jamk.fi

PWA-teknologian toiminta eri alustoilla sekä PWA-ominaisuuksien toteuttaminen verkkosovelluksessa

Joel Harjunheimo

Opinnäytetyö
Huhtikuu 2019
Tekniikan ja liikenteen ala
Insinööri (AMK), tieto- ja viestintätekniikan tutkinto-ohjelma
Ohjelmistotekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Harjunheimo, Joel	Opinnäytetyö, AMK	Huhtikuu 2019
	Sivumäärä 40	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi

PWA-teknologian toiminta eri alustoilla sekä PWA-ominaisuuksien toteuttaminen verkkosovelluksessa

Tutkinto-ohjelma

Tieto- ja viestintätekniikka

Työn ohjaaja(t)

Pasi Manninen, Esa Salmikangas

Toimeksiantaja(t)

Nolwenture Oy

Tiivistelmä

Progressive Web App (PWA), eli progressiivinen verkkosovellus, on uusi kehitteillä oleva lajitelma teknologioita, jonka tarkoitus on tuoda uusia ominaisuuksia verkkosovelluksiin. Opinnäytetyön toimeksiantaja Nolwenture Oy kehittää Base-verkkosovellusta, joka pohjautuu React.js-kirjastoon ja Node.js-ympäristöön. Toimeksiantaja halusi saada lisää tietoa PWA:sta, koska kyseessä on uusi teknologia ja jatkossa mahdollisesti vaihtoehtoinen teknologia mobiilikehityksessä käytettäville sovelluskehyksille.

Tutkimuksen tavoite oli selvittää, mikä PWA on, miten se toimii, mikä on tämän hetkinen PWA tuki eri alustoilla sekä miten yrityksen omaan Base-sovellukseen lisätään PWA-ominaisuuksia. Opinnäytetyö käsitti kaksi osaa, joista toinen käsitteli teorian ja toinen teknistä toteutusta.

Tutkimus toteutettiin kvalitatiivisena tutkimuksena, koska tarkoitus ei ollut luoda parannuksia olemassa olevaan, vaan tutkia PWA-teknologiaa ja sen toimintaa ilmiönä, jotta toimeksiantaja saisi paremman kuvan tutkimuskohteesta. Tutkimus alkoi PWA:n toimintaan liittyvät teorian tutkimisella, jonka jälkeen tutkittiin eri alustojen tukea. Viimeiseksi tutkimuksessa toteutettiin teknillinen toteutus Base-sovellukseen käyttäen Googlen kehittämää Workbox-kirjastoa, joka tarjoaa valmiita liitännäisiä PWA-ominaisuuksien toteuttamiseen.

Tuloksena saatiin yleiskuva PWA-teknologian toiminnasta, selvennystä tuesta eri alustoilla sekä toimivia PWA-ominaisuuksia Base-sovellukseen. Tuloksista voitiin tehdä johtopäätöksiä, että tulevaisuudessa progressiivinen verkkosovellus voi olla vaihtoehtoinen toteutustapa mobiilisovellukselle, mutta tällä hetkellä laitteiden tuki ei välttämättä riitä. Välimuistien käyttö kuitenkin antaa verkkosovellukselle paremman käyttökokemuksen, koska sivujen lataus on nopeampaa.

Avainsanat (asiasanat)

PWA, progressiivinen verkkosovellus, verkkosovellus, mobiilisovellus, Workbox

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Harjunheimo, Joel	Bachelor's thesis	April 2019
		Language of publication: Finnish
	Number of pages: 40	Permission for web publication: x

Title of publication

Operating principles of PWA technology on different platforms and implementation of PWA features in a Web application

Degree programme

Information and communications technology

Supervisor(s)

Manninen, Pasi & Salmikangas, Esa

Assigned by

Nolwenture Oy

Abstract

Progressive Web Application (PWA) is a new set of technologies in development aiming to bring new features to web applications. The thesis assigner, Nolwenture Oy, is developing a web application called Base, which is built by using React.js library and Node.js runtime-environment. Nolwenture Oy wanted to have more information on PWA, because the technology is new in the field and it might be an alternative to frameworks used in mobile development.

The aim of the research was to find out and clarify, what PWA is, how it works, what is the current support for PWA on different platforms and how to add PWA features to the Nolwenture Base-application. The research consisted of two parts, the first part of which was about the theory of PWA and how it works, and the second part that was about the technical implementation of PWA features to the Base.

The research was carried out as a qualitative research, because the aim of the study was not to make improvements but to study PWA technology as a phenomenon to give the assigner clearer comprehension of it. The research began with theory about how PWA works which was followed by researching the PWA support on different platforms. Last part of the research was to implement PWA features to Base with Google's Workbox library, which offers readymade plugins to achieve wanted features.

The result of the research was an overview of how the PWA technology works, clarification on the support on different platforms and working PWA features in Base. A conclusion can be drawn from the results that PWA might be a viable alternative for mobile development in the future, however the current support might not be enough as it is. A better user experience can, however, be achieved by using caching on web applications.

Keywords/tags (subjects)

PWA, Progressive Web App, web application, mobile application, Workbox

Miscellaneous (Confidential information)

Sisältö

Kuviot

1 Base-sovelluksen kehittäminen

1.1 Lähtökohta

Älypuhelinten määrän kasvettua yritykset ovat alkaneet täyttää sovellustarpeita mobiilisovelluksilla. Syitä tähän voi olla esimerkiksi hyvä käytettävyys, laitteen helppo liikuteltavuus tai offline-toiminnot. Yhtenä ongelmana mobiilisovelluskehityksessä kuitenkin on Android- ja iOS-sovellusten pohjautuminen eri kieliin, mikä vaatii luomaan molemmille alustoille erilliset toteutukset, mikä taas nostaa sovelluskehityksen hintaa. Ongelmaan ratkaisuksi on kehitetty esimerkiksi React Native -sovelluskehys, jolla voidaan luoda sovelluksia molemmille alustoille käyttäen samaa pohjaa. React Nativen kaltaisella sovelluskehyksellä ei kuitenkaan välttämättä pystytä tuottamaan yhtä tehokasta tai monipuolista sovellusta kuin natiivitoteutuksella. Verkkosovelluksilla ei ole mobiilisovellusten kaltaisia ongelmia eri alustojen kanssa, vaikka selainten välillä eroja onkin. Ongelmana perinteisissä verkkosovelluksissa on kuitenkin niiden riippuvaisuus verkkoyhteydestä. Ratkaisuna verkkosovellusten offline-käyttöön voi nykyään käyttää PWA:ta eli progressiivista verkkosovellusteknologiaa.

Tästä kehityksestä syntyi tarve tälle opinnäytetyölle, tarve ymmärtää uutta teknologiaa ja luoda toimintaympäristöä sen hyödyntämiselle. Opinnäytetyö koostuu kahdesta osuudesta: kirjallisesta tuotoksesta sekä teknisestä toteutuksesta. Kirjallinen tuotos on opinnäytetyön raportti, jossa kuvataan teknistä toteutusta sekä kootaan yhteen siihen liittyvää teoriaa. Opinnäytetyön teoriaosan tarkoitus on antaa hyvä kuva progressiivisten verkkosovellusten teknologiasta ja toimintaperiaatteista, jotta teknisessä toteutuksessa tehtyjen toimintojen tarkoitus ja hyöty avautuu parhaiten. Teknisen toteutuksen tarkoitus on tuottaa käyttökelpoisia parannuksia toimeksiantajan Base-sovellukseen sekä antaa käytännön esimerkki progressiivisen verkkosovelluksen toiminnasta.

1.2 Toimeksiantaja

Tämän opinnäytetyön toimeksiantajana toimiva Nolwenture Oy on yksityisomisteinen vuonna 2012 perustettu digitaalisia palveluita tarjoava yritys Joensuusta. Yrityksessä on tällä hetkellä viisitoista vakituista työntekijää. Nolwenture kehittää omaa Base-nimistä verkkosovellusta, joka on rakennettu käyttämällä React.js-kirjastoa ja Node.js-ympäristöä. Basea voidaan käyttää pohjana asiakkaiden tilaamissa ohjelmistoissa, jolloin sovelluskehitykseen tarvittava on lyhyempi. Nolwenture tarjoaa sovelluskehityspalveluita niin verkko- kuin mobiilisovelluksien ja IoT-toteutuksien osalta. Yritys tarjoaa myös asiantuntijapalveluita vuokraamalla työntekijöitä asiakkaille ja antamalla konsultaatiota markkinoinnista ja tietoteknisistä aiheista. Nolwenturen asiakkaat sijoittuvat suurilta osin Itä-Suomen alueelle, mutta sillä on asiakkaita myös Singaporessa ja Saksassa.

1.3 Työ tavoite

Tutkimuksen tavoite on tutkia progressiivisen verkkosovellusteknologian nykytilaa sekä sovelletaan sitä toimeksiantajan Base-sovellukseen. Opinnäytetyön teknisen toteutuksen tavoitteena on tehdä Base-sovelluksesta tehokkaampi ja antaa sille offline-toimivuutta. Teknologian tutkiminen ja testaaminen on yritykselle tärkeää, jotta se pystyy pysymään mukana alan teknologian kehityksessä sekä mahdollisesti käyttämään PWA-teknologiaa tulevissa projekteissa. PWA-toiminnallisuus toteutetaan käyttämällä Googlen kehittämää Workbox JavaScript -kirjastoa ja sen Webpack-liitännäistä, koska käytössä on myös Webpack-kirjasto. Workbox tarjoaa yksinkertaisemman tavan toteuttaa palvelunvälittäjä (service worker) kirjaston omilla liitännäisillä, mutta antaa myös mahdollisuuden toteuttajalle toteuttaa palvelunvälittäjä alusta loppuun itse.

Tämä opinnäytetyö on laadullinen tutkimus, sillä sen tavoitteena on tutkia progressiivisen verkkosovellusteknologian nykytilaa, toimivuutta eri alustoilla, antaa hyvä kuvaus sen toiminnasta sekä käyttää teknologiaa tuomaan PWA-toiminnallisuutta olemassa olevaan verkkosovellukseen. Tekninen toteutus tuottaa uutta tietoa ilmiön

syvällisemmän ymmärtämisen avulla. Vaikka tekninen toteutus kohdistuukin toimeksiantajan tiettyyn sovellukseen, tutkimuksen tuloksia voidaan soveltaa ja hyödyntää muihinkin verkkosovelluksiin. Kvalitatiivisen eli laadullisen tutkimuksen tarkoitus onkin luoda tietoa aiheesta ja saada siitä hyvä kuvaus. Tavoitteena laadullisella tutkimuksella on kuvata, ymmärtää ja tulkita tutkittavaa aihetta ilman määrällisiä keinoja kuten tilastollisia menetelmiä. Laadullisen tutkimuksen tavoite on esimerkiksi kokonaisvaltainen aiheen ymmärtäminen. Aineistoa kerätään, kunnes tutkimusongelma ratkeaa ja tutkija ymmärtää ilmiön riittävän syvällisesti. Tutkimustulosta ei voida yleistää laadullisessa tutkimuksessa, sillä se pätee ainoastaan tutkimuskohteen osalta. (Kananen 2017, 33-36.)

1.4 Tutkimusasetelma

Tutkimusongelma ohjaa tutkimusta, joten sen määrittely on tärkeää, koska ongelmasta johdetaan tutkimuskysymykset. Tutkimuskysymyksen avuksi voidaan määrittää metakysymyksiä, jotka auttavat vastauksen löytämisessä korkeamman tason tutkimuskysymyksille. Työn kannalta kriittisiä ovat tutkimusongelmat ja kysymykset, sillä ne ovat lupauksia, jotka tulee täyttää empiirisessä osuudessa. (Kananen 2017, 60-62.) Opinnäytetyön tutkimusongelmaksi kehittyi progressiivisen verkkosovellusteknologian tutkiminen ja sen soveltaminen valmiiseen verkkosovellukseen. Tutkimusongelmasta voidaan johtaa seuraavat tutkimuskysymykset:

- Mitä tarkoittaa PWA eli progressiivinen verkkosovellus?
- Miten PWA toimii eri alustoilla?
- Miten Base-sovellukseen toteutetaan PWA-toimintoja?

Tässä työssä aineisto kerätään pääsääntöisesti verkosta saatavista lähteistä. Tutkimuksen kohteen ollessa uusi, ei painettua kirjallisuutta aiheesta ole juurikaan saatavilla. Toisaalta tällä voidaan varmistaa myös lähteiden paikkaansa pitävyys, sillä tuore verkosta saatu materiaali ei ole vanhentunut. Laadullisen tutkimuksen aineistopohjana voidaan käyttää kaikkia dokumentteja ja tutkimusongelma pyritään ratkaisemaan eri tietolähteitä käyttäen. Kirjallisen aineiston luotettavuutta täytyy analysoida

ja käyttää useampaa tietolähdettä, jotta tieto olisi mahdollisimman yhtäläistä. (Kananen 2010, 63.) Tutkimuskysymyksiin vastataan kerätystä aineistoista saadulla tiedolla eli tiedon avulla etsitään ratkaisua tutkimusongelmaan. Aineiston keräämiseen käytettävät menetelmät ovat riippuvaisia tutkimusotteesta. Laadullisessa tutkimuksessa aineisto koostuu erilaisista dokumenteista, haastatteluista ja havainnoinneista. Aineistoja voidaan käsitellä ainoastaan niille sopivin analyysikeinoin. (Kananen 2017, 67).

Tieteellisen tutkimuksen luotettavuus ja laatu täytyy varmistaa. Kvalitatiivisen tutkimuksen kohdalla luotettavuusarvion tekeminen on vaikeampaa kuin kvantitatiivisessa tutkimuksen kohdalla. (Kananen 2010, 68-69.) Opinnäytetyön ollessa laadullinen tutkimus, joka on kohdistettu tiettyyn toimeksiantajan kehitteillä olevaan sovellukseen, on tutkimuksen luotettavuuden arviointi haasteellista. Tutkimuksen validiteettia voidaan tarkastella helpommin, sillä tutkimuksen teknisen osuuden edetessä saadaan nopeasti selville, testataanko sovelluksesta oikeita asioita. Reliabiliteetin arviointi taas on vaikeampaa, koska PWA-teknologia kehittyy nopeaa tahtia ja tutkimuksen tulos voi toimia ainoastaan toteutuksen kohteena olevan sovelluksen tämän hetkisessä kehitysvaiheessa.

2 Progressive Web App

2.1 Hyödyt ja vaatimukset

PWA eli Progressive Web App on moderni web-sovellus, joka käyttää moderneja verkon rajapintoja perinteisten progressiivisten parannusstrategioiden ohella tarjotakseen järjestelmäriippumattoman web-sovelluksen. PWA-sovelluksien tarkoitus on toimia missä tahansa sijainnissa, tarjota useita ominaisuuksia, jotka tarjoavat saman käyttäjäkokemuksen kuin laitteiden natiivisovellukset sekä pystyä hyötymään myös verkon hyödyistä. (Progressive web apps 2018.) Verkkosivun hyötyihin voidaan laskea niiden parempi löydettävyys ja nopeampi käyttö sekä sivuston linkin jakamisen helppous. Natiivisovelluksien hyötyihin puolestaan kuuluu niiden parempi integraatio laitteiden kanssa sekä mahdollisuus käyttää sovelluksia suoraan laitteen

aloitusnäytöstä. PWA:n tarkoitus on antaa mahdollisuus luoda sovelluksia, jotka voivat hyödyntää molempien toteutuksien hyötyjä. (Introduction to progressive web apps 2018.)

Google on koonnut Baseline Progressive Web App Checklist -listan, josta kehittäjä voi katsoa PWA-sovellukseen tarvittavat ominaisuudet. Lähtökohtaisesti sivuston tulee käyttää HTTPS:ää, koska salaamaton yhteys yhdistettynä service workerin toimintaan on suuri turvallisuusriski. Sivuston tulisi myös toimia responsiivisesti ja sulavasti kaikilla mobiililaitteilla ja tietokoneilla. Sivuston toimintaa ja käyttökokemusta parantaa myös yhtäläinen toimivuus eri selainten välillä, siirtymien toimiminen sulavasti hitaassakin verkossa, sivuston kaikkien URL-osoitteiden toimiminen offline-tilassa ja sivuston nopea käynnistyminen myös 3G-verkossa. Esimerkillisessä PWA-sovelluksessa tulisi lisäksi käyttää esimerkiksi History API, Credential Management API, Payment Request API -rajapintoja, Cache-first-toimintoa sekä sen tulisi sisältää ilmoitus, kun käyttäjä on offline-tilassa. Googlen tekemä indeksointi on myös tärkeää PWA-sovellukselle. (Progressive Web App Checklist 2018.)

2.2 PWA:n toimintaperiaatteet

Verkkosivun voi renderöidä kahdella tapaa client-side (CSR) eli käyttäjä puolella tai server-side (SSR) eli palvelimen puolella. PWA-sovellus voidaan rakentaa käyttämällä kumpaa tapaa tahansa. Yleisin tapa on kuitenkin app shell -konsepti, joka käyttää molempia CSR:ää ja SSR:ää sekä lisäksi noudattaa offline first -käytäntöä. App shell -konseptin tarkoitus on ladata minimaalinen käyttöliittymä heti kun se on mahdollista, tallentaa se välimuistiin, jotta se on saatavilla yhteydettömässä tilassa peräkkäisillä vierailuilla ja sitten ladata sovelluksen sisältö. Tällä tavalla käyttöliittymä voidaan ladata seuraavalla vierailukerralla välittömästi välimuistista ja uusi sisältö voidaan pyytää palvelimelta, mikäli sitä ei ole jo tallennettu välimuistiin. Tällainen toiminta antaa käyttäjälle paremman käyttökokemuksen, koska sovellus näyttää sisältöä heti ilman, että käyttäjä joutuisi odottamaan tyhjällä sivulla tai katsomaan latausanimaatiota. Välimuistista saatavaa sisältöä voidaan myös näyttää ilman verkkoyhteyttä.

Välimuistin käyttöä ja pyyntöjä palvelimelle voidaan kontrolloida service worker -skriptillä. (Progressive web app structure, 2018.)

Käyttökokemusta mobiililaitteella parantaa Web App Manifest -tiedosto, jonka avulla verkkosovelluksen voi muotoilla näyttämään mobiililaitteen natiivisovellukselta. Tiedoston avulla sovelluksen voi käynnistää koko näytön kokoisena ilman osoitepalkkia ja määrittää miten päin sovellus on ruudulla. Manifestin avulla sovellukseen voi myös esimerkiksi lisätä Splash screen -aloitusnäytön ja teemavärit. (Osmani, 2015.) Web App Manifestin toiminnasta tarkemmin kappaleessa 2.4.

2.3 Service Worker

Service worker eli palvelunvälittäjä on eräänlainen web worker eli JavaScript-tiedosto, joka toimii erillään selaimen pääsäikeestä. Se pysäyttää tietoverkon pyyntöjä, tallentaa välimuistiin (cache) tai hakee välimuistista tietoja ja välittää push-ilmoituksia. Service worker on siis ohjelmoitava tietoverkon välityspalvelin, joka mahdollistaa tietoverkon pyyntöjen käsittelyn halutulla tavalla, mutta se toimii ainoastaan salatussa yhteydessä (HTTPS) tietoturvan takia. (Introduction to Service Worker, 2019.)

Service worker on itsenäinen ja riippumaton muusta sovelluksesta, minkä seurauksena se voi toimia täysin asynkronisesti. Service worker ei siis estä muun ohjelmiston toimintoja eli toisin sanoen se on luonteeltaan non-blocking. Tästä johtuen synkronista XHR:ia tai selaimen local storagea ei voida käyttää, mutta hyvänä puolena on se, että service worker voi ottaa vastaan push-ilmoituksia palvelimelta ja näyttää ne käyttäjän laitteella, vaikka sovellus ei olisi aktiivinen selaimessa. Service workerilla ei ole suoraa pääsyä Document Object Modeliin (DOM), joten se joutuu käyttämään “postMessage()” -metodia lähettääkseen dataa ja message-tapahtumankuuntelijaa vastaanottaakseen dataa. (Mt.)

Service workerin mahdollisuus käsitellä tietoverkon pyyntöjä ja niiden tallentamista välimuistiin mahdollistaa sovelluksen käytön yhteydettömässä tilassa eli offline-tilassa sen mukaan mitä välimuistiin on tallennettu. Service worker on riippuvainen

kahdesta rajapinnasta: fetch, jota käytetään sisällön noutamiseen verkosta ja cache, joka on säilyvä (persistent) välimuisti ja erillinen selaimen välimuistista. Sovelluksen tietojen tallentaminen service workerin välimuistiin nopeuttaa sisällön lataamista. Muita service workerin rajapintoja ovat esimerkiksi Notifications API, Push API, Background Sync API ja Channel Messaging API. (Mt.)

Notifications API mahdollistaa ilmoituksien näyttämisen ja niiden kanssa vuorovaikuttamisen laitteen natiivi-ilmoitusjärjestelmän kautta. Push API on rajapinta, joka tekee laitteelle mahdolliseksi vastaanottaa push-viestejä. Push-viesti toimitetaan service workerille, joka voi käyttää viestissä olevaa tietoa päivittääkseen sovelluksen paikallista tilaa tai näyttääkseen käyttäjälle ilmoituksen. Service workerin itsenäisyys muusta sovelluksesta mahdollistaa viestien lähettämisen ja vastaanottamisen laitteen selaimen ollessa suljettuna. Background Sync API -rajapinta mahdollistaa toimenpiteiden lykkäämistä huonossa verkkoyhteydessä, kunnes laite saa taas paremman yhteyden. Tiedon välittämisen lykkäys varmistaa sen, että tieto varmasti lähetetään niin kuin käyttäjä on sen tarkoittanut. Rajapinta myös antaa palvelimen päivittää sovellusta jaksoittain, jotta sovellus voidaan päivittää, kun se on seuraavan kerran yhteydessä. Channel Messaging API -rajapinta taas mahdollistaa service workerien ja web workerien kommunikoinnin keskenään ja isäntäsovelluksen kanssa. (Mt.)

2.4 Web App Manifest

Web App Manifest on yksinkertainen JSON-tiedosto, joka toimii ohjeena selaimelle siitä, miten sovelluksen tulisi käyttäytyä, kun se tallennetaan käyttäjän aloitusnäyttöön mobiililaitteella tai työpöydälle tietokoneella. Sovelluksen tulee sisältää manifestitiedosto, jotta selain voi näyttää ”Lisää aloitusnäyttöön” -ilmoituksen käyttäjän laitteella. Tiedosto lisätään verkkosovellukseen samalla tavalla kuin esimerkiksi tyylitiedostot eli HTML-tiedoston head-osioon käyttämällä link-elementtiä. Kuviossa 1 näkyy manifestitiedosto, jossa on määriteltynä järjestyksessä: Sovelluksen nimi ja lyhyt versio nimestä, ikonit ja niiden sijainnit, koot ja tyypit, sovelluksen aloitussivu, display

eli käyttöliittymän esitystapa laitteen näytöllä, taustaväri ja teemaväri. (The Web app Manifest, 2019.)

```
{
  "name": "Web Starter Kit",
  "short_name": "WSK",
  "icons": [{
        "src": "images/touch/icon-128x128.png",
        "sizes": "128x128",
        "type": "image/png"
      }, {
        "src": "images/touch/apple-touch-icon.png",
        "sizes": "152x152",
        "type": "image/png"
      }, {
        "src": "images/touch/ms-touch-icon-144x144-precomposed.png",
        "sizes": "144x144",
        "type": "image/png"
      }, {
        "src": "images/touch/chrome-touch-icon-192x192.png",
        "sizes": "192x192",
        "type": "image/png"
      }],
  "start_url": "/index.html?homescreen=1",
  "display": "standalone",
  "background_color": "#3E4EB8",
  "theme_color": "#2F3BA2"
}
```

Kuvio 1. Webb App Manifest (Web starter kit, 2015.)

Lisäksi manifestiin voidaan vielä määritellä esimerkiksi orientation, description ja scope. Orientation määrittää sovelluksen orientaation eli sen missä asennossa sovellus on laitteen näytöllä. Vaihtoehtoja on landscape eli vaakakuva tai portrait eli pystykuva, any, natural, landscape-primary, landscape-secondary, portrait-primary ja portrait-secondary. Scope määrittelee sen liikkuma-alan, jonka selain näkee olevan osa sovellusta. Alan ulkopuolelle siirtyessä selain siirtää käyttäjän takaisin sovelluksesta selainikkunaan. Manifestissa määritetyn sovelluksen aloitussivun tulee olla scope-määrityksen sisällä. Mikäli ominaisuutta ei määritellä, koko sovelluksen ala on katettu. Mikäli sivuston https://example.com scope on määritelty arvolla /example, on osoite https://example.com/example/anotherpage alueen sisällä, mutta osoite

https://example.com/thirdpage alueen ulkopuolella. Molemmat osoitteet ovat alueen sisällä arvolla /. (The Web app Manifest, 2019.)

3 PWA:n tuki eri alustoilla

3.1 PWA mobiilialustalla

3.1.1 Android

Android-laitteisiin service worker tuki tuli joulukuun 2018 alussa Chrome-selaimen version 71 myötä (Can I Use? 2019). Android-laitteissa selain luo ja asentaa automaattisesti laitteelle PWA-sovelluksesta APK:n (Android Application Package), kun käyttäjä antaa hyväksyntänsä “Lisää aloitusnäyttöön” -ilmoitukseen. Tätä APK:ta voidaan kutsua myös WebAPK:ksi. Selain käyttää sovelluksen Web App Manifest -tiedostoa ja muuta metadataa APK:n luomiseen. Selain tarkastaa sen hetkisen manifestin, kun sovellus käynnistetään ja päivittää WebAPK:n, mikäli manifestista on muuttunut sellaisia kohtia, jotka vaikuttavat sovelluksen lisäämiseen aloitusnäyttöön. Sovelluksen asentaminen APK:n kautta mahdollistaa sen, että sovellus näkyy laitteen sovellusluettelossa ja siihen liittyviä asetuksia voi muuttaa laitteen sovellusasetuksista. Asennus myös rekisteröi intent-filttereitä laitteelle. (Kotwicz, Lepage & Posnic 2019.) Intent on eräänlainen viesti, jolla pyydetään toimintoja sovelluksen toiselta komponentilta ja intent filter on työkalu, jolla voi suodattaa intent viestejä niiden toiminnan, sisällön ja kategorian perusteella (Intents and Intent Filters, 2019). Filttereiden avulla voidaan muuttaa sovelluksen toiminnallisuutta siten, että sovelluksessa avatut linkit avataan sovelluksessa eikä erikseen selaimen välilehteen, mikäli ne ovat manifestin scope-ominaisuuden alueen sisällä (Kotwicz, Lepage & Posnic 2019.)

Chrome käyttää senhetkistä profiilia tallentaakseen dataa, eikä sitä erotella mihinkään. Tällä tavalla voidaan antaa käyttäjälle jaettu kokemus selaimen ja asennetun sovelluksen välillä. Evästeet ja client side storage -tallennustila jaetaan myös selaimen ja sovelluksen välillä. Sovelluksen toiminnalle voi aiheutua ongelmia, mikäli käyttäjä tyhjentää Chrome profiilinsa tai poistaa sivuston datan. (Mt.)

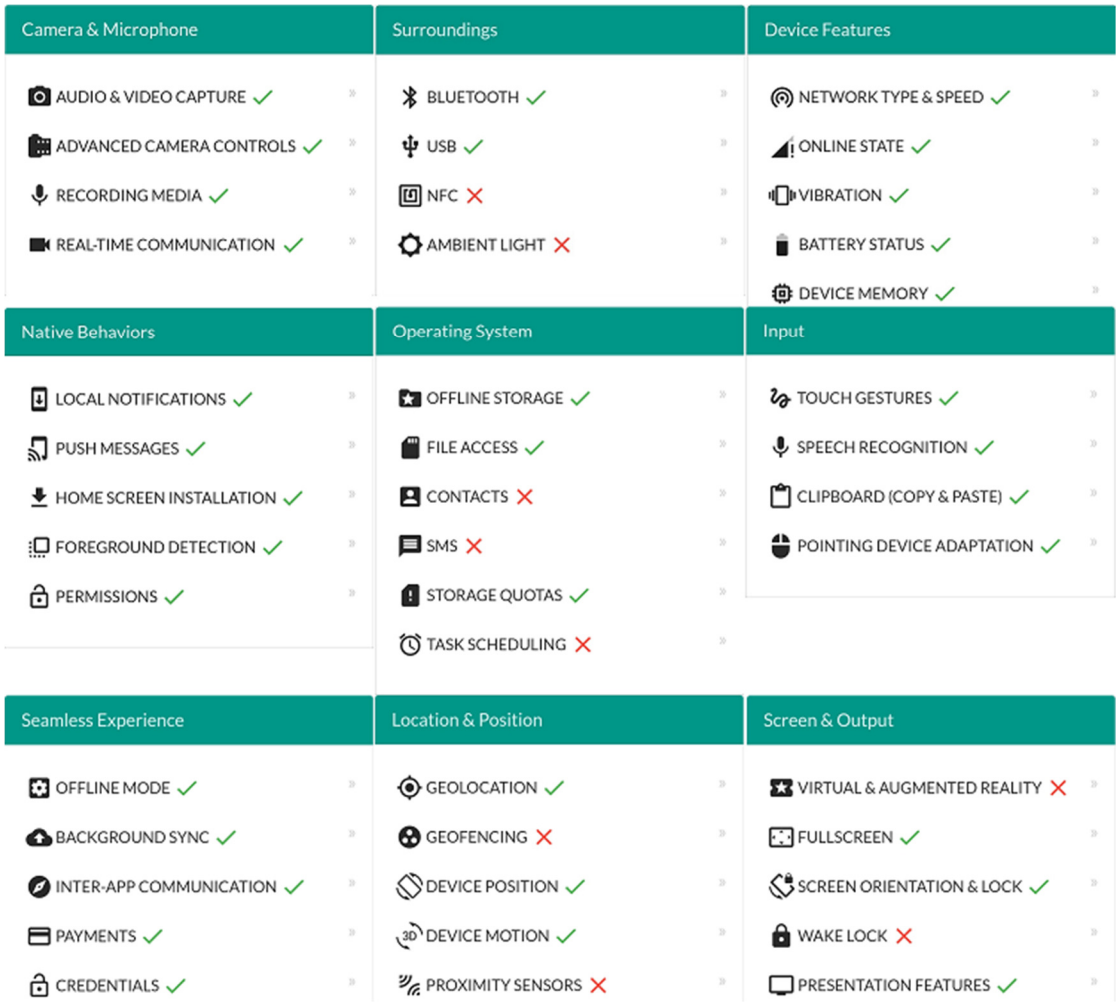
Android 8.0 version eli Android Oreon myötä Androidiin lisättiin uusia toimintoja, jotka voivat olla hyödyllisiä PWA-sovelluksille, mutta jotka eivät vielä toimi verkkosovelluksien kautta. Toimintoja ovat esimerkiksi:

- Notification dots eli merkit, jotka näkyvät sovelluksen aloitusnäytön ikonissa, mikäli sovelluksessa on avaamattomia ilmoituksia
- Adaptive Icons eli sovellusten ikonit voivat olla erimuotoisia eri laitteilla
- Notification channels, joka antaa käyttäjälle mahdollisuuden määritellä verkkosivujen tai sovellusten haluttuja ilmoituksia kategorioittain
- Shortcuts eli pikavalinnat, jotka näkyvät, kun käyttäjä painaa sovelluksen ikonia pitkään

(Firtman, 2017.)

3.1.2 Android-laitteen What Web Can Do Today -testi

Testilaitteena toimi Honor 9 puhelin Android 8.0 versiolla ja selaimena Chrome versio 71.0.3578.99 ja Firefox versio 64.0.2. Testiin käytettiin What Web Can Do Today -nimistä sivustoa. Kuviossa 2 vihreällä merkityt laitteen ominaisuudet ovat PWA-sovelluksen käytettävissä Chrome-selaimessa ja punaisella merkityt eivät ole.



Kuvio 2. Android-laitteen testi What Web Can Do -sivustolla (What Web Can Do Today, 2019.)

Testien perusteella Firefox-selaimella ei ole pääsyä seuraaviin ominaisuuksiin, joihin Chrome-selaimella on: Advanced Camera Controls, Bluetooth, USB, Battery Status, Device Memory, Home Screen Installation, Storage Quotas, Speech Recognition, Background Sync, Inter-App Communication, Payments, Credentials, Presentation Features. Firefoxilla on pääsy seuraaviin ominaisuuksin, joihin Chromella ei ole: Ambient Light, Proximity Sensors. (What Web Can Do Today, 2019.)

3.1.3 iOS

Applen älypuhelimiin tuli iOS-version 11.3 myötä enemmän tukea PWA-sovelluksille. Tuki tarkoitti käytännössä Web App Manifest ja Service Worker tukea ja sitä, että sovelluksia pystyi asentamaan iOS-laitteelle ilman App Store -sovelluskaupan hyväksyntää. Ilman sovelluskaupan hyväksyntää verkkosovellus ei kuitenkaan voi tällä hetkellä

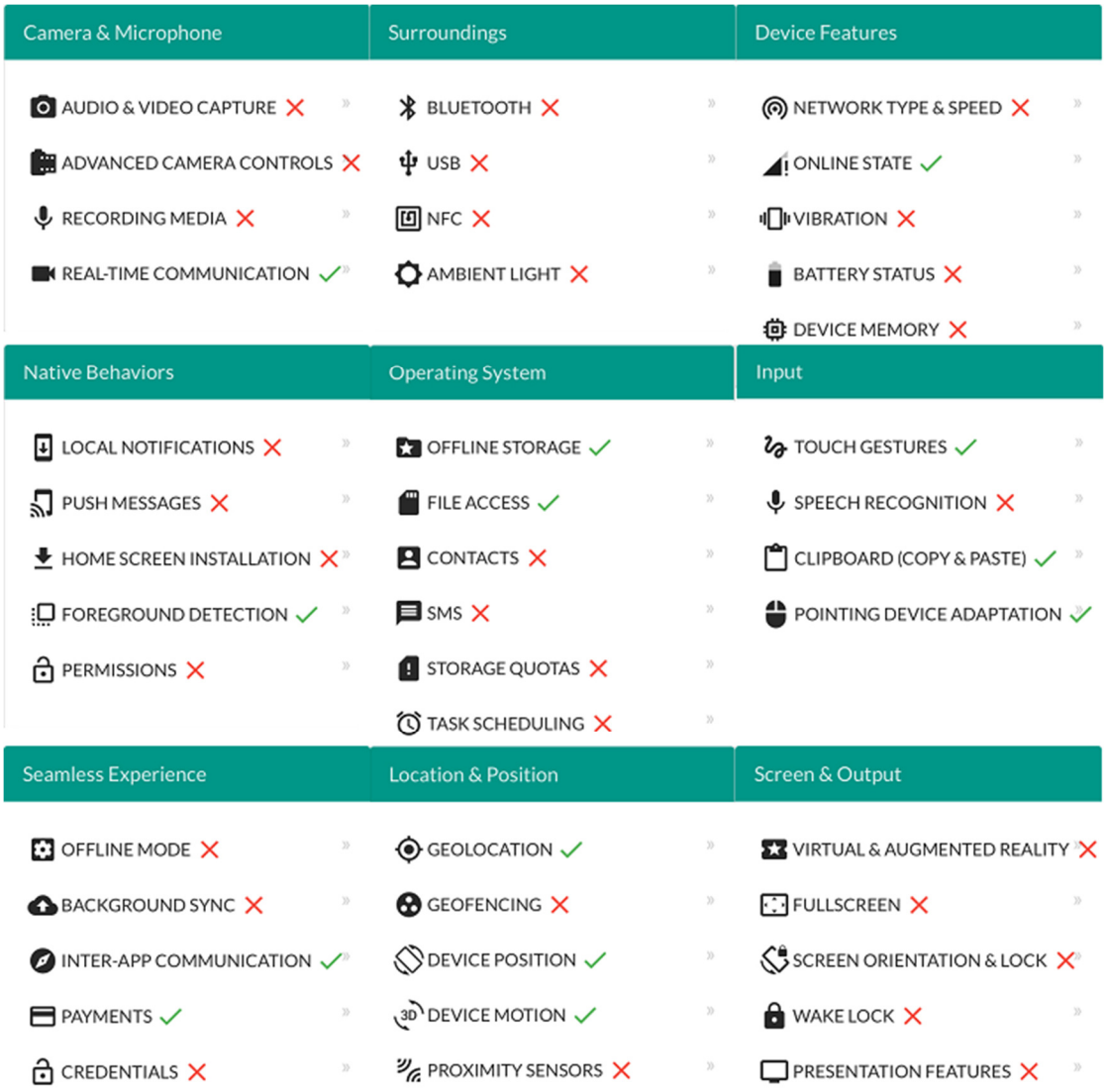
käyttää puhtaasti natiiveja ominaisuuksia kuten Face ID:tä tai ARKit:iä iPhone X:ssä. (Firtman, 2018.)

PWA-sovelluksella on tällä hetkellä joitain rajoituksia natiivisovelluksiin verrattuna. PWA-sovellus ei voi esimerkiksi tallentaa laitteelle yli 50 megabittiä dataa yhteydettöntä tilaa varten, eikä se voi käyttää ominaisuuksia kuten Bluetoothia, Beaconsia, Touch ID:tä tai Face ID:tä ARKit:iä, korkeusmittaria tai akun tietoja. Lisäksi PWA-sovelluksella ei ole oikeuksia ajaa koodia taustalla tai päästä In App Payments -toimintoon tai muihin Apple-pohjaisiin palveluihin. Laite myös vapauttaa PWA-sovelluksen käyttämän tilan muutaman viikon kuluttua, mikäli käyttäjä ei käytä sovellusta. (Mt.)

PWA-sovelluksien käytössä iOS-laitteilla on vielä ongelmia verrattuna Android-laitteisiin. Käyttäjälle ei voi antaa selaimesta mitään indikaatiota siitä, että sivusto on PWA-sovellus. Sovelluksen tallentaminen vaatii käyttäjältä ”Jaa” ja ”Lisää aloitusnäyttöön” -nappien manuaalista painamista toisin kuin Android-laitteella, jossa verkkosovellus voi antaa käyttäjälle ilmoituksen selaimen kautta. PWA-sovelluksia ei myöskään voi asentaa muiden sovelluskaupasta löytyvien selaimien kuten Chromen, Firefoxin, Braven tai Edgen kautta. Käyttäjä voi myös asentaa sovelluksen useampaan kertaan, jolloin aloitusnäytöllä näkyy kaksi sovelluksen ikonia. Sovelluksen tiedostot jaetaan kuitenkin asennuksien kesken. 3D touch -valikko ei myöskään toimi PWA-sovelluksien ikoneiden kanssa. (Mt.)

3.1.4 iOS-laitteen What Web Can Do Today -testi

Testilaitteena toimi iPhone SE iOS-versiolla 12.2 ja selaimina Chrome versio 73.0.3683.68 ja Safari. Testiin käytettiin What Web Can Do Today -nimistä sivustoa kuten Android-laitteen testissä. Kuviossa 3 vihreällä merkityt laitteen ominaisuudet ovat PWA-sovelluksen käytettävissä Chrome-selaimessa ja punaisella merkityt eivät ole. Testien perusteella Safarilla on pääsy seuraaviin ominaisuuksiin, joihin Chromella ei ole pääsyä: Audio & Video Capture ja Offline-mode.



Kuvio 3. iOS-laitteen testi Chrome-selaimessa (What Web Can Do Today, 2019)

3.2 PWA työpöytäympäristössä

3.2.1 Tuki eri laitteilla

PWA-sovelluksia tuetaan tällä hetkellä Chrome-selaimessa Chrome OS:ssä (Chrome versio 67+), Linuxissa (Chrome versio 70+) ja Windowsissa (Chrome versio 70+). Tuki on myös tullut MacOS:lle Chrome version 73 myötä. PWA-sovellukset toimivat työpöytäympäristössä samalla periaatteella kuin mobiilialustallakin eli ne voidaan niin sanotusti asentaa laitteelle ja käyttää selaimesta erillisessä ikkunassa. Service workerin ja välimuistin ansiosta sovelluksen toiminta on luotettavaa. Vaikka mobiililaitteiden käyttö kasvaa vahvasti, niiden huiput käyttäjämäärissä tapahtuvat

aamuisin ja iltaisin, kun taas työpöytälaitteita käytetään tasaisemmin koko päivän aikana ja enemmän toimistoaikoina, kun ihmiset ovat työpaikallaan. (LePage, 2019.)

3.2.2 Windows

PWA-sovelluksen jakaminen Microsoft Storen kautta tuo sovelluksen yli 600 miljoonan Windows 10 käyttäjän ulottuville. Asennettuna PWA-sovellus toimii Windows 10:ssä Universal Windows Platform -sovelluksen tavoin. Windows 10 ympäristössä toimiessaan PWA-sovellus hyötyy seuraavista asioista kuten: sovelluksen oma ikkuna, selaimesta erillään oleva prosessi, ei rajoja säilytykselle, yhteydettömän tilan ja tausta-ajon prosessit, pääsy natiiveihin Windows Runtime (WinRT) -rajapintoihin JavaScriptin avulla, sovelluksen näkyminen Windowsin aloitusvalikossa sekä Cortanan hakutuloksissa. (McCormick, Navarra, Purwar & Wojciakowski, 2018.)

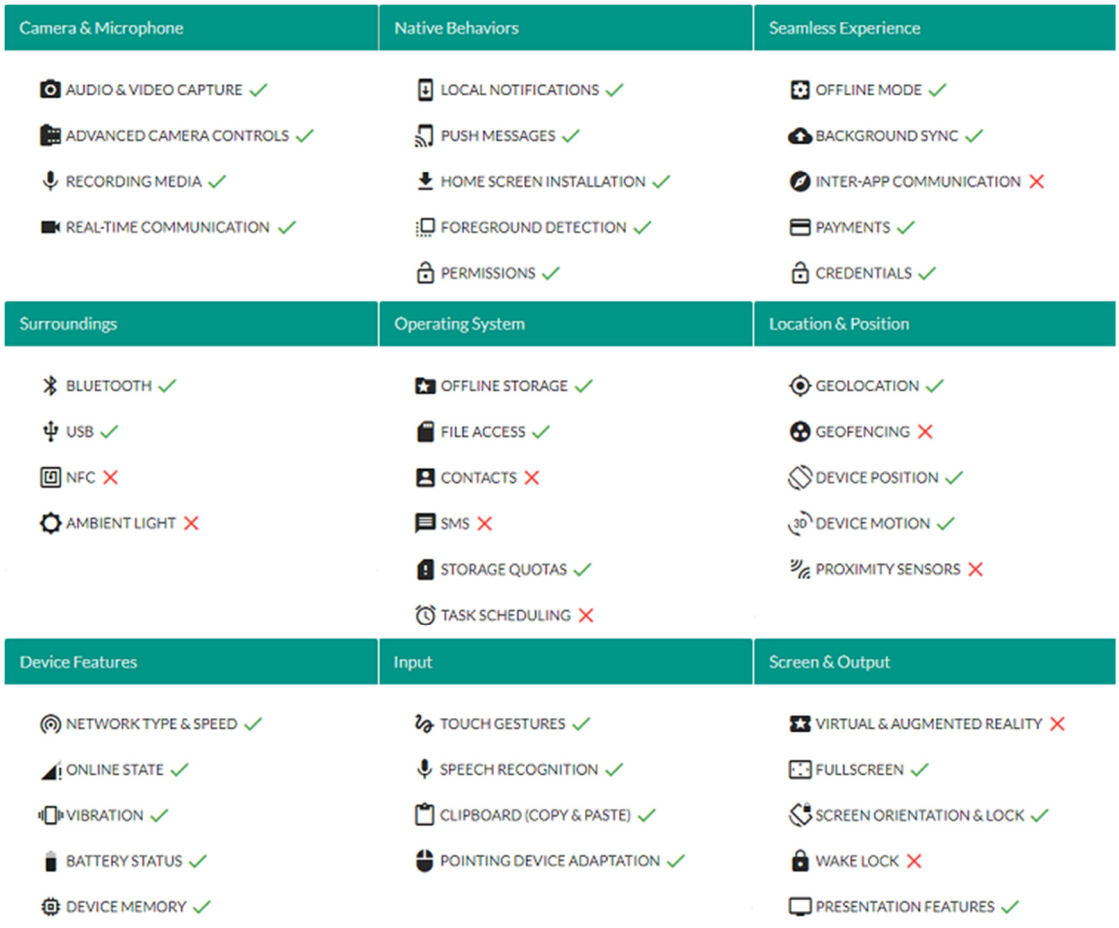
Windows 10 ympäristössä toimiakseen PWA-sovellus tarvitsee vähintään:

- HTTPS-salatun liikenteen
- Service Worker -skriptin
- Web App Manifest -tiedoston
- Parempaan käyttökokemukseen PWA-sovelluksessa olisi hyvä olla:
    - Yhteensopivuus eri selaimilla
    - Responsiivinen käyttöliittymä
    - Deep linking, eli jokaisen sovelluksen sivun reitittäminen uniikkiin URL-osoitteeseen

(Mt.)

3.2.3 Windows 10 -laitteen What Web Can Do Today -testi

Testilaitteena toimi Lenovon kannettava tietokone Windows 10 käyttöjärjestelmällä ja selaimina Chrome versio 71.0.3578.98 ja Firefox versio 64.0.2. Testiin käytettiin Android- ja iOS-laitteiden testien tapaan What Web Can Do Today -nimistä sivustoa. Kuviossa 4 vihreällä merkityt laitteen ominaisuudet ovat PWA-sovelluksen käytettävissä Windows 10 -laitteella Chrome-selaimessa ja punaisella merkityt eivät ole.



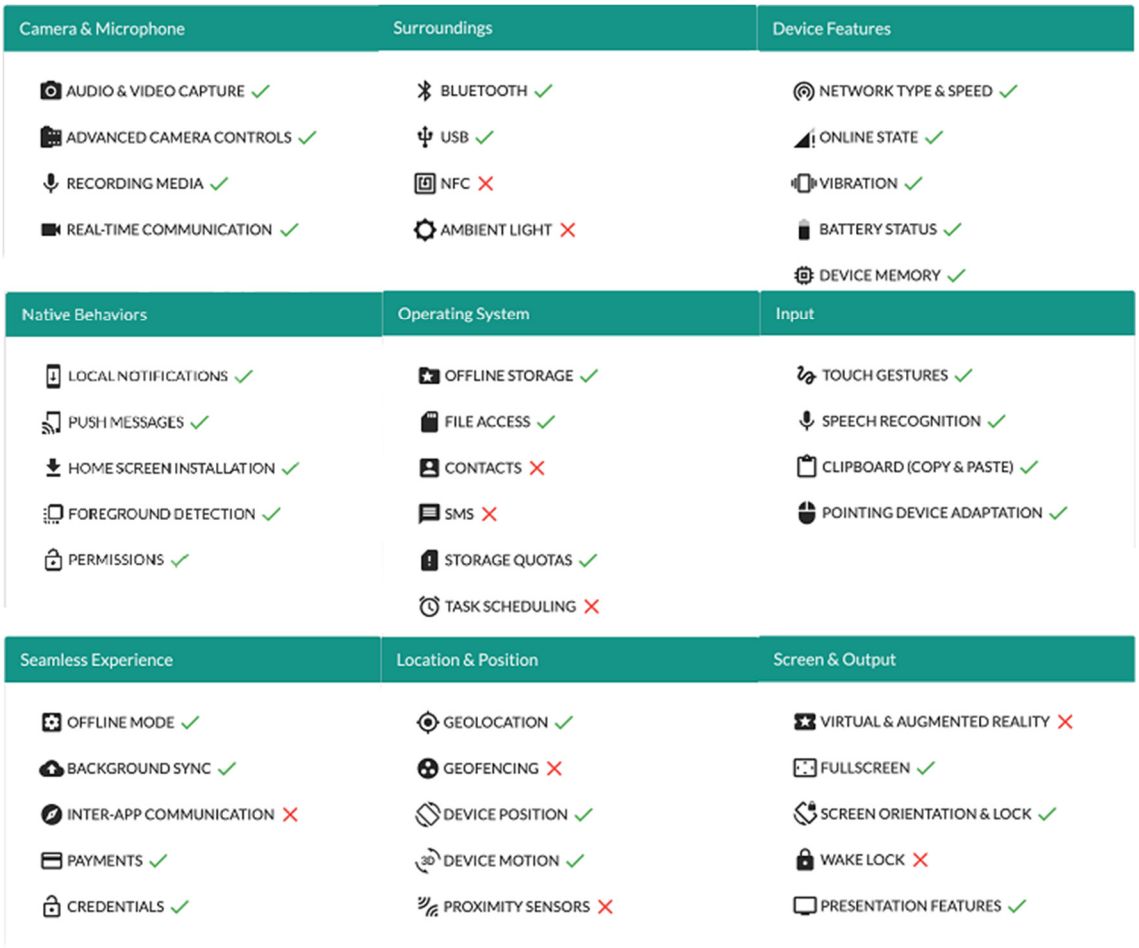
Kuvio 4. Windows 10 -laitteen testi Chrome-selaimessa (What Web Can Do Today, 2019.)

Testien perusteella Firefox-selaimella ei ole pääsyä seuraaviin ominaisuuksiin, joihin Chrome-selaimella on: Advanced Camera Controls, Home Screen Installation, Background sync, Payments, Credentials, Network Type & Speed, Battery Status, Device Memory, Speech Recognition, Presentation Features. Firefoxilla on pääsy seuraaviin ominaisuuksin, joihin Chromella ei ole: Ambient Light, Proximity Sensors. (What Web Can Do Today, 2019.)

3.2.4 MacOS-laitteen What Web Can Do Today -testi

Apple lisäsi Service Worker tuen Safari-selaimen versioon 11.1 (Gazdecki, 2018). Safari ei kuitenkaan tällä hetkellä tue Web App Manifest -tiedostoa, vaan tuki on kehitteillä (WebKit Feature Status, 2019). MacOS sai kuitenkin täydellisen PWA-tuen Chrome-selaimen version 73 avulla (LePage, 2019). Testilaitteena toimi Macbook Pro Mid 2015, jonka käyttöjärjestelmänä oli macOS versio 10.14.3. Testaukseen

käytettiin Chrome-selaimen versiota 73.0.3683.86 ja Safari-selaimen versiota 12.0.3. Kuviossa 5 vihreällä merkityt laitteen ominaisuudet ovat PWA-sovelluksen käytettävissä macOS-laitteella Chrome-selaimessa ja punaisella merkityt eivät ole.



Kuvio 5. MacOS-laitteen testi Chrome-selaimessa (What Web Can Do Today, 2019.)

Testien perusteella Safari-selaimella ei ole pääsyä seuraaviin ominaisuuksiin, joihin Chromella on: Advanced Camera Controls, Recording Media, Background Sync, Credentials, Bluetooth, USB, Storage Quotas, Device Position, Device Motion, Screen Orientation & Lock, Presentation Features, Speech Regocnition, Touch Gestures, Home Screen Installation ja Persmissions. (What Web Can Do Today, 2019.)

4 Toteutetut PWA-sovellukset

Tutkimalla ja seuraamalla toteutettujen PWA-sovellusten toimintaa ja käyttöönoton jälkeisiä tuloksia voidaan nähdä yleistä kuvaa siitä, mitä hyötyjä ja haittoja PWA-

teknologian käytöstä on. Paremman kuvan avulla voidaan tehdä helpommin päätöksiä siitä, kannattaako PWA-teknologiaa harkita esimerkiksi React Nativen sijasta mobiilikehityksessä. PWA-teknologia tarjoaa monia työkaluja, joiden avulla voidaan parantaa verkkosovelluksien toimintaa ja käyttökokemusta, mikä voi tarjota asiakkaille uutta ja tuoda sovellukseen enemmän arvoa. Mahdollista lisäarvon tuomista voi arvioida tutustumalla toteutettuihin PWA-sovelluksiin.

4.1 Twitter Lite

Twitterin noin 330 miljoonasta käyttäjästä 80 prosenttia käyttää palvelua mobiililaitteella. Twitter julkaisi Twitter Lite -nimisen PWA-toteutuksen ohjelmistostaan huhtikuussa 2017. Twitter Liten tavoitteena on lyhentää latausaikoja, lisätä käyttäjän sitouttamista (user engagement) ja vähentää datankäyttöä. Twitter saavutti PWA-toteutuksellaan 65 prosentin lisäyksen sivujen määrässä sessiota kohden ja 75 prosentin lisäyksen tweettien määrässä. Myös bounce rate eli prosentuaalinen osuus kävijöistä, jotka vain vilkaisevat sivua, laski 20 prosenttia. Twitter Lite on suorituskyvyltään kilpailukykyinen, kun sitä verrattaan natiiviin Android sovellukseen, mutta se käyttää ainoastaan 3 prosenttia natiivisovelluksen tarvitsemasta tallenustilasta. (Twitter Lite PWA - -, 2017.)

Twitter saavutti 250 000 uniikin päivittäisen käyttäjän käyttäjämäärän lisättyään sovellukseen Add to Homescreen eli ”Lisää aloitusnäyttöön” -kehotuksen, joka lisää sovelluksen kuvakkeen laitteen aloitusnäyttöön natiivisovelluksen tavoin. Käyttäjät avasivat sovelluksen keskimäärin neljä kertaa päivässä, mikä on auttanut käyttäjien sitouttamisessa mobiiliverkossa. PWA-sovellus voi lähettää push-viestejä eli ilmoituksia, samalla tavalla kuin natiivisovellus myös silloin kun laitteen selain on suljettu. Twitter Lite lähettää käyttäjille noin 10 miljoonaa ilmoitusta päivässä. (Mt.)

Twitter Lite turvautuu välimuistiin tallennettuun tietoon mahdollisimman paljon, mikä auttaa pitämään datasiirron vähäisempänä. PWA-toteutus optimoi myös kuvia, mikä on auttanut vähentämään käyttäjien datan käyttöä jopa 70 prosenttia. Vähäinen datan käyttö ja vähemmän suorituskykyä vaativa sovellus auttaa tavoittamaan

käyttäjiä myös maista, joissa verkon nopeus on hitaampi tai epäluotettavampi ja laitteiden suorituskyky huonompaa. (Mt.)

Twitter Lite käynnistyy ensimmäisellä kerralla noin viidessä sekunnissa 3G-verkossa useimmilla laitteilla ja seuraavat käynnistykset ovat lähestulkoon välittömiä hitaammassakin verkossa. Sovellus lataa palvelimen vastauksen vähitellen selaimeen lähettäen samalla ohjeita kriittisten resurssien ennalta lataamiseen. Resurssit jaetaan pieniin osiin siten, että ensimmäinen lataus tarvitsee ainoastaan näytöllä näkyvän sisällön resurssit. Service worker tallentaa muut resurssit välimuistiin, mikä mahdollistaa nopean navigoinnin toisiin näkymiin. Välimuistin käyttö on vähentänyt käyttäjien latausaikoja 30 prosenttia ja 50 prosenttia Time To Interactive -aikaa. (Mt.)

4.2 Tinder

Tinderin toteuttaman PWA-sovelluksen MVP-toteutus tehtiin kolmessa kuukaudessa käyttämällä Reactia UI-kirjastona ja Reduxia tilan hallintaan (state management). Lopputuloksena PWA-sovellus tarvitsee vain 10 prosenttia tilan määrästä, jonka vanha sovellus käytti. Natiivin Android sovelluksen koko oli noin 30 MB ja PWA-sovelluksen koko noin 2,8 MB. Tinder PWA-sovelluksen Time To Interactive -aika on alle 5 sekuntia, johon päästiin käyttämällä reitityspohjaista koodin jakoa, suorituskyky budjettia, late-discovered resurssien ennakkolatausta ja pitkäaikaista resurssien säilyttämistä välimuistissa. PWA-sovelluksen myötä huomattiin, että sovelluksen toimintoja käytettiin enemmän verkossa kuin natiivi sovelluksissa, viestejä lähetettiin enemmän verkossa kuin natiivisovelluksissa, ostosten määrä pysyi samana verkossa kuin mobiilissa, profiileja muokattiin enemmän verkossa kuin natiivisovelluksissa ja se, että käyttäjät viettivät aikaa enemmän sovelluksessa verkossa kuin natiivisovelluksissa. (Osmani, 2017.)

Koodin jakaminen

Tinder pystyi Route-level code-splitting -menetelmällä jakamaan isoja JavaScript-paketteja, jotka hidastivat sovelluksen lataamista käyttövalmiiksi. Paketit sisälsivät

koodia, jota ei tarvittu heti käynnistyksen yhteydessä, joten tarvittava koodi voitiin ladata ensin ja muu koodi myöhemmin. Koodin jakamiseen käytettiin React Routeria ja React Loadablea. Jakamisella saavutettiin JavaScript-pakettien koon pienentyminen noin 40 prosenttia 166 KB:sta 101 KB:iin, DOMContentLoad ajan lyheneminen noin 15 prosenttia 5,46 sekunnista 4,69 sekuntiin ja latausajan lyheneminen noin 60 prosenttia 11,91 sekunnista 4,69 sekuntiin. (Mt.)

Pitkäaikainen resurssien säilyttäminen välimuistissa ja late-discovered resurssien ennakkolataus

Resurssien säilyttämistä pitkäaikaisesti välimuistissa haittasi Tinderin käyttämä chunkhash, joka mitätöi välimuistin, kun ulkopuolisiin kirjastoihin tuli muutoksia. Jotta resurssien säilyminen pidempiä aikoja olisi mahdollista, Tinder alkoi määritellä ns. valkolistaa (whitelist) ulkopuolisista riippuvuuksista ja erotti niiden Webpack manifestin sovelluksen pääosasta. Molempien osien kooksi jäi noin 160 KB. Tinder toteutti myös kriittisten JavaScript- ja Webpack-pakettien ennakkolatauksen (precaching). Single-page-sovelluksen tapauksessa resurssit voivat olla esimerkiksi JavaScript-paketteja. Precaching johti latausajan lyhenemiseen sekunnilla ja first paint -ajan lyhenemistä 1000 millisekunnista 500 millisekuntiin. (Mt.)

Suorituskykybudjetti

Halutun suorituskyvyn takaamiseksi mobiililaitteilla, Tinderillä otettiin käyttöön suorituskykybudjetti. Budjetti tarkoitti ennalta sovittua tiedostojen kokoa. Budjetin alaisiin tiedostoihin kuului main chunk ja vendor chunks, joiden koko sai olla noin 155 KB, asynchronous chunks, joiden koko sai olla noin 55 KB, other chunks, joiden koko sai olla 35 KB ja CSS tiedostot, joiden koko sai olla 20 KB. (Mt.)

4.3 Nikkei

Nikkei tunnisti sivustollaan nopean kasvun mobiililaitteiden osalta vierailijoiden siirryttyä käyttämään enenevissä määrin mobiililaitteita sivuston selaamiseen. Nikkein

tekemän auditoinnin jälkeen huomattiin, että sivustoa ei oltu täysin optimoitu mobiilikäyttöön, joten sivuston lataus oli hidasta. Sivuston interaktiivisuuteen meni 20 sekuntia, joka oli paljon ottaen huomioon, että 53 prosenttia mobiilikäyttäjistä hylkää sivuston, mikäli sen lataus kestää yli 3 sekuntia. (Nikkei - -, 2018.)

Nikkei alkoi kehittää ongelmaan ratkaisuksi PWA-toteutusta, joka käyttää responsiivista suunnittelua, vakio JavaScriptiä ja useamman sivun arkkitehtuuria, tavoitteena antaa käyttäjille miellyttävä käyttökokemus. Service worker -skriptin avulla Nikkei pystyi tarjoamaan ennalta-arvattavan suorituskyvyn verkon laadusta huolimatta. Skripti takaa myös sen, että pääartikkelit ovat aina saatavilla ja ne latautuvat lähestulkoon välittömästi välimuistin ansiosta. Web App manifest -tiedoston ja service worker -skriptin ansiosta käyttäjä pystyy tallentamaan sovelluksen laitteen alkunäyttöön samalla tavalla kuin natiivisovelluksenkin. (Mt.)

Sovelluksen nopeampaan toimintaan pystyttiin vaikuttamaan seuraavilla teknologioilla ja tekniikoilla:

- http/2 Server Push
- Priorisoidaan käyttäjälle kriittisiä JavaScript- ja CSS-tiedostoja
- Kolmannen osapuolen resurssien ennakkokysely resolverilta (pre-resolve DNS/TCP/SSL handshake)
- Mahdollistaa välittömän navigoinnin sivuilla
- Inline Critical-path CSS
- 0 renderöinnin estävää tyylitiedostoa
- JavaScript-pakettien uudelleenkirjoitus ja moderni pakettien optimointi
- JavaScript-pakettien koko tippui 300 KB:stä 60 KB:iin

(Mt.)

Lopputuloksena PWA-toteutus johti 14 sekuntia parempaan Time To Interactive -aikaan, 75 prosenttia nopeampaan latausaikaan, 2,3 kertaiseen ogranic traffic määrään, eli käyttäjiin, joita ei ohjattu sivustolle toiselta sivustolta tai mainoksesta, 58 prosenttia korkeampaan maksavien käyttäjien määrään, 49 prosenttia korkeampaan

päivittäisten käyttäjien määrään ja kaksinkertaiseen määrään sivujen katsontakertoja sessiota kohden. (Mt.)

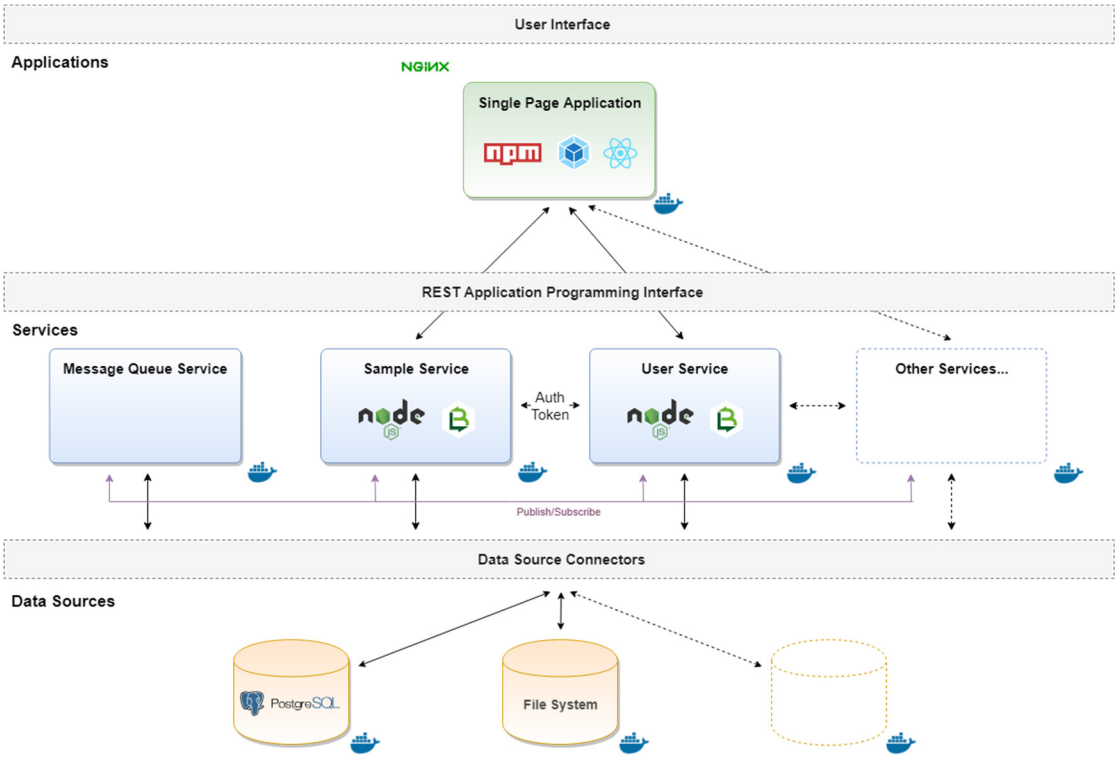
5 PWA-ominaisuuksien lisäys Nolwenture Baseen

5.1 Teknisen toteutuksen tarkoitus

Teknisen toteutuksen tarkoitus oli selvittää miten Nolwenture Base-verkkosovellukseen voidaan lisätä PWA-ominaisuuksia, kuten offline-toimivuus, push-ilmoitukset ja taustasynkronisaatio. Tutkimuksen tuloksien avulla Base-sovellusta voidaan kehittää haluttuun suuntaan ja ottaa huomioon PWA-teknologian tarpeita sekä vaatimuksia jo hyvissä ajoin kehityksen aikana. Lisäksi toteutuksen tulosten avulla voidaan arvioida PWA-toiminnallisuuksien toteuttamiseen tarvittavaa työtä.

5.2 Nolwenture Base

Nolwenture Base on kasvava toisiinsa yhdistyneiden ns. boilerplate-osien eli toistuvien ohjelmisto-osien kokonaisuus. Ohjelmistojen osat on kehitetty käyttämällä samankaltaisia työtapoja ja ideologiaa, mikä mahdollistaa monimutkaisten järjestelmien nopean kehittämisen. Basen arkkitehtuuri koostuu mikropalveluista (microservice) sekä mobiili- ja verkkosovellusasiakasohjelmista. Basen ohjelmisto-osat pääasiallisesti toteutettu käyttämällä JavaScript-, Node.js-, Loopback-, React-, React Native, PostgreSQL-, RabbitMQ- ja Docker-teknologioita. Microservice-pohjainen arkkitehtuuri mahdollistaa kokonaisuuden osien muokkaamisen ja päivittämisen vaivattomammin, koska yksittäinen palvelu voidaan vaihtaa tarpeen vaatiessa toiseen ilman koko kokonaisuuden muokkaamista. Mikropalveluarkkitehtuuri helpottaa myös kieleltään poikkeavien palveluiden, kuten koneoppimispalveluiden (machine learning), jotka yleensä toteutetaan Python- tai R-kielellä, sisällyttämisen muuhun palvelukokonaisuuteen. Kuviossa 6 on esitetty Nolwenture Basen arkkitehtuuri.



Kuvio 6. Nolwenture Base rakenne

5.3 Teknisen toteutuksen läpikäynti

PWA-toiminnallisuus toteutettiin käyttämällä Googlen kehittämää Workbox JavaScript-kirjastoa, koska käytössä oli myös moduulien paketointiin käytettävä Webpack. Workbox hoitaa tiedostojen käsittelyn Webpackin kanssa, joten käyttäjän ei tarvitse huolehtia siitä, että tehdyt muutokset ovat mukana Webpackin konfiguraatiossa. Workbox-kirjaston käytössä on kaksi liitännäisvaihtoehtoa: generateSW ja injectManifest, joista ensimmäiseksi kokeiltiin generateSW-liitännäistä.

GenerateSW

GenerateSW-liitännäisen avulla oli helppo ottaa käyttöön service workerin tarjoama välimuistin ennakkokäyttö (precache), ajonaikainen välimuistin käyttö (runtime cache) ja offline-toiminnallisuus. Sen käyttöön ottamiseksi tarvitsi lisätä Webpackin config-tiedostoon liitännäiset (plugins) kohtaan kuviossa 7 näkyvät rivit. Kyseiset määritykset ottavat käyttöön generateSW-liitännäisen ja sille määritykseksi kuvien poissulkeminen (exclude) precachesta, runtime cachen käyttö kuvien kohdalla,

cacheFirst-strategian käyttö, välimuistin nimeäminen images-nimiseksi ja välimuistiin tallennettavien kuvien määrän rajoittaminen kymmeneen.

```
new WorkBoxPlugin.GenerateSW({
    swDest: 'sw.js',
    exclude: [/\.(?:png|jpg|jpeg|svg)$/],
    runtimeCaching: ({
        urlPattern: [/\.(?:png|jpg|jpeg|svg)$/],
        handler: 'cacheFirst',
        options: {
            cacheName: 'images',
            expiration: {
                maxEntries: 10
            }
        }
    })
})
```

Kuvio 7. GenerateSW

GenerateSW:n avulla saatiin vaivatta käyttöön hyödyllisiä PWA-ominaisuuksia. Precache mahdollistaa nopeamman sivuston lataamisen peräkkäisillä käynneillä, koska resursseja ei välttämättä tarvitse ladata uudelleen verkon yli. Runtime cache taas auttaa tekemään sivustosta nopeamman, kun kertaalleen ladattuja resursseja ei tarvitse välttämättä ladata uudelleen, mikäli ne eivät ole muuttuneet. GenerateSW-liitännäinen ei kuitenkaan ole kustomoitavissa, mistä syystä se vaihdettiin InjectManifest-liitännäiseen.

InjectManifest

InjectManifest-liitännäinen tukee esimerkiksi taustasynkronisaatiota (backgroundsync) ja tarkempaa reittien määrittelyä (routing) sekä antaa kehittäjälle mahdollisuuden kirjoittaa service worker -skriptin kokonaan tai osin itse. InjectManifest-liitännäinen lisätään samalla tavalla Webpackin config-tiedostoon, kuin generateSW-liitännäinen. InjectManifestin tapauksessa tiedostoon ei tarvinnut lisätä kuin kuviossa 8 näkyvät rivit. InjectManifest ei välttämättä tarvitse muita määrityksiä kuin service worker -skriptin osoitteen, koska koko toiminnallisuus rakennetaan

kyseiseen tiedostoon. Tiedosto täytyy luoda manuaalisesti ja polun täytyy täsmätä tiedoston osoitteeseen tai muuten injectManifest ei voi toimia.

```
new WorkBoxPlugin.InjectManifest({
    swSrc: './src/src-sw.js'
})
```

Kuvio 8. InjectManifest

Precache

Precache eli eräänlainen ennakkovälimuisti on service workerin ominaisuus, joka mahdollistaa resurssien tallentamisen välimuistiin service workerin asennusvaiheessa. Precache on yksi tärkeä osa sovelluksen offline-toimivuutta. Workbox-precache liitännäinen tarkastaa välimuistiin tallennetut tiedot jokaisella vierailukerralla ja päivittää muuttuneet tiedot lataamalla ne uudelleen ja poistamalla vanhat, mikäli tiedostojen versionumero (revisioning) on muuttunut. Kyseinen ominaisuus saadaan injectManifest-liitännäisen tapauksessa käyttöön lisäämällä service worker -skriptiin rivi:

workbox.precaching.precacheAndRoute(self.__precacheManifest);

Workbox lisää eli injektoi service worker -skriptiin niin sanotun precache-manifest listan sovelluksen käynnistyessä. Precache-manifest kertoo service workerille, mitä tiedostoja tallennetaan precacheen. Precache-manifestia ei pidä sekoittaa kohdassa 2.4 läpikäytyyn Web App Manifestiin.

Runtime cache

Runtime cache saatiin käyttöön käyttämällä Workboxin routing-moduulia. Moduulin käyttö oli yksinkertaista, koska sille tarvitsi vain antaa osoite, jota kuunnella ja yksi strategia, joita ovat Stale-While-Revalidate, Cache First, Network First, Network only ja Cache only, workbox-strategies-moduulista.

Stale-While-Revalidate (myös Cache Then Network) vaihtoehto mahdollistaa nopean vastauksen, koska se käyttää ensisijaisesti välimuistista löytyviä resursseja. Mikäli välimuistista ei löydy tarvittavaa resurssia, se ladataan verkosta tai muuttuneen resurssin tapauksessa päivitetään välimuistiin. Stale-While-Revalidate kuitenkin käyttää välimuistista löytyvää resurssia, vaikka se olisi muuttunut, joten kyseinen vaihtoehto ei sovi sellaisiin tilanteisiin, joissa käyttäjälle on tarkoitus näyttää kaikkein tuoreimmat resurssit (Workbox Strategies 2019). Tämä toiminnallisuus ilmeni toteutuksen aikana esimerkiksi poistettaessa sisältöä selaimella käyttöliittymän kautta, minkä jälkeen poistettu sisältö hävisi selaimesta, mutta palasi uudelleen sivun päivityksen jälkeen ja hävisi lopullisesti toisen päivityksen jälkeen. Tietokannasta sisältö poistui heti kun se poistettiin käyttöliittymässä, mutta service worker tarjosi vanhaa sisältöä välimuistista ennen kuin välimuisti päivittyi sivun päivityksen yhteydessä.

Cache First ja Network First vaihtoehdot toimivat nimensä mukaisesti siten, että service worker käyttää resursseja ensisijaisesti välimuistista tai verkosta. Mikäli ensisijainen lähde ei ole saatavilla, käytetään toista lähdettä. Cache First toimii samalla periaatteella kuin Stale-While-Revalidate, mutta siinä ei tarkasteta välimuistia ja ladata muuttuneita tai poistettuja tiedostoja uudelleen verkon yli. Network First vaihtoehtoa on hyvä käyttää sellaisille resursseille, jotka päivittyvät tiuhaan. Network Only ja Cache Only käyttävät ainoastaan valittua lähdettä resursseille. Tällöin Network Only vaihtoehdolla resurssit ladataan määritetylle reitille ainoastaan verkon yli, eikä niitä käytetä missään vaiheessa välimuistista. Cache Only käyttää pelkästään välimuistia ja on hyödyllinen, jos haluaa käyttää omaa kustomoitua precache-toiminnallisuutta. (Workbox Strategies 2019).

Reitityksen toteutus

Sovellukseen määritettiin viisi eri reittiä, jotka näkyvät kuviossa 9. Ensimmäisenä määritetään reitti service worker -skriptin tallentamiseksi cacheen, koska muuten service worker ei olisi saatavilla offline-tilassa ja sovelluksen offline-käyttö olisi siten mahdotonta. Seuraavaksi määritetään reitti, jotta sisäänkirjautumisen validointi on mahdollista myös offline-tilassa. Cachelle määritettiin myös oma nimi validation-Cache, jotta sen sisältöä oli helpompi seurata. Seuraavat kaksi reittiä hoitavat kahden

muun sivun tallentamisen cacheen. Viimeisenä määritetään sivulla /user olevan käyttäjälistan tallentaminen cacheen, jotta lista olisi saatavilla myös offline-tilassa ja cachelle annetaan nimi userListCache. Tässä tapauksessa käytettiin networkFirst-strategiaa edellä mainittujen Stale-While-Revalidate ominaisuuksien takia, jotta näytettävä käyttäjälista olisi aina ajan tasalla.

```
workbox.routing.registerRoute(
    new RegExp('/src-sw.js'),
    workbox.strategies.staleWhileRevalidate()
);

workbox.routing.registerRoute(
    new RegExp('/user/Sessions/validate'),
    workbox.strategies.staleWhileRevalidate({
        cacheName: 'validationCache'
    })
);

workbox.routing.registerRoute(
    new RegExp('/another'),
    workbox.strategies.staleWhileRevalidate()
);

workbox.routing.registerRoute(
    new RegExp('/user$'),
    workbox.strategies.staleWhileRevalidate()
);

workbox.routing.registerRoute(
    new RegExp('/api/user/Users$'),
    workbox.strategies.networkFirst({
        cacheName: 'userListCache'
    })
);
```

Kuvio 9. Workbox routing

Workboxin avulla pystyi myös määrittämään nimet eri cacheille (kuvio 10). Tässä tapauksessa precachelle määritettiin nimeksi install-time ja runtime cachelle runtime. Prefix on ennakkoliite, joka lisätään cachen nimen eteen, jolloin cachen nimeksi tulee esimerkiksi nolwenture-base-runtime. Reittien määrittelyssä annetut nimet ylikirjoittavat setCacheNameDetails kohdassa määritetyt nimet.

```
workbox.core.setCacheNameDetails({
    prefix: 'nolwenture-base',
    precache: 'install-time',
    runtime: 'runtime'
})
```

Kuvio 10. Välimuistien nimeäminen

Ongelmia reitityksessä

Reitityksien määrittämisen jälkeen huomattiin, että kirjautuminen ulos ei toiminut, vaan istunto pysyi edelleen voimassa ja sivustoa pystyi selaamaan normaalisti. Syy löytyi reitistä /user/Sessions/validate. Kirjautumistiedot pysyivät tallessa cachessa ja service worker vastasi kirjautumisen validointiin välimuistista löytyvillä resursseilla. Ongelmaan ratkaisuksi täytyi kehittää lyhyt logiikka kuuntelijalla (kuvio 11), joka kuunteli pyyntöjä osoitteeseen /api/user/Users/logout, johon sovellus lähetti pyynnön, kun käyttäjä painoi uloskirjautumista. Pyynnön jälkeen validationCache yksinkertaisesti poistetaan ja käyttäjän uloskirjautuminen onnistuu, koska kirjautumistietoja ei ole enää välimuistissa.

```
self.addEventListener('fetch', (event)=>{
    if(event.request.url.endsWith('/api/user/Users/logout')){
        caches.delete('validationCache')
            .catch((error)=>console.log(error))
    }
})
```

Kuvio 11. Uloskirjautumisen kuuntelija

Taustasynkronisaatio

Background Sync API eli taustasynkronisaatio rajapinta auttaa parantamaan sovelluksen toimivuutta ja käytettävyyttä, mikäli laitteella on huonolaatuinen verkkoyhteys tai ei yhteyttä ollenkaan. Rajapinta toimii service workerin kanssa tallentaen epäonnistuneet pyynnöt ns. jonoon (queue), josta pyynnöt lähetetään uudelleen verkkoyhteyden palauduttua (replay). Rajapinta kuuntelee sync-tapahtumaa, jonka selain lähettää, kun se näkee olevansa yhteydessä verkkoon. Tapahtuman jälkeen

Background Sync yrittää lähettää jonossa olevat pyynnöt uudelleen. (Karlin & Kruisselbrink, 2018).

Taustasynkronisaation testaus oli tärkeää, koska se on iso osa offline-käytön parantamista. Kuviossa 12 näkyy Workboxin tarjoama valmis liitännäinen taustasynkronisaatiolle, jonka lisäksi täytyi määrittää reitti. Tässä tapauksessa reitti käsitti koko sivuston, mutta se kuunteli ainoastaan POST-metodilla lähetettyjä pyyntöjä. Liitännäiseen lisätty määritys *maxRetentiontTime* määrittää kuinka kauan pyyntöjä säilytetään jonossa. Tässä tapauksessa pyynnöt säilyvät vuorokauden ajan.

```
const bgSyncPlugin = new workbox.backgroundSync.Plugin('backgroundQueue', {
    maxRetentionTime: 24*60 //24h
});

workbox.routing.registerRoute(
    new workbox.strategies.NetworkOnly({
        plugins: [bgSyncPlugin]
    }),
    'POST'
);
```

Kuvio 12. Taustasynkronisaation liitännäinen

Taustasynkronisaation testauksessa ongelmaksi muodostui se, että Chromen kehittäjätyökaluista network-välilehdestä valittaessa offline-vaihtoehto vaikuttaa ainoastaan selaimesta lähteviin pyyntöihin, mutta ei service workerin kautta kulkeviin pyyntöihin. Tämän takia taustasynkronisaation testaus oli mahdotonta, koska pyynnöt eivät missään vaiheessa epäonnistuneet oikealla tavalla. Vaihtoehtona olisi ollut myös tietokoneen tai palvelimen verkkoyhteyden katkaiseminen, mutta koska koko kehitysversio sovelluksesta toimi paikallisesti tietokoneella, ei verkkoyhteydellä ollut toiminnan kannalta merkitystä.

Toinen Workboxin tarjoama vaihtoehto oli käyttää kuuntelijaa kuuntelemaan fetch-tapahtumia (kuvio 13), kopioimaan pyyntö request.clone()-metodilla ja lisäämään pyyntö jonoon, mikäli se epäonnistuu. Lisäämällä logiikkaan vielä kuunneltavaksi metodiksi ”POST” saatiin kiinni ainoastaan halutut pyynnöt.

```
self.addEventListener('fetch', (event)=>{
    const promiseChain = fetch(event.request.clone())
        .catch(()=>{
            return queue.addRequest(event.request);
        })
    event.waitUntil(promiseChain);
});
```

Kuvio 13. Fetch-metodi kuuntelija

Toiminnon testaamiseksi kaadettiin tietty Docker-kontti, jonka jälkeen pyynnöt eivät menneet enää perille. Pyynnön vastauksena saatiin 503 Service Unavailable, joten pyyntö ei epäonnistunut oikealla tavalla mennäkseen kuuntelijan catch-osioon. Tapahtuma saatiin kiinni lisäämällä then-osio ennen catch-osiota, jonka jälkeen epäonnistunut pyyntö saatiin kiinni ja lisättyä jonoon. Ongelmana tässä testaustavassa oli se, että selain ei ollut missään välissä yhteydettömässä tilassa, koska ainoastaan Docker-kontti oli alhaalla, joten jonossa olevia pyyntöjä ei lähetetty uudelleen. Myöskään Docker-kontin alasajon ja kehittäjätyökalujen offline-valinnan käyttäminen yhtä aikaa ei tuottanut haluttua tulosta. Jonoon tallennettujen pyyntöjen lähettäminen saatiin toimimaan satunnaisia kertoja, mutta ei ikinä tarpeeksi johdonmukaisesti. Parhaiten liitännäistä olisi voinut testata, mikäli sovellus olisi ollut erillisellä palvelimella.

Push-ilmoitukset

Push-ilmoitukset käyttävät Notifications API ja Push API rajapintoja. Notifications-rajapinta mahdollistaa järjestelmäilmoitusten näyttämisen käyttäjälle ja Push-rajapinta antaa service workerille mahdollisuuden käsitellä palvelimelta tulevia ilmoituksia sovelluksen ollessa toimettomassa tilassa. Molemmat rajapinnat on rakennettu Service Worker API -rajapinnan päälle. Service Worker -rajapinta reagoi push-ilmoitustapahtumiin ja hoitaa ilmoitusten välittämisen sovellukselle. (Introduction to Push Notifications, 2019.)

Push-ilmoituksien lähettämiseen tarvitaan käyttäjän suostumus. Suostumusta voidaan pyytää lisäämällä kuviossa 14 näkyvä osa samaan skriptiin, jossa service worker rekisteröidään. Kun käyttäjä avaa sivun, ruudulla näytetään pyyntö ilmoitusten

sallimisesta erillisessä ponnahdusilmoituksessa. Mikäli käyttäjä hyväksyy ilmoitusten vastaanottamisen, voi service worker jatkossa välittää push-ilmoituksia käyttäjän laitteeseen.

```
Notification.requestPermission((status)=>{
    console.log('Notification permission status:',status);
});
```

Kuvio 14. Push-ilmoituksien suostumuksen pyyntö

Push-ilmoitusten näyttämiseksi service worker -skriptiin lisättiin kuuntelija, joka kuuntelee push-tapahtumia. Chrome-selaimen kehittäjätyökalujen Application välilehden Service Workers valikosta voidaan lähettää push-ilmoituksia, jonka jälkeen kuuntelija reagoi push-tapahtumaan näyttämällä laitteella ilmoituksen. Ilmoitusta pystyy kustomoimaan antamalla sille otsikko ja lisämääritteitä kuten viestin teksti, viestin ikoni ja viestissä näkyvä kuva, värinä sitä tukevilla laitteilla sekä toimintoja. Toiminnoilla voi lisätä ilmoitukseen nappeja, joiden painamista voidaan kuunnella erillisellä notificationonclick-kuuntelijalla. Tapahtuman jälkeen voidaan suorittaa toimenpiteitä kuten käyttäjän siirtäminen tiettyyn osoitteeseen sivuston sisällä tai uuden ilmoituksen näyttäminen.

6 Tutkimustulokset ja johtopäätökset

6.1 PWA

Tutkimustyössä oli tavoitteena tutkia PWA-teknologian nykytilannetta ja sen toimivuutta eri laitteilla sekä tuottaa teknillinen toteutus toimeksiantajan verkkosovellukseen. Tutkimustyö toteutettiin keräämällä aineistoa teknologian dokumentaatiosta, jota on tarjolla lähinnä verkossa, käyttämällä laitteiden ja selaimen testaukseen tarkoitettua sivustoa sekä toteuttamalla PWA-toiminnallisuutta sovellukseen koodaamalla ja havainnoimalla tuloksia. Tutkimuksen avulla saatiin selvyyttä siitä, että PWA on teknologiana hyödyllinen ja sen avulla verkkosovelluksiin voidaan tuoda offline-toiminnallisuutta ja muita ominaisuuksia, jotka ovat aiemmin olleet mahdollisia lähinnä mobiilisovelluksissa. PWA:n toimintaperiaatteista selvisi se, että se tarvitsee

sitä tukevan selaimen ja service worker -skriptin, jonka tarkoitus on ”kaapata” sovelluksen pyyntöjä ja käsitellä niitä sekä tallentaa niitä cacheen, eli välimuistiin. Lisäksi tarvitaan niin sanottu Web App Manifest -tiedosto, jonka avulla käyttäjää voidaan kustomoida sovellus sopimaan yrityksen teemaan ja pyytää asentamaan sovellus laitteen työpöydälle tai kotinäyttöön mobiililaitteilla.

6.2 PWA eri ympäristöissä

PWA toimii tällä hetkellä parhaiten Android-pohjaisissa älypuhelimissa, koska suurin osa niistä käyttää Chromea selaimena. Applen iPhone puhelimissa Safarin PWA-tuki on kuitenkin aktiivisesti kehitteillä, joten tilanne voi parantua niiden osalta nopeasti. PWA-sovelluksen toteuttaminen mobiiliympäristöön natiivikielellä tai esimeriksi React Nativella kehitetyn sovelluksen sijasta ei kuitenkaan tällä hetkellä näytä parhaalta vaihtoehdolta juurikin selainten puutteellisen PWA-tuen takia. PWA-toiminnallisuuden lisääminen verkkosovellukseen on kuitenkin erinomainen tapa lisätä toiminnallisuutta ja parantaa käyttökokemusta myös mobiilisovelluksessa, joten PWA-teknologian käyttö on suositeltavaa, mutta ei kuitenkaan natiivisovelluksen korvaajana. Suurimman hyödyn mobiiliympäristössä PWA:sta saa tällä hetkellä, jos sen tarjoaa muiden mahdollisuuksien ohella tai tietää, että kohderyhmä käyttää PWA:ta tukevia laitteita ja selaimia. Tällainen voi olla mahdollista esimerkiksi yrityksen sisäiseksi sovellukseksi tarkoitetun sovelluksen tapauksessa, mikäli työntekijät käyttävät samoja laitteita. PWA-teknologialla voidaan tuottaa silloin sovellus, joka toimii samaan aikaan kaikilla laitteilla (tietokoneet ja älypuhelimet tai tabletit) ilman tarvetta ostaa tai kehittää erillistä sovellusta jokaiselle eri alustalle.

PWA:n toiminta työpöytäkäytössä on tällä hetkellä hyvin riippuvainen Google Chromesta. Tuki Safariin on kuitenkin kehitteillä ja tilanne voi muuttua macOS-laitteissa iOS-laitteiden lailla lähitulevaisuudessa, minkä jälkeen PWA:lla on jälleen enemmän käyttökohteita ja mahdollisuuksia. Työpöytäkäytössä PWA-teknologia mahdollistaa saman sovelluksen käytön eri laitteilla (Windows, MACOS ja Linux) kuten mobiiliympäristössäkin, joten tarvetta usealle eripohjaiselle sovellukselle ei ole tarvetta, eikä erillistä asennusta (pl. Sovelluksen asennus selaimesta) ja asetuksien

säätöä tarvita, koska sovellus toimii selaimessa ja on siten helposti liikutettavissa. PWA tarjoaa työpöytäympäristössäkin parempaa ja varmempaa offline-käyttömahdollisuutta, sekä pienempää tallennustilan käyttöä.

Kuitenkin kuten laitteiden testituloskuvioissa (kuviot 2,3,4 ja 5) nähdään, on laitteiden ominaisuuksien käyttömahdollisuudet rajallisia varsinkin iOS-pohjaisilla mobiililaitteilla. Tuki voi ulottua puuttuviin ominaisuuksiin hyvin nopeastikin, mutta tällä hetkellä samanlaisen käyttökokemuksen ja toiminnallisuuden tarjoaminen PWA-sovelluksen kautta mobiililaitteilla ei välttämättä onnistu tasapuolisesti Android- ja iOS-laitteilla. Sovelluksen toimivuus riippuu kuitenkin mitä sovellus tarvitsee toimiakseen, joten yksinkertainen sivusto ei välttämättä tarvitsekaan pääsyä esimerkiksi Bluetoothiin. PWA-sovelluksien yleistyminen mobiililaitteilla muussakin, kuin yksinkertaisten sovellusten tapauksessa, on kuitenkin riippuvainen laitteiden ominaisuuksien tuesta varsinkin, jos PWA-sovelluksella haluttaisiin korvata natiivisovelluksia.

6.3 Nolwenture Base toteutus

Teknisen toteutuksen tarkoitus oli testata PWA-toiminnallisuutta ja osoittaa siitä hyötyjä. Workbox-kirjasto tarjosi hyvän ja helposti käytettävän työkalun, jonka avulla eri ominaisuuksia pystyi testaamaan ilman aikaa vaativaa koodausta. Kirjasto tarjosi valmiit liitännäiset niille ominaisuuksille (offline-toimivuus, taustasynkronisaatio ja push-viestit), joita haluttiin kokeilla. Kirjaston käyttö oli tehokkaampi vaihtoehto sen sijaan, että ominaisuuksien koodaamiseen ja sen opetteluun olisi käytetty aikaa. Sama pätisi todennäköisesti oikeassa ohjelmistoprojektissa, mikäli kehittäjän ei ole tarvetta tehdä monimutkaisia toimintoja PWA-teknologialla.

Offline-toiminnallisuuden toteuttaminen ei ollut vaikeaa, kun käytössä oli niin sanotut precache- ja runtime cache -välimuistit. Välimuistien käyttö nopeutti sivuston latausaikoja sen jälkeen, kun resurssit olivat latautuneet välimuistiin sekä mahdollisti sivuston toimivuuden offline-tilassa. Välimuistien käyttö antaa sovellukselle paremman käyttökokemuksen, joten niiden käyttö on suositeltavaa sulavamman käytön takia, vaikka varsinaista offline-käytettävyyttä ei haluttaisikaan toteuttaa. Offline-tilaa

varten sovellukseen olisi voinut vielä lisätä niin sanottu fallback-sivun eli sivun, jolle käyttäjä ohjataan offline-tilassa. Fallback-sivu voisi sisältää ainoastaan ilmoituksen offline-tilasta tai pahoittelut siitä, että haluttu sivu tai sivusto on saavuttamattomissa. Fallback-sivu parantaisi käyttökokemusta, koska käyttäjälle esitettäisiin edes jonkinlainen sivu selaimen vakio offline-ilmoitussivun sijaan.

Taustasynkronisaation toteutus ja testaus osoittautui ongelmalliseksi, koska sovellus toimi lokaalisti, joten verkkoyhteydellä tai sen puutteella ei ollut vaikutusta sovelluksen toimintaan. Epäonnistuneita pyyntöjä saatiin kuitenkin tallennettua jonoon ja epäsäännöllisesti myös lähetettyä uudelleen jonosta, mikä on indikaatio siitä, että taustasynkronisaatio toimii kuten sen pitäisikin. Sovelluksessa ei ollut toteutushetkellä myöskään muuta toiminnallisuuden testausmahdollisuutta kuin lisätä ja poistaa käyttäjiä, joten taustasynkronisaation toimiminen muussa tapauksessa jäi testaamatta ja siten epäselväksi. Toimimisen varmistamiseksi olisi ollut hyvä pystyä testaamaan ominaisuutta muussakin tapauksessa.

Push-ilmoitusten ja -viestien lähettämisen toteutuksessa ei ilmennyt ongelmia, mutta käyttötarkoituksia sovelluksessa ei juurikaan ollut. Chromen kehittäjätyökalujen avulla lähetetyt push-viestit toimivat juuri niin kuin niiden pitikin. Muuta reittiä lähettää viestejä ei tosin ollut, koska sovellus ei oikeastaan sisältänyt sellaista toiminnallisuutta, jonka kanssa olisi voinut käyttää push-ilmoituksia tai -viestejä. Vaikka tässä toteutuksessa ei käynyt ilmi, push-ilmoituksilla on paljon käyttökohteita, joiden avulla voidaan parantaa sovelluksen käyttökokemusta sekä lisätä käyttäjän aktiivisuutta sovelluksen käytön osalta. Tällaisia käyttökohteita voi olla esimerkiksi käyttäjän tiedottaminen uudesta sisällöstä tai muistutuksen lähettäminen.

Toteutuksen hyötynä on lähinnä parannettu käyttökokemus välimuistien käytön ansiosta. Taustasynkronisaatio ja push-ilmoitukset ovat toteutuksen kohteen osalta käytännössä turhia, mutta niillä voi olla käyttöä muualla, joten toiminnan testaaminen on siltä osin tarpeellista ja tärkeää. Taustasynkronisaatiota ja push-ilmoituksia voisi käyttää paremmin hyväksi, mikäli sovellukseen lisäisi enemmän toiminallisuutta pelkän käyttäjien poistamisen ja lisäämisen lisäksi. Kuitenkin, koska kyseessä on sovellus, jota käytetään pohjana asiakasprojekteissa, ei siihen välttämättä ole kannattavaa

lisätä toiminallisuutta, mitä ei käytetä muualla. Jos PWA-teknologiaa olisi tarkoitus myydä asiakkaalle, olisi luonnollisesti tärkeää luoda sille sopivia ominaisuuksia teknologian testaamiseen. Tekninen toteutus toi kuitenkin haluttua informaatiota PWA-ominaisuuksien lisäämisestä verkkosovellukseen sekä huomioita sovelluksen osista ja toiminnallisuuksista, joita todennäköisesti joutuu suunnittelemaan uudelleen, mikäli PWA-toiminnallisuutta halutaan tuoda osaksi sovellusta. Yksi tällainen toiminto on esimerkiksi kirjautuminen ja se tapa, jolla kirjautunut käyttäjä voidaan validoida myös offline-tilassa.

6.4 Tutkimuksen luotettavuus ja pätevyys

Aineistoa ja dokumentaatiota PWA-teknologiasta on tarjolla lähestulkoon ainoastaan verkosta. Koska kyse on uudesta teknologiasta, ei painettua kirjallista aineistoa ole juurikaan saatavilla. Tutkimuksessa aineisto kerättiin esimerkiksi Googlen ja Mozillan lähteistä, joten voidaan olettaa, että aineisto on luotettavaa. Google toimii yhtenä PWA-teknologian kehittäjänä, joten sen aineisto on kattavinta ja parhaiten ajan tasalla olevaa.

Koska teknisen toteutuksen kohteena on sovellus, ei tutkimustuloksista voida välttämättä vetää johtopäätöksiä muiden sovellusten kohdalla, koska erot eri sovellusten välillä voi olla suuriakin. Tutkimuksesta voidaan kuitenkin päätellä, että PWA-teknologia on toimiva teknologia ja se tarjoaa uusia tärkeitä ominaisuuksia verkkosovellusten kehittämiseen. Kyseessä on myös uusi teknologia, joten sen tulevaisuuden merkityksestä ja olemassaolosta ei voi tällä hetkellä olla lainkaan varma.

Tutkimuskysymyksiin tutkimus vastaa hyvin, mutta tutkimustuloksista ei voi tehdä suoria johtopäätöksiä siitä, kannattaako esimerkiksi mobiilisovelluksien kehityksessä harkita PWA-teknologiaa React Nativen sijasta, koska teknisessä toteutuksessa ei testattu mobiililaitteisiin liittyviä ominaisuuksia kuten kameraa, koska sovelluksessa ei ollut sopivaa toiminnallisuutta. Teoriaosa antaa kuitenkin selvyyttä PWA-teknologian toiminnasta mobiililaitteilla tämän opinnäytetyön kirjoittamisen ajankohdassa.

6.5 Jatkotutkimukset

PWA-teknologian käytöstä natiivisovelluksen korvaajana mobiiliympäristössä olisi tarvetta tehdä tarkempaa juuri tarkasti aiheeseen keskittyvää tutkimusta. Tutkimus vaatisi enemmän opiskelua ja syvempää osaamista PWA-teknologiasta. Koska ainakaan tässä työssä käytetty Workbox-kirjasto ei tarjoa sopivia liitännäisiä valmiina, tarkemman tutkimuksen aikana joutuisi tekemään enemmän koodausta itse. Tutkimukseen tarvitsisi myös toisenlaisen pohjan, joka soveltuisi paremmin mobiiliympäristöön. Teknisessä toteutuksessa käytetty Base-sovellus vaatisi myös muutoksia esimerkiksi kirjautumisen validoinnin osalta, jotta se toimisi oikein ja turvallisesti myös offline-tilassa. Lisäksi esimerkiksi Redux-kirjaston toimintaa tulisi tutkia tarkemmin PWA-ominaisuuksien kanssa.

Lähteet

Android Developers. 2019. Intents and intent filters. Viitattu 30.3.2019. https://developer.android.com/guide/components/intents-filters

Firtman, M. 2017. Android Oreo takes a bite out of Progressive Web Apps. Viitattu 30.3.2019. https://medium.com/@firt/android-oreo-takes-a-bite-out-of-progressive-web-apps-30b7e854648f

Firtman, M. 2018. Progressive Web Apps on iOS are here. Viitattu 18.2.2019. https://medium.com/@firt/progressive-web-apps-on-ios-are-here-d00430dee3a7

Gaunt, M., Kinlan, P. 2019. The Web App Manifest. Viitattu 18.2.2019. https://developers.google.com/web/fundamentals/web-app-manifest/

Gazdecki A. 2018. Welcoming PWAs: Apple now supports service workers on Safari. Viitattu 1.3.2019. https://betanews.com/2018/02/22/welcoming-pwas-apple-now-supports-service-workers-on-safari/

Introduction to Push Notifications. 2019. Artikkeli Google Developers sivustolla. Viitattu 30.3.2019. https://developers.google.com/web/ilt/pwa/introduction-to-push-notifications

Introduction to Service Worker. 2019. Google Developers www-sivuston esittely Service Workerista. Viitattu 18.2.2019. https://developers.google.com/web/ilt/pwa/introduction-to-service-worker

Kananen, J. 2010. Opinnäytetyön kirjoittamosen käytännön opas. Tampere: Tampereen Yliopistopaino.

Kananen, J. 2017. Laadullinen tutkimus prograduna ja opinnäytetyönä. Jyväskylä: Suomen Yliopistopaino.

Karlin, J., Kruisselbrink, M. 2018. Web Background Synchronization. Viitattu 18.2.2019. https://wicg.github.io/BackgroundSync/spec/

Kotwicz, P., LePage, P., Posnick J. 2019. WebAPKs on Android. Viitattu 15.2.2019. https://developers.google.com/web/fundamentals/integration/webapks

McCormick, L., Navarra, E., Purwar, S., Wojciakowski, M. 2018. Progressive Web Apps on Windows. Viitattu 8.4.2019. https://docs.microsoft.com/en-gb/microsoft-edge/progressive-web-apps

MDN Web Docs. 2019. MDN:n sivuston dokumentointi PWA:n rakenteesta. Viitattu 18.3.2019. https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/App_structure Progressive web app structure

MDN Web Docs. 2019. MDN:n sivuston Web App Manifest dokumentointi. Web App Manifest. Viitattu 24.3.2019. https://developer.mozilla.org/en-US/docs/Web/Manifest

Nikkei achieves a new level of quality and performance with their multi-page PWA. 2018. Artikkeli Google Developers sivustolla. Viitattu 18.1.2019. https://developers.google.com/web/showcase/2018/nikkei

Osmani, A. 2017. A Tinder Progressive Web App Performance Case Study. Viitattu 18.1.2019. https://medium.com/@addyosmani/a-tinder-progressive-web-app-performance-case-study-78919d98ece0

Osmani, A. 2015. Web Starter Kit. Viitattu 24.3.2019. https://github.com/google/web-starter-kit/blob/master/app/manifest.jso.n

Semenov, A. 2017. How Progressive Web Apps make the Web great again. Viitattu 18.3.2019. https://webagility.com/posts/how-progressive-web-apps-make-the-web-great-again

Twitter Lite PWA Significantly Increases Engagement and Reduces Data Usage. 2017. Artikkeli Google Developers sivustolla. Viitattu 18.1.2019. https://developers.google.com/web/showcase/2017/twitter

Webkit Feature Status. 2019. Webkitin ominaisuuksien tilannelista kehitysvaiheista. Viitattu 18.3.2019. https://webkit.org/status/#specification-web-app-manifest

What Web Can Do Today, 2019. Whatwebcando.today testisivusto verkkosovellus rajapinnoille. Viitattu 7.4.2019. https://whatwebcando.today/

Workbox Strategies. 2019. Google Developers www-sivuston Workbox-työkalun dokumentaatio Workboxin strategioista. Viitattu 18.2.2019. https://developers.google.com/web/tools/workbox/modules/workbox-strategies

Workbox Strategies. 2019. Google Developers www-sivuston Workbox-työkalun dokumentaatio Workboxin precachin-liitännäisestä. Viitattu 18.2.2019. https://developers.google.com/web/tools/workbox/modules/workbox-precaching