Mira Hiltunen

Creating multiplatform experiences with Progressive Web Apps

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Thesis

13 November 2018

Author Title Number of Pages Date	Mira Hiltunen Creating multiplatform experiences with Progressive Web Apps 35 pages 13 November 2018
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Mobile Solutions
Instructors	Petri Vesikivi, Principal Lecturer Kalle Varisvirta, CTO, Exove Oy

The purpose of this thesis was to research and learn about Progressive Web Applications and then to create an application that implements some Progressive Web App features. The thesis looks into application development in general level and what Progressive Web Applications are, how they are built and what they are good at and the phases of the application development.

Progressive Web Apps have recently been grown in popularity. They enable features that used to available only on native applications. They have several benefits over native and conventional web applications and they are very cost-effective way to develop applications. According to several studies, Progressive Web Apps increase number of users and engagement compared to traditional web-based applications.

A time tracking application was developed for a company specialized in software development. For the first launch the purpose was just to create a minimum viable product that would implement only the mandatory features required for tracking worktime. The application front-end is based on React, a JavaScript framework developed by Facebook. Application development started with design and coming up with requirements through user stories. Development process proceeded mostly without problems even though deadlines posed some difficulties. The application was tested mainly with built-in tools in the browsers and actual user testing was left to be done after the first release.

The result of the development work was a React application implemented as a Progressive Web App. All the functionalities of the product were not finished, and the development of the application will continue. However, the Progressive Web App functionalities were successfully implemented in the final product.

Keywords	JavaScript, Progressive Web App, React

Tekijä Otsikko Sivumäärä Päiväys	Mira Hiltunen Alustariippumattomien käyttäjäkokemusten toteutus progressiivisilla verkkosovelluksilla 35 sivua 13.11.2018
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Mobile Solutions
Ohjaajat	Yliopettaja Petri Vesikivi Teknologiajohtaja Kalle Varisvirta

Insinöörityön tarkoitus oli tutkia progressiivisia verkkosovelluksia ja toteuttaa sovellus, johon on sisällytetty progressiivisten verkkosovellusten ominaisuuksia. Insinöörityössä perehdyttiin sovelluskehitykseen yleisellä tasolla: mitä progressiiviset verkkosovellukset ovat, miten niitä tehdään ja mitä etuja niillä on muihin sovelluksiin verrattuna.

Progressiiviset verkkosovellukset ovat kasvattaneet suosiotaan viime aikoina. Ne mahdollistavat toimintoja, jotka oli ennen mahdollista toteuttaa vain natiivisovelluksilla. Niillä on useita hyötyjä natiivisovelluksiin ja perinteisiin verkkosovelluksiin verrattuna, ja ne ovat kustannustehokas tapa kehittää sovelluksia. Useiden tutkimusten mukaan progressiiviset verkkosovellukset lisäävät käyttäjämääriä ja käyttäjät ovat sovelluksiin sitoutuneempia kuin perinteisiin selainpohjaisiin verkkosovelluksiin.

Insinöörityössä toteutettiin tuntiseurantasovellus ohjelmistokehitykseen erikoistuneelle yritykselle. Käyttöönottoa varten tarkoituksena oli toteuttaa mahdollisimman yksinkertainen, toimiva tuote, joka sisältäisi aluksi vain tuntikirjauksen kannalta pakolliset toiminnot. Sovellus tehtiin Reactilla, joka on Facebookin kehittämä JavaScript-sovelluskehys. Sovelluskehitys alkoi suunnittelulla ja vaatimusmäärittelyllä käyttäjätarinoiden kautta. Kehitystyö sujui suurimmaksi osaksi ilman ongelmia, vaikka vaikeuksia projektissa tuli tiukasta aikataulusta. Sovellusta testattiin kehitystyön aikana lähinnä selaimeen sisäänrakennetuilla työkaluilla ja varsinanen käyttäjätestaus päätettiin toteuttaa vasta ensimmäisen julkaisun jälkeen.

Kehitystyön tuloksen syntyi progressiivinen React-sovellus. Kaikkia ominaisuuksia ei saatu kehitettyä loppuun asti, ja sovelluksen kehitys jatkuu edelleen. Kaikki progressiiviset ominaisuudet saatiin kuitenkin toteutettua sovellukseen.

Avainsanat	JavaScript, progressiivinen verkkosovellus, React

Contents

List of Abbreviations

A2HS	Add to Home Screen. Modern browser feature that allows adding an application shortcut to the home screen of the device.
API	Application Programming Interface. Set of defined available methods of communicating with the underlying system.
CRA	Create React App. React app boilerplate without build configuration.
DOM	Document Object Model. HTML interface that defines the document structure.
HTTP	Hypertext Transfer Protocol. Application layer protocol for hypermedia systems.
OS	Operating system. Program that runs on a computer managing the hardware and other programs.
PWA	Progressive web app. Website or application that uses
UI	User interface.
UX	User experience.

1 Introduction

1.1 Background

Information and communication technologies are evolving constantly and their impact on people’s lives is higher than ever. Types of applications range from entertainment to education, healthcare to sports and almost any other genre. Different services are being developed more and more on mobile and delivering a great user experience across all devices is vital as the use of mobile web and applications is growing at a fast pace.

Responsive design has had a positive impact on user experience especially on mobile devices, though native solutions often provide better experience both on desktop and on mobile. Traditionally, application development for mobile and desktop environments has been done separately from each other and it has not been very cost-effective. Native software distribution and updates to software are difficult. However, the web is an extremely convenient way to provide services on all the platforms, but its capabilities have been very limited in the past and web apps have required stable internet connections.

Progressive Web Apps tackle some of these issues improving the user experience on web and changing the way applications are built. Progressive Web Apps, developed by Google, have been around for a few years now, and they address many of the issues with native and web experiences. Progressive Web Apps bring the native look and feel to the web, enabling new features such as offline usage.

Exove is a company that focuses on developing and designing digital solutions. Exove’s hour tracking system, Inside 2, was outdated and lacked in user experience. In 2018 Exove started a process of taking a new ERP (Enterprise Resource Planning) system into use and needed an hour tracking application that could be integrated into it, synchronizing working time data with the new system. Hence it was decided to build a new application that would be implemented as a Progressive Web App, making it as easy as possible for employees to keep track of working time on mobile and desktop, wherever they are working.

1.2 Objectives

The objectives of this thesis are:

- Researching features and possibilities of Progressive Web Apps.
- Learning about Progressive Web App development.
- Designing and developing a front-end application for hour tracking.
- Growing and sharing in-house knowledge at Exove.

Backend development, ERP integration or any server architecture are not part of this thesis. The frontend application relies on a separately developed backend and API.

1.3 Structure of the report

This thesis is divided into five chapters. The next chapter is an overview to application development, and it covers different types of applications and ways to develop those. Chapter three focuses on Progressive Web Apps: what they are, how they work and how they are implemented. Fourth chapter discusses the implementation of the hour tracking app and the final chapter discusses the reflections and evaluation of the application and the possibilities of Progressive Web Apps.

2 Overview of application development

Front-end applications can generally be divided to three groups: native, hybrid and web apps. Applications for desktop, mobile and web are often developed individually for all platforms as there are few cross-platform solutions that would work seamlessly in all the platforms.

2.1 Web applications

The web has developed substantially during the past few decades. Originally the internet was only used in a very few selected companies and research facilities, like CERN. Tim Berners Lee, an engineer at CERN, developed HTML and HTTP in order to make the large number of scientific documents easier to manage, as it had become almost impossible. In November 1992 only 26 websites existed; today the number of websites can be counted in millions. [1.]

Web apps are run in browser engines and they are mostly platform independent - there are some differences in how different browsers render content, but generally web apps work similarly across all platforms. Developing web applications over native apps has numerous benefits; the same code base works on all the platforms and distributing new applications is effortless as it does not require any marketplace application submissions and waiting for approval. This makes updating and fixing bugs simpler too, as users always have the newest version in their hands since the latest version is always loaded when visiting the app.

Implementation of web apps vary a lot. HTML, CSS and often JavaScript are the basic technologies that web apps are built on. In the past websites used to be very static and implementations simple. Today there is a wide range of frameworks, boilerplates and content management systems (CMS) to choose from, when building a website. Trends are ever changing, JavaScript frameworks come and go, and the technologies develop rapidly. Responsive design is one trend that changed how the web is viewed and built on mobile, and that is still relevant today.

Mobile first, graceful degradation and progressive enhancement all rely on responsive design. Mobile first is a methodology for designing and developing web applications where content is designed to fit smaller screens first, expanding to larger breakpoints after that [2]. Most users are visiting web apps using smartphones with small screens and mobile connection that often have limited bandwidth. Therefore, it is important to take user experience on mobile into consideration. Building mobile views first and then gradually make the content responsive for larger viewports lets developers and designers to optimize responsiveness maintaining as few the breakpoints as possible [2].

Graceful degradation, on the other hand, is somewhat upside-down thinking as opposed to mobile first. Designers and developers often want to take advantage of the latest features and browser capabilities. However, that often means excluding some users whose devices or connections fail to support those. For example, older versions of Internet Explorer are notorious for their lacking feature support and developers often want to avoid optimizing apps to work in those versions. In graceful degradation the focus is on building as advanced systems as possible and taking the possible degradations and lack of support into account in later phases during the development, ensuring that the app would remain somewhat functional in older systems, too. In other words, a website packed with features would gradually scale and remove content and features when the system becomes simpler and viewports smaller. [1.]

Progressive enhancement is a web development strategy that focuses on establishing a solid foundation for the user experience and including basic functionalities that all browsers can support. More advanced features are included automatically for all the browsers that support it. Thus, while graceful degradation focuses on delivering complex and fancy solutions and then simplifying on demand, progressive enhancement is a complete opposite approach: it emphasizes delivering strong, simple foundations and then adding complexity progressively on top of the foundations, delivering a usable product regardless of environment. [1.]

2.2 Native applications

Native apps are developed for a specific operating system (OS), such as Windows, Linux, macOS (previously OS X) or Android and they generally only work on the platform they have been developed for. Two major mobile platforms are Android and iOS, and they both have their own software development kits (SDK). They are able to use device-specific built-in hardware and software and perform generally better than equivalent hybrid applications. Native applications often provide better user experience with a UI that matches the OS.

Android applications are programmed using Java, Kotlin or C++. Native iOS applications, on other hand, are made with Swift or Objective-C. Swift is a programming language introduced in 2014 and is intended to replace Objective-C eventually [3]. For developing native desktop applications, there are even more programming languages to choose from. Popular programming languages include Java, C# and C++.

Native applications are often distributed through centralized app stores, such as Google Play, App Store or Microsoft Store. Having a native application published in an app store can be quite cumbersome, as there are a lot of requirements an application must meet in order to have the application listed. For example, for publishing in Google Play developers must consider policies, API levels, device compatibility, quality guidelines and planning store listing and among other issues that are not relevant when publishing apps in the web [4]. Additionally, developing and publishing native apps for iOS and macOS requires a computer capable of running macOS. Updating the applications that are listed in these marketplaces is not that straightforward either. After publishing an update in a store, is not certain that the app will get updated unless the user has opted in for automatic updates, so often getting the latest version to users requires user interaction. The big positive in having an app listed in a store is visibility to a large userbase, and they are generally safe to use as they are always reviewed before listing. When it comes to desktop apps, even though App Store and Microsoft Store are popular, desktop software is not as often distributed in a dedicated app stores, especially on Windows OS.

In addition to problematic publishing, updating and distribution, another disadvantage of developing native applications is having to possibly maintain several codebases and versions of the same application if the target audience uses multiple devices. This means that development consumes a lot of time and human resources. Developing for different platforms requires specialized skill sets, as programming languages and techniques vary between platforms. Additionally, services might have presence in web too, adding up to the number of code bases to be maintained. In summary, developing native applications or all of its strong suites can be quite a large investment.

2.3 Hybrid applications

Hybrid apps are typically cross-platform applications that are made with web technologies such as HTML, CSS and JavaScript. Hybrid apps are based on frameworks, such as Apache Cordova or Electron, that enable the application to function similarly to native apps. Hybrid applications usually work in a browser-based WebView, and they have access to certain hardware capabilities as well. However, those capabilities are somewhat limited depending on the selected framework as some functionalities rely on external sources and third-party plugins. For example Cordova does not support fingerprint scanning by default. [5.]

Electron is an open-source library for cross-platform desktop application development with JavaScript, HTML and CSS. It uses Chromium and node.js together, building them to a single runtime. Electron apps can be packaged for all three major operating systems. Many popular apps, such as Slack, Skype and Discord are built with Electron. [6.]

As mentioned, hybrid applications are usually cross-platform apps, which makes them very cost-efficient compared to developing separate native apps. Developing hybrid applications is easier for web developers who are already familiar with web technologies and want to develop cross-platform apps without having to implement native solutions individually to each desired platform, or mobile developers who want to distribute their app without having to re-implement it for each platform [5].

Hybrid apps come with their downsides too. Their performance is often lower than performance in native apps and their functionalities do not match native ones. Even so, the reduced cost of development for multiplatform applications makes hybrid app development very appealing.

2.4 Cross-platform applications

Software that can run in multiple different platforms is called a cross-platform application. Cross-platform apps are often hybrid apps implemented using web technologies, but there are also several frameworks that enable cross-platform native development, such as React Native or Xamarin.

React Native is a framework that enables native mobile app development with JavaScript and React. As React, a JavaScript framework developed by Facebook, turned out very successful, Facebook created React Native for streamlining the application development process for multiple platforms. Traditionally native applications for Android are made with Java, while applications for iOS have been made with Objective-C, or lately, Swift. However, in order to publish an application in multiple platforms, learning several programming languages takes time, and maintaining multiple codebases consumes a lot of resources. React Native pursued to make this easier allowing developers to use a single framework for cross-platform development. [7.]

Even though it is theoretically enough to know just JavaScript for developing a React Native application, there can often be platform specific issues where knowledge of native application development is beneficial; according to Discord, development is significantly easier for native developers, than pure web developers [6]. Additionally, there are also some performance issues when developing with React Native. Java, Objective-C and Swift are considerably faster than JavaScript, especially when heavy calculations are included. [7.]

Xamarin is another cross-platform tool that enables native development for Android, iOS, OS X and UWP (Universal Windows Platform) with C# using Visual Studio. Xamarin applications can share codebase that includes most of the app logic and user interface, counting about 75 % of the code in general. In addition to the shared code base, the applications usually need some platform specific code. Developing with Xamarin makes it easier to develop for multiple platforms but some platform specific programming is needed, nevertheless. [9.]

3 Progressive Web Apps

3.1 What are Progressive Web Apps

Progressive enhancement is a web development strategy that has an emphasis on creating great user experiences regardless of the device capabilities or web connection speeds. Progressively enhanced applications get progressively better when connections and device capabilities improve. In case of connectivity, everything loads instantly if the connections are fast but on slower speeds the most important parts will be prioritized, and less relevant items are loaded later. [10.]

Progressive Web Apps are web apps with a progressively enhanced, native-like user experience. PWAs can be installed on a user's device, they have access to some native functionalities like push notifications and they are able work offline as well. Once pinned to home screen, user experience should not be very different from that of a native app. They also integrate in the device settings and the app launcher. PWAs still run in the browser but they have their own app window, and the browser UI elements are not visible making the app appear as a native application. [10.]

Google describes Progressive Web Apps with the following keywords:

- Reliable,
- Fast,
- Integrated,
- Engaging. [11.]

For an application to be considered a Progressive Web App, it needs to meet certain technical requirements:

- Have a manifest file declared in the HTML head. Web manifest should be a standard JSON file placed in the app directory that includes information about the application and its behavior. Mandatory fields for PWAs are *background_color*, *display*, *icons*, *start_url* and either *name* or *short_name*. [13.]

- Include an appropriate icon for home screen. There should be several differently sized icons for optimal display in all different devices and device sizes. Recommended sizes are 128*128 px as a base and a range from 152*152 px to 512*512 px to accommodate a range of different screen and device sizes. [10.]

- Register a service worker. Service workers are web workers that reside between the app and the network, handing network requests and caching, for example. [10.]

- Serve the application over HTTPS. HTTPS (Hypertext Transfer Protocol Secure) is an internet protocol. HTTPS prevents intruders from exploiting resources and tampering with data because the communication is encrypted and so it protects the integrity of the website. Service workers only work on applications where HTTPS is enabled. During application development, testing and developing is usually done on localhost, which is an exception for this rule. It is always a good practice to secure applications in the web, regardless of the nature of the data that is handled inside the application. [12.] Services like Let’s Encrypt by Internet Security Research Group (ISRG) allow anyone owning a domain to get the needed certificates for free.

Essentially almost any web app can be made a Progressive Web App as long as they meet the requirements. React, Angular, Vuejs and SPAs in general are very popular frameworks for PWA development, but it’s possible to make a CMS website, like Drupal or WordPress, a PWA too.

3.1.1 A2HS

A2HS (Add to Home Screen) is a feature in modern browsers that allows adding a shortcut to the application in user’s home screen, making it easy to access repeatedly. Shortcuts work on both mobile and desktop platforms, although desktop support is still fairly new. After accepting the prompt, app icon is added to the launcher and the newly added PWA will run like any other app installed on the system. [13.]

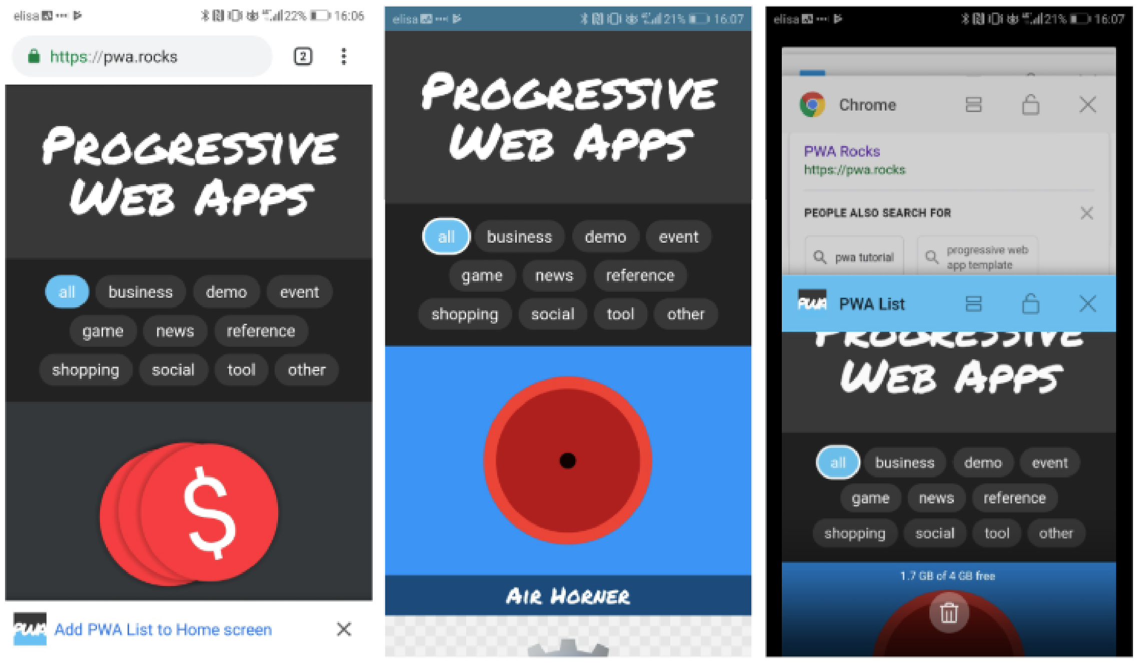
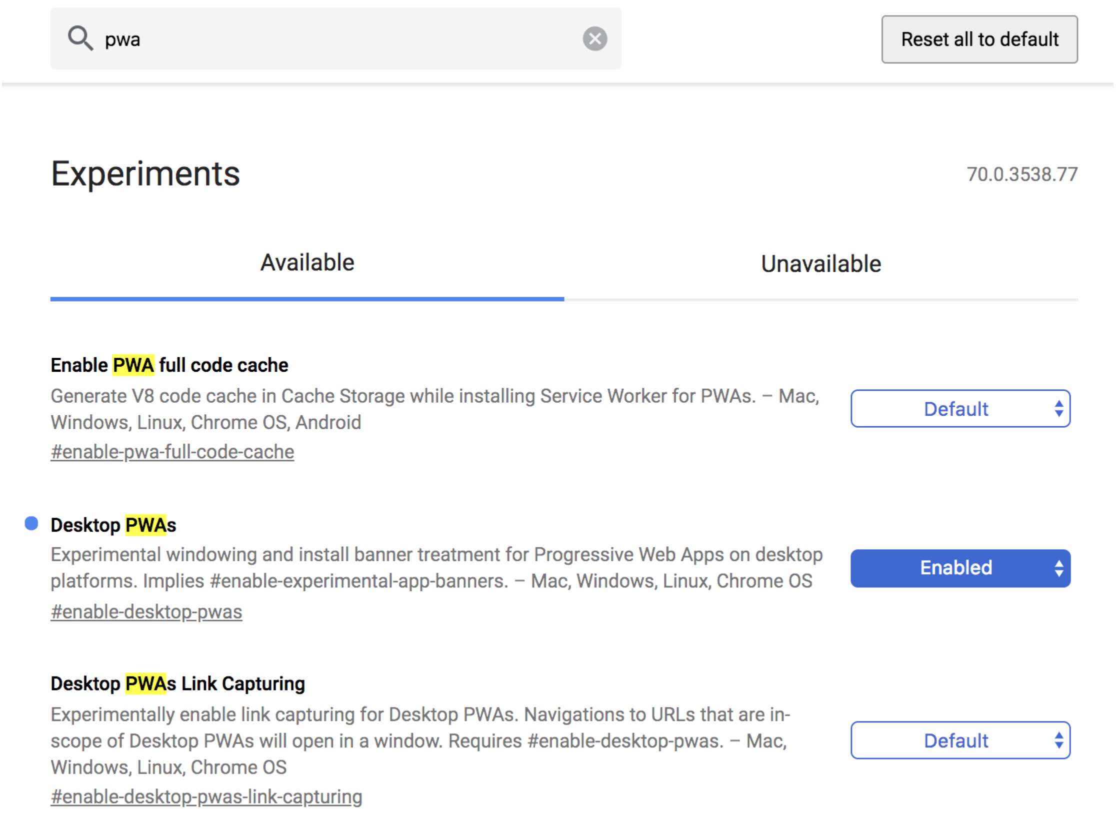


Image 1. Progressive Web App opened in Chrome for Android on the left, prompting the user to add the app to home screen. After installation, the app is seen just like any native app without the browser UI in its own window.

When launching the app using the icon in the home screen, it might take some time to get the browser up and running. During this time a splash screen will be shown. The splash screen is generated automatically, and it shows a background color and app icon in the middle, both defined in the manifest properties. [14.]

The installation prompt will be shown to mobile user after a so-called user engagement heuristic, which vary between browsers. In Chrome this means that the user has to interact with the domain for at least 30 seconds. For the dialog to show up, the app must listen to *beforeinstallprompt* event, which will trigger after the 30 seconds have passed. Function *prompt()* can then be called in order to show the popup. [15.]

Desktop browsers so far do not automatically prompt users to install the applications. In fact, the desktop PWA feature must be specifically enabled in the browser settings on Chrome by enabling a “Desktop PWA” flag as seen in image 2. Even with the flag enabled, A2HS prompt is shown only on mobile but it is possible to, for example, add a button to the web app and use JavaScript to fire an event asking user to install the application [13].


pwa

Reset all to default

Experiments

70.0.3538.77

Available

Unavailable

Enable PWA full code cache
Generate V8 code cache in Cache Storage while installing Service Worker for PWAs. – Mac, Windows, Linux, Chrome OS, Android
#enable-pwa-full-code-cache

Default

Desktop PWAs
Experimental windowing and install banner treatment for Progressive Web Apps on desktop platforms. Implies #enable-experimental-app-banners. – Mac, Windows, Linux, Chrome OS
#enable-desktop-pwas

Enabled

Desktop PWAs Link Capturing
Experimentally enable link capturing for Desktop PWAs. Navigations to URLs that are in-scope of Desktop PWAs will open in a window. Requires #enable-desktop-pwas. – Mac, Windows, Linux, Chrome OS
#enable-desktop-pwas-link-capturing

Default

Image 2. Enabling Desktop PWA flag in *chrome://flags*.

Alternatively, PWAs on desktop can be installed from the browser menu. Without any visual cues on the web page, such as an installation button, many users will likely not find this functionality, resulting in a bad user experience.

3.1.2 Manifest

Manifest is a JSON file that provides information about the application, like name, icons and color scheme, to the browser. Manifest.json is deployed to the app by injecting it in the head using a link tag. [15.]

```
<link rel="manifest" href="%PUBLIC_URL%/manifest.json">
```

Listing 1. Manifest file included in the head of an app.

Here are listed some of the properties in manifest file and their function. Although there are more properties than the ones listed here, these ones are the most relevant ones for PWAs.

short_name and name

Name is the application name and it is visible in the installation prompt and home screen. Alternatively, if short name is declared, it can be used in home screen where space is limited instead of name.

icons

Declares an array of images that are used in various places, like shortcut icon, task switcher and so on. It is recommended to include at least a 192x192 pixel icon and a 512x512 pixel icon.

start_url

URL relative to the manifest file that tells where the application should be launched. This is often the index file, but sometimes it is better to lead the users elsewhere, depending on the purpose of the application. For example, in a website where the frontpage acts like a landing page for a product it might be better to open the app in a page where the users can actually do something.

background_color

Used as a background color in the splash screen when the app is loading after launch, before any stylesheets are loaded. Its purpose is just to provide a better user experience and a smoother transition.

theme_color

Used as the default theme color for the app. This affects some OS elements, like background color in Android task switcher or Windows app title bar color.

display

Display is used to define the preferred browser mode for the app. There are four possible values.

Display mode	**Description**	**Fallback**
fullscreen	Uses all the space available. No user agent is visible.	standalone
standalone	App looks and feels like a native app.	minimal-ui
minimal-ui	App looks and feels like standalone but with some minimal browser con-trols, that vary depending on platform and browser.	browser
browser	App opens in a browser tab or window Default display mode.	-

Table 1. Browser mode display values and their descriptions with their fallbacks.

orientation

Orientation lets the developer to enforce it for apps that are supposed to work in only one orientation, such as games. Users often prefer to choose the orientation themselves, so it is not advised to specify this unless needed.

scope

Scope defines the set of urls that are considered part of the app.

3.1.3 App shell architecture

App shell is a way to build the application so that it loads instantly and reliably. Essentially the app shell holds the minimal UI structure and related CSS required for the application to work as demonstrated in image 3. The architecture is a key point in the performance and reliability of Progressive Web Apps. Application shell should load rapidly and use very little data. [16.]

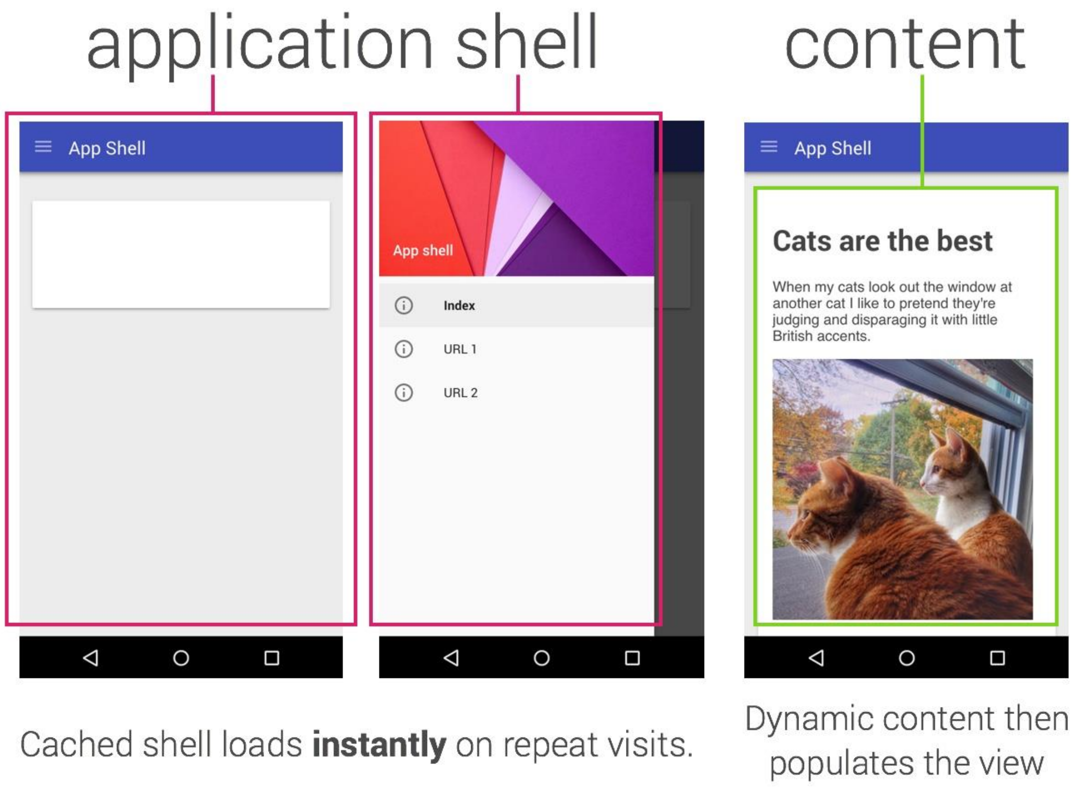

application shell
content
App Shell
App shell
Index
URL 1
URL 2
App Shell
Cats are the best
When my cats look out the window at another cat I like to pretend they're judging and disparaging it with little British accents.
Cached shell loads **instantly** on repeat visits.
Dynamic content then populates the view

Image 3. App Shell model as defined by Google [16].

App shell model makes a lot of sense for applications with unchanging parts in UI, such as navigation or footer, and dynamically changing content. Application shell should be cached using a service worker so that no data would be retrieved on subsequent loads unless there is new or changed data. [17.]

3.1.4 Service Workers

Service worker is a type of web worker, which is JavaScript running in the background of an application independently of other scripts. Web workers don't affect the performance of the web app as the scripts are executed separately from the main browser thread. In other words, service workers are scripts running in the background that sit between the network and the application; they act as a client-side proxy which makes it possible to control all the network requests done by the application. Additionally, they can receive messages even if they are inactive, for example when the app is in the background. [10.]

Service workers enable features that do not demand user interaction or browser, such as caching, push notifications and background synchronization. As network conditions vary a lot and data might be expensive for some, configuring service workers for caching is very useful. If the application requests a file from the server, the service worker can intercept the request, check if the file exists already in the local cache, and return it in case it is found, so no actual network requests are made. In case the file is not found, the service worker allows the request to go through, fetch the file and save it in the cache for future requests. [10.]

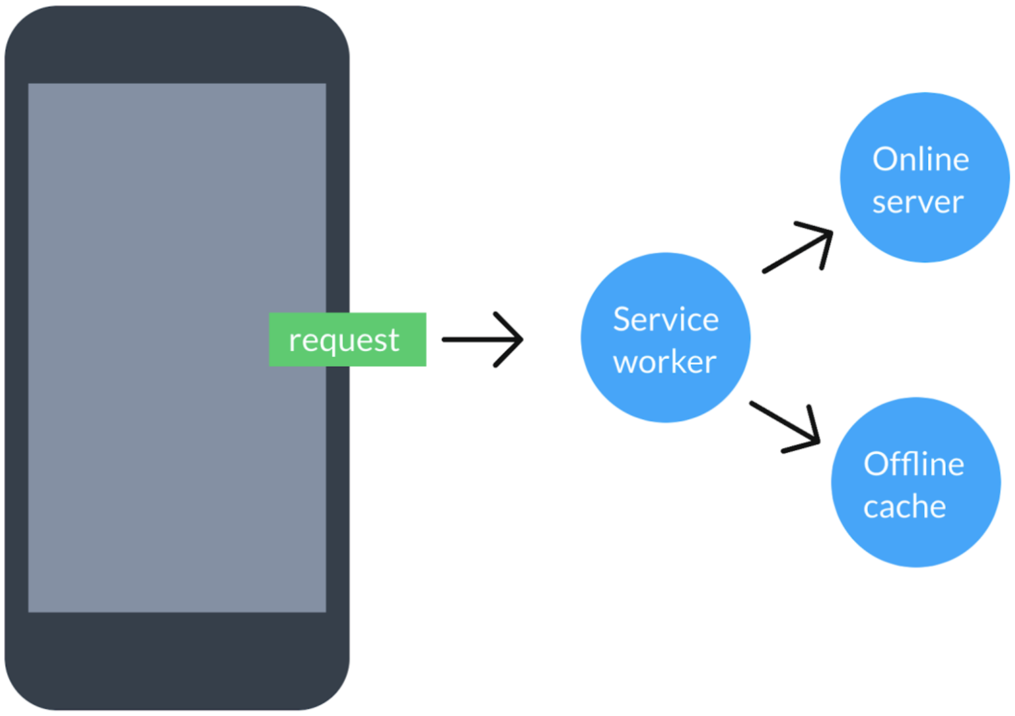


Image 4. Illustration of service worker handling network requests.

In a normal application architecture, the app and its scripts stop running when it's not active. Thus, receiving push notifications there is not possible unless the app is in the foreground, listening actively. Because service workers stay in the background, they enable for example background sync and push notifications. Even if the app goes in the background, service workers are always working in the background. [10.]

Service worker lifecycle

Service worker's lifecycle is separate from the web page, and it begins when a user visits the application and the service worker is installed (image 5). Before installation, it is necessary to check whether the current browser supports service workers since as of now, some browsers still consider it an experimental feature. If the browser supports service workers, it will start installation. The service worker is then registered to work with a specific scope, which refers to a path, such as '/', which is a global scope that refers to the entire application. In some cases, it might be useful to limit the scope to a specific part of the application, such a '/users', for example. [18.]

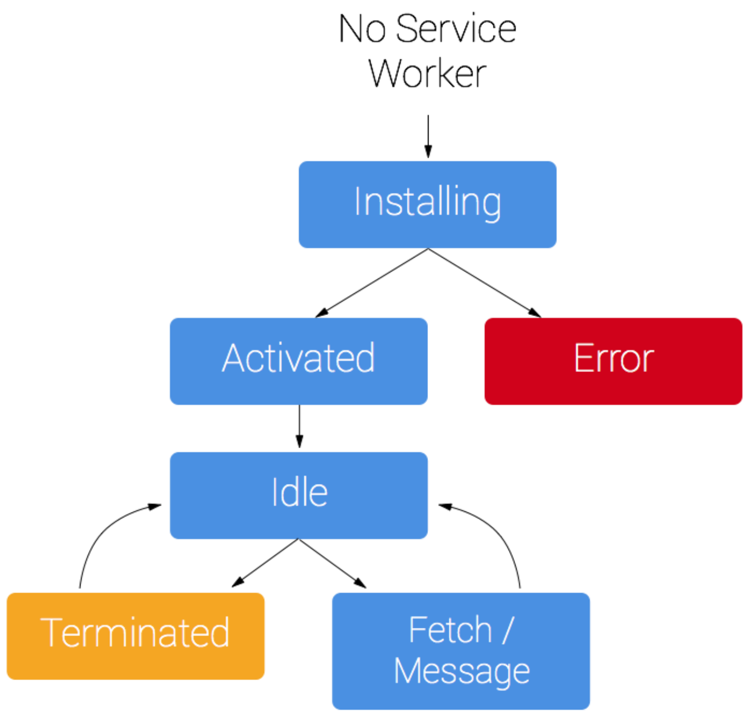


Image 5. Simplified service worker lifecycle [17].

After registration, the service worker is activated whenever the service worker is required. Service workers have two states: terminated and active. Service workers are idle until they are called upon: they react to *fetch* and *message* events. It is possible to register listeners for those events inside the service worker. [10.]

Registering and installing a service worker

For installing the service worker, it needs to be registered in the page in order to let the browser know where the file resides. Service worker API availability is checked, and if it's available, the service worker will be registered once the page has been loaded. [18.]

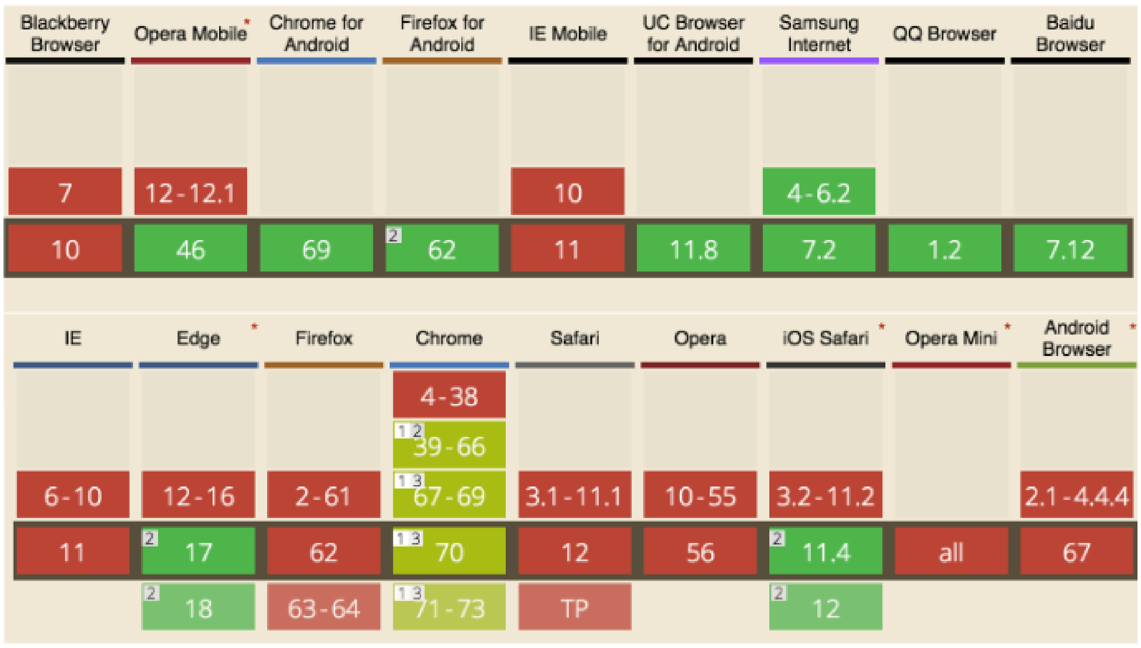
```
if ('serviceWorker' in navigator) {
  window.addEventListener('load', function() {
navigator.serviceWorker.register('/sw.js').then(function(registration) {
      // Registration was successful
      console.log('ServiceWorker registration successful with scope: ', regis-
tration.scope);
    }, function(err) {
      // registration failed :(
      console.log('ServiceWorker registration failed: ', err);
    });
  });
}
```

Listing 2. Service worker registration [18].

Service workers are updated when users visit the app if need be. The service worker is always redownloaded and if there are any differences, it will be updated. [19.]

3.2 Browser and platform support

Progressive Web Apps for mobile are currently supported in all major browsers: Chrome, Safari and Firefox [19]. Desktop support is still on its way, Chrome and Edge being the only browsers that currently support PWAs on desktop. Chrome desktop PWA support is still experimental and requires enabling a flag in *chrome://flags*.



Blackberry Browser	Opera Mobile *	Chrome for Android	Firefox for Android	IE Mobile	UC Browser for Android	Samsung Internet	QQ Browser	Baidu Browser
7	12-12.1			10		4-6.2		
10	46	69	[2] 62	11	11.8	7.2	1.2	7.12

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *
			4-38					
			[1 2] 39-66					
6-10	12-16	2-61	[1 3] 67-69	3.1-11.1	10-55	3.2-11.2		2.1-4.4.4
11	[2] 17	62	[1 3] 70	12	56	[2] 11.4	all	67
	[2] 18	63-64	[1 3] 71-73	TP		[2] 12		

Image 6. Manifest.json browser support according to caniuse.com [19].

Progressive Web Apps can be installed on a desktop computer just like in the mobile devices, and they can be run from the same place as other native software. And just like on mobile, they look and feel a lot like native applications. They run in a native app window instead of a browser window.

3.3 Push notifications

Push notification is a feature that has been available on native mobile and desktop apps for a long time now and now they are available in web apps too, ever since Chrome version 52 was released. Notification is a small popup message outside the browser in the user's device and they help keeping the user engaged even when they are not using the app.

Push notifications take advantage of two APIs: Push API, which is needed to handle push messages inside the service worker and Notifications API, which is an interface for the notifications themselves. Push notifications always need a permission from the user. If declined, no push notifications from that application can be shown in the user's device. Service workers are needed for listening the incoming push messages as they remain functional even when the app itself is closed. When the service worker receives a message, it displays the notification and then goes back in the idle state. Notifications can be interacted with by either dismissing the notification or acting on it if it has any custom actions defined. Service worker handles these interactions as well. [20.]

3.4 Benefits of Progressive Web Apps

Reach and engagement

Web is the biggest platform and with billions of connected devices it has the biggest reach. The number or monthly unique visitors in top 1000 applications is three times greater in the web than in mobile apps; number of unique mobile app users in 2016 was 4 million, whereas mobile web had 11,4 million users [21]. Study shows that on average, users install zero apps per month, but they visit around a hundred websites [22]. However, web apps are not as engaging as mobile apps and users are not that likely to revisit websites. Average time spent is significantly higher on mobile apps than on web apps, and users keep coming back to use the applications they have installed. Native functionalities, like push notifications keep users engaged even when the app is not in the foreground. Ideally, the reach of web and engagement on mobile could be combined; this is exactly what Progressive Web Apps are designed to do. [21.] The threshold for users to install and use a native application is very high compared to web apps that can be easily accessed in the browser.

Conversion rate is a metric used for tracking conversions in websites; the higher the better. Conversions are defined as valuable actions for business a user has taken after interacting with the website, or an advertisement. Valuable actions could be for example making a purchase in a web store or subscribing to a newsletter. They are calculated by dividing the number of conversions by the number of sessions. [23.] In 2016 AliExpress implemented a PWA and increased conversion rates by 104 % and 82 % in iOS - even though there was not yet PWA support in Safari. Number of pages visited doubled and time spent increased over 70 % in all browsers. AliExpress took advantage of features such as ability to work offline and focused on good performance. [24.]

Data consumption and performance

Connectivity is an everlasting issue in the web. Data might come very expensive in some places, especially so when traveling to another country. Poor connections usually lead to poor experiences. Service workers, App Shell architecture and caching improve this, resulting in a more reliable and fast application that consumes considerably less data. For example Konga, a leading Nigerian e-commerce, implemented a Progressive Web App in 2016 and managed to reduce as much as 92 % for initial loads when compared to native application [25].

Bounce is an analytical term for sessions where the users visit the site once and then leave, i.e. bounce. Bounce rate tells how many of your users leave your site after the visit, and a high bounce rate is generally unwanted [26]. Performance of web apps is very important. Time is money, and long loading times increase bounce rates enormously - most users don’t want to wait more than 3 seconds for a page to load [22].

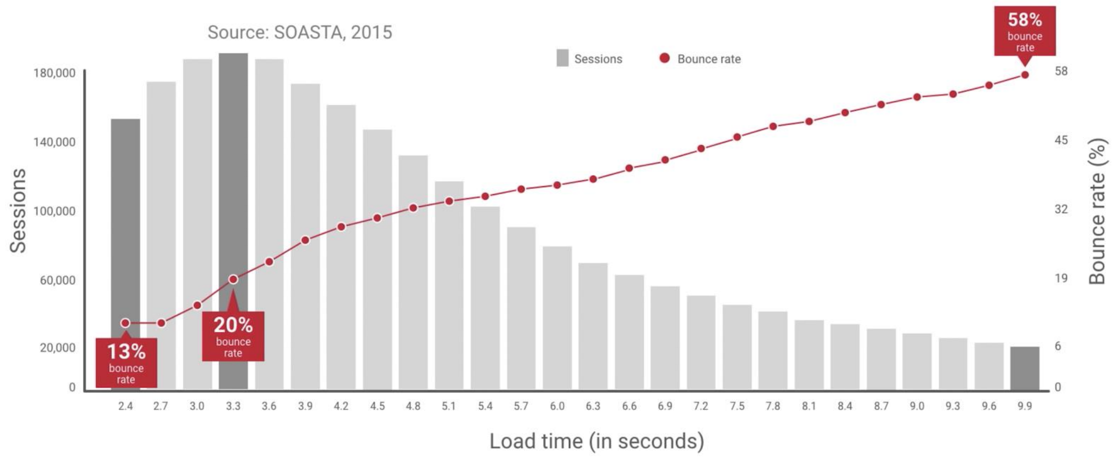


Image 7. Bounce rates are increased as the loading speed decreases [22].

Cross-platform support

There are numerous benefits in using PWA over native or regular web apps. PWAs provide an effortless way to implement cross-platform applications, decreasing development costs and reducing the number of codebases to maintain to just one. In addition to regular desktop and mobile environments, PWAs work perfectly in other contexts too, such as Chrome OS or desktop docks for smartphones, such as Samsung DeX.

Chrome OS is an operating system developed by Google. It primarily runs on Chromebooks, and it is designed for users who use most of their time in the browser. Chrome OS is, in a sense, a browser-based system. It is not possible to install native applications, as the features are really stripped down to browser support. This makes it really fast compared to traditional desktop computers. [27]. PWAs are installable in Chrome OS - and assuming that it implements responsive design, it is a great solution in Chromebooks.

Samsung DeX (see image 8) is a dock that provides desktop experiences for Samsung Galaxy smartphones. Samsung DeX allows user to use a mouse, keyboard and a monitor when using the dock together with a Galaxy phone [28].

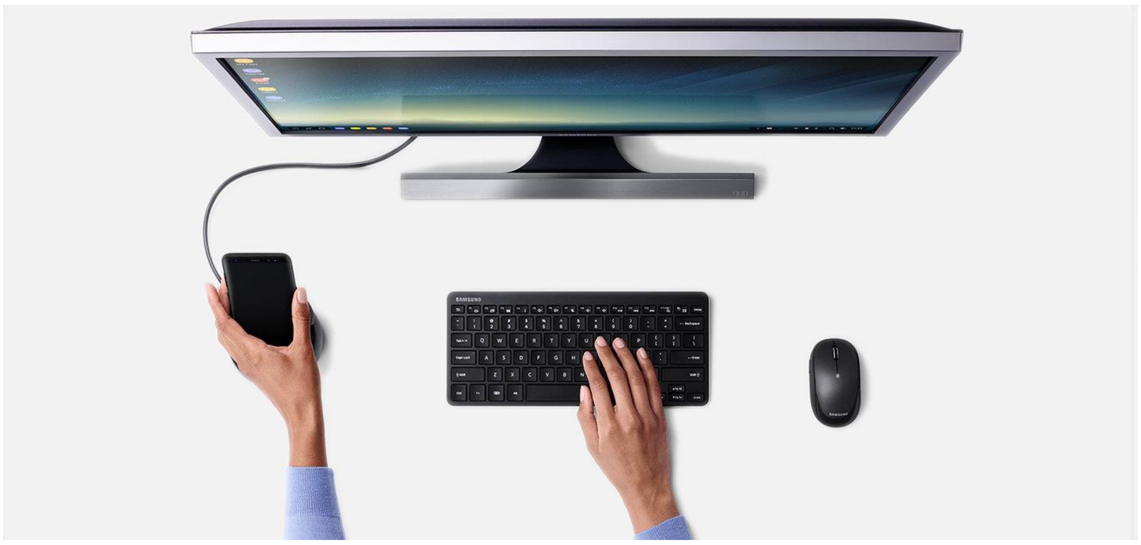
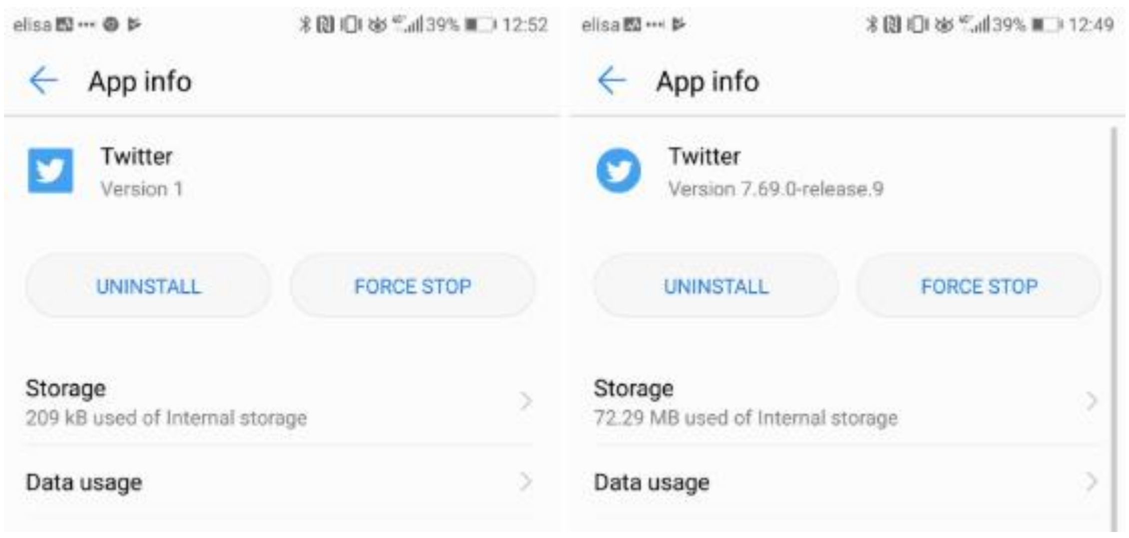


Image 8. Samsung DeX [28].

While the concept is innovative, native mobile applications are usually not designed to support desktop resolutions, leading to poor user experience. However, provided that PWAs is designed to be responsive and scalable for all the resolutions, they work seamlessly with Samsung DeX [21]. Even though smartphone docks are not yet that widely used, this is a great example of how versatile Progressive Web Apps are.

Storage

Progressive Web Apps can take significantly less space than their native counterparts. For example a Twitter app for Android takes over 70 MB of memory, whereas Twitter Lite PWA only takes up 208 kB (see image 9). That means less than 1 % of the needed storage space compared to native app.



Twitter
Version 1

Storage
209 kB used of Internal storage

Data usage

Twitter
Version 7.69.0-release.9

Storage
72.29 MB used of Internal storage

Data usage

Image 9. Twitter storage space, PWA installation details on the left and native Android application on the right.

To summarize some of the benefits of Progressive Web Apps, they:

- have better reach than mobile apps,
- are more engaging and native-like than regular web apps,
- perform better than web apps,
- consume less data and use less local storage,
- have the ability to work offline,
- work across different platforms,
- can be delivered more conveniently than native apps,
- are able to utilize certain native functionalities like push notifications.

4 Implementation of a Progressive Web App

4.1 Introduction

The purpose of the application is to provide a UI for tracking working time. The planned launch date for production was first of January 2019. The first version of the application was developed as an MVP (minimum viable product) due to a tight schedule.

Working time tracker needed to be able to save working time as entries where one entry represents one task done. One piece of work, for example programming a feature, meeting or writing documentation, would be logged as an entry with information such as description, selected task from predefined list, start time and end time. Logging working time should be made easy so that they would be as accurate as possible, and no working time would be lost trying to make sense in complicated work hour logs.

Even though this thesis is focused on development of Progressive Web Apps, the MVP requirements for this application did not include any mobile optimization or PWA functionalities. Minimum requirements for first working version included just a possibility to input, edit and save hour entries in the browser. Nevertheless, some PWA features were tested and included in the early implementation for research and learning purposes.

Backend solution was implemented separately from this project and was not included in the scope of this thesis. Some of the required functionalities could not be fully implemented as the web app development was pending on issues in the backend.

4.1.1 User stories

User stories are an agile way of writing down software requirements but instead of listing the features from a technical perspective, the requirements should be written from the perspective of the end user. The requirements are written in a non-technical way outlining the end goal or desired outcome of the task with a few sentences. The purpose is to describe the value for the end user (or another relevant party, like an administrator). This helps the developers to better understand the meaning of the feature and giving better time estimations. [29.]

User stories were used in this project to define the functionalities that needed to be completed for a minimum viable product.

Must have for launch:

- As a user, I need to be able to login.
- As a user, I need to be able to logout.
- As a user, I want to navigate in the site.
- As a user, I want to have a UI for inputting an entry.
- As a user, I want to be able to edit my hour entries in draft mode.
- As a user, I want to be able to delete my hour entries.
- As a user, I want to be able to save my inputted hour entries.

- As a user, I shouldn’t be able to edit the entries that are already saved.
- As a user, I can’t save invalid hour entries.

Could have features:

- As a user, I want to use the app while offline.
- As a user, I can pin the app to my home screen on mobile.
- As a user, I should get push notifications about unsaved entries.
- As a user, I can see a list of my available tasks.
- As a user, I can favorite a task.

4.2 Design

Based on the requirements, a couple wireframes and layouts were created using InVision (see images 10 and 11) and Marvel prototyping tools. Both tools allow quick prototyping and creating animated interactive prototypes.

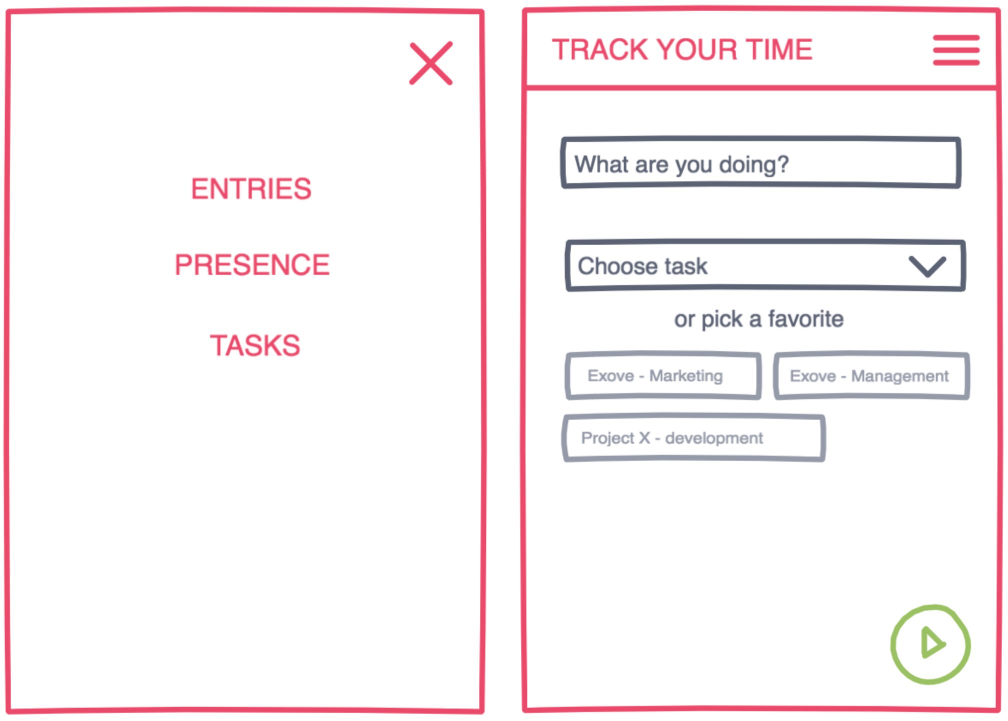
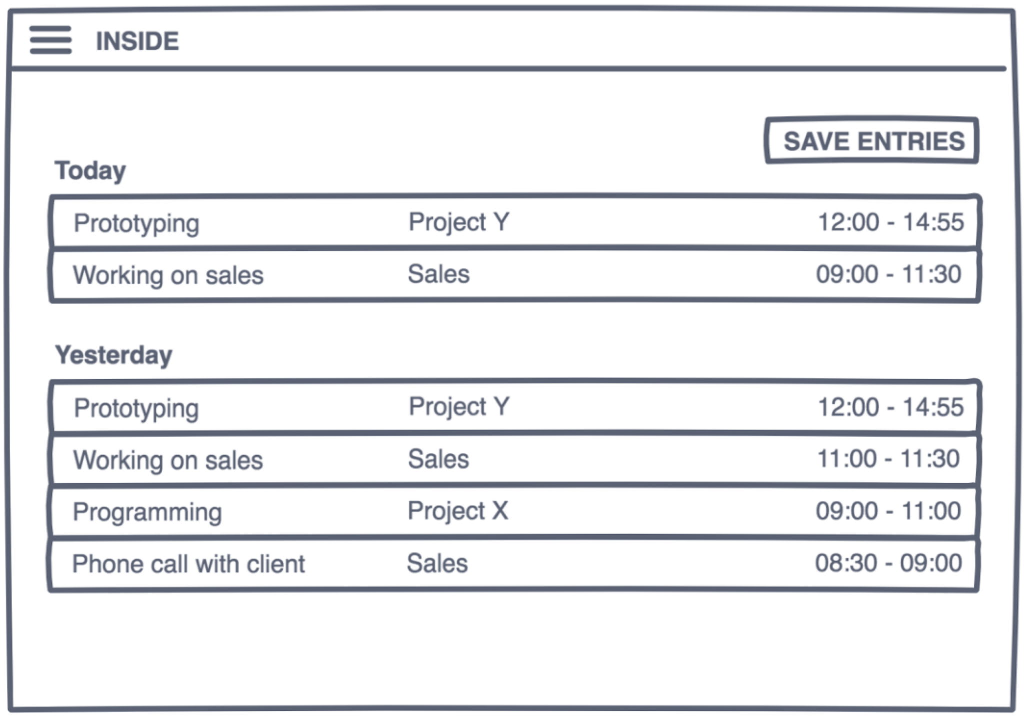


Image 10. Early mobile wireframe created with InVision.

First wireframes were low-fidelity prototypes for early stage concept testing. After creating the wireframes, interactive high-fidelity layouts were created with Marvel, providing more realistic user experience to that of the final release (see image 12). Marvel comes with a set of ready UI components and icons, making prototyping without any design software such as Sketch or Photoshop quite easy.



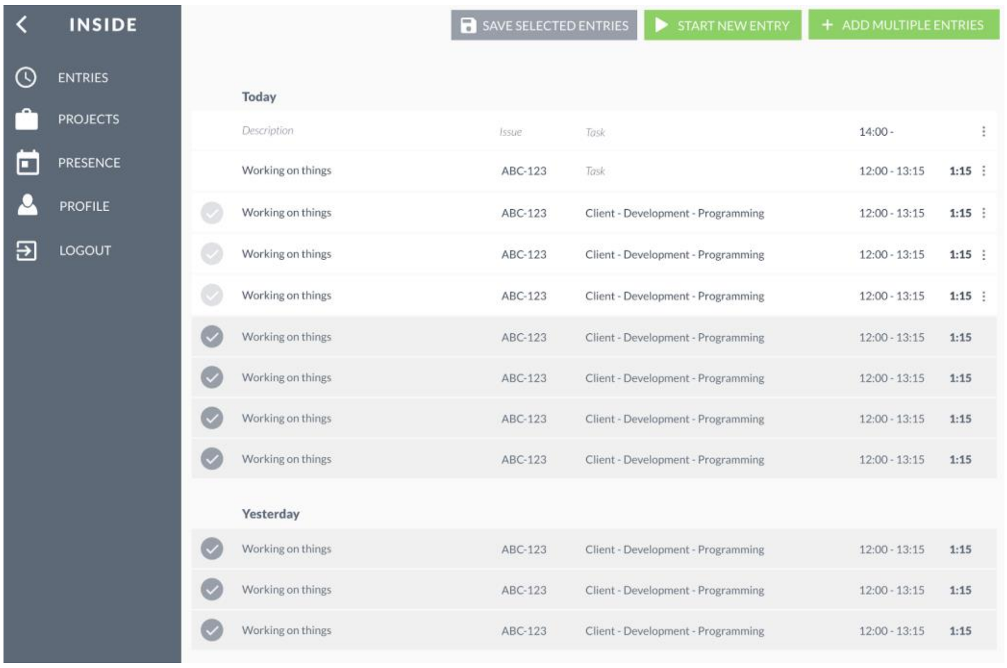
Today

Prototyping	Project Y	12:00 - 14:55
Working on sales	Sales	09:00 - 11:30

Yesterday

Prototyping	Project Y	12:00 - 14:55
Working on sales	Sales	11:00 - 11:30
Programming	Project X	09:00 - 11:00
Phone call with client	Sales	08:30 - 09:00

Image 11. Desktop wireframe created with InVision.



Today

Description	Issue	Task	14:00 -	
Working on things	ABC-123	Task	12:00 - 13:15	1:15
Working on things	ABC-123	Client - Development - Programming	12:00 - 13:15	1:15
Working on things	ABC-123	Client - Development - Programming	12:00 - 13:15	1:15
Working on things	ABC-123	Client - Development - Programming	12:00 - 13:15	1:15
Working on things	ABC-123	Client - Development - Programming	12:00 - 13:15	1:15
Working on things	ABC-123	Client - Development - Programming	12:00 - 13:15	1:15
Working on things	ABC-123	Client - Development - Programming	12:00 - 13:15	1:15
Working on things	ABC-123	Client - Development - Programming	12:00 - 13:15	1:15

Yesterday

Working on things	ABC-123	Client - Development - Programming	12:00 - 13:15	1:15
Working on things	ABC-123	Client - Development - Programming	12:00 - 13:15	1:15
Working on things	ABC-123	Client - Development - Programming	12:00 - 13:15	1:15

Image 12. Desktop layout created with Marvel.

The layouts designed (image 12) were not very accurate in the end, as they were created very early in the project and some requirements had changed since then.

4.3 Development

4.3.1 Development environment

The application was developed using a MacBook Pro. Node.js is a JavaScript runtime environment and npm is the default package manager for Node packages. Node.js version 10.11.0 and npm (Node package manager) version 6.4.1 were installed in the system. Visual Studio Code, a popular open-source text editor developed by Microsoft, was used as the main code editor. Project management and issue tracking were handled in Jira, a project and issues tracking software by Atlassian.

Version control was handled with git in Atlassian's Bitbucket. Feature branch git workflow was used during the development. In this workflow, all development happens in dedicated branches. After finishing a feature, a pull request is made from a feature branch to a master branch. Another developer reviews the code, and if there are no improvements to make or bugs to fix, pull request is accepted and merged to master. This ensures that master branch has no broken or ugly code and it makes it easier for multiple developers to contribute.

As the actual backend was still under development, there was a need for a mockup API that could be used to test API calls during application development. Postman is an API development environment and it is a useful tool for designing, mocking and testing APIs. It is available for macOS, Windows and Linux as a native application [31]. Postman was used for creating some mock API calls during the development while backend was being still developed.

4.3.2 Technologies used

The frontend app was built with React, a JavaScript library for building user interfaces. React is a frontend library and not a complete framework, as it only deals with the view layer of application in the MVC model (Model, View, Controller). It is a component-based framework which takes advantage of a Virtual Document Object Model (Virtual DOM), making it possible to update the interface without re-rendering the entire DOM whenever there is new data. This makes React applications very effective.

As such, React meets all the requirements set for the application. It has also been quite a popular framework at Exove, so it made sense to choose a framework that people around were familiar with in order to be able to make code reviews and also in case the project team would grow in the future. Additionally, according to Stack Overflow Developer survey React is in the top three most popular and loved technologies [29] and there seems to be a lot of demand for React in the job market. I also wanted to learn more about React myself, so choosing React as the framework was an easy choice to make.

Application was bootstrapped with Create React App (CRA). CRA is also created by Facebook and it comes with a preconfigured environment for both development and production. It includes React, JSX, ES6, TypeScript and Flow syntax support, unit test runner, live development server, build script bundling and it supports PWAs by default. Using CRA instead of having to configure environments by hand saved considerable amount of time. The project initially used a custom self-made webpack configuration, but that was later replaced with CRA to avoid all the hassle with configurations.

4.3.3 Application architecture

Application structure was mostly defined by CRA. Public folder includes the index.html file, and files inside the public folder are accessible from the public/index.html. Assets, such as icons, were placed inside the public folder. Index.html is the template, and index.js is entry point of the application. Tests were not created at all so the test file was left unused.

```
app root
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   ├── favicon.ico
│   ├── index.html
│   └── manifest.json
```

```
└── src
    ├── actions
    ├── components
    ├── containers
    ├── reducers
    ├── App.css
    ├── App.js
    ├── App.test.js
    ├── index.css
    ├── index.js
    └── serviceWorker.js
```

Listing 3. Application folder architecture.

All the React components were included in subfolders inside src: actions, components, containers and reducers. The application consists of six different components, illustrated in image 13.

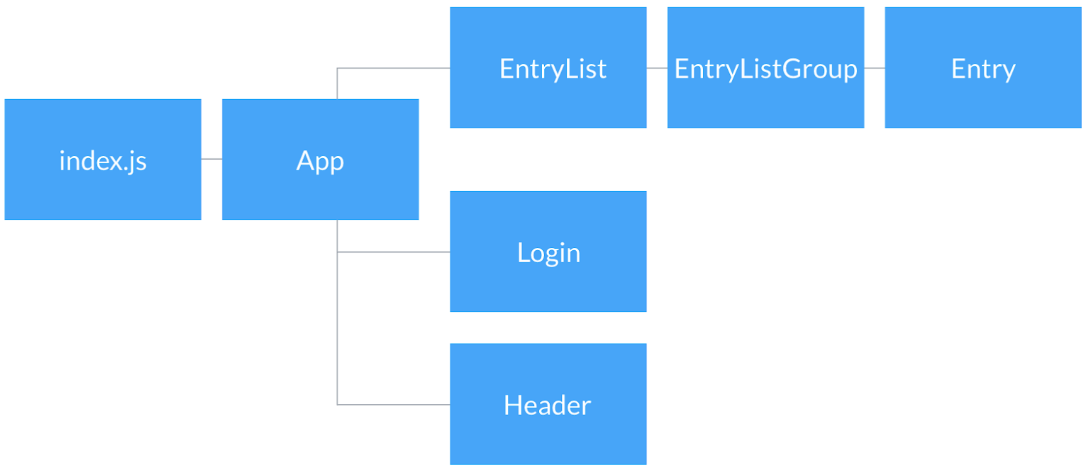


Image 13. React app component architecture.

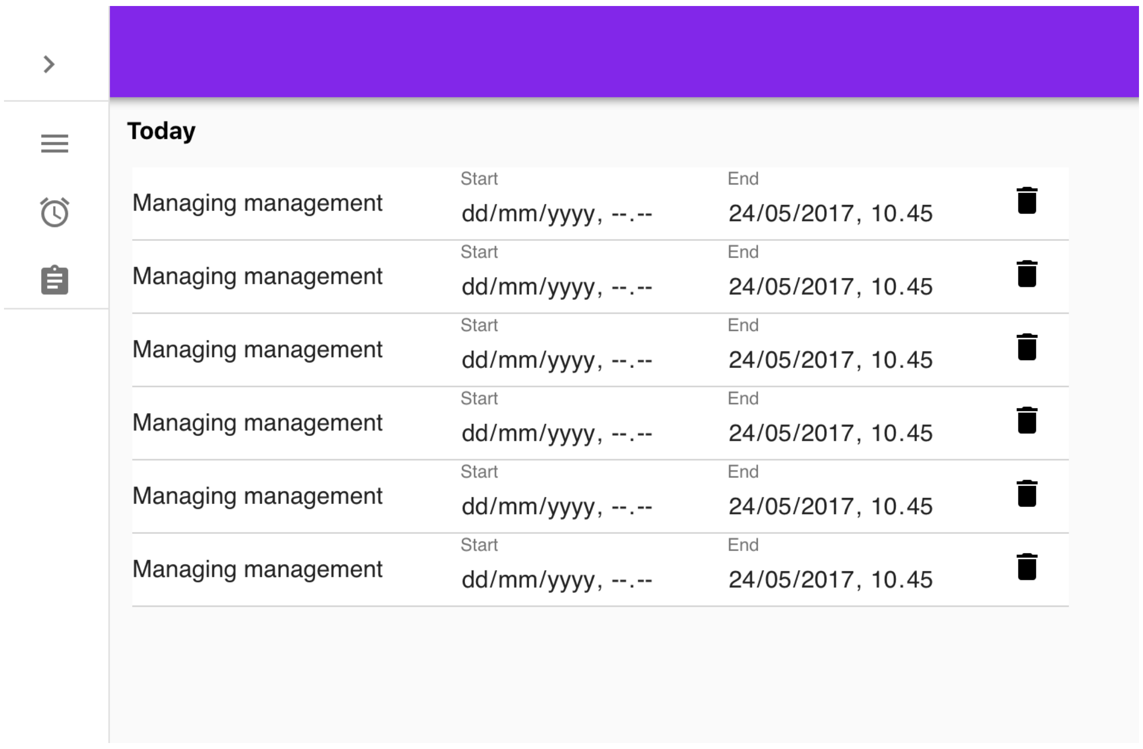
App component takes care of routing and wrapping the application. Navigation and app header are included in the Header component which is always visible. EntryList, EntryListGroup and Entry are used for creating, fetching and rendering time entries. Component for logging in was created but was not functional as it was pending on server-side implementation.

Material UI

Material-UI is an open source React UI framework that implements Google’s Material Design and it has a large number of built-in components [32]. It looks relatively good by default, allowing faster prototyping and development without having to focus on making the front-end appearance. Default theme was customized (listing 4) and part of the user interface created with Material-UI can be seen in image 14.

```
const theme = createMuiTheme({
 palette: {
   primary: {
     main: '#8f00f2'
   },
   secondary: {
     main: '#ffdbeb'
   },
   error: {
     main: '#ff3c1b'
   }
 }
})
```

Listing 4. Color palette customization using MuiThemeProvider.



Today

	Start	End
Managing management	dd/mm/yyyy, --.--	24/05/2017, 10.45
Managing management	dd/mm/yyyy, --.--	24/05/2017, 10.45
Managing management	dd/mm/yyyy, --.--	24/05/2017, 10.45
Managing management	dd/mm/yyyy, --.--	24/05/2017, 10.45
Managing management	dd/mm/yyyy, --.--	24/05/2017, 10.45
Managing management	dd/mm/yyyy, --.--	24/05/2017, 10.45

Image 14. Part of the UI constructed using components and icons provided by Material UI. Colors were customized using MuiThemeProvider component. Purple primary color is used in the app bar.

Material UI allows customising the look and feel of the applications with themes, which specifies elements such as color palette, surfaces, shadows, typography and several other variables. MuiThemeProvider component was used for theme customisation with brand colors and fonts.

Routing

As opposed to traditional website file structure, index.html and bundle.js are the only served files and routing happens on the client side. JavaScript interprets the URL and decides which part of the markup it will render on the page. This is also where the name Single Page Application (SPA) is derived from: technically the users stay in the same page all the time. Reactjs doesn't have a built-in router, so React Router DOM was used to handle routing as shown in listing 5.

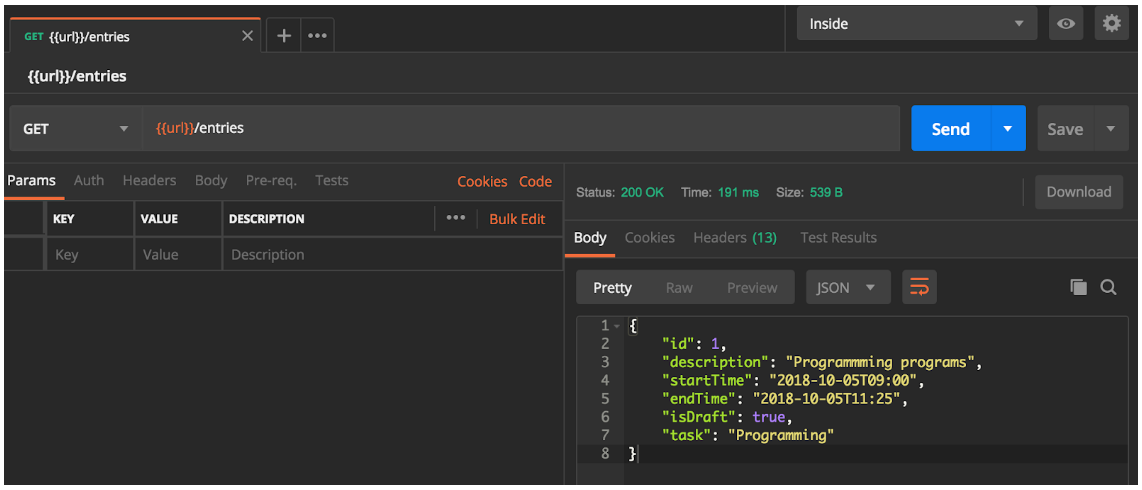
```
<BrowserRouter>
     <Switch>
        <Route exact path='/' component={Home} />
        <Route path='/entries' component={EntryList} />
        <Route path='/login' component={Login} />
     </Switch>
</BrowserRouter>
```

Listing 5. Code sample of routing application.

React Router is an npm package that provides routing functionality for both React and React Native. React Router is not usually installed directly, as React Router DOM is meant for applications that run in the browser and it comes with React Router as a dependency. React Router provides the core routing components and React Router DOM comes with components that work with the browser elements, such as BrowserRouter.

API endpoint

As the actual backend was still under development and there was no available API or even documentation about the endpoints that were going to be available, a mock server was created in Postman in order to simulate the endpoints and responses in a Postman Collection (image 15). Mock examples are very helpful especially during the early development phases allowing the project team to work in parallel without delays in front-end or back-end development.



```
{
    "id": 1,
    "description": "Programmming programs",
    "startTime": "2018-10-05T09:00",
    "endTime": "2018-10-05T11:25",
    "isDraft": true,
    "task": "Programming"
}
```

Image 15. Simplified example of a mock GET request response created in Postman.

The backend data model was ready, so based on the known properties some sample requests were created in the collection. For example, a GET request to */entries* endpoint would return an object with a list of all the saved entries with properties such as id, description, start time, end time and task name.

4.3.4 Implementing PWA features in a React app

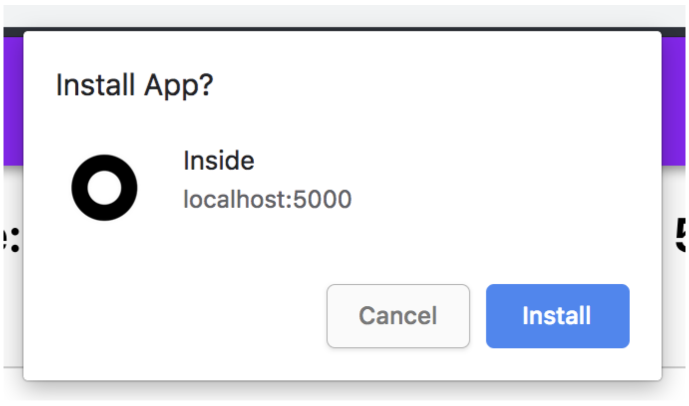
Meeting PWA requirements in React app created with CRA is fairly straightforward since it supports service workers by default and CRA applications come with a web manifest file and a simple service worker. Manifest for the application was customized as seen in listing 6.

```
{
 "short_name": "Inside",
 "name": "Inside",
 "manifest_version": 2,
 "icons": [
   {
     "src": "favicon.ico",
     "sizes": "64x64 32x32 24x24 16x16",
     "type": "image/x-icon"
   },
   {
     "src": "\/icon-144.png",
     "sizes": "144x144",
     "type": "image\/png"
   },
   {
     "src": "\/icon-192.png",
     "sizes": "192x192",
     "type": "image\/png"
   },
   {
     "src": "\/icon-512.png",
     "sizes": "512x512",
     "type": "image\/png"
   }
 ],
 "start_url": ".",
 "display": "standalone",
 "orientation": "any",
 "theme_color": "#ff3c1b",
 "background_color": "#fff"
}
```

Listing 6. Manifest JSON.

The default manifest file already contains the properties needed to meet PWA requirements except for icons. Manifest properties were customized with the application's own name, color scheme and icons as seen in the code snippet and it was good to go.

Service worker is not enabled by default but that was easily changed in the index.js file. As the PWA requirements were met, the app was installable on desktop (image 16) and mobile. The service worker included a script for creating a prompt for desktop users.


Install App?

Inside
localhost:5000

Image 16. Inside application's A2HS installation prompt in Chrome on macOS.

Push notifications were also implemented in the app but mostly for testing purposes (image 17). They are not providing any value to users so far, as the use cases had not been decided so final implementation was left for future development.

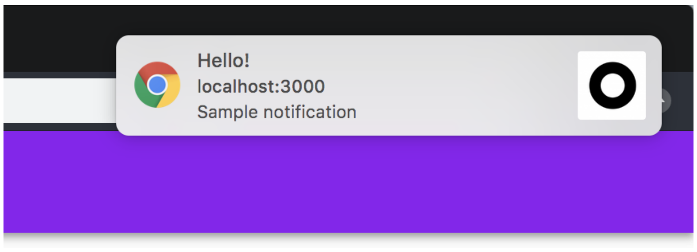

Hello!
localhost:3000
Sample notification

Image 17. Push notification in Chrome on macOS.

Implementing a push notification locally was quite simple. First, user permission was asked using the Notifications API. Notifications cannot be used without user granting the permission to do so.

```
Notification.requestPermission.then((result) => {
    console.log(result);
});
```

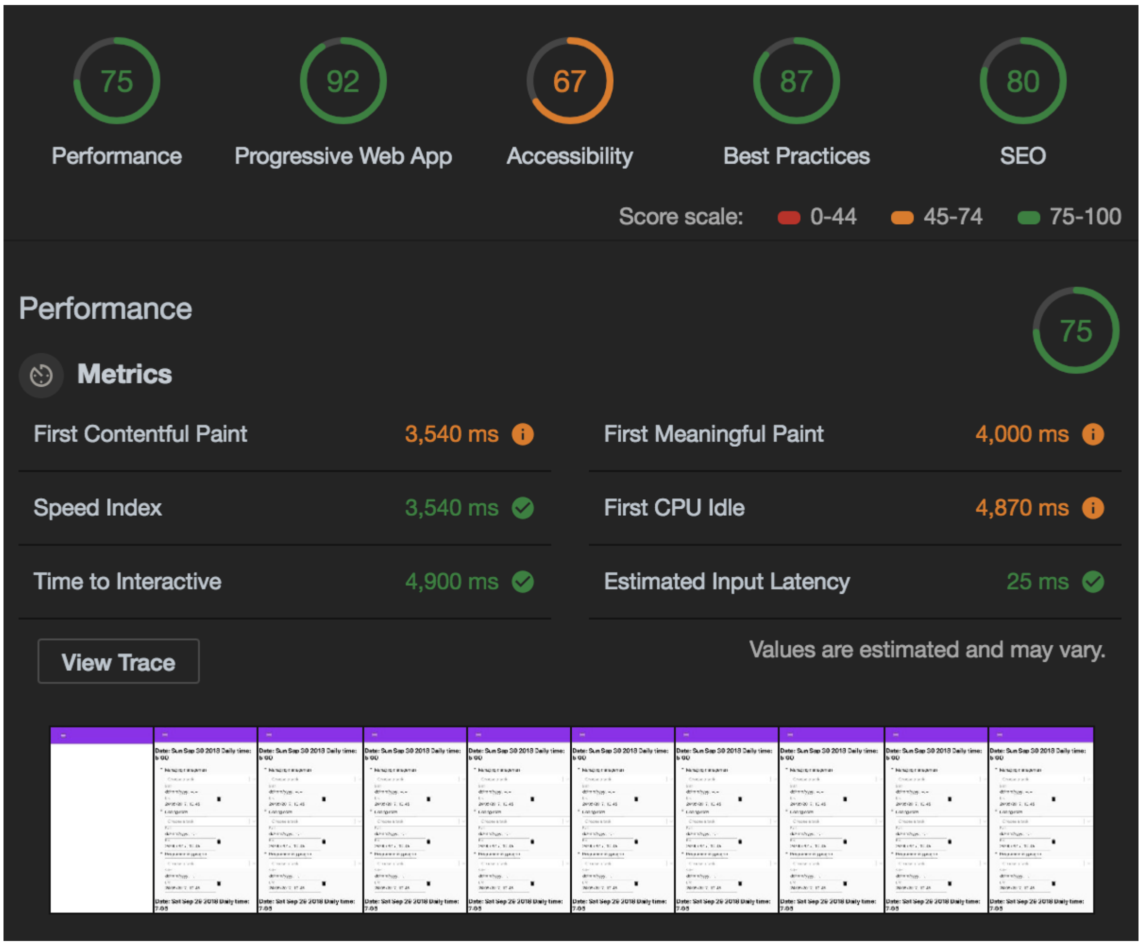
Listing 7. Asking user permission for showing push notifications and writing the result in console.

The notification itself was created inside the service worker with a notification constructor. Alternatively, the notification could be called on the service worker registration object with *showNotification()* function, passing the icon, title and body text as parameters to it. Notifications created with the constructor are not interactive as can be seen in image 16; there are no options for dismissing or acting on the message. Notifications created with the active service workers, however, can handle interactions with it since they are paired with the service worker, and the worker is then listening for it.

4.3.5 Testing

Code quality was reviewed with pull requests and linters were configured in the text editor to ensure consistency and reduce errors. The app was primarily tested on Chrome. It was also tested with an iOS Simulator in Xcode for a couple times. Testing in the staging web server with physical devices was not possible as it did not implement HTTPS, so testing PWA features was possible only on desktop having the applications locally installed. User testing was planned to start after the first launch.

Lighthouse is an open source developer tool for assessing websites. It is built in Chrome and its accessible via Chrome DevTools, command line or Node modules. Lighthouse generates a report based on a series of audits run against the page. It can measure performance, accessibility, best practices, PWA abilities and search engine optimization (SEO) and it gives each part a score and suggestions for improving the score if needed.



75 Performance · 92 Progressive Web App · 67 Accessibility · 87 Best Practices · 80 SEO

Score scale: 🟥 0-44 🟧 45-74 🟩 75-100

Performance 75

Metrics

First Contentful Paint	3,540 ms	First Meaningful Paint	4,000 ms
Speed Index	3,540 ms ✓	First CPU Idle	4,870 ms
Time to Interactive	4,900 ms ✓	Estimated Input Latency	25 ms ✓

View Trace

Values are estimated and may vary.

Image 18. Lighthouse report from Inside application.

As seen from the Lighthouse report, there are some improvements that could be made but general score is quite good. SEO is not an important metric for now as the app is not meant to be available for the general public and does not need to be indexed by search engines at all. All the Progressive Web App features were successfully implemented. The only concern was that the app was not automatically redirected from HTTP to HTTPS but that was not possible to achieve on localhost and will be solved in production environment once it gets there, which should then increase the score to 100.

Production builds were occasionally tested with an npm package called serve as the service worker worked only in production build, and not on localhost. Serve enables static pages and single page applications in localhost environments. Production builds were created with *yarn build* command, and the application was then served with *serve -s build*.

While testing a PWA on desktop, a few issues did arise. Firstly, since the app is essentially running in a browser, some browser extensions were behaving unexpectedly. For example, a bookmark extension still showed up like it would in a regular browser window but because it was running in the PWA window, clicking the bookmarks did not lead anywhere as seen on the left in image 19. Even though it was not a huge problem, it made the user experience somewhat clumsy and less native-like.

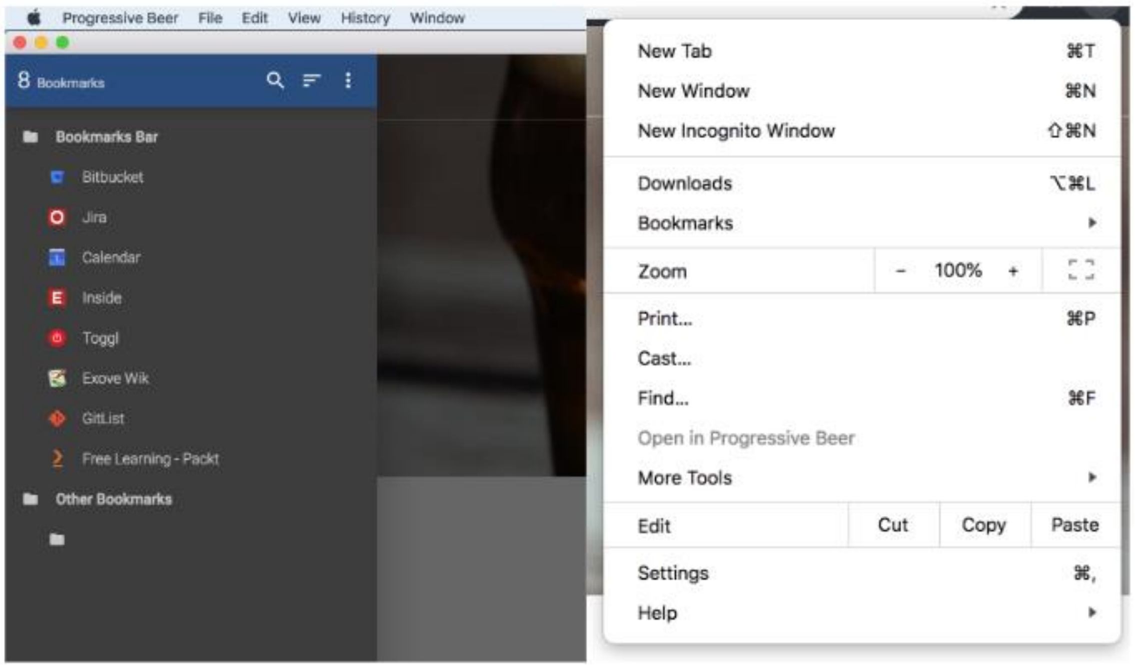


Image 19. Bookmark extension unexpectedly showing up in a PWA installed on macOS on the left and browser option window after a failed attempt on uninstalling the PWA on the right.

Deleting PWAs proved to be trickier than expected as well. Normally desktop applications on Mac can be deleted by moving the apps to trash can. However, while the app wasn’t usable anymore after having moved it and emptying the trash, it was not installable in the browser either as demonstrated on the right in image 19. The app needed to be removed at *chrome://apps* in the browser, which might feel counterintuitive to many users.

5 Conclusion

The goal of this thesis was to learn about Progressive Web Apps and develop a working time tracker app that would also implement some PWA features. Additionally, as a result of this research a training about Progressive Web Apps was also planned to be held at Exove.

Progressive Web Apps have a lot of benefits to then and they make better experiences in the web and implementing the required PWA functionalities is not very difficult. Considering how remarkable number of users use mobile devices to access the web, popularity of Progressive Web Apps will likely increase significantly. They will probably replace many native applications in the future, especially apps that don’t really need the native functionalities or hardware. However, Progressive Web Apps are not yet that well-known for end users, and browser support is lacking to some extent. PWAs rely only on the available web APIs, and while the number of them will most likely grow significantly in the future, this means that native apps still support a much wider range of features for the time being.

The application developed was a time tracking application that allowed users to keep track of their working time. Implementing the PWA functionalities in the app was simple enough and learning curve was not too high. Unfortunately due to the timeline and parallel development of both front-end and back-end it was not possible to get all the features fully functional; some mandatory functionalities such as user authorization were not implemented at all but the development will still continue.

All in all, Progressive Web Apps are a very interesting topic and a lot was learned from the research. They will probably be a significant part of the web in the future.

References

1 Wellens, Paul. 2015. [E-book]. Practical Web Development. Packt Publishing Limited.

2 LePage, Pete. [Online]. Responsive Web Design Basics. Google Developers. https://developers.google.com/web/fundamentals/design-and-ux/responsive. Updated 21.9.2018. Accessed 11.11.2018.

3 Swift.org - About Swift. [Online]. Apple Inc. https://swift.org/. Accessed 20.10.2018.

4 Launch checklist. [Online]. Google Developers. https://developer.android.com/distribute/best-practices/launch/launch-checklist Updated 22.5.2018. Accessed 10.10.2018.

5 Documentation - Apache Cordova. [Online]. The Apache Software Foundation. https://cordova.apache.org/docs/. Accessed 10.11.2018.

6 Build cross platform desktop apps with JavaScript, HTML, and CSS. [Online]. https://electronjs.org/. Accessed 19.10.2028.

7 Boduch, Adam. 2017. [E-book]. React and React Native. Packt Publishing Limited.

8 Chen, Fanghao. 26.6.2018. [Online]. Why Discord is Sticking with React Native. Discord. https://blog.discordapp.com/why-discord-is-sticking-with-react-native-ccc34be0d427. Accessed 23.10.2018.

9 Xamarin Documentation. [Online]. Microsoft. https://docs.microsoft.com/en-us/xamarin/. Accessed 1.11.2018.

10 Domes, Scott. 2017. [E-book]. Progressive Web Apps with React. Packt Publishing Limited.

11 Progressive Web Apps. [Online]. Google Developers. https://developers.google.com/web/progressive-web-apps/. Accessed 16.7.2018.

12 Basques, Kayce. [Online]. Why HTTPS Matters. Google Developers. https://developers.google.com/web/fundamentals/security/encrypt-in-transit/why-https. Updated 21.9.2018. Accessed 25.10.2018.

13 Add to Home Screen. [Online]. Mozilla. https://developer.mozilla.org/en-US/docs/Web/Apps/Progressive/Add_to_home_screen. Updated 24.10.2018. Accessed 29.10.2018.

14 Gaunt, Matt; Kinlan, Paul. The Web App Manifest. [Online]. Google Developers. https://developers.google.com/web/fundamentals/web-app-manifest/. Updated 18.10.2018. Accessed 20.10.2018.

15 LePage, Pete. [Online]. Add to Home Screen. Google Developers. https://developers.google.com/web/fundamentals/app-install-banners/. Updated 30.10.2018. Accessed 5.11.2018.

16 Osmani, Addy. [Online]. The App Shell Model. Google Developers. https://developers.google.com/web/fundamentals/architecture/app-shell. Updated 21.9.2018. Accessed 30.9.2018.

17 Gaunt, Matt; Osmani, Addy. [Online]. Instant Loading Web Apps with Application Shell Architecture. Google Developers. https://developers.google.com/web/updates/2015/11/app-shell. Updated 2.8.2018. Accessed 20.8.2018.

18 Gaunt, Matt. [Online]. Service Workers: an Introduction. Google Developers. https://developers.google.com/web/fundamentals/primers/service-workers. Updated 21.9.2018. Accessed 10.10.2018.

19 Can I use... Support tables for HTML5, CSS3, etc. [Online]. https://caniuse.com/#feat=web-app-manifest. Accessed 30.10.2018.

20 Introduction to Push Notifications. [Online]. Google Developers. https://developers.google.com/web/ilt/pwa/introduction-to-push-notifications. Updated 9.4.2918. Accessed 5.11.2018.

21 Progressive Web Apps: Great Experiences Everywhere (Google I/O '17). 17.5.2017. [Video]. Google Chrome Developers. https://www.youtube.com/watch?v=m-sCdS0sQO8. Accessed 20.8.2018.

22 Why Build Progressive Web Apps? 26.4.2017. [Video]. Google Chrome Developers. https://www.youtube.com/watch?time_continue=15&v=1QILz1lAzWY. Accessed 30.9.2018.

23 Conversion rate: Definition. [Online]. Google. https://support.google.com/google-ads/answer/2684489?hl=en. Accessed 9.11.2018.

24 AliExpress. [Online]. Google Developers. https://developers.google.com/web/showcase/2016/aliexpress. Updated 9.2.2017. Accessed 21.9.2018.

25 Konga. [Online]. Google Developers. https://developers.google.com/web/showcase/2016/konga. Updated 9.2.2017. Accessed 21.9.2018.

26 Bounce rate: Definition. [Online]. Google. https://support.google.com/analytics/answer/1009409?hl=en. Accessed 7.11.2018.

27 What is Google Chrome OS? 18.11.2009. [Video]. Google Chrome. https://www.chromium.org/chromium-os. Accessed 30.10.2018.

28 Samsung DeX. [Online]. Samsung. https://www.samsung.com/us/explore/dex/. Accessed 5.11.2018.

29 Rehkopf, Max. [Online]. User Stories. Atlassian. https://www.atlassian.com/ag-ile/project-management/user-stories. Accessed 8.11.2018.

30 Developer Survey Results 2018. [Online]. Stack Overflow. https://insights.stacko-verflow.com/survey/2018/. Updated 19.3.2018. Accessed 5.11.2018.

31 Postman | API Development Environment. [Online]. https://www.getpost-man.com/. Accessed 30.10.2018.

32 The world's most popular React UI framework - Material-UI. [Online]. https://ma-terial-ui.com/. Accessed 5.10.2018.