otherwise be unavailable without user interaction (Gaunt, 2020; MDN Web Docs, 2021c).

As a user, having a poor network connection or being fully offline can make most web applications completely unusable, thus significantly impacting user experience. The introduction of service workers has allowed web developers to tackle this major issue, which is the loss of connectivity or offline mode. According to an article by MDN Web Docs (MDN Web Docs, 2021c), offline support is one of the overriding reasons why native applications are often chosen over web applications. The utilization of service workers allows an application to provide an offline-first experience by initially using its cached assets to provide a default experience, and then retrieving more data from the network if available (MDN Web Docs, 2021c). A type of architecture utilizing service worker caching is also referred to as an application shell (see Figure 3), where the cached assets include the skeleton of the user interface and its core components (Osmani, 2019). As a result, the application can provide a default experience similar to native applications, since it does not need to fetch these cached assets from the server.

![Application Shell Diagram](image)

Figure 3. Illustration of an application shell and its dynamically updated content (Osmani, 2019).
Progressive web apps aiming to be as capable and reliable as native applications crucially need service workers, as it not only provides an offline experience for the user using application shells, but also provides a faster and smoother experience with control over how much data is being downloaded according to the network strength.

2.2 HTTPS

HTTPS, abbreviated from Hypertext Transfer Protocol Secure, is an internet communication protocol. It is used to protect the confidentiality and integrity of data being transferred between the site and its user. Specifically, HTTPS is used to encrypt the data traffic between a website and its users, preserve the integrity of exchanged data, and authenticate that the user is communicating with the intended website (Google, 2021c). To prevent these types of vulnerabilities within progressive web apps, such as man-in-the-middle attacks (Google, 2021c), using HTTPS is mandatory for registering service workers (MDN Web Docs, 2021c). In addition, the importance of using HTTPS in progressive web apps comes from the permission workflows of new and updated browser APIs, such as the Geolocation API or the Notifications API (Basques, 2020; MDN Web Docs, 2020, 2021g) As the documentations of these two example APIs suggest, HTTPS is required for the developer to gain access to certain functionalities when developing progressive web apps.

Usage of HTTPS has gotten increasingly common over the years, since it provides a good baseline of security for modern websites that surpasses the safety and security of the previously used HTTP. According to the trends and statistics provided by the Internet Security Research Group (2021), also known as ISRG, the usage of HTTPS has been on an upwards trajectory ever since 2013. Further observations regarding the statistics provided by ISRG indicate that 84.6% of websites loaded through the Firefox browser were using HTTPS at the end of February 2021. Hence, it could be said that the technology has matured enough to be widely used, which can also play a part in its requirement within progressive web apps.

2.3 Web app manifest

A web app manifest is a file formatted in JSON that provides all the information as to how a progressive web app should behave when installed on a user’s device (LePage et al., 2021). The information stored in a manifest file ranges from the name and icon of a progressive web app once it has been installed, to more niche features such as application shortcuts that can be accessed directly from the home screen. According to LePage (2021) a progressive web app must include a web app manifest with the main set of properties to be installable. This includes:

- short_name or name,
3 Benefits and Drawbacks of Developing Progressive Web Apps

Developing and using progressive web apps can bring several benefits to both the developer and the user. These benefits can sometimes even outweigh the benefits of platform-specific native applications. However, possible drawbacks concerning progressive web apps also exist. This chapter introduces some of the benefits and drawbacks of progressive web apps.

3.1 Benefits of progressive web apps

3.1.1 Reach alongside capabilities

Unlike platform-specific native applications, a web application is inherently platform agnostic. Web applications run on the browser, which makes it accessible on any device that has a screen and a browser. By contrast, native mobile applications are distributed through proprietary application distribution platforms, such as Apple App Store and Google Play on iOS and Android operating systems, respectively. This limits the reach of native applications past other platforms, such as desktop operating systems, unless separate desktop versions are also implemented. Progressive web apps seek to bridge the gap that is the difference in reach alongside capabilities between native and web applications (see Figure 4).

![Figure 4. Difference in reach and capabilities between native, web and progressive web apps (Richard & LePage, 2020a).](image-url)
3.1.2 Single codebase for cross-platform software

Building the same application natively on different platforms usually means that the application needs to be written in multiple different codebases; native iOS applications are primarily written in Swift (2021) via XCode, an Apple-only IDE, while native Android applications are primarily written in Kotlin (2021) via the Android SDK. As a result, it is common that a single application is written in many different languages to fully capitalize on the capabilities of each platform, such as sending push notifications, adding offline functionality, handling local files, and utilizing other hardware features. Developing the same app for multiple platforms poses its own challenges, such as varying expertise within the development team, platform-specific bugs and issues, and even disbalances in features and capabilities between each platform. A survey conducted in 2013 aimed to understand the main challenges in mobile app development, and the results suggested that one of the biggest challenges was developing apps across multiple platforms (Joorabchi et al., 2013).

Progressive web apps, however, are cross-platform compatible with only a single codebase, while also utilizing emerging browser APIs to be able to acquire the same capabilities as native applications (Richard & LePage, 2020a). A single codebase, consisting of HTML, CSS and JavaScript, makes the development and maintenance process easier, since it covers the whole spectrum of devices that are able to run web applications. Ultimately, a single codebase helps save time and money for the developers, as all the required development and testing is done on a single application.

3.1.3 Flexibility

Flexibility in the context of progressive web apps can have multiple meanings. This includes different methods of distribution and installation. For instance, progressive web apps can be installed from a browser or from an app store, such as Google Play (Renzulli & LePage, 2020). Companies such as Google, Facebook and Twitter have developed numerous progressive web apps and published them onto the Google Play store using Trusted Web Activities (also known as TWA) (Renzulli & LePage, 2020), mainly by creating lightweight (or lite) versions of their popular native applications. Some of these lightweight applications in question include Google Maps Go (Google, 2021b), Facebook Lite (Facebook, 2021) and Twitter Lite (Twitter, 2021). The appeal in lightweight applications is that they take significantly less storage space compared to their fully featured counterparts, while also maintaining the core features that the user wants.

Apart from developing lite versions of popular native applications, progressive web apps can also be used for a specific platform’s version of an application. For example, popular
social networking service TikTok (2021) develops their application on iOS and Android as platform-specific native applications. On the other hand, TikTok utilizes a progressive web app for its desktop experience, the core features of the mobile application persist, and the performance is still fast and smooth. To summarize, the distribution, installation and development of progressive web apps is quite flexible, and each developer can choose whether to implement a progressive web app based on their own goals.

3.2 Possible drawbacks when utilizing progressive web apps

3.2.1 Web security flaws

In general, web applications can suffer from many types of security flaws, including vulnerabilities such as SQL injections and cross-site scripting (Fonseca et al., 2014). However, online resources related to preventing such vulnerabilities are quite common and accessible. This includes different online articles and documentation, and even JavaScript modules that can be installed onto the application that’s being developed in order to prevent such vulnerabilities. As a result, web developers can implement basic security features that prevent such vulnerabilities, even without specific security or privacy experts working on the project simultaneously. It is important to note that developers need to be aware of such vulnerabilities and willing to actively improve on the security and privacy features of the web application.

Regarding security and privacy vulnerabilities that directly concern progressive web apps, a study conducted by Lee and others (2018) was made regarding specific progressive web app features resulting in vulnerabilities such as phishing via push notifications, side-channel attacks that abuse cache and cryptocurrency attacks by taking advantage of service workers. The study also suggests practical defense mechanisms against the vulnerabilities in question, and has curated separate sections for library providers, progressive web app developers, and users. Findings regarding these security vulnerabilities have also been reported to the involved vendors. Like any technology, browsers and browser APIs develop and mature over time, resulting in more secure progressive web apps.

3.2.2 Poor web performance

Web performance optimization for progressive web apps is crucial, especially if there are alternative native applications that have better performance. Poor web performance optimization can negatively affect user experience, potentially leading to site abandonment or even impacting conversion rates. Negative impact on conversion rates means that users would less likely perform a measured or desired action, such as making a purchase or reading an article (MDN Web Docs, 2021a). As a result, tracking web performance should be at the forefront of all web developers, regardless of the app being a progressive
web app or not. The following chapter will breakdown progressive web app performance in more detail.

4 Performance of Progressive Web Apps

Performance optimization plays a crucial role in developing progressive web apps, as poorly optimized applications can negatively affect user experience, thus potentially leading to fewer users. Knowing how to increase performance and improve the user experience of a progressive web app means it is essential to know which tools are used to measure web performance, and what techniques are used to boost web performance. This chapter will briefly introduce the tools commonly used to measure and evaluate progressive web app performance, in particular the tools mentioned by Google engineers Bojan Pavic and Chris Anstey (2019). In addition, an overview of certain performance terminology and optimization techniques used in real-world cases will also be introduced.

4.1 Web performance measuring tools

When measuring performance, it is important to understand that there is variability of performance within different real-world scenarios. For example, performance can vary depending on the users’ devices and network connections. As a result, performance measurement data can be separated as lab data and field data (Pavic & Anstey, 2019). Like any lab experiment, lab data is collected within a controlled environment. Certain variables such as device and network settings are predefined, thus providing reproducible performance results. On the other hand, field data aims to capture real-world scenarios, where user experience may vary drastically depending on different variables.

Different types of data require different measurement tools. Most commonly, DevTools provided on browsers such as Chrome (Google, 2021a) and Firefox (MDN Web Docs, 2021b) are used to analyze the runtime performance of a website. Excluding DevTools, Pavic and Anstey (2019) also recommend two other tools: Lighthouse (Google, 2021e) and the Chrome User Experience Report (Google, 2021d). Lighthouse is used for lab data measurements by running a series of audits against a website and generating a report, while the Chrome User Experience Report (also known as CrUX) is used for field data by showing real-world Chrome user experience metrics of popular websites (Pavic & Anstey, 2019). There is also the PageSpeed Insights page, which is powered by both Lighthouse and CrUX, providing a way to measure both lab and field metrics simultaneously. In the context of developing progressive web apps, Lighthouse is an extremely powerful tool, as it additionally provides metrics for progressive web app standards, accessibility, search engine optimization (also known as SEO), and even coding best practices. Other performance metrics tools also exist, however this thesis mostly focuses on Chrome DevTools and Lighthouse, as they are the most popular tools when developing
progressive web apps and have been used in certain case studies. Specific case studies and their progressive web app performance optimizations will be presented later in this chapter.

4.2 Common web performance metrics
When developing a web application and optimizing its performance, it is important to know what the most common web performance audits are and how improving on them can improve user experience. For Lighthouse, this includes (Lighthouse performance scoring, 2021):

- **First Contentful Paint (FCP):** marks the time at which the first text or image is painted.
- **Speed Index:** shows how quickly the contents of a page are visibly populated.
- **Largest Contentful Paint (LCP):** marks the time at which the largest text of image is painted.
- **Time to Interactive (TTI):** shows the amount of time it takes for the page to become fully interactive.
- **Total Blocking Time:** sum of all time periods between FCP and Time to Interactive when task length exceeds 50ms.
- **Cumulative Layout Shift:** measures the movement of visible elements within the viewport.

The breakdown of these metrics is to ensure a better understanding of the upcoming case studies and the performance optimization methods that have been implemented.

4.3 Web performance optimization techniques
There are certain web performance optimization techniques commonly deployed on well-known applications. This section provides a high-level overview on some of the techniques used to boost web performance, specifically mentioned in the progressive web app case studies of Tinder, Pinterest, and Twitter Lite (Osmani, 2018a, 2018b; Armstrong, 2018). The case studies were selected specifically with progressive web app performance optimization as the primary condition. Pinterest, for example, saw significant improvements in their core business metrics after progressive web app optimizations, including *time spent, user-generated ad revenue* and *core engagements* (see Figure 5).
While there exists an endless amount of web performance optimization techniques depending on the libraries and frameworks used, the following subsections focus solely on the optimizations shared among the case studies. As a result, certain commonly known web performance optimizations could be unlisted. The web performance optimizations shared among the case studies include *bundling* and its analysis, *code-splitting, caching* and *pre-caching* assets, and mentions of *preload*, which will be introduced in the following paragraphs.

### 4.3.1 Bundling

Bundling, in the context of software development, is defined as a process of taking multiple files or modules and combining them into a singular file. Tinder, Pinterest and Twitter Lite use *Webpack* (2021) as their bundler of choice, and the analysis process proved to be much easier by utilizing the *Webpack Bundle Analyzer* -software (Osmani, 2018a, 2018b; Armstrong, 2018). It is common practice to bundle all resources required by the application in order to reduce network requests, for example during the initial load time (Yberg, 2018). However, having a single bundle containing all the dependencies of the application can also be a bottleneck, as large bundles can negatively impact performance, specifically metrics such as First Contentful Paint, Largest Contentful Paint and Time to Interactive (Osmani, 2018a, 2018b; Armstrong, 2018). As a result, analyzing and optimizing large JavaScript bundles was a key aspect of increasing web performance for these applications.

### 4.3.2 Code-splitting

One of methods used to optimize bundles is code-splitting, allowing the developer to split code into various smaller bundles or *chunks* that are loadable on demand or in parallel, thus giving the developer more control over resource load prioritization (Code Splitting, 2021). Tinder, Pinterest and Twitter Lite applied code-splitting on a *route-based level*,
allowing critical pieces of code to be loaded upfront, while lazy-loading non-critical resources once they are needed (Osmani, 2018a, 2018b; Armstrong, 2018). As a result, sizes of main bundle decreased, noticeably improving application load times, creating a faster and smoother experience. For a progressive web app with numerous assets, code-splitting can be a significant factor in creating user experiences that feel fast.

4.3.3 Caching and pre-caching

Service worker caching and pre-caching can reduce the number of requests being sent towards the application server, as well as increase performance metrics such as Time to Interact on repeat visits. Tinder used service workers to pre-cache all their route-level bundles (Osmani, 2018a), allowing routes to be cached before they are requested and making pages load faster with less unnecessary network traffic. In addition to route-level bundles, Tinder also caches their application shell as well as their core static assets, including JavaScript bundles and CSS (Osmani, 2018a). Pinterest uses similar methods of caching and pre-caching as Tinder, while also implementing per-locale service workers so that locale bundles could also be cached (Osmani, 2018b). As a result, it had a positive impact on repeat load performance as well as enables basic offline rendering for most of their users. Pre-cached assets also play a significant role in subsequent application load performance of Twitter Lite, even after frequent deployment of newer versions of the application’s JavaScript assets (Armstrong, 2018). On a 3G network, load times when returning to the application saw a noticeable improvement of 75% by pre-caching assets (Armstrong, 2018). As a result, caching and pre-caching can also help applications become more accessible, especially in the context of weaker network connections or slower internet speeds.

4.3.4 Preloading resources

Preload is a technique used on Tinder, but also mentioned as an interesting addition to Pinterest (Osmani, 2018a, 2018b). Using preload on certain resources allows the browser to fetch them sooner than the browser would otherwise discover them (Djirdeh & Mihajlija, 2020). In the case of Tinder, preload was used fetch their critical JavaScript and webpack bundles that were important for the core experience, reducing both load time and first paint time (Osmani, 2018a). In other words, users are able to interact with the application noticeably faster, resulting in better user experience.

5 Other Studies Regarding Progressive Web Apps

The development of progressive web apps has also introduced academic studies and research beyond the scope of performance. The following subchapters will introduce two
academic research articles, specifically about progressive web app energy efficiency and user experience, respectively.

5.1 Energy efficiency
The year 2017 saw the first empirical investigation regarding service workers and progressive web apps, as Malavolta and others (2017) had published a research paper assessing energy efficiency of service workers in progressive web apps. Research regarding the energy efficiency of service workers within progressive web apps was thought to be important, as energy is a scarce resource in mobile devices, and no evidence was found regarding service worker battery drain (Malavolta et al., 2017). The research involved two different devices with different specifications, two different network conditions of 2G and Wi-Fi, as well as seven different progressive web apps for testing, handled within a replicable environment. The two research questions proposed by Malavolta and others (2017) were: “Does the use of service workers impact the energy efficiency of [progressive web apps]” and “Does the use of service workers impact the energy efficiency of [progressive web apps] under different network conditions.”

According to Malavolta and others (2017), results indicated that service workers do not have a statistically significant impact on energy efficiency. Interestingly, results showed that the median energy consumption was always lower when service workers were enabled on the progressive web app (Malavolta et al., 2017). Therefore, it is recommended to develop progressive web apps with service workers, as lower energy consumption can play a positive role in deciding whether users are willing to use progressive web apps or not. However, results also indicated that low-end mobile devices consumed noticeably more energy compared to higher-end mobile devices when running progressive web apps (Malavolta et al., 2017), suggesting that improvements in device hardware and platform positively impacts the energy efficiency of progressive web apps. Energy consumption under different network conditions also resulted in marginal differences, suggesting that the features and benefits gained by the utilization of service workers have a minimal impact on the energy consumption of the mobile device running them (Malavolta et al., 2017).

5.2 User experience
Experimental studies specifically regarding the usability and user experience of progressive web apps had not been done, until de Andrade Cardieri and Zaina (2018) published their work in 2018. The focus of the study was to compare the user experiences of using web, mobile and progressive web apps. Keyword-based research was initially conducted to find out which interface elements within native and web applications were commonly reported with usability issues. The interface elements selected included search, date
picker, select, map, and tab menu and table. The user experience comparison between a progressive web app and a native application was done with Trivago, a popular travel application that has both a progressive web app and a native application.

According to de Andrade Cardieri and Zaina (2018), many users experienced satisfaction after interacting with the progressive web app and its interface elements. On the other hand, all native interface elements were observed to provoke negative emotions from participants, specifically from either slow feedback after interaction or slow loading (de Andrade Cardieri & Zaina, 2018). This is a surprising yet important result for developers, as it indicates that progressive web apps can compete against native applications regarding user interface elements and overall user experience.

6 Conclusions
This literature study aimed to provide an understanding for what progressive web apps are, why should developers create them, and what are factors for consideration during development and optimization. This study begins by defining progressive web apps and the critical components that differentiate it from a standard web application. The benefits and possible drawbacks listed in chapter three aim to answer our second research question, which is why developers should create progressive web apps. And finally, chapters three to five introduce the specific factors for consideration when developing progressive web apps.

Web technologies and browser APIs are constantly improving and advancing. Progressive web apps surpassing native application performance and user experience is extremely plausible with proper optimization, and the reach that differentiates the web from native ecosystems can provide an edge for developing a large userbase. In addition, it can be expected to see an increasing number of existing web applications make the transition to progressive web apps, such as the prior mentioned applications of Tinder and Pinterest.

In the future, it is important to monitor the development of progressive web apps, including the web technologies and browser APIs that they are built on. In particular, the improvement and advancement of progressive web apps can make them an appealing option for both the developers and the user, regardless of the platform or environment in which the progressive web app is running on.
References


