



From Native to Progressive Web App A Proof of Concept

Gunharth Randolph

1810738273

Entwicklung & Betrieb Mobiler Informationssysteme

Web Communication & Information Systems

29. July 2019

Abstract

Lorem ipsum

Contents

1	Introduction	2
2	Methods	4
2.1	Literature Search	4
2.2	Related Works	4
2.3	Technical Implementation	5
3	Concepts & Technical Implementation	6
3.1	Requirements	6
3.2	Technical Implementation of Requirements	7
3.2.1	Single Page App (SPA) & Progressive Web App	8
3.2.2	Cloud Provider and Realtime Functionalities	12
4	Results	17
5	Conclusion	18

1. Introduction

In 2015 the Google Developer Alex Russel first drafted the term Progressive Web Apps (PWAs) and formulated the concept of PWAs as it stands today.

Full Progressive App support will distinguish engaging, immersive experiences on the web from the “legacy web”. Progressive App design offers us a way to build better experiences across devices and contexts within a single codebase but it’s going to require a deep shift in our understanding and tools. (Russell, 2015)

Since then, notably Google and its Chrome browser focus on the implementation of all necessary technologies in order to fully support PWAs. Nearly every new release offers the Chrome browser adds further support for PWAs. As an example, at the time of writing Chrome 76 shows an install icon to the left of the address bar, if a site meets the Progressive Web App installability criteria¹.

So far, the bedrock for engaging and immersive experiences on mobile devices has been bound to the development of native and/or hybrid apps. However, developing a native application is an expensive solution. Programmers spend years learning to code heavy native apps and app owners usually invest a huge budget finding a development team (Nguyen, 2019). That said, at least two versions of an app need to be developed, maintained and published to the Apple and Google App Stores respectively.

As PWAs are still a rather new concept there are known limitations in terms of unified browser support of APIs², missing filesystem access and platform-level features integrations like calendar and contacts (Biørn-Hansen et al., 2018;

¹<https://developers.google.com/web/fundamentals/app-install-banners#criteria>

²<https://whatwebcando.today>

Malavolta, 2016). However, there are many applications that do not depend on the before mentioned limitations. On 15th July 2019 Twitter announced its new website, which is a PWA build on one code base feeding all devices and browsers that are accessing the site on the Web. The goal for the new site was to make it easier and faster to develop new features for people worldwide and to provide each person and each device with the right experience (Croom and Baker, 2019).

The purpose for this study is to look at the technical side of PWAs, whereby the following questions will be answered. First, whether it is possible to take an existing mobile app as a prototype and re-build the app as a PWA. Second, what services and programming languages are best fitted in doing so and how well do they support the development of a PWA? Finally, I discuss my findings and any limitations I might have encountered.

2. Methods

This study implements three research methods to gain a broader understanding of the possibilities, state of research and technological practices regarding Progressive Web Apps.

2.1 Literature Search

Our searches for academic involvement in Progressive Web Apps returned a very limited amount of results as per July 2017 ([Biørn-Hansen et al., 2018](#)). The same search performed two years later doesn't reveal any further major releases on this topic. Scientific research is still pretty sparse. However, technology related publishers have picked up the topic with a handful of publications in 2017 and 2018 ([Hume and Osmani, 2018](#); [Ater, 2017](#)). In 2019 though, outlining the latest advances on the technology side of PWAs, there only seems to be one notable publication: a practical guide to PWAs by Liebel written in the German language ([Liebel, 2019](#)).

Interestingly, I was not able to find any scientific research on the commercial aspects of using PWA. A comparison on budgets and running costs required by native/hybrid apps versus a PWA. As noted in the conclusion of this study this leaves room for future research.

2.2 Related Works

The Google Web Fundamentals group acts as the driving force behind the documentation on PWA related topics and the creation of blog posts and tutorials.

As the supporting technologies advance with every Chrome browser release, the Web Fundamentals website¹ is the foundation for upcoming research and studies.

Other than Google-created content, online platforms such as Medium blog posts² and Youtube tutorials³ lay the foundation for the technical research of this study. Notably, the technical series of articles about PWA published by Eder Ramírez Hernández⁴ on Medium were extremely informative.

2.3 Technical Implementation

To gain a better understanding of the possibilities of Progressive Web Apps a PWA was developed. The motivation for the PWA was to take an existing native application as a reference and to re-create the application as a Progressive Web App. For this purpose I picked the *Beer With Me*⁵ application, which is available on the Apple App Store as well as on Google Play. As stated by the Swedish developers Antonsson, Knutsson and Tidbeck "Beer With Me is a social application to notify your friends when you are drinking so that they can join".

¹<https://developers.google.com/web/fundamentals>

²<https://medium.com/search?q=Progressive%20Web%20Apps>

³https://www.youtube.com/results?search_query=Progressive+Web+Apps

⁴<https://medium.com/@eder.ramirez87>

⁵<https://beerwithme.se/>

3. Concepts & Technical Implementation

This chapter provides a detailed look at the required concepts and the technical implementation of the PWA. The project has been open-sourced to allow verification of the results¹. Further, the application is available online².

3.1 Requirements

The following section outlines the concepts and requirements used for the project by looking at the core functionality of the original *Beer with Me* app.

Progressive Web App. The functional clone of the *Beer with Me* app must implement all relevant techniques as outlined by the Progressive Web App specifications. Technologies include Service Workers, Application Shell (AppShell), Web App Manifest and serving the demo installation over HTTPS. Further, it is the intention to follow the 10 principal concepts, which lay the foundation for Progressive Web Applications, as closely as possible: Progressive, Responsive, Installable, Connectivity Independent, App-Like, Fresh, Safe, Discoverable, Re-engageable and Linkable (Osmani, 2015).

Authentication and Authorization. Users must be able to register for the service with a combination of E-Mail and Password. As an alternative an existing Google account may be used in order to login directly.

Realtime Database. Data is stored in a cloud-hosted database and synchronized in realtime to every connected client.

¹<https://github.com/gunharth/bwm>

²<https://bwm.gunicode.com>

Realtime Notifications. If enabled by the user push-notifications shall be send to the user in realtime, whenever a new user registers for the service or when a new post was published by a user.

Realtime Map. If enabled by the user his/her current geolocation is saved and displayed on an online map.

Single Page App (SPA). The app shall be implemented as a Single Page App with one of the main JavaScript frameworks (React, Angular or Vue.js).

An additional requirement was intentionally added to round-off PWA specific topics. When posting a new entry a user should be provided with the option to post a photo using the camera of the device.

The following topics were defined as being not part of the requirements, as they do not have an impact on the used technologies of the PWA, nor do they have an effect on the results of this study. Hence, design considerations of the application can be neglected, nor is it the intention to copy the interface and design of the original *Beer with Me* app. The resulting PWA focusses on the implementation of the functionality and not on offering a identical clone of the original. Further, all functions are implemented as a proof-of-concept following the MVP spirit ([Wikipedia, 2019](#)). Thus, to make it a user centric production ready PWA further development cycles are required, i.a creating social groups and multi-lingual support.

3.2 Technical Implementation of Requirements

As per the actual implementation of the PWA the goal was to base the solution on the least different providers on the one hand, as well as a minimum in terms of coding and programming languages. Hence, the frontend of the application uses the basic building blocks of regular web sites being HTML, CSS and JavaScript in form of the JavaScript framework Vue.js. To satisfy the backend Firebase was picked as the sole solution provider.

3.2.1 Single Page App (SPA) & Progressive Web App

For the development of the frontend the JavaScript framework Vue.js³ and its ecosystem was used. The Vue Cli⁴ offers great tooling support for Vue.js developments - among other features it comes with a preset for installing a Progressive Web App skeleton.

Listing 1: Installation and project creation commands with the Vue Cli

```
1 $ npm install -g @vue/cli
2 $ vue create beerwithme
```

On project creation Vue Cli offers you to manually select the features that will be installed. Next to Progressive Web Application support the Vue Router and Vuex was installed. The Router enables navigation services within a SPA and Vuex is responsible for the management of state within the application.

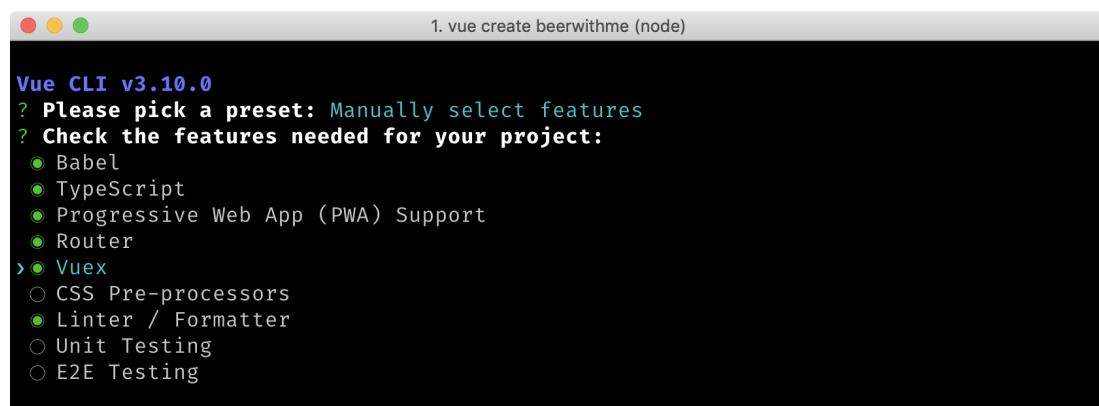


Figure 1: Installation options for a Vue.js project

The Vue Cli PWA support comes with Workbox⁵ installed - a library developed by Google for adding offline support to web applications. Among other features it mainly helps with the automatic creation and management of application pre-cache and the application shell (or app shell)⁶.

³<https://vuejs.org>

⁴<https://cli.vuejs.org>

⁵<https://developers.google.com/web/tools/workbox>

⁶<https://developers.google.com/web/fundamentals/architecture/app-shell>

Listing 2: Service Worker with Workbox and Firebase specific initiation (firebase-messaging-sw.js)

```
1  importScripts("https://www.gstatic.com/firebasejs
    /5.6.0/firebase-app.js");
2  importScripts("https://www.gstatic.com/firebasejs
    /5.6.0/firebase-messaging.js");
3
4  self.__precacheManifest = [].concat(self.
    __precacheManifest || []);
5  workbox.precaching.suppressWarnings();
6  workbox.precaching.precacheAndRoute(self.
    __precacheManifest, {});
7
8  workbox.routing.registerRoute(
9    new RegExp(
10     "https://firebasestorage.googleapis.com/v0/b/
        bmwgunicode.appspot.com/.*"
11    ),
12    workbox.strategies.staleWhileRevalidate()
13  );
14
15  firebase.initializeApp({
16    messagingSenderId: "ID"
17  });
```

For the frontend design of the app Vuetify⁷ was used. Vuetify is a responsive design component framework for Vue.js based on the material design⁸ guidelines developed by Google.

Listing 3: Command to add Vuetify to a Vue.js project

```
1  $ vue add vuetify
```

The map functionality was implemented by using the NPM (Node Package Manager⁹) package vue2-leaflet¹⁰, which is a Vue wrapper library for the open-source JavaScript library Leaflet¹¹. Map tiles are served from OpenStreetMap¹².

⁷<https://vuetifyjs.com>

⁸<https://material.io>

⁹<https://www.npmjs.com>

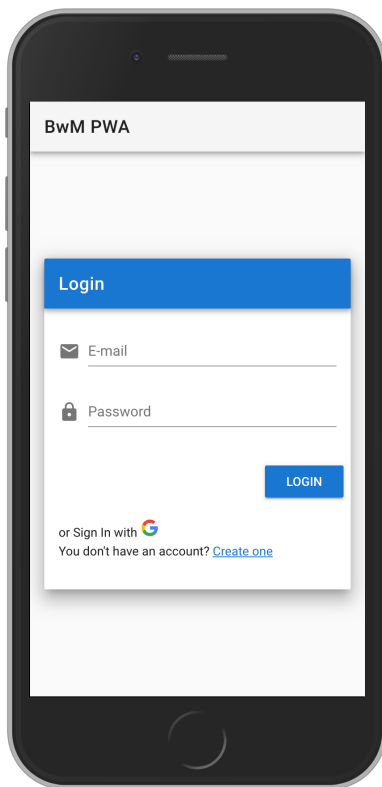
¹⁰<https://www.npmjs.com/package/vue2-leaflet>

¹¹<https://leafletjs.com>

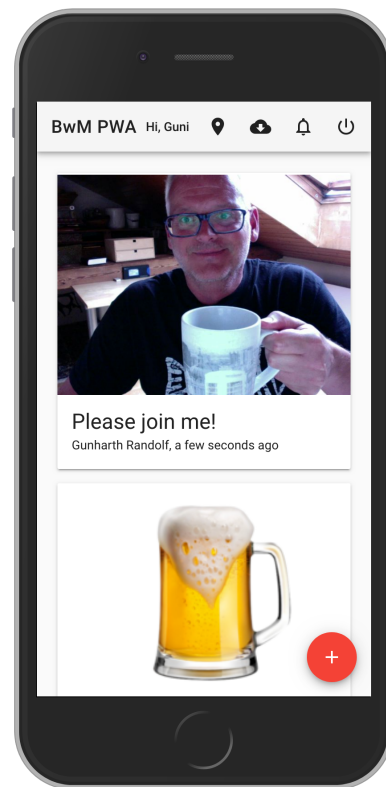
¹²<https://www.openstreetmap.org>

Listing 4: All final npm dependencies in packages.json

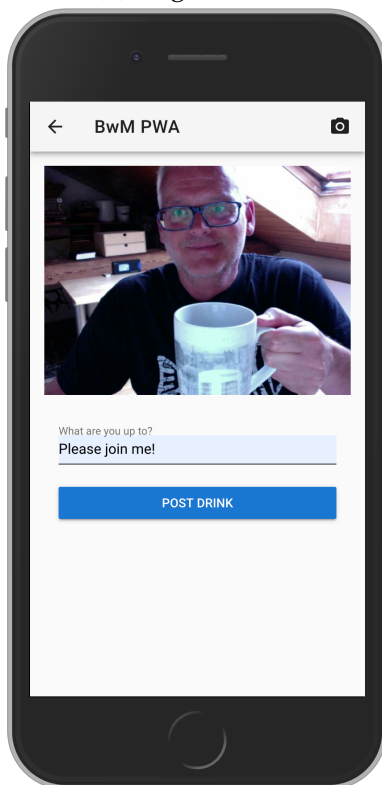
```
1  "dependencies": {  
2    "axios": "^0.19.0",  
3    "core-js": "^2.6.5",  
4    "firebase": "^6.2.2",  
5    "leaflet": "^1.5.1",  
6    "material-design-icons-iconfont": "^5.0.1",  
7    "register-service-worker": "^1.6.2",  
8    "vue": "^2.6.10",  
9    "vue-moment": "^4.0.0",  
10   "vue-router": "^3.0.3",  
11   "vue2-leaflet": "^2.1.1",  
12   "vuetify": "^1.5.5",  
13   "vuex": "^3.0.1"  
14 },
```



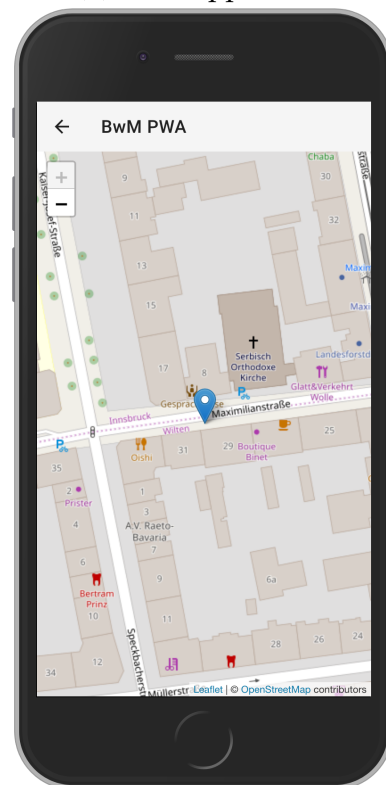
(a) Login screen



(b) Main app screen



(c) New post screen with photo option



(d) The map

Figure 2: Frontend screenshots showing main app screens, Vuetify material design components and the map.

Figure 3 shows the Lighthouse¹³ Progressive Web App report for the *Beer with Me* PWA.

















Progressive Web App			
These checks validate the aspects of a Progressive Web App. Learn more.			
	Fast and reliable		
1	Page load is fast enough on mobile networks		✓
2	Current page responds with a 200 when offline		✓
3	start_url responds with a 200 when offline		✓
	Installable		
4	Uses HTTPS		✓
5	Registers a service worker that controls page and start_url		✓
6	Web app manifest meets the installability requirements		✓
	PWA Optimized		
7	Redirects HTTP traffic to HTTPS		✓
8	Configured for a custom splash screen		✓
9	Sets an address-bar theme color		✓
10	Content is sized correctly for the viewport		✓
11	Has a <meta name="viewport"> tag with width or initial-scale		✓
12	Contains some content when JavaScript is not available		✓

Figure 3: Lighthouse report

3.2.2 Cloud Provider and Realtime Functionalities

For the backend I picked Firebase¹⁴ as the provider. Around the same time in 2015, when the term Progressive Web Apps was born, Google acquired this San Francisco start-up. Since then Google has expanded upon the original service and integrated it closely into their other cloud ambitions - whereby the Firebase service is dedicated to providing a specialized Platform as a Service (PaaS) for mobile and web applications.

Firebase offers a javascript API which is installed through NPM. The Firebase CLI enables development directly from the command line. All Firebase services

¹³<https://developers.google.com/web/tools/lighthouse>

¹⁴<https://firebase.google.com>

can be configured and maintained through the CLI.

Listing 5: Firebase configuration and services (firebaseConfig.js)

```
1 import firebase from "firebase/app";
2 import "firebase/firestore";
3 import "firebase/messaging";
4 import "firebase/storage";
5 import "firebase/auth";
6 import { config } from "@/firebaseCredentials.js";
7
8 firebase.initializeApp(config);
9 let db = firebase.firestore();
10 db.enablePersistence({ synchronizeTabs: true });
11 const storage = firebase.storage();
12 let messaging = null;
13 if (firebase.messaging.isSupported()) {
14   messaging = firebase.messaging();
15 }
16 const auth = firebase.auth();
17 const socialAuth = firebase.auth;
18 export default {
19   db,
20   storage,
21   messaging,
22   auth,
23   socialAuth
24 };
```

The following section demonstrates the Firebase services used for the *Beer with Me* PWA.

Authentication and Authorization. Firebase Auth offers multiple methods to authenticate, including email and password and third-party providers like Google. It integrates tightly with other Firebase services, and it leverages standards like OAuth 2.0 and OpenID Connect.

Listing 6: Firebase Auth initiation using VueJS (main.js)

```
1 // Init Firebase auth before Vue inits the App
2 firebase.auth.onAuthStateChanged(firebaseUser => {
3   if (firebaseUser) {
4     store.dispatch("autoSignIn", firebaseUser);
```

```
5     }
6     if (!app) {
7         app = new Vue({
8             router,
9             store,
10            render: h => h(App)
11        }).$mount("#app");
12    }
13    });
```

Realtime Database. Firebase offers a realtime Database service with Firestore, which stores and syncs data between users and devices using a cloud-hosted, JSON based NoSQL database. An important feature of Firestore in terms of PWAs is offline support and live synchronization.

Listing 7: Realtime query for new posts (Home.vue)

```
1  firebase.db
2    .collection("drinks")
3    .orderBy("created_at", "desc")
4    .onSnapshot(snapshot => {
5        this.drinks = [];
6        snapshot.forEach(drink => {
7            this.drinks.push({
8                id: drink.id,
9                url: drink.data().url,
10               comment: drink.data().comment,
11               author: drink.data().author,
12               created_at: drink.data().created_at,
13           });
14       });
15   });
```

Photos taken by the user are stored using the Firebase Cloud Storage service, a CDN (Content Delivery Network) driven service by Google tightly integrated with the authentication service. At present there is no offline support available, i.e photos taken offline are not synced to the storage once there is a connection. One solution to this caveat is to save the photo as a base64 encoded string into the Firestore database. Once the actual database query runs Firebase Functions converts the string to an actual image, stores the file to the Firebase Cloud Storage and saves the resulting image reference back into Firestore.

Listing 8: Store a photo to Firebase Cloud Storage (Camera.vue)

```
1 firebase.storage
2   .ref()
3   .child('images/picture-${new Date().getTime()}')
4   .put(blob)
5   .then(res => {
6     res.ref.getDownloadURL().then(pictureUrl => {
7       console.log("File available at", pictureUrl);
8     });
9   })
```

Realtime Push Notifications. The Firebase messaging service in combination with Firebase functions allows to easily listen for hooks to send out push notifications to a single user or a group of users. Firebase Cloud Functions¹⁵ is a dedicated service that seamlessly ties into the other Firebase offerings.

Listing 9: Firebase Functions calling the messaging service when a new post is created in the Firestore database(functions/index.js)

```
1 exports.createDrink = functions.firestore
2   .document("drinks/{drinkId}")
3   .onCreate(event => {
4     var drink = event.data();
5     console.log(drink.comment);
6     return axios
7       .post(
8         "https://fcm.googleapis.com/fcm/send",
9         {
10           to: "/topics/general",
11           priority: "high",
12           notification: {
13             title: drink.author + " is thirsty!",
14             body: drink.comment || "Join in for a drink",
15             click_action: "https://bwm.gunicode.com",
16             icon: "https://bwm.gunicode.com/img/icons/
17               notification-128x128.png"
18           }
19         })
20   });
```

¹⁵<https://firebase.google.com/products/functions>

Listing the full integration of Firebase into the is out of the scope of this paper. Refer to the codebase at <https://github.com/gunharth/bwm> in the functions folder.

Finally, **Firebase Hosting** service is used to deploy the demo to <https://bwm.gunicode.com>. Using the firebase CLI this is a straight forward process. Once the project is registered as a service through the Firebase CLI new versions of a project can be easily deployed.

Listing 10: Firebase CLI commands

```
1  # Install the CLI globally through npm
2  $ npm install -g firebase-tools
3  # login to the firebase service
4  $ firebase login
5  # register a new project with firebase
6  $ firebase init
7  # deploy the project to firebase hosting
8  $ firebase deploy
```

4. Results

Overall, re-building the prototype of the original application was a success. The foundational question of this study was if it will be possible to duplicate the features of the original application as a PWA. This can be answered in a positive way.

Frontend frameworks, like the one used in this study, strive to support the development of optimized PWA skeletons to streamline the development and promise future enhancements to limit the time to market (TTM). It is assumed that this tendency continues to unveil further productive environments.

Firebase, as a representative for a solution provider to deliver server-less application delivery is an example of a rather new industry. It is expected that Firebase will expand on its services and other service providers will enter this market in the near future to cater for the quickly rising demand in this field.

In terms of the actual development process, more streamlined processes are required. The modularity of npm packages still seems to be too scattered and present a hurdle for rapid developments. A simpler implementation on all ends in terms of modularity should be the focus of frontend frameworks as well as cloud specific service providers. Overall, it is observed that in some cases reading the documentation is a time consuming task, whereby leading away the focus from the actual product development.

Finally, it is advised to closely look at the application requirements at an early stage. Not every project is suitable to be developed as a PWA yet. Hence, if an application depends on platform or device specific services native and hybrid solutions still are a better choice.

5. Conclusion

In conclusion, PWAs might not replace native apps completely within the next few years. However, the techniques involved have leveled up the standard of modern websites. Rapid enhancements of browser technologies and related standards promise further integration with devices and enduser behaviors.

The new Twitter website demonstrates that PWAs stepped out of the teething phase and are a serious contender for platform independent production ready applications. We can expect further businesses to entice PWA technologies in the near future in order to streamline the delivery of their products.

Next too technical research on PWAs there is plenty of room for related future research. The new technology introduces new user behavior in the likes of installing a web site on a desktop PC, which implies research studies on user behavior and usability. Further, PWAs have the potential to make app stores obsolete. This financial aspect needs to be researched and new ways to commercialize the production of applications are suggested.

Bibliography

Ater, T. (2017). *Building Progressive Web Apps: Bringing the Power of Native to the Browser*. O'Reilly Media, Sebastopol, CA, first edition edition. OCLC: ocn956340441.

Biørn-Hansen, A., Majchrzak, T. A., and Grønli, T.-M. (2018). Progressive Web Apps for the Unified Development of Mobile Applications. In Majchrzak, T. A., Traverso, P., Krempels, K.-H., and Monfort, V., editors, *Web Information Systems and Technologies*, volume 322, pages 64–86. Springer International Publishing, Cham.

Croom, C. and Baker, G. (2019). Building the new Twitter.com. https://blog.twitter.com/engineering/en_us/topics/infrastructure/2019/buildingthenewtwitter.html. Retrieved 2019-08-02T09:20:08Z.

Hume, D. A. and Osmani, A. (2018). *Progressive Web Apps*. Manning Publications, Shelter Island, New York. OCLC: on1017992176.

Liebel, C. (2019). *Progressive Web Apps: das Praxisbuch*. Rheinwerk Computing. Rheinwerk Verlag, Bonn, 1 edition.

Malavolta, I. (2016). Beyond native apps: Web technologies to the rescue! (keynote). In *Proceedings of the 1st International Workshop on Mobile Development - Mobile! 2016*, pages 1–2, Amsterdam, Netherlands. ACM Press.

Nguyen, P. H. (2019). Progressive Web App in Enhancing App Experience: Life Care Insurance Claim Application.

Osmani, A. (2015). Getting started with Progressive Web Apps. <https://addyosmani.com/blog/getting-started-with-progressive-web-apps/>. Retrieved 2019-06-08.

Russell, A. (2015). Progressive Web Apps: Escaping Tabs Without Losing Our Soul – Infrequently Noted. <https://infrequently.org/2015/>

06/progressive-apps-escaping-tabs-without-losing-our-soul/. Retrieved 2019-08-02.

Wikipedia (2019). Minimum viable product. https://en.wikipedia.org/w/index.php?title=Minimum_viable_product&oldid=906835119. Retrieved 2019-08-07.