Tampereen yliopisto

Minna Jarsma

PROGRESSIIVISTEN WEB-SOVELLUSTEN SOVELTUVUUS KORVAAMAAN MOBIILISOVELLUKSET

Diplomityö
Informaatioteknologian ja viestinnän tiedekunta
Tarkastaja: Prof. Hannu-Matti Järvinen
Maaliskuu 2021

TIIVISTELMÄ

Minna Jarsma: Progressiivisten web-sovellusten soveltuvuus korvaamaan mobiilisovellukset
Diplomityö
Tampereen yliopisto
Tietotekniikan tutkinto-ohjelma
Maaliskuu 2021

Mobiilisovellusten etuina ovat olleet asennettavuus mobiililaitteelle sekä pääsy sen laitteistoon, kuten kameraan. Tavanomaisesti on toteutettu erikseen web-sovellus selaimia varten ja mobiilisovellus mobiilikäyttöä varten. Web-selainten rajapintojen kehittymisen myötä web-sovelluksien mahdollisuudet ovat lähestyneet natiivisovelluksia: selaimet voivat hyödyntää muun muassa laitteen kameraa ja mikrofonia sekä lukea käyttäjän sijainnin. Progressiivisten web-sovellusten ansiosta yhdellä koodipohjalla voidaan toteuttaa sovellus, joka toimii saumattomasti sekä perinteisessä selainnäkymässä että laitteelle asennettuna. Asennettuna versiona web-sovellus vastaa käyttökokemukseltaan natiivisovellusta.

Tämän työn tavoitteena oli tutkia, voidaanko mobiilisovellus korvata progressiivisella web-sovelluksella. Ensin määriteltiin vaatimukset, jotka täyttämällä progressiivinen web-sovellus olisi varteenotettava vaihtoehto mobiilisovellukselle. Vaatimukset koostuivat kuvitteellisen kohdeyrityksen asettamista vaatimuksista sekä progressiivisia web-sovelluksia koskevista vaatimuksista. Kohdeyrityksen vaatimusten perusteella oli tarpeen toteuttaa sovellus, jonka avulla voitiin hallinnoida huoltokäyntejä. Vaatimusten perusteella määriteltiin toteutettava web-sovellus sekä sen tarvitsema palvelinpuolen sovellus ja tietokanta.

Määrittelyn perusteella työssä toteutettiin progressiivinen web-sovellus Reactilla. Palvelinpuolella hallinnoitiin web-sovelluksen tarvitsemia tietoja sekä lähetettiin push-ilmoituksia. Toteutettu prototyyppisovellus täytti kaikki progressiivisia web-sovelluksia koskevat vaatimukset sekä lähes kaikki kohdeyrityksen asettamista vaatimuksista. Yksi täyttämättä jääneistä vaatimuksista olivat push-ilmoitukset, joita mobiiliselain Safari ei tue. Safarin markkinaosuus sekä puuttuvien push-ilmoitusten merkittävyys osoitti, että mobiilisovellusta ei voida korvata progressiivisella web-sovelluksella. Jatkokehitystä varten tulisi selvittää, onko Safaria varten olemassa vaihtoehtoa push-ilmoituksille. Lisäksi mobiiliselainten lukumäärä ja tuettujen rajapintojen vaihtelevuus tuottaa haasteita web-sovellusten kehityksessä.

Avainsanat: progressiivinen web-sovellus, PWA, palvelutyöläinen

ABSTRACT

Minna Jarsma: The suitability of progressive web applications to replace mobile applications
Master of Science Thesis
Tampere University
Degree Programme in Information Technology
March 2021

The advantages of mobile applications have been installability and access to the device hardware, for example the camera. Conventionally the applications for web browsers and mobile devices have been created separately. Due to the development of APIs of web browsers the potential of web applications has been approaching that of native applications: browsers can utilize the device camera and microphone, and also read the location of the user. Thanks to progressive web applications you can use one codebase to create an application which works seamlessly on traditional browser view as well as an installed application. The user experience of an installed web application matches that of a native application.

The goal of this thesis was to evaluate if a progressive application can replace a mobile application. First the requirements the progressive web application needed to meet were defined. The requirements consisted of requirements set by an imaginary target company and of requirements concerning progressive web applications. The progressive web application should be capable of managing maintenance visits according to the requirements of the target company. The web application was specified according to the requirements, along with the backend application and the database.

According to the specification the progressive web application was developed with React. The data required by the web application was managed on server-side, and the server was also used for sending push notifications. The developed prototype application met all of the progressive web application requirements and almost all of the requirements set by the target company. One of the unmet requirements were push notifications which are not supported in Safari mobile browser. The market share of Safari and the significance of the missing push notifications proved that a mobile application cannot be replaced with a progressive web application. Alternatives for push notifications in Safari should be researched for further development. Also the number of different mobile browsers and the varied support for APIs produce challenges for the development of web applications.

Keywords: progressive web application, PWA, service worker

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

ALKUSANAT

Kun koitan miettiä toista vastaavanlaista urakkaa elämässäni, mieleeni nousee ainoastaan Soulissa sijaitsevalle Bukhansan-vuorelle kapuaminen. Diplomityötä tehdessä ei onneksi esiintynyt vaaratilanteita, mutta usko omiin kykyihin horjui matkan varrella. Näistä hetkistä selvittiin läheisten tuella, ja matkan päätteeksi olo on uupunut mutta onnellinen.

Tämän diplomityön aihe on lähtöisin Mikko Viskarilta, jota haluan kiittää sekä aiheen keksimisestä että tuesta työelämän haasteissa. Kiitokset myös työnantajalleni Etteplan MORElle joustavuudesta kirjoitusprosessin aikana.

Kiitokset Hannu-Matti Järviselle, jonka ohjauksen ansiosta sain diplomityön maaliin. Haluan kiittää perhettäni ja ystäviäni tärkeästä tuesta opintojeni aikana. Erityiskiitokset Tuomakselle korvaamattomasta avusta ja rohkaisusta diplomityön suhteen sekä kumppanuudesta elämän ylä- ja alamäissä.

Tampereella, 8. maaliskuuta 2021

Minna Jarsma

SISÄLLYSLUETTELO

KUVALUETTELO

OHJELMA- JA ALGORITMILUETTELO

LYHENTEET JA MERKINNÄT

APK	engl. Android Package, Android-käyttöjärjestelmän käyttämä pakettitiedostotyyppi
CSS	engl. Cascading Style Sheets, kieli tyyliohjeiden määrittelyyn
DOM	engl. Document Object Model, tapa kuvata rakenteisen dokumentin rakenne puuna
FCM	Firebase Cloud Messaging, viesti- ja notifikaatiopalvelu
FMP	engl. First Meaningful Paint, ensimmäinen merkitsevä piirto
HTML	engl. Hypertext Markup Language, avoimesti standardoitu kuvauskieli
HTTP	engl. Hypertext Transfer Protocol, WWW-palvelinten ja selainten käyttämä tiedonsiirtoprotokolla
HTTPS	engl. Hypertext Transfer Protocol Secure, suojattu tiedonsiirtoprotokolla
IPA	engl. iOS App Store Package, iOS-käyttöjärjestelmän käyttämä pakettitiedostotyyppi
JavaScript	dynaaminen korkean tason ohjelmointikieli erityisesti web-ympäristöjä varten
JSON	engl. JavaScript Object Notation, tiedonvälitykseen tarkoitettu tiedostomuoto
JSX	JavaScript XML, JavaScript-kielen syntaksilaajennus
NFC	engl. Near Field Communication, lyhyen kantaman tiedonsiirtoteknologia
PWA	engl. Progressive Web Application, progressiivinen web-sovellus
SDK	engl. Software Development Kit, kokoelma ohjelmistokehitystyökaluja
SEO	engl. Search Engine Optimization, hakukoneoptimointi
SPA	engl. Single-Page Application, dynaamisesti sisältöään päivittävä web-sovellus
TLS	engl. Transport Layer Security, verkkoliikenteen salausprotokolla
TTI	engl. Time To Interactive, aika interaktiiviseen käyttöliittymään

1 JOHDANTO

Web-selainten tarjoamat rajapinnat ja ominaisuudet ovat kehittyneet vuosien saatossa, ja lähestyvät enemmän ja enemmän mobiilisovellusten ominaisuuksia. Nykypäivänä web-sovellukset voivat hyödyntää käyttäjän laitteen kameraa ja mikrofonia, lukea bluetooth- ja USB-laitteita sekä vastaanottaa push-ilmoituksia. Sen sijaan että toteutettaisiin web-sovellus ja mobiilisovellus erikseen, ne voidaan toteuttaa samalla koodipohjalla yhtenä sovelluksena.

Progressiiviset web-sovellukset toimivat sekä web-selaimessa tavallisen web-sivuston tapaan että laitteelle asennettuna, jolloin ne tarjoavat saumattoman natiivisovelluksen kaltaisen kokemuksen. Responsiivisuuden lisäksi progressiivisten web-sovellusten etu on nopeus. Tutkimuksen mukaan käyttäjät odottavat korkeintaan noin kaksi sekuntia tietohaun tuloksia ennen sivuston hylkäämistä [53]. Progressiiviset web-sovellukset pystyvät tallentamaan tarvitsemiaan resursseja selaimen välimuistiin, jolloin käyttäjän ei tarvitse odottaa niiden hakemista verkkoyhteyden yli.

Progressiivisten web-sovellusten etuna on myös sovelluksen jakamisen helppous: ei tarvitse huolehtia sovelluksen jakamisesta keskitetyn sovelluskaupan kautta, sillä vierailu web-sivustolla riittää sovelluksen asentamiseksi laitteelle. Mobiilisovellusten kohdalla sovelluksen julkaiseminen sovelluskaupassa voi olla työlästä ja aikaa vievää. Web-sovellusten kehittäminen on mahdollista eri käyttöjärjestelmillä ja kehitysympäristöillä, kun taas esimerkiksi iOS-sovellusten kehittäminen on tiukasti rajoitettu macOS-käyttöjärjestelmille.

Tässä työssä on tavoitteena tutkia, onko mobiilisovellus mahdollista toteuttaa progressiivisena web-sovelluksena. Aihetta lähestytään konstruktiivisen tutkimusmenetelmän mukaisesti: ensin määritellään vaatimukset, jotka täyttämällä progressiivinen web-sovellus korvaa mobiilisovelluksen. Vaatimusten perusteella määritellään toteutettava sovellus, ja määritelmän perusteella toteutetaan progressiivinen web-sovellus. Lopuksi päätellään tutkimuksen lopputulos sen perusteella, täyttääkö toteutettu sovellus aiemmin määritellyt vaatimukset.

Luvussa 2 käydään läpi web-sovellusten kehityskulkua webin alkuajoista nykypäivään asti sekä tarkastellaan mobiilisovellusten nykytilannetta ja niiden eri toteutustapoja. Progressiivisten web-sovellukset ja niiden merkittävimmät ominaisuudet määritellään luvussa 3.

Työssä toteutettavaan prototyyppisovellukseen kohdistuvat toiminnalliset ja ei-toiminnalliset vaatimukset käsitellään luvussa 4. Vaatimusten perusteella määritellään prototyyppi-

sovelluksen frontend- sekä backend-ominaisuudet luvussa 5, ja luvussa 6 käydään läpi määrittelyn perusteella toteutettu sovellus ominaisuuksineen. Toteutettu prototyyppisovellus arvioidaan sekä tutkimuksen havainnot käydään läpi luvussa 7. Yhteenvedossa eli luvussa 8 tiivistetään työhön liittyvä tieto sekä tutkimustulokset.

2 TAUSTAA

Tässä luvussa luodaan katsaus web-sovellusten historiaan ja selvitetään, millaisin kehitysaskelein ensimmäisistä web-sivuista päädyttiin moderneihin web-sovelluksiin. Lisäksi käydään läpi mobiilisovellusten nykytilannetta sekä eri toteutustapoja.

2.1 Web-sovellukset

Webin alkuaikoina web-sivut koostuivat ainoastaan staattisesta sisällöstä, joka määritellään HTML-merkintäkieltä (engl. Hypertext Markup Language) käyttäen. Web-sivut voivat sisältää hyperlinkkejä muihin web-sivuihin, mutta niihin siirtyminen vaati sisällön lataamisen uudelleen palvelimelta. Sisällön uudelleenlataaminen vaadittiin myös siinä tapauksessa, jos jo avatun web-sivun sisältöä oli tarpeen muuttaa.

Web-sivujen dynaaminen sisältö oli palvelinpuolen hallussa, kunnes JavaScriptin myötä sitä saatiin myös asiakaspuolelle web-selaimeen. JavaScript on dynaaminen ja yksisäikeinen ohjelmointikieli, joka tukee oliokeskeisen ohjelmointityylin lisäksi myös imperatiivista ja funktionaalista tyyliä. Nimestään huolimatta JavaScriptillä ei ole yhteyttä Java-ohjelmointikieleen. JavaScript noudattaa ECMAScript-standardia, josta julkaistaan uusi versio vuoden välein. [45]

JavaScriptin ansiosta osa web-sivun toimintalogiikasta voitiin siirtää web-selaimen puolelle, joten tarve web-sivun lataaminen uudelleen palvelimelta väheni. HTML:n ja JavaScriptin lisäksi CSS-kuvauskieli (engl. cascading style sheets) on yksi web-teknologian kulmakivistä. CSS:ää käyttäen voidaan määritellä tyylisääntöjä, joiden perusteella web-sivun ja sen ulkoasu esitetään web-selaimessa.

HTML-ominaisuuksien kehittyminen on jatkunut webin alkuajoista lähtien. Vuonna 2014 julkaistu HTML5 toi mukanaan muun muassa uusia mediaelementtejä: audio- ja video-elementtien avulla web-sivulla voidaan soittaa ääni- ja videosisältöä [57]. Nämä ominaisuudet löytyvät suoraan web-selaimesta, eli esimerkiksi videon näyttämiseen web-sovelluksessa ei tarvita erillistä ohjelmakirjastoa. Tammikuussa 2021 suosituin web-selain on Chrome noin 67 prosentin markkinaosuudella; seuraavaksi suosituimpia ovat Safari noin 10 prosentin markkinaosuudella ja Firefox noin 8 prosentin markkinaosuudella [19].

JavaScriptin ansiosta voidaan toteuttaa yhden sivun web-sovelluksia eli SPA:ita (engl. single-page application), joissa sivuston rakenne ja logiikka haetaan palvelimelta yhdel-

lä sivun latauksella. Näissä sovelluksissa näkymien välillä siirtyminen on toteutettu ilman sivun uudelleenlataamista, jolloin lopputuloksena on natiivisovelluksia muistuttava sujuvampi käyttökokemus.

Viimeisin kehitysaskel web-sovellusten saralla ovat progressiiviset web-sovellukset eli PWA:t (engl. progressive web application), jotka ottavat askeleen kohti natiivisovelluksia. Progressiivinen web-sovellus on alun perin Googlen käyttämä termi mukautuville ja joustaville web-sovelluksille, jotka on toteutettu ainoastaan web-teknologioita käyttäen [42]. Nämä sovellukset voivat hyödyntää enemmän käyttäjän laitteen ominaisuuksia natiivisovellusten tavoin, ja ne voidaan asentaa käyttäjän laitteelle.

2.2 Mobiilisovellukset

Suosituimpia mobiilikäyttöjärjestelmiä kirjoitushetkellä ovat Android reilulla 70 prosentin markkinaosuudellaan ja iOS noin 28 prosentin markkinaosuudella [52]. Muita mobiilikäyttöjärjestelmiä ovat muun muassa Sailfish OS ja Ubuntu Touch, joista kumpikin on avoimen lähdekoodin käyttöjärjestelmä. Sailfish OS on suomalaisen Jolla-yrityksen kehittämä, ja Ubuntu Touch on UBPorts Communityn kehittämä ja ylläpitämä.

Android-mobiilisovellusten kehittämiseen vaaditaan Android-ohjelmistokehitystyökalut, joihin sisältyy muun muassa tarvittavat kirjastot, emulaattori sekä debuggeri. Virallinen Android-sovelluksien kehittämiseen tarkoitettu ohjelmointiympäristö on Android Studio, joka on rakennettu Intellij IDEA-ohjelmointiympäristön päälle [67]. Ensisijainen ohjelmointikieli Android-kehitykseen on Kotlin, mutta Android Studio tukee myös Java- ja C++-ohjelmointikieliä [5, 27]. Koontiversion (engl. build) luomiseksi Android Studio kääntää sovelluksen lähdekoodin sekä sen käyttämät resurssit, ja kokoaa ne Android Package- eli APK-pakettitiedostoon [37]. APK-tiedostossa on kaikki sovelluksen sisältö, ja sitä käyttämällä sovellus voidaan asentaa Android-laitteelle [37]. Virallinen sovelluskauppa Android-sovelluksia varten on Googlen ylläpitämä Google Play.

iOS-sovellusten kehittämiseen vaaditut iOS-ohjelmistokehitystyökalut ovat kirjoitushetkellä saatavilla ainoastaan Mac-käyttöjärjestelmälle, joten kehittäminen ei ole mahdollista esimerkiksi Windows-käyttöjärjestelmällä. Myös ensisijainen iOS-ohjelmointiympäristö, eli Xcode, on kirjoitushetkellä saatavilla vain Mac-käyttöjärjestelmälle. Suositeltu ohjelmointikieli iOS-kehittämiseen oli Objective-C, kunnes Apple korvasi sen Swift-ohjelmointikielellä vuonna 2014 [46]. iOS-ohjelmistokehitystyökalujen avulla sovelluksen lähdekoodeista voidaan koostaa IPA-pakettitiedosto (iOS App Store Package), jonka avulla sovellus voidaan asentaa iOS-laitteelle [61]. Applen ylläpitämä App Store on virallinen sovelluskauppa iOS-sovelluksille.

Mobiilisovelluksia voidaan myös kehittää usealle käyttöjärjestelmälle samaa koodipohjaa käyttäen esimerkiksi React Nativen tai Flutterin avulla. Facebookin kehittämä React Native on avoimen lähdekoodin ohjelmistokehys (engl. framework) natiivisovellusten toteuttamiseen JavaScript-ohjelmointikielellä Android- tai iOS-alustoille [18]. Elementit mobiiliso-

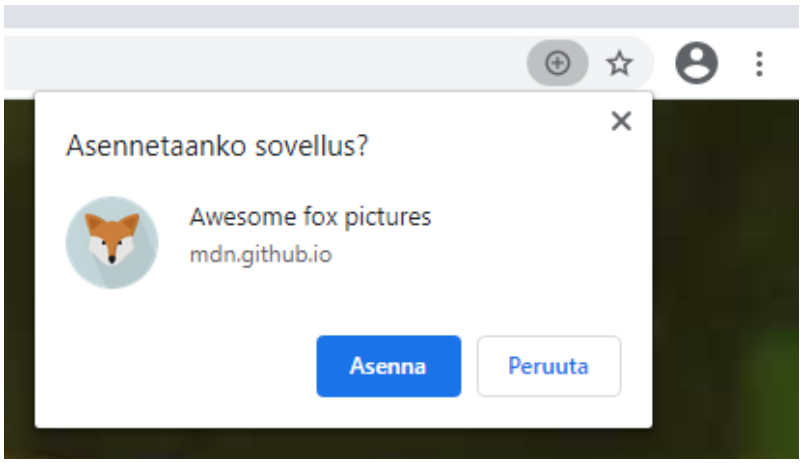
vellukseen toteutetaan React-kirjastoa hyödyntäen, joista React Native luo joko Android- tai iOS-versiot. React Nativella voidaan myös luoda alustariippuvaista koodia, jolla voidaan esimerkiksi rajata ulkoasua koskevat tyylit tiettyyn käyttöjärjestelmään tai sen versioon [18].

Flutter on Googlen kehittämä UI-työkalupaketti, jolla voidaan toteuttaa sovelluksia ChromeOS-, Android- ja iOS-alustoille samalla koodipohjalla. Kirjoitushetkellä Flutterin tuki työpöytä- ja web-sovelluksille on kehitteillä. Flutter-sovelluksia kirjoitetaan Dartilla, joka on nopea olioperustainen ohjelmointikieli. Dart tukee sekä ajonaikaista kääntämistä (engl. just-in-time compilation) että ennenaikaista kääntämistä (engl. ahead-of-time compilation). Flutter-sovelluksista luodaan tuotantoversiot kääntämälle ne natiiviksi konekieleksi tai selaimissa ajettavaksi JavaScript-koodiksi. [82]

3 PROGRESSIIVISET WEB-SOVELLUKSET

Viimeisin askel web-sovellusten kehityksessä ovat progressiiviset web-sovellukset, jotka hälventävät eroa natiivisovellusten ja web-sovellusten välillä. Termi progressiivisille web-sovelluksille on lähtöisin Frances Berrimanin ja Alex Russelin määritelmästä, jonka mukaan progressiiviset web-sovellukset hyödyntävät modernien selainten uusia ominaisuuksia, kuten palvelutyöläisiä sekä verkkosovellusmanifesteja [74]. Googlen nykyisen määritelmän mukaan progressiiviset web-sovellukset ovat nopeita, luotettavia sekä mukaansatempaavia: ne toimivat sujuvasti myös hitaammalla verkkoyhteydellä, ja houkuttelevat käyttäjän takaisin sovelluksen pariin esimerkiksi mobiilisovelluksista tuttujen push-ilmoitusten avulla [65, 74]. Kyseinen määritelmä ei aseta rajoituksia toteutustapaan tai käytettäviin teknologioihin, joten progressiivinen web-sovellus voidaan toteuttaa perinteiseen tapaan JavaScriptillä tai esimerkiksi C#-kielellä WebAssemblyn avulla [11, 65]. Erona tavanomaisiin web-sovelluksiin nähden progressiivisilla web-sovelluksilla on natiivisovelluksista tuttuja ominaisuuksia, kuten offline-käyttö sekä sovelluksen asentaminen laitteelle [65]. Mahdollisuus näiden ominaisuuksien hyödyntämiseen riippuu vahvasti käytetyn web-selaimen tarjoamasta tuesta: selaimia on tarjolla useita, ja kaikki eivät välttämättä tue uusimpia ominaisuuksia [65].

Progressiivinen web-sovellus voidaan asentaa käyttäjän laitteelle käyttämällä web-selainta, josta löytyy tuki kyseisille sovelluksille. Käyttäjän navigoitua web-sovelluksen osoitteeseen selain tarjoaa asentamistoimintoa, mikäli sovellus täyttää asennettavuusvaatimukset. Kuvassa 3.1 esitetään Chrome-selaimen desktop-versiossa varmistusdialogi web-



Kuva 3.1. *Progressiivisen web-sovelluksen asennuskehoite Chrome-selaimessa.*

sovelluksen asentamiselle. Asentamisen myötä käyttäjän laitteelle luodaan ikoni, jolloin asennettu web-sovellus näkyy natiivisovellusten kaltaisesti. Erona natiivisovelluksiin nähden kyseisestä ikonista avautuu asennettu web-sovellus selainnäkymässä, josta puuttuu selaimen oma käyttöliittymä osoitepalkkeineen. Lopputuloksena on natiivisovelluksen kaltainen käyttökokemus.

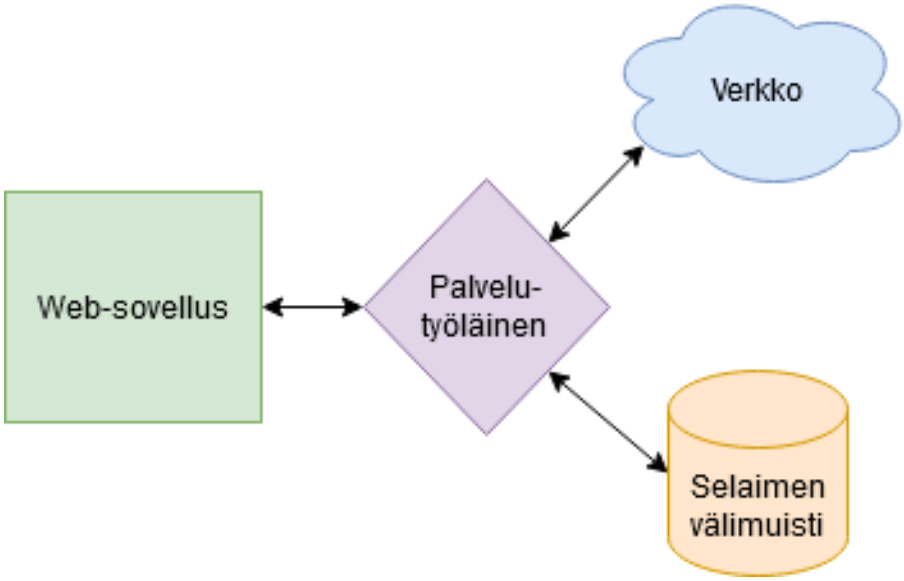
3.1 Verkko- ja palvelutyöläiset

Progressiivisten web-sovellusten ominaisuuksista osa vaatii palvelutyöläisen (engl. service worker), joka on eräs verkkotyöläistyyppi. Verkkotyöläiset (engl. web worker) voivat toimia taustalla omassa säikeessään varsinaisen web-sovelluksen pääsäikeen rinnalla. Tämän ansiosta verkkotyöläisen toiminta ei häiritse tai keskeytä web-sovelluksen ja sen käyttöliittymän toimintaa. Vaikka web-sovellus ja verkkotyöläinen toimivat erillään, ne voivat kommunikoida keskenään viestien välityksellä. Verkkotyöläiset ovat verrattain raskaita suorittaa ja siksi tarkoitettu käytettäväksi vähälukuisina, mutta oikein käytettyinä ne voivat auttaa jopa energiankulutuksen vähentämisessä [62, 79]. [73]

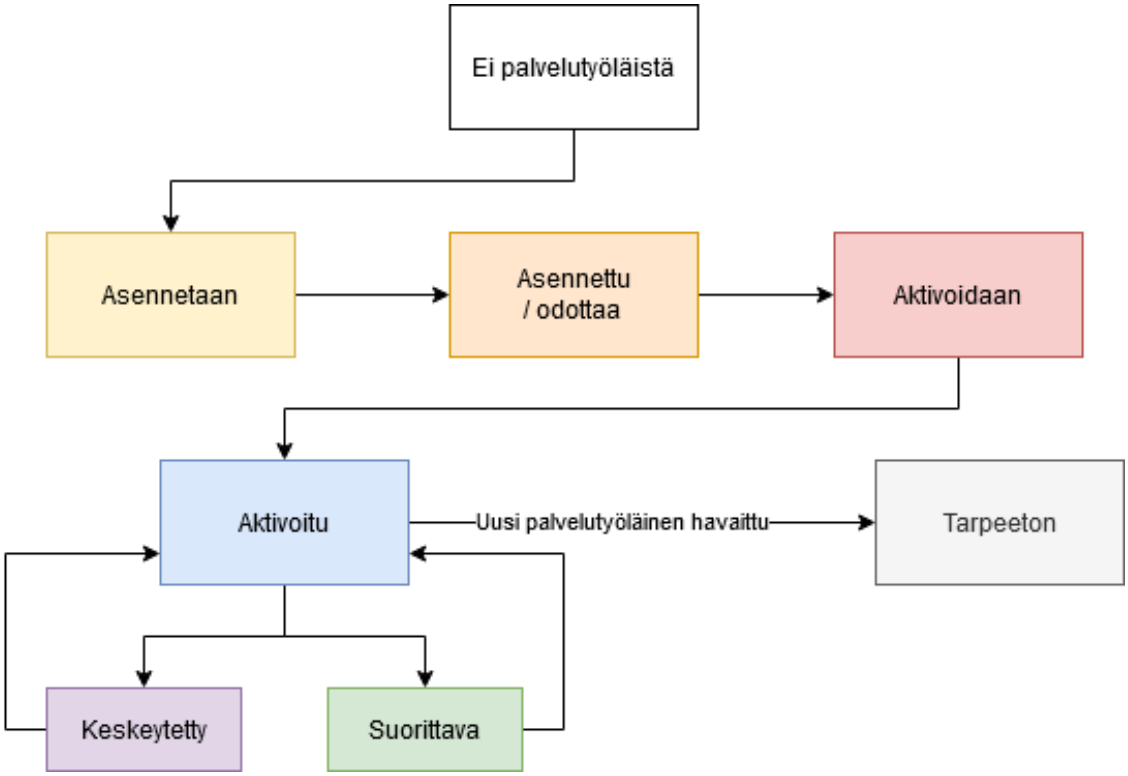
Verkkotyöläisiä varten luodaan uusia käyttöjärjestelmätason säikeitä. Tästä huolimatta rinnakkaisuudesta aiheutuvat ongelmat ovat harvinaisia, sillä verkkotyöläisillä ei ole pääsyä ei-säieturvallisiin komponentteihin sekä niiden kommunikointi toisten säikeiden kanssa on tarkoin rajattu. Lisäksi verkkotyöläisen pääsy varsinaiseen web-sovellukseen on rajallista: DOM:n (engl. document object model) muokkaaminen ole mahdollista, ja globaaliin window-objektin ominaisuuksien käyttäminen on rajoitettu. Verkkotyöläisen säikeessä ajettava koodi voi kuitenkin hyödyntää monia selaimen tarjoamia rajapintoja, kuten IndexedDB-tietokantaa sekä verkkopistokkeita (engl. web socket). Lisäksi verkkotyöläisillä on pääsy globaaliin importScripts-funktioon, joten ne pystyvät hyödyntämään myös ulkopuolisia skriptejä. [73]

Progressiivisten web-sovellusten hyödyntämä verkkotyöläistyyppi, eli palvelutyöläiset, toimivat itse web-sovelluksen, web-selaimen sekä verkon välissä kuvan 3.2 mukaisesti [80]. Tämän ansiosta palvelutyöläiset voivat vaikuttaa siihen, miten web-sovelluksesta lähteviä verkkopyyntöjä käsitellään: palvelimelle osoitettuun verkkopyyntöön voidaan esimerkiksi vastata selaimen välimuistista löytyvällä tiedolla palvelimelta hakemisen sijaan. Tämän ansiosta web-sovelluksessa voidaan esittää sisältöä, vaikka verkkoyhteyttä ei olisi juuri sillä hetkellä saatavilla. Palvelutyöläisellä on paljon valtaa, ja haittapuolena on väärinkäytön mahdollisuus: palvelutyöläinen voi napata ja muokata web-sovelluksesta lähteviä pyyntöjä. Väliintulohyökkäykset (engl. man-in-the-middle attack) ovat mahdollisia, joten palvelutyöläisen käyttö vaatii suojatun HTTPS-yhteyden. Tällöin voidaan varmistua siitä, että web-selain on vastaanottanut oikean palvelutyöläisen, eikä sen koodia ole peukaloitu matkan varrella. [34]

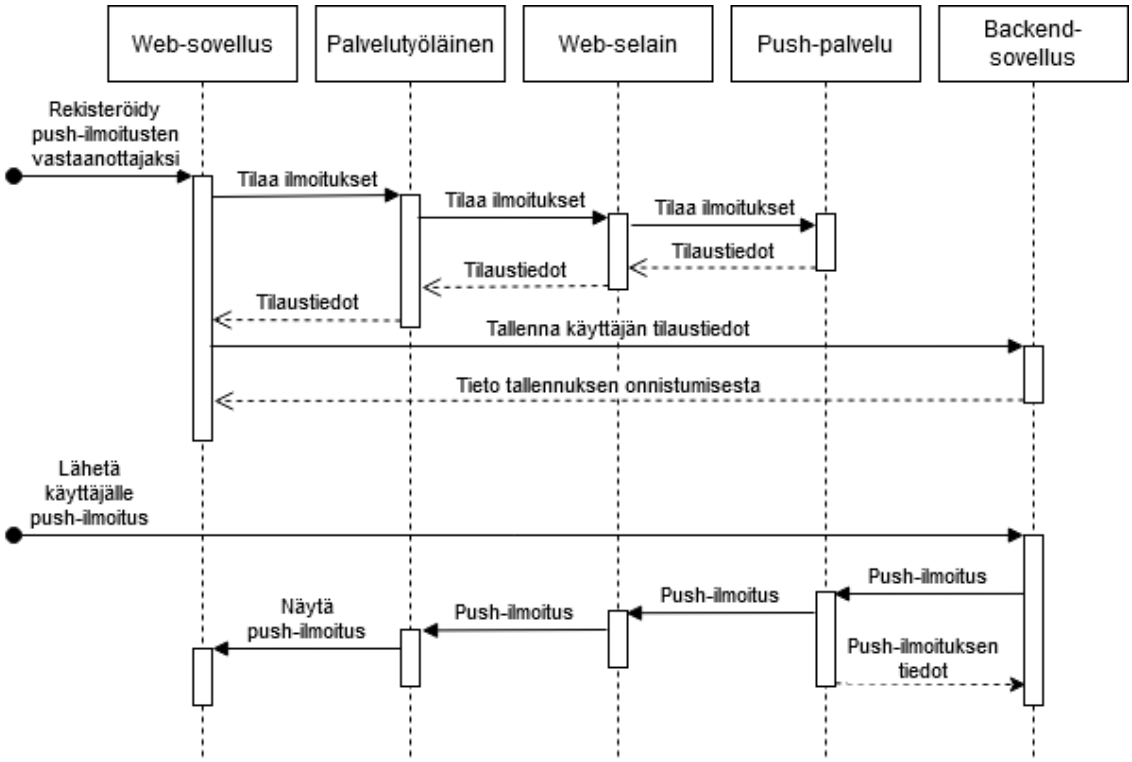
Jotta palvelutyöläinen voidaan ottaa web-sovelluksessa käyttöön, se tulee rekisteröidä. Kuvassa 3.3 esitetään palvelutyöläisen elinkaari sellaisessa tapauksessa, jossa aiem-



Kuva 3.2. *Palvelutyöläisen sijainti web-sovelluksen, selaimen ja verkon välissä.*



Kuva 3.3. *Palvelutyöläisen elinkaari ensimmäisellä asennuskerralla.*



Kuva 3.4. *Kaavio push-palveluun rekisteröitymisestä ja ilmoitusten välityksestä.*

paa palvelutyöläistä ei ole asennettuna. Käyttöönottoprosessin alussa haetaan palvelutyöläisen koodi ja asennetaan se. Asennusvaiheessa voidaan alustaa esimerkiksi resursseja offline-käyttöä varten. Jos asennus onnistuu, palvelutyöläinen siirtyy asennettuun eli odottavaan tilaan. Mikäli web-sovelluksessa olisi jo käytössä toinen palvelutyöläinen, asennettavan palvelutyöläisen käyttöönotto edellyttäisi tässä vaiheessa sovelluksen uudelleen avaamista. Tämän jälkeen palvelutyöläinen siirtyy aktivointivaiheeseen, jonka aikana voidaan suorittaa tarvittavia käyttöönottotoimia. Näitä voivat olla esimerkiksi edellisen palvelutyöläisen käyttämien resurssien siivoaminen. Aktivointivaiheen jälkeen palvelutyöläinen on aktivoidussa tilassa, jolloin se pystyy suorittamaan sille määriteltyjä toimia, kuten web-sovelluksesta lähtevien pyyntöjen hallintaa. Mikäli palvelutyöläinen on toimeton, se siirretään keskeytettyyn tilaan muistin säästämiseksi [34]. Palvelutyöläinen siirtyy lopuksi tarpeeton-tilaan, jos havaitaan nykyisen korvaava, uusi palvelutyöläinen. [72]

3.1.1 Push-ilmoitukset

Palvelutyöläisten avulla on mahdollista lähettää ilmoituksia käyttäjälle, vaikka sovellus ei ole juuri samalla hetkellä aktiivisena. Push-ilmoituksilla (engl. push notification) voidaan kertoa käyttäjälle esimerkiksi web-sovelluksen uusista päivityksistä. Kyseisiä ilmoituksia tulee kuitenkin käyttää harkiten: jos push-ilmoituksia käytetään käyttäjälle ei-relevantin tiedon jakamiseen, ne voivat ärsyttää käyttäjää ja aiheuttaa sovelluksen käytön vähenemistä [9].

Push-ilmoitusten käyttämiseen vaaditaan push-palvelu, joka hallinnoi ilmoituksiin rekisteröityneiden tietoja sekä välittää ilmoitukset halutuille kuuntelijoille [43]. Kuvassa 3.4 esitetään prosessit rekisteröitymiseen sekä push-ilmoituksen lähettämiseen. Rekisteröityminen lähtee web-sovelluksesta: jos käyttäjältä on saatu suostumus push-ilmoitusten näyttämiseen, tilauspyyntö lähetetään push-palvelulle. Vastauksena saadaan tilaustiedot, jotka tallennetaan backend-sovellukseen. Varsinaisen push-ilmoituksen lähettäminen tapahtuu backend-sovelluksen kautta: push-palvelulle lähetetään push-ilmoituspyyntö, jonka mukana on aiemmin tallennetut tilaustiedot ilmoituksen kohdentamista varten. Push-ilmoitus lähetetään palvelusta käyttäjän web-selaimelle, jonka kautta se päätyy palvelutyöläiselle. Keskeytetty palvelutyöläinen herätetään push-ilmoituksen saapuessa, ja ilmoituksen käsittelyn jälkeen se siirretään takaisin keskeytetty-tilaan [43]. Kuvan 3.4 esimerkissä push-ilmoituksen näyttäminen tapahtuu web-sovelluksen puolella, mutta toteutuksesta riippuen se voidaan myös hoitaa palvelutyöläisen puolella.

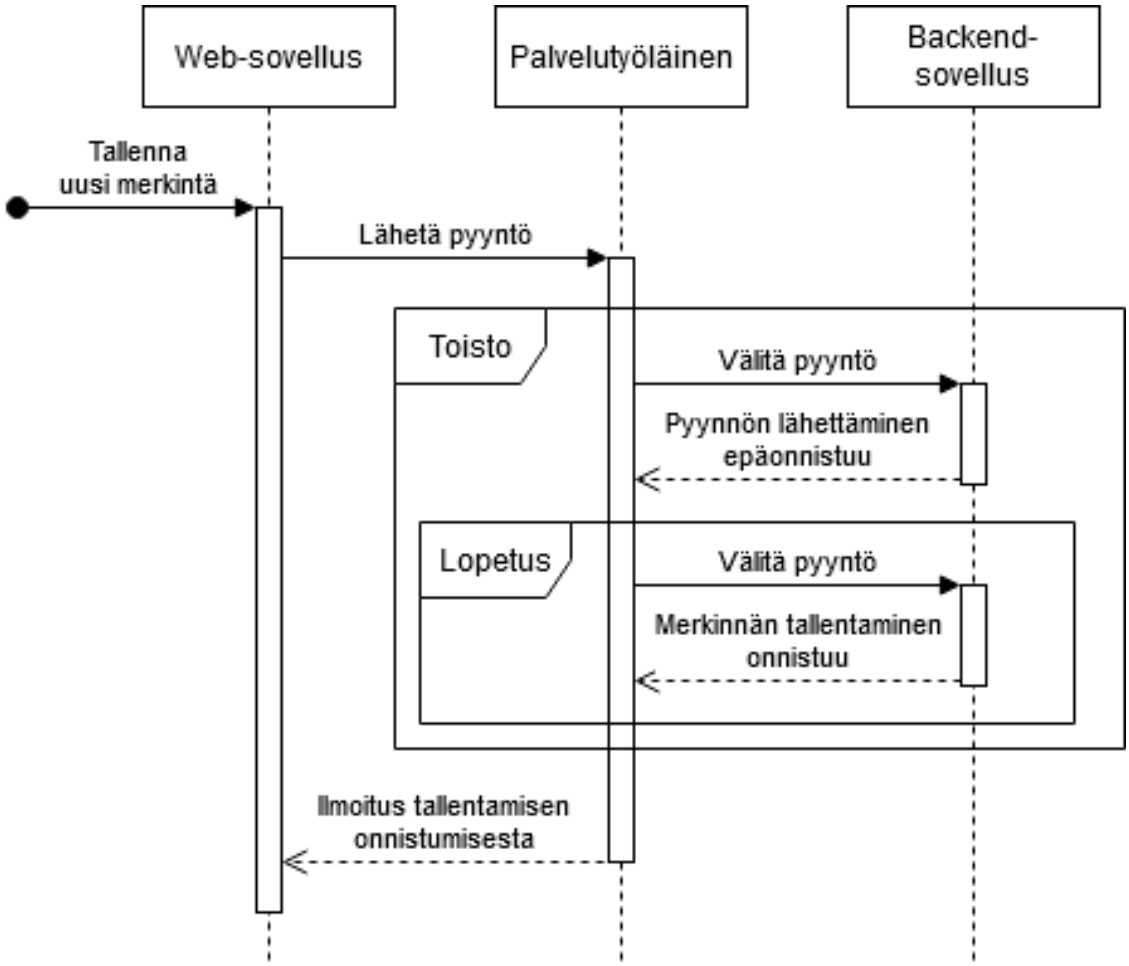
Push-ilmoitukset mahdollistava selaimen push-rajapinta, eli Push API, vaatii aktiivisen palvelutyöläisen, joka voi rekisteröityä push-ilmoitusten vastaanottajaksi. Rekisteröitymisen myötä saadaan ilmoituksen lähettämiseen tarvittavan julkisen avaimen lisäksi rajapinnan osoite, jonka kautta push-ilmoitukset välitetään laitteille. Kyseisen osoitteen avulla muutkin palvelut voivat lähettää push-ilmoitus tietylle palvelutyöläiselle, joten osoite on pidettävä salassa väärinkäytön estämiseksi. [58]

Selaimissa on push-ilmoitusten näyttämistä varten ilmoitusrajapinta eli Notifications API, jonka luomat ilmoitukset sijaitsevat selaimen näkymän ulkopuolella. Ilmoitusten näyttämiseen edellytetään käyttäjän suostumus, joka tavanomaisesti pyydetään kun käyttäjä vierailee web-sivustolla. Jos pyyntö push-ilmoitusten esittämiseen näytetään välittömästi käyttäjän siirryttyä sivustolle, se voi vaikuttaa käyttökokemukseen negatiivisesti. Parempi keino suostumuksen pyytämiseen on käyttäjän toiminnan seurauksena, esimerkiksi napin painamisen yhteydessä. [54]

3.1.2 Taustasynkronointi

Tavanomaisten web-sovellusten tapauksessa sovelluksen käyttäminen hankaloituu merkittävästi, mikäli palvelin ei ole käytössä tai verkkoyhteyttä ei ole saatavilla. Jos käyttäjä haluaa esimerkiksi tallentaa uuden muistiinpanon eikä tallentaminen onnistu, tallentamisen yrittäminen uudelleen on käyttäjän vastuulla. Lisäksi tallennettava sisältö menetetään sovelluksesta poistuttaessa.

Verkkoyhteysongelmien kohdalla progressiivisen web-sovelluksen käyttöä auttaa taustasynkronointi (engl. background synchronization). Taustasynkronoinnin ansiosta web-sovelluksesta lähteviä pyyntöjä voidaan suorittaa uudelleen niin kauan, kunnes pyyntö onnistuu. Kuvassa 3.5 on esitetty esimerkkitapaus uuden merkinnän tallentamisesta. Kun web-sovelluksesta lähtee tallennuspyyntö, palvelutyöläinen nappaa pyynnön ja välittää sen edelleen backend-sovellukselle. Pyynnön epäonnistuessa palvelutyöläinen voi



Kuva 3.5. *Kaavio uuden resurssin tallentamisesta taustasynkronoinnin avulla.*

tallentaa kyseisen pyynnön web-selaimen IndexedDB-tietokantaan myöhempää uudelleenlähettämistä varten [6]. Palvelutyöläinen lähettää pyyntöä uudelleen, kunnes pyyntö onnistuu ja merkintä tallennetaan backend-sovelluksessa.

4 VAATIMUKSET

Toteutettava web-sovellus määritellään kuvitteellisen huoltoyrityksen asettamien vaatimusten pohjalta. Määrittelyssä otetaan huomioon myös progressiivisen web-sovelluksen määritelmää koskevat kriteerit. Tässä luvussa käydään läpi edellä mainitut toiminnalliset ja ei-toiminnalliset vaatimukset.

4.1 Kohdeyrityksen vaatimukset

Sovelluksen kuvitteellinen käyttäjäkunta koostuu huoltokäyntejä suorittavien henkilöiden lisäksi vastaanottohenkilöistä, jotka kirjaavat huoltokäyntejä huoltohenkilöille. Näin ollen sovelluksessa tulee olla kaksi eri käyttäjäroolia, vastaanottaja sekä huoltaja. Vastaanottajan pääasiallinen käyttö kohdistuu sovelluksen desktop-näkymään, ja huoltaja suorittaa huoltokäyntejä älypuhelimen avulla mobiilinäkymässä. Tästä syystä sovelluksen tulee olla responsiivinen, jotta sovelluksen käyttäminen onnistuu erikokoisilla näytöillä.

Huoltokäynneillä tulee olla nimi, sijainti, lista varsinaisista tehtävistä eli huollettavista laitteista, sekä tieto käyttäjästä, jolle huoltotyö on määrätty. Vastaanottajien pitää pystyä edellä mainittujen huoltokäynnin tietojen lisäksi luomaan uusia huoltokäyntejä ja poistamaan tallennettuja huoltokäyntejä, sekä muokkaamaan edellä mainittuja huoltokäynnin tietoja. Huoltokäyntejä tulee pystyä listaamaan ja rajaamaan käyttäjän perusteella, jotta huoltaja näkee vain hänelle määrätyt huoltokäynnit. Tavallisen käyttäjän pitää pystyä myös merkkaamaan tälle määrätyn huoltokäynnin tehtäviä suoritetuksi.

Huoltokäyntikohteet voivat sijaita katvealueella, jossa mobiiliverkkoyhteyttä ei ole saatavilla. Tässäkin tapauksessa sovelluksen käyttäminen tulee olla mahdollista, eli sovelluksen tulee tallentaa tarpeeksi resursseja lokaalisti offline-käyttöä varten. Tällä mahdollistetaan esimerkiksi huoltokäyntilistan tarkastelu katvealueella, mutta huoltokäyntien tietoja tulee pystyä myös muokkaamaan offline-tilassa. Kun huoltaja muokkaa huoltokäynnin tietoja esimerkiksi merkitsemällä tehtävän suoritetuksi, sovelluksen pitää lähettää muokkauspyyntö palvelimelle kun verkkoyhteys on seuraavan kerran saatavilla.

Huoltokäyntien sijaintien näyttämistä varten sovellukseen tarvitaan karttapalvelu. Kartan avulla voidaan valita uuden huoltokäynnin sijainti, ja sen koordinaatit tallennetaan järjestelmään. Sovelluksen pitää pystyä paikantamaan käyttäjä, jotta tämän sijainti voidaan näyttää sovelluksen kartalla huoltokäynnin sijainnin lisäksi.

Kun vastaanottaja luo uuden huoltokäynnin ja määrää sen huoltajalle, sovelluksen tulee ilmoittaa uudesta huoltokäynnistä välittömästi kyseiselle käyttäjälle. Ilmoituksen tulee lähteä myös jos olemassa olevan huoltokäynnin suorittaja vaihdetaan toiseen. Käyttäjän tulee huomata ilmoitus, vaikka sovellus ei olisi sillä hetkellä aktiivisena. Tätä tarkoitusta varten mobiilisovelluksista tutut push-ilmoitukset ovat sopivia. Näin huoltaja saa välittömästi tiedon uudesta huoltokäynnistä, jolloin niitä voidaan määrätä lyhyelläkin varoitusajalla.

Huoltokäyntiä suorittaessa huoltajan tulee identifioida huollettavat laitteet, joita kyseinen huoltokäynti koskee. Jokaiseen huoltotehtävään sisältyvään tehtävään on tallennettu koodi eli huollettavan laitteen ID. Jos kyseessä on bluetooth-yhteyteen kykenevä laite, huoltajan tulee pystyä yhdistämään siihen älypuhelimen bluetoothia käyttämällä ja sitä kautta vertaamaan laitteen ID:tä. Sovelluksen tulee pystyä hyödyntämään älypuhelimensa NFC-lukijaa, jotta huollettava laite voidaan tunnistaa siinä olevan NFC-tunnisteen avulla. Jos huollettavassa laitteessa on QR-koodi, huoltajan pitää pystyä lukemaan QR-koodin sisältö älypuhelimensa kameran avulla.

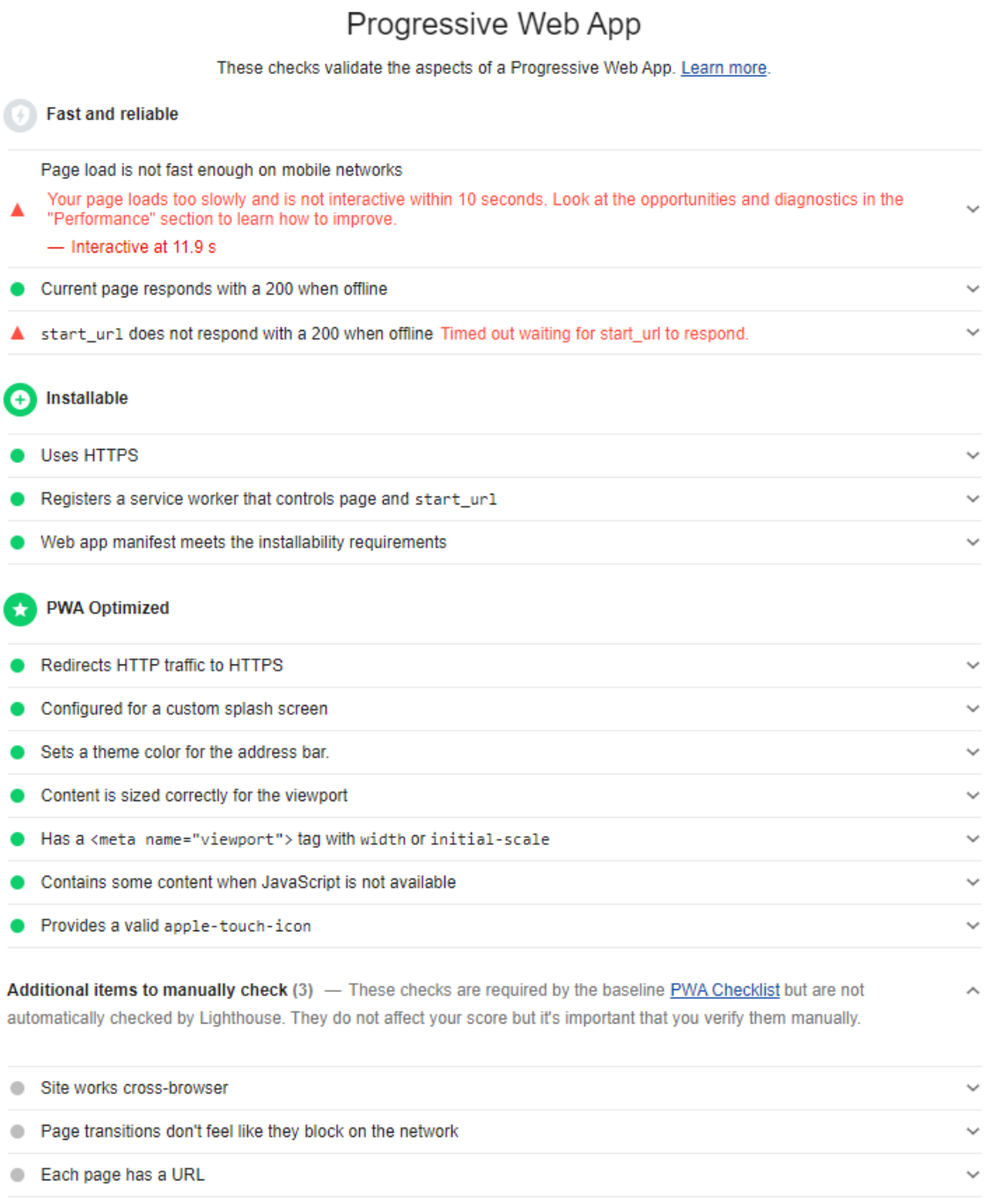
4.2 PWA-vaatimukset

PWA-sovelluksen vaatimuksia voidaan tarkastella käyttämällä Googlen kehittämää Lighthousea, joka on avoimen lähdekoodin työkalu verkkosivujen laadun tarkasteluun. Lighthousea käytetään ajamalla se komentoriviltä, Node-moduulina tai Chrome-selaimen lisäosana. PWA-auditoinnin lisäksi Lighthousella voidaan suorittaa auditointeja SEO- (engl. Search Engine Optimization) ja saavutettavuusvaatimusten osalta. [47]

Kuvassa 4.1 on esimerkki Lighthousen PWA-auditin tuloksista. Vaatimuksia löytyy kolmesta eri kategoriasta: nopeus ja luotettavuus, asennettavuus ja PWA-optimointi. Lisäksi on muita vaatimuksia, joita ei voida tarkistaa Lighthousella. Ne eivät vaikuta auditoinnin tulokseen, mutta ne tulee käydä läpi manuaalisesti.

4.2.1 Nopeus ja luotettavuus

Lighthouse-raportin nopeus ja luotettavuus-kategorian ensimmäisen kohdan mukaan verkkosivun tulee latautua nopeasti myös hitaammilla mobiiliverkkoyhteyksillä, sillä useat käyttäjät vierailevat sivustoilla mobiiliverkkoyhteyden välityksellä [55]. Verkkosivun nopeutta voidaan mitata kahdesta näkökulmasta, FMP:n (engl. first meaningful paint) sekä TTI:n (engl. time to interactive) osalta [55]. FMP kertoo kuinka kauan kestää ladata sivun pääsisältö näkyville, ja TTI mittaa milloin sivu on täysin interaktiivinen [32, 71]. Sivu on täysin interaktiivinen, kun FMP on saavutettu, tapahtumakäsittelijät (engl. event handler) on rekisteröity suurimmalle osalle sivun näkyvistä elementeistä, ja sivu vastaa käyttäjän toimintaan 50 millisekunnin sisällä [71]. Jos TTI on esimerkiksi kymmenen sekuntia, käyttäjän näkökulmasta verkkosivun latausaika on kymmenen sekuntia vaikka FMP olisi vain



Kuva 4.1. *Esimerkki Lighthousen PWA-raportista.*

yksi sekunti [55]. Verkkosivu täyttää Lighthousen mobiiliverkkolatausaikaa koskevan vaatimuksen, jos TTI on maksimissaan kymmenen sekuntia hitaalla 4G-yhteydellä [55].

Kuvan 4.1 nopeus ja luotettavuus-kategorian toisen ja kolmannen vaatimuksen perusteella sivuston tulee tarjota sisältöä myös offline-tilassa. Verkkosivun manifestin tulee sisältää start_url-arvon, joka kertoo sovelluksen käynnistyksen yhteydessä ladattavan URL:n [68]. Offline-tilassa tarjottava sisältö mahdollistetaan palvelutyöläisen avulla [17].

4.2.2 Asennettavuus

Lighthouse-raportin toinen kategoria, asennettavuus, vaatii että sovellus käyttää salattua HTTPS-yhteyttä HTTP-yhteyden sijaan. Googlen mukaan kaikkien verkkosivustojen tulisi käyttää HTTPS-yhteyttä, vaikka ne eivät käsittelisi arkaluontoisia tietoja [25]. Jokainen suojaamaton HTTP-pyyntö voi paljastaa tietoa käyttäjän identiteetistä ja käyttäytymisestä [8]. Keräämällä suojaamatonta tietoa eri lähteistä tunkeutujat voivat päätellä käyttäjän henkilöllisyyden. [8].

Asennettavuus-kategorian toisena vaatimuksena on palvelutyöläisen rekisteröinti sekä start_url-arvon määrittäminen. Palvelutyöläisen rekisteröinnin myötä sovelluksessa voidaan hyödyntää PWA-ominaisuuksia, kuten push-ilmoituksia sekä offline-toimintoja [23]. Google suosittelee Workbox-kirjastopaketin käyttöä palvelutyöläisen rekisteröintiin [23].

Kategorian kolmas vaatimus koskee verkkosovellusmanifestia (engl. web app manifest) ja sen asennettavuusominaisuuksia. Manifestin tulee sisältää seuraavat arvot, jotta vaatimus täyttyy:

- short_name tai name
- icons
- start_url
- display
- prefer_related_applications.

Jos edellä mainitut arvot on määritelty oikein, Chrome lähettää beforeinstallprompt-tapahtuman (engl. event) automaattisesti. Tämän tapahtuman avulla voidaan pyytää käyttäjää asentamaan PWA-sovellus omalle laitteelle. Muilla selaimilla, kuten Firefoxilla, on omat edellytyksensä kyseisen tapahtuman lähettämiselle. [75]

4.2.3 PWA-optimointi

Lighthouse-raportin kolmannen kategorian, PWA-optimoinnin, ensimmäisen vaatimuksen mukaan sovelluksen tulee ohjata HTTP-liikenne uudelleen salattuun HTTPS:ään. Alikohdassa 4.2.2 selitetään HTTPS-vaatimus tarkemmin. HTTP-liikenteen uudelleenohjaamisen pystyy mahdollistamaan palvelimen asetuksia muuttamalla.

Kategorian mukaan PWA-sovellukselle täytyy määritellä oma aloitusruutu (engl. splash screen), jotta sovelluksesta saadaan natiivisovelluksen kaltainen. Aloitusruutu näytetään laitteen ruudulla, kun laitteelle asennettu sovellus käynnistetään. Androidilla aloitusruutu on oletuksena tyhjä valkoinen ruutu. Androidin Chrome-selain näyttää kustomoidun aloitusruudun, jos seuraavat arvot on määritelty oikein verkkosovellusmanifestissa: name, background_color ja icons. [44]

Kolmas vaatimus kategoriassa koskee teeman värin asettamista. Teeman väri, jota käytetään selaimen osoitepalkissa, voidaan määritellä esimerkiksi brändin mukaiseksi. Tavallista selainkäyttöä varten teeman väri määritellään sivun HTML:ssä meta-elementillä. PWA-käyttöä varten teeman väri täytyy lisätä myös theme_color-arvona verkkosovellusmanifestiin. [24]

PWA-optimoinnin kategoria vaatii myös, että verkkosovelluksen sisältö on sovitettu selaimen näköporttiin (engl. viewport). Näköportilla tarkoitetaan selainikkunan aluetta, jossa näytetään verkkosivun sisältö. Valitun sivun sisältö voi skaalautua huonosti tai osa siitä voi jäädä piiloon, jos vaatimus on jätetty huomiotta. [16]

Kategorian viidennen vaatimuksen mukaan sovelluksen HTML:ssä tulee määritellä viewport-metatunniste (engl. meta tag), jonka content-attribuutissa on mukana width- ja initial-scale-arvot. Width-arvolla asetetaan näköportin koko ja initial-scale-arvo määrittää zoomauksen tason. Viewport-metatunniste ja sen attribuutit varmistavat, että verkkosovellus on oikean kokoinen eri laitteilla. [20]

Kuudes PWA-optimoinnin vaatimus koskee tilannetta, jossa JavaScript on estetty selaimessa. Verkkosivun tulee tässäkin tapauksessa tarjota sisältöä HTML:ssä, jolla voidaan esimerkiksi pyytää käyttäjää ottamaan JavaScript käyttöön selaimessa. Noscript-elementtiä voidaan hyödyntää, jos verkkosivu ehdottomasti vaatii JavaScriptin. Noscript määrittää HTML-sisällön, joka näytetään jos sivun ohjelmakoodi ei ole tuettu selaimessa [22]

PWA-optimointikategorian seuraavan vaatimuksen mukaan sovelluksessa tulee määritellä apple-touch-icon -ikoni iOS-laitteita varten. Kyseinen ikoni ja ikonin polku tulee määritellä HTML:n head-osiossa, jonka myötä kuva näytetään iOS-laitteen työpöydällä asentamisen jälkeen. Kuvalle on kokosuositukset, ja kuvan taustan ei tule olla läpinäkyvä hyvän käyttökokemuksen varmistamiseksi. [21]

Kategorian viimeinen vaatimus käsittelee Android-laitteelle asennetun PWA-sovelluksen työpöytäikonia. Sovellukselle tulee määritellä maskitettava (engl. maskable) ikoni, jotta laitteen työpöydällä näytettävä ikoni täyttää kokonaan sille varatun tilan. Muussa tapauksessa ikoni saa valkoisen taustan. Ikoni tulee määritellä verkkosovellusmanifestin icons-kenttään, ja antaa sille purpose-arvoksi maskable. [49]

4.2.4 Muut vaatimukset

Lighthouse-raportin muiden, manuaalisesti tarkistettavien vaatimusten mukaan verkkosivustojen tulisi toimia kaikilla merkittävimmillä selaimilla. Googlen mukaan sovellusta tulisi testata Chromella, Firefoxilla, Edgellä sekä Safarilla, sekä korjata niillä mahdollisesti ilmenevät ongelmat. Lisäksi Google suosittelee Workbox-työkalupaketin käyttöä. [66]

Siirtymät verkkosivuston eri sivujen välillä tulisi vaikuttaa nopeilta myös hitaalla verkkoyhteydellä. Tyhjän sivun näyttämisen sijaan informoidaan käyttäjää meneillään olevasta tietojen hakemisesta esimerkiksi ikonilla. Latausindikaattorin voi näyttää heti, kun käyttäjä on painanut linkkiä ja seuraavan sivun tietojen hakeminen on aloitettu. Toinen tapa on ensin siirtyä uudelle sivulle, jossa näytetään latausindikaattori tietojen hakemisen ajan. Yhden sivun sovelluksen tapauksessa voidaan välittömästi siirtyä seuraavaan näkymään, jossa tietojen hakemisen ajan näytetään luurankonäkymä (engl. skeleton screen). [56]

Lighthousen muiden vaatimusten mukaan jokaisella verkkosivuston sivulla tulisi olla oma, yksilöivä URL. Tällöin niitä voidaan helposti jakaa sosiaalisessa mediassa. Sivuston jokaisen sivun URL:n toimivuus tulisi testata syöttämällä se uuteen selainikkunaan. Jos kyseessä on yhden sivun sovellus, tulee tarkistaa että asiakaspuolen (engl. client-side) reitittäjä pystyy rakentamaan sovelluksen tilan. [26]

5 PROTOTYYPIN MÄÄRITTELY

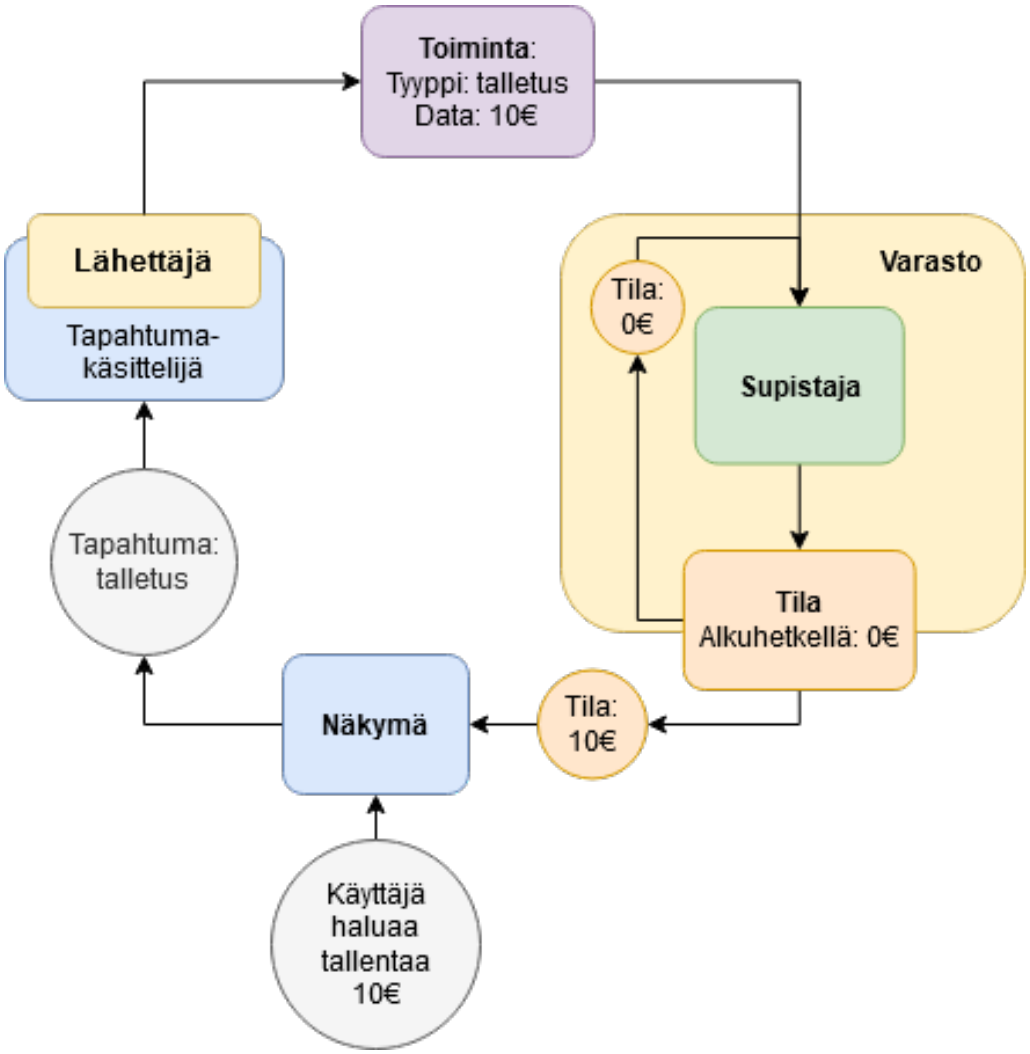
Avaamalla asennettu progressiivinen web-sovellus desktop- tai mobiililaitteella sovellus avautuu pelkistetyssä web-selainikkunassa, josta puuttuu muun muassa osoitepalkki. Tällöin päästään lähemmäs natiivisovelluksen käyttökokemusta. Itse web-sovellukselta kuitenkin vaaditaan tiettyjä ominaisuuksia, jotta erot natiivisovellukseen voidaan häivyttää. Sovelluksen näkymien välillä siirtyminen tulee olla sujuvaa, joten yhden sivun sovellus eli SPA (engl. single-page application) on luonteva valinta. SPA-sovellusten tapauksessa näkymästä toiseen siirtyminen ei vaadi sivun lataamista palvelimelta, vaan näkymien esittämiseen vaadittu koodi haetaan yhdellä latauskerralla. Lisäksi web-sovelluksen ulkoasu tulee olla responsiivinen, jotta käyttökokemus olisi lähellä natiivisovellusta. Web-sovelluksen rakenne ja tiedostot alustetaan Create React App-paketin avulla, joka luo yksinkertaisen SPA-sovelluksen automaattisesti.

5.1 Käyttöliittymä

Projektin web-sovellusta varten valittiin Facebookin kehittämä avoimen lähdekoodin React-kirjasto, jonka avulla voidaan toteuttaa React-komponenteista koostuvia käyttöliittymiä. Kyseisiä komponentteja voidaan toteuttaa JSX-syntaksilla, joka yhdistää logiikan ja merkintäkielen samaan tiedostoon. Jokaisella React-komponentilla on oma tilansa, jonka muuttuessa komponentti piirretään uudelleen (engl. rerender). Sovelluksen käyttöliittymä toteutetaan Material-UI:n avulla, joka on Material Design-muotokieltä mukaileva käyttöliittymäohjelmistokehys. Material-UI tarjoaa valmiin teeman sekä React-komponentteja, joiden avulla käyttöliittymästä saadaan responsiivinen. Lisäksi mukana tulee laaja valikoima valmiita ikoneja eri käyttötarkoituksiin. Sovelluksen näkymät suunniteltiin sovelluksen vaatimusten pohjalta.

5.2 Tilanhallinta

React tarjoaa käyttöliittymäkomponenttien tilanhallinnan, mutta web-sovelluksen globaalia tilanhallintaa varten projektiin otettiin mukaan Redux-kirjasto. Mukana on myös Redux Toolkit -kirjasto, joka sisältää työkaluja Reduxin käytön helpottamiseksi. React-komponenteille voidaan antaa pääsy Reduxissa säilytettävään tietoon, jolloin komponentit piirretään uudelleen mikäli niille jaettu Redux-tieto muuttuu.



Kuva 5.1. *Tiedonkulku Redux-tilanhallinnassa [3].*

Kuvassa 5.1 esitetään tiedonkulkua Redux-tilanhallinnassa esimerkkitapauksessa, jossa käyttäjä tallentaa summan Redux-tilanhallintaan. Käyttäjän painaessa tallennusnappia näkymässä laukaistaan tieto tallennustapahtumasta, jonka tapahtumakäsittelijä (engl. event listener) nappaa. Tapahtumakäsittelijässä oleva lähettäjä (engl. dispatch) välittää tiedon tallennuksesta luomalla uuden toiminnan (engl. action). Toiminta lähetetään varastolle (engl. store), joita voi olla sovelluksessa useita. Toiminta päätyy sille varastolle, jonka lähettäjää on käytetty. Varastossa sijaitsee tila (engl. state), eli Reduxin säilömä data, ja supistaja (engl. reducer).

Supistajalle välitetään tallennustoiminta sekä sen hetkinen tila. Supistaja ei suoraan muokkaa tilan tietoa, vaan kopioi tilan ja tekee kopioon muokkaukset saamansa toiminnan tyypin perusteella. Esimerkkitapauksessa supistaja lisää sen hetkiseen summaan käyttäjän tallentaman summan, ja palauttaa uuden tilan. Päivitetty tilatieto välitetään näkymälle, jolloin sovelluksen käyttöliittymä päivitetään vastaamaan uutta tilaa.

5.3 Reititys

Projektin web-sovelluksessa hyödynnetään React Router -kirjastoa, jotta se voidaan toteuttaa yhden sivun sovelluksena. Perinteisessä staattisessa reitityksessä, jota esimerkiksi Angular ja Express käyttää, sovelluksen reitit määritellään jo sovelluksen alustusvaiheessa. React Routerin käyttämän dynaamisen reitityksen tapauksessa reittejä voidaan muokata dynaamisesti. Tämän ansiosta reitin näkymästä voidaan esimerkiksi jättää tietty osio näyttämättä pienemmillä näytöillä.

5.4 Progressiivisen web-sovelluksen ominaisuudet

Projektin vaatimusten perusteella web-sovelluksen tulee olla progressiivinen web-sovellus, joka kykenee muun muassa taustasynkronointiin sekä offline-käyttöön. Nämä ominaisuudet toteutetaan Googlen toteuttaman Workbox-kirjastopaketin avulla. Workbox on kokoelma kirjastoja, jotka helpottavat palvelutyöntekijöiden toteuttamista ja kehittämistä. Offline-käytön ja taustasynkronoinnin lisäksi esimerkiksi resurssien, kuten JavaScript- ja CSS-tiedostojen, tallentaminen välimuistiin voidaan toteuttaa kätevästi Workboxin avulla.

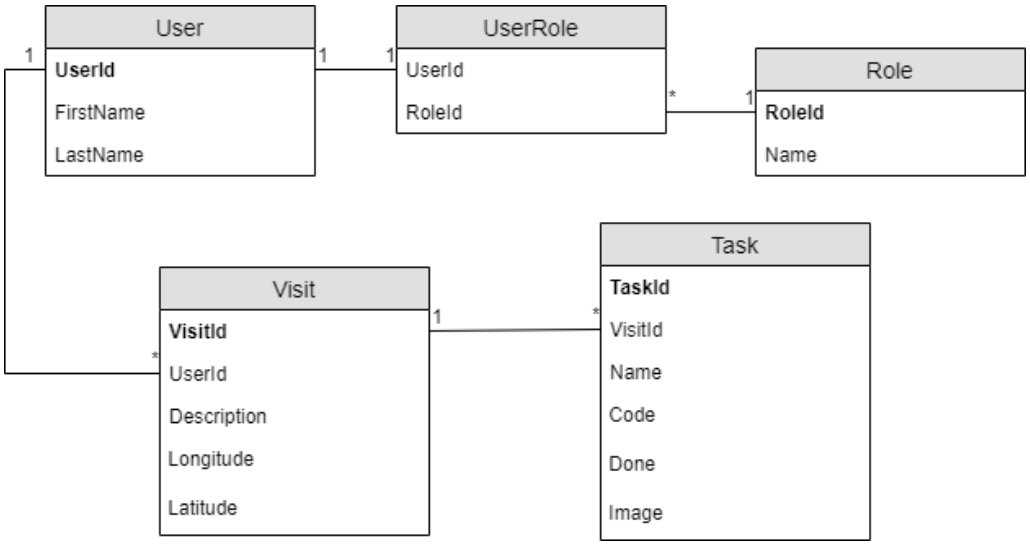
5.5 Sijainti

Web-sovelluksen tulee näyttää kartta, sekä piirtää siihen tarvittaessa sijainteja. Tähän tarkoitukseen valittiin avoimen lähdekoodin Leaflet-kirjasto, jolla voidaan luoda mobiili-käyttöönkin soveltuvia interaktiivisia karttoja. Leafletin ohella käytetään myös React Leaflet -kirjastoa, joka abstrahoi Leafletin ominaisuudet React-komponenteiksi ja joka hyödyntää Reactin elinkaarimetodeja (engl. lifecycle method). Leafletin avulla esitettävä karttadata haetaan Mapbox-karttapalvelun rajapinnasta.

5.6 Backend

Web-sovelluksen vaatimuksiin kuuluu huoltokäynteihin ja niihin kuuluvien tehtävien listaaminen, muokkaaminen ja poistaminen. Sovelluksen tulee myös hallinnoida käyttäjiä sekä käyttäjärooleja. Huoltotehtävä- ja käyttäjätietojen säilöminen tapahtuu tietokannassa, ja backend-sovelluksen kautta tietoihin voidaan kohdistaa erilaisia toimenpiteitä.

Backend-sovellus toteutetaan Microsoftin ylläpitämällä, avoimen lähdekoodin .NET Core -ohjelmistokehyksellä, jonka avulla voidaan luoda web-sovellusten lisäksi muun muassa IoT- tai koneoppimissovelluksia. Vuoden 2020 helmikuussa teetetyn Stack Overflow -kyselyn mukaan .NET Core on ollut suosituimpien web-ohjelmistokehysten joukossa [1]. .NET Corella toteutettava backend-sovellus hyödyntää OpenIddict-kirjastoa, jonka avulla voidaan toteuttaa käyttäjien ja käyttäjäroolien hallinta. Backend-sovelluksen käyttämä



Kuva 5.2. *Tietokantamalli.*

tietokanta toteutetaan Microsoftin SQL Server -hallintajärjestelmällä, jota hyödyntäen voidaan luoda relaatiotietokantoja myös laajempia järjestelmiä varten. Toteutettava tietokantarakenne, joka on esitetty kuvassa 5.2, suunniteltiin sovelluksen vaatimusten perusteella.

Tietokannassa säilytetään tietoa käyttäjistä User-tietokantataulussa, johon tallennetaan käyttäjien pääavain sekä etu- ja sukunimi. Ennalta määrätyt käyttäjäroolien pääavaimet ja nimet tallennetaan Role-tauluun. Käyttäjille annetut roolit saadaan UserRole-liitostaulusta, joka liittää yksittäisen käyttäjän pääavaimen roolin pääavaimeen. Esimerkkikäyttäjät luodaan suoraan tietokantaan, sillä käyttäjien rekisteröinti ei ole olennainen toiminto tämän työn kannalta.

Sovelluksessa luodut huoltokäynnit tallennetaan Visit-tietokantatauluun, joka sisältää pääavaimen lisäksi huoltokäynnin kuvauksen, sijainnin leveys- ja pituusasteina sekä vierasavaimen User-tauluun, joka määrittelee käyttäjän jolle kyseinen käynti on määrätty. Task-tietokantataulu sisältää sovelluksessa luodut työtehtävät. Task-taulu sisältää työtehtävän pääavaimen, nimen, koodin, kuvan, tiedon onko tehtävä suoritettu sekä vierasavaimen Visit-tauluun, joka kertoo mihin huoltokäyntiin työtehtävä on liitetty. Laite, jota työtehtävä koskee, on tarkoitus tunnistaa kooditiedon avulla. Työtehtävän suoritus voidaan varmistaa kuvadatan avulla, jonka käyttäjä tallentaa suoritettuaan tehtävän.

Backend-sovellukseen toteutetaan rajapinta huoltokäyntejä, työtehtäviä ja käyttäjiä varten kuvan 5.3 mukaisesti. GET-pyyntöjen avulla voidaan listata resurssit tai hakea yksittäisen resurssin tiedot. Vastaanottaja-roolissa oleva käyttäjä pystyy hakemaan kaikkien resurssien tiedot, mutta huoltaja näkee vain itselleen määrätyt huoltotehtävät ja niihin liittyvät resurssit. Ainoastaan vastaanottajat voivat luoda uusia resursseja voidaan luoda POST-pyynnöillä, sillä kyseiset toiminnot on rajattu pois huoltajilta. Huoltajat pystyvät muokkaamaan omien työtehtävien tietoja esimerkiksi merkkaamalla ne suoritetuiksi, kun taas vastaanottajat voivat rajatta käyttää PUT-pyyntöjä resurssien muokkaamiseen. Myös

HTTP-verbi	Polku	Toiminto
GET	Api/Visits	Hakee listan huoltokäynneistä
GET	Api/Visits/<ID>	Hakee ID:n osoittaman huoltokäynnin tiedot
POST	Api/Visits	Luo uuden huoltokäynnin
PUT	Api/Visits/<ID>	Päivittää ID:n osoittaman huoltokäynnin tiedot
DELETE	Api/Visits/<ID>	Poistaa ID:n osoittaman huoltokäynnin
GET	Api/Tasks	Hakee listan työtehtävistä
GET	Api/Tasks/<ID>	Hakee ID:n osoittaman työtehtävän tiedot
POST	Api/Tasks	Luo uuden työtehtävän
PUT	Api/Tasks/<ID>	Päivittää ID:n osoittaman työtehtävän tiedot
DELETE	Api/Tasks/<ID>	Poistaa ID:n osoittaman työtehtävän
GET	Api/Users	Hakee listan rekisteröityneistä käyttäjistä

Kuva 5.3. *Backend-sovelluksen rajapintakuvaus.*

resurssien poistaminen DELETE-pyynnöillä on mahdollista vain vastaanottajille.

6 TOTEUTUS

Tässä luvussa esitellään sovelluksen toteutuksesta osat, joiden avulla vastataan luvussa 4 esitettyihin vaatimuksiin. Kyseiset vaatimukset sisältävät sekä kuvitteellisen asiakasyrityksen tarpeita että Lighthouse-työkalun asettamia PWA-vaatimuksia.

6.1 Palvelutyöläinen

Sovelluksen palvelutyöläisen käyttöönotto aloitetaan index.js-tiedostossa, josta React-sovelluksen suoritus aloitetaan. Kyseisessä tiedostossa kutsutaan erillisessä tiedostossa sijaitsevaa register-funktiota. Tälle funktiolle on mahdollista antaa parametrin mukana onSuccess- ja onUpdate-takaisinkutsufunktiot (engl. callback function). Näiden funktioiden avulla voidaan suorittaa toimia React-sovelluksen puolella palvelutyöläisen rekisteröinnin ja päivittymisen myötä.

Palvelutyöläisen rekisteröinti on rajoitettu tuotantoversioon, jotta sovelluksen kehittäminen olisi sujuvampaa. Nähdäkseen tekemänsä koodimuutokset kehittäjän tarvitsee vain päivittää web-sovellus selaimessa. Mikäli palvelutyöläistä käytettäisiin myös kehitysversiossa, kehittäjä tulisi avata web-sovellus uudelleen nähdäkseen tekemänsä muutokset.

Ennen palvelutyöläisen rekisteröintiä tarkistetaan palvelutyöläisen osoite. Sovellukselle ei voida rekisteröidä palvelutyöläistä, joka sijaitsee eri lähteessä (engl. origin) kuin sovellus itse. Palvelutyöläisen toiminta-alue (engl. scope) rajoittuu palvelutyöläisen omaan sijaintiin sekä tiedostohierarkiassa sen alla oleviin dokumentteihin. Toisiin lähteisiin tehdyt resurssihaut menevät kuitenkin palvelutyöläisen läpi, jos ne on tehty palvelutyöläisen toiminta-alueella sijaitsevan dokumentin kautta. [72]

Tarkistusten jälkeen luodaan tapahtumakuuntelija, joka ajetaan sivun latauduttua kokonaisuudessaan. Tapahtumakuuntelijassa palvelutyöläisen rekisteröinnin suoritus muuttuu riippuen siitä, ajetaanko sovellusta paikallisessa ympäristössä. Testattaessa eri sovelluksia paikallisessa ympäristössä on mahdollista, että palvelutyöläistä käytetään väärän sovelluksen kohdalla. Tästä syystä paikallisessa ympäristössä ajettaessa tarkistetaan, löytyykö palvelutyöläisen koodia sille määrätystä osoitteesta. Jos koodia ei löydy tai osoitteesta saatu data ei ole JavaScript-koodia, poistetaan aktiivisen palvelutyöläisen rekisteröinti sekä ladataan sivu uudelleen. Jos palvelutyöläisen koodi löytyy, tai kyseessä ei ole paikallinen ympäristö, siirrytään palvelutyöläisen varsinaiseen rekisteröintiin registerServiceWorker-funktiossa.

```
navigator.serviceWorker
  .register(serviceWorkerUrl)
  .then(registration => {
    registration.onupdatefound = () => {
      // Haetaan asentuva palvelutyöläinen.
      const installingWorker = registration.installing;
      if (installingWorker == null) {
        return;
      }
      installingWorker.onstatechange = () => {
        if (installingWorker.state === 'installed') {
          if (navigator.serviceWorker.controller) {
            // Uusi palvelutyöläinen haettu, edellinen vielä käytössä.
            if (config && config.onUpdate) {
              config.onUpdate(registration);
            }
          } else {
            // Palvelutyöläinen ensimmäistä kertaa otettu käyttöön.
            if (config && config.onSuccess) {
              config.onSuccess(registration);
            }
          }
        }
      };
    };
  })
  .catch(error => console.error(error));
```

Ohjelma 6.1. *RegisterServiceWorker-funktion sisältö.*

RegisterServiceWorker-funktio, joka on toteutettu ohjelman 6.1 mukaisesti, rekisteröi palvelutyöläisen sille annetulla osoitteella. Kyseisessä funktiossa voidaan ajaa myös onSuccess- tai onUpdate-takaisinkutsufunktio riippuen siitä, onko kyseessä ensimmäinen asennuskerta. OnSuccess-funktio ajetaan ensimmäisen asennuskerran kohdalla, eli sitä hyödyntämällä voidaan esimerkiksi näyttää ilmoitus juuri käyttöönotetuista offline-toiminnallisuuksista React-sovelluksen puolella. Vastaavasti onUpdate-funktion avulla voidaan huomauttaa käyttäjälle, että sovelluksesta on saatavilla uusi versio, jonka käyttöönotto edellyttää sivun uudelleenlataamista.

```
importScripts("https://storage.googleapis.com/.../workbox-sw.js");
importScripts("/precache-manifest.60b...9354.js");
self.addEventListener('message', (event) => {
  if (event.data && event.data.type === 'SKIP_WAITING') {
    // Aktivoidaan palvelutyöläinen.
    self.skipWaiting();
  }
});
// Asetetaan palvelutyöläinen ohjaajaksi.
workbox.core.clientsClaim();
// Päivitetään ennakkoon tallennettavat tiedostot.
self.__precacheManifest = [].concat(
  self.__precacheManifest || []
);
workbox.precaching.precacheAndRoute(
  self.__precacheManifest, {}
);
workbox.routing.registerNavigationRoute(
  workbox.precaching.getCacheKeyForURL("/index.html"),
  { blacklist: [/^\/_/,/\/[^/?]+\.[^/]+$/] }
);
```

Ohjelma 6.2. *Service-worker.js-tiedoston sisältö.*

Ohjelmassa 6.2 on palvelutyöläisen koodi, jota käytetään palvelutyöläisen rekisteröinnissä. Ohjelman skipWaiting- ja clientsClaim-metodien avulla asennettava palvelutyöläinen voidaan tarvittaessa aktivoida ja asettaa ohjaajaksi (engl. controller) vapaana oleville sovelluksille ilman web-sovelluksen lataamista uudelleen [7, 87]. Näitä metodeja tulee käyttää harkiten, sillä ne voivat johtaa ongelmiin esimerkiksi ennakkotallennettujen tiedostojen kanssa [87].

Ohjelma 6.2 kattaa myös ennakkoon välimuistiin tallennettujen tiedostojen käsittelyn. Ennakkotallennusmanifestin (engl. precache manifest) perusteella ohjelmassa luodaan reitit, joihin vastataan ennakkotallennusdatalla.

Palvelutyöläisen rekisteröintiin hyödynnettyjen rajapintojen, ServiceWorkerContainerin ja ServiceWorkerRegistrationin, ominaisuudet on tuettu kaikissa selaimissa paitsi Internet Explorerissa [63, 64]. Workbox-kirjasto, jonka avulla palvelutyöläinen luodaan, on tuettu kaikissa suosituissa selaimissa paitsi Internet Explorerissa [33]. Palvelutyöläisen yhteydessä käytetty ImportScripts-metodi on tuettu kaikissa selaimissa [88].

6.2 Asennettavuus

Progressiivisen verkkosovelluksen asennettavuus ilman erillistä verkkosovelluskauppaa mahdollistetaan verkkosovellusmanifestin avulla. Sovelluksen ja sen manifestin pitää täyttää tietyt kriteerit, jotta sovellus voidaan asentaa laitteelle: sovellus on palveltuna suojatun verkkoyhteyden eli HTTPS:n yli, asennettua sovellusta varten löytyy ikoni, ja manifesti sisältää background_color-, display-, icons-, name-, short_name- sekä start_url-arvot. Chromium-pohjaiset selaimet, kuten Edge ja Chrome, vaativat myös rekisteröidyn palvelutyöläisen. [38]

```
{
  "short_name": "Maintenance",
  "name": "Maintenance App",
  "icons": [
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192",
      "purpose": "any maskable"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#0057ae"
}
```

Ohjelma 6.3. *Manifest.json-tiedoston sisältö.*

Sovelluksen verkkosovellusmanifesti on toteutettu ohjelman 6.3 mukaisesti, ja se on liitetty sovellukseen index.html-tiedoston head-elementissä. Manifestissa on sovelluksen nimen lisäksi lyhyt versio nimestä, joka näytetään asennetun sovelluksen ikonin yhteydessä laitteen työpöydällä. Manifestin icons-taulukosta löytyy sovelluksen eri päätelaitteisiin soveltuvat ikonit. Tuki progressiivisen verkkosovelluksen asentamiselle löytyy kaikista desktop- ja mobiiliselaimista paitsi Internet Explorerista, desktop-Safarista ja desktop-Operasta, ja Firefoxin desktop-versioon tuki on tulossa [13].

PWA-optimoidulla web-sovelluksella tulee olla maskitettava ikoni. Kyseessä on adaptiivinen ikonityyppi, joka näyttää tarkoituksenmukaiselta eri päätelaitteilla. Käytettävän ikonin muoto ja koko riippuvat laitteen käyttöjärjestelmästä sekä selaimesta. Mikäli progressiivisen web-sovelluksen ikoni ei ole maskitettava, ikonille annetaan valkoinen taustaväri Android-laitteilla [49].

Kuva 6.1 on kuvakaappaus Maskable.app Editor-sivustolta, jonka avulla voidaan generoida maskitettavia ikoneja. Editorissa ikonille voidaan määritellä kerroksia, jotka tässä

Kuva 6.1. *Kuvakaappaus maskitettavan ikonin generointipalvelusta.*

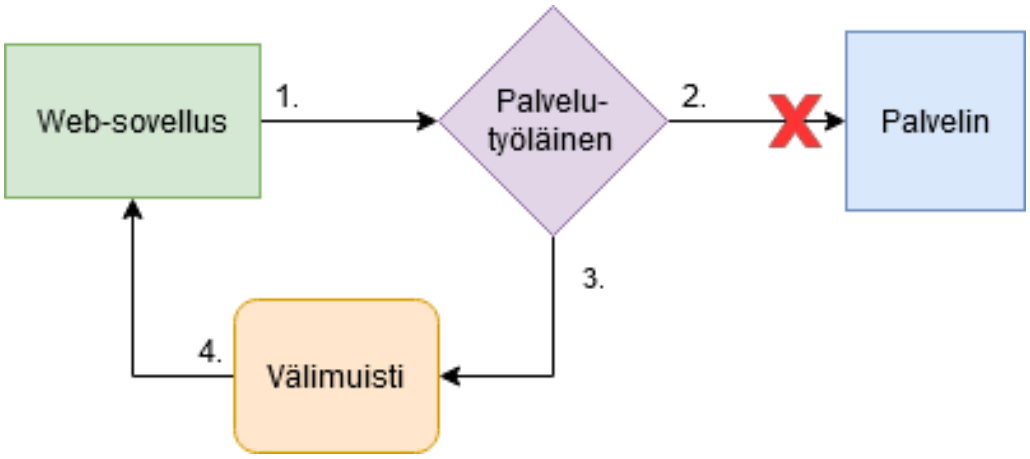
tapauksessa ovat kuva rattaista sekä sininen taustaväri. Kuvassa 6.1 näkyy edellä mainittujen kerrosten yhdistelmän lisäksi maskit, joilla voidaan testata ikonin ulkoasua eri tilanteissa. Turvallinen minimialue (engl. minimum safe area) on standardisoitu kuvamuoto, jota kaikkien alustojen tulee tukea. Ikonin toimiessa kaikilla maskeilla, se voidaan exportata generaattorista lopullisena versiona. Maskitettavan ikonin käyttöönottamiseksi se tulee lisätä manifest.json-tiedostoon sekä antaa sille purpose-arvoksi "any maskable"ohjelman 6.3 mukaisesti.

PWA-optimointia varten manifest.json-tiedostossa tulee olla theme_color-arvo, joka määrittää selaimen osoitepalkin värin mikäli ominaisuus on tuettu web-selaimessa [24]. Lisäksi se voi vaikuttaa käyttöjärjestelmän muuhun käyttöliittymään, kuten Androidin statuspalkkiin. Ohjelmassa 6.3 teemaväri on asetettu vastaamaan maskitetun ikonin taustaväriä.

Web-sovelluksen käynnistämisen yhteydessä näytettävää käynnistyskuvaa varten manifest.json-tiedostosta tulee löytyä sovelluksen nimi, sopiva ikoni sekä taustaväri, joka asetetaan arvolla background_color. Jos taustaväri vastaa web-sovelluksen käyttöliittymän taustaväriä, siirtymä käynnistyskuvasta varsinaiseen sovellukseen on sulava.

6.3 Offline-käyttö

Huoltajan tulee pystyä lukemaan tälle määrätyt huoltokäynnit sekä niiden sisältämät työtehtävät myös offline-tilassa. Palvelimelta haetut tiedot tulee tallentaa selaimen muistiin, jotta viimeisimmät huoltokäynnit ja työtehtävät voidaan näyttää myös mobiiliverkkoyhteyden puuttuessa. Tallentaminen on toteutettu palvelutyöläisen koodissa ohjelman 6.4 mukaisesti, jossa rekisteröidään uusi Workbox-reitti. Kyseiseen reittiin sisältyvät kaikki sovelluksesta lähtevät pyynnöt, jotka kohdistuvat palvelimen rajapintaan. Workbox-kirjaston ominaisuudet on tuettu kaikissa suosituimmissa selaimissa paitsi Internet Explorerissa [33].



Kuva 6.2. *Resurssin haku Workboxin network first-strategialla.*

```
workbox.routing.registerRoute(
  ({ url }) => url.pathname.startsWith('/api/'),
  new workbox.strategies.NetworkFirst({
    cacheName: 'api-cache',
    plugins: [
      new workbox.cacheableResponse.Plugin({ statuses: [200] }),
    ],
  }),
);
```

Ohjelma 6.4. *Palvelimelta tulevien resurssien tallentaminen palvelutyöläisessä.*

Pyynnöt käsitellään kuvassa 6.2 esitetyllä Workboxin network first -strategialla, eli pyydetty resurssi pyritään ensin hakemaan palvelimelta. Jos palvelimelta ei saada vastausta, palvelutyöläinen vastaa pyyntöön välimuistista löytyvällä resurssilla. Välimuistiin tallennettu resurssi päivitetään jokaisen palvelimelta saadun vastauksen yhteydessä. Tämä strategia sopii resursseille, jotka muuttuvat usein. [85]

Muita Workboxin strategioita ovat network only, cache first, cache only sekä stale-while-revalidate. Network only-strategiassa resurssit haetaan vain palvelimelta, eli sen kohdalla ei käytetä välimuistia. Cache first hyödyntää ensisijaisesti välimuistista löytyviä resursseja, mutta niiden puuttuessa hakee tarvittavan datan palvelimelta. Kyseinen strategia sopii ei-kriittisille resursseille, joita voidaan tallentaa välimuistiin vähitellen. Cache only sen sijaan käyttää ainoastaan välimuistiin tallennettuja resursseja. Stale-while-revalidaten kohdalla pyydetty resurssi haetaan samaan aikaan sekä välimuistista että palvelimelta. Jos se löytyy välimuistista, vastataan sillä sovelluksesta lähdettyyn pyyntöön. Muussa tapauksessa odotetaan palvelimen vastausta. Välimuistiin tallennettu resurssi päivitetään jokaisen palvelimelta saadun vastauksen yhteydessä. [85]

6.4 Taustasynkronointi

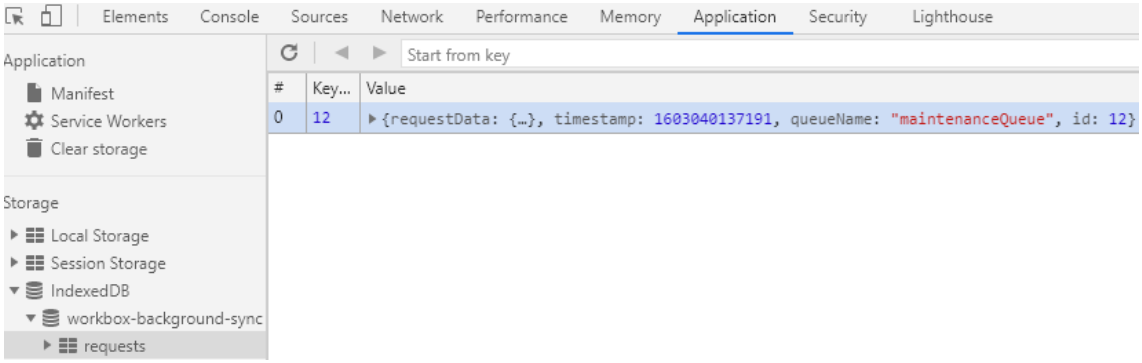
Huoltokäyntien ja työtehtävien lukemisen lisäksi huoltajan tulee pystyä muokkaamaan tietoja offline-tilassa esimerkiksi merkitsemällä työtehtävä suoritetuksi. Offline-käyttöä tukeva taustasynkronointi voidaan toteuttaa Workboxin tarjoaman WorkboxBackgroundSync-moduulin avulla. WorkboxBackgroundSync-moduulin Queue-luokkaa käyttämällä voidaan luoda uusi jono, jonka avulla epäonnistuneet pyynnöt voidaan tallentaa selaimen IndexedDB-säilöön myöhempää uudelleenlähettämistä varten [86]. Sekä IndexedDB että Workbox-kirjaston ominaisuudet on tuettu kaikissa web-selaimissa paitsi Internet Explorerissa [12, 33].

```
1  // Luodaan uusi jono.
2  const queueName = 'maintenanceQueue';
3  const queue = new workbox.backgroundSync.Queue(queueName);
4  // Kuunnellaan sovelluksesta lähteviä pyyntöjä.
5  self.addEventListener('fetch', (event) => {
6    // Ohitetaan GET-pyynnöt.
7    if (event.request.method === 'GET') {
8      return;
9    }
10   const promiseChain = fetch(event.request.clone())
11     .catch(() => {
12       // Tallennetaan epäonnistuneet pyynnöt jonoon.
13       return queue.pushRequest({ request: event.request });
14     });
15   event.waitUntil(promiseChain);
16 });
```

Ohjelma 6.5. *Palvelutyöläisen taustasynkronoinnin toteutus.*

Taustasynkronointi otetaan käyttöön palvelutyöläisen koodissa ohjelman 6.5 mukaisesti, jossa luodaan maintenanceQueue-niminen jono. Jonoon tallennetaan sovelluksesta lähteneet epäonnistuneet HTTP-pyynnöt, jos ne koskevat tiedon muokkaamista. GET-pyynnöt suoritetaan tavanomaiseen tapaan. Palvelutyöläinen yrittää lähettää jonoon tallennettuja pyyntöjä uudelleen oletuksena viikon verran [59].

Taustasynkronointia voidaan testata Chrome-selaimen kehittäjän työkalujen avulla. Jotta palvelutyöläinen asentuu ja aktivoituu, tulee aluksi vierailla testattavan verkkosivuston online-sivulla. Seuraavaksi verkkoyhteys otetaan pois käytöstä, tai testipalvelin sammutetaan, jos sovellusta ajetaan lokaalissa ympäristössä. Tämän jälkeen lähetetään esimerkiksi uuden huoltokäynnin luontipyyntö, jolloin pyyntö epäonnistuu ja se tallennetaan taustasynkronointijonoon. Taustasynkronointijonon sisältöä voidaan tarkastella kehittäjän työkalujen IndexedDB-osion avulla, jossa tallennettu pyyntö näkyy kuvan 6.3 mukaisesti. Kun verkkoyhteys palautetaan käyttöön, taustasynkronointi ajetaan automaattisesti, jos



Kuva 6.3. *Kuvakaappaus Chromen taustasynkronointijonosta.*

kyseessä on Chrome- tai Edge-selain [70]. Taustasynkronointi voidaan ajaa myös manuaalisesti joko päivittämällä verkkosivu tai Chrome-selaimen kehittäjän työkalujen palvelutyöläisosiosta.

6.5 Sijainti

Web-sovelluksessa tulee näyttää käyttäjän ja huoltotehtävien sijainnit kartalla. Käyttäjän laitteen sijaintia tarkkaillaan selaimen Geolocation-rajapinnan ja sen watchPosition-metodin avulla, jotka ovat tuettuja kaikilla selaimilla [36]. Huoltotehtävien koordinaatit on tallennettu tietokantaan, josta ne voidaan hakea palvelimen rajapinnan kautta. Kartta esitetään Leaflet-kirjaston avulla, joka on kaikilla selaimilla tuettu avoimen lähdekoodin karttakirjasto [4].

```
1  if (navigator.geolocation) {
2    // Tarkkaillaan laitteen sijaintia.
3    navigator.geolocation.watchPosition((position) => {
4      // Tallennetaan saadut koordinaatit.
5      setLongitude(position.coords.longitude);
6      setLatitude(position.coords.latitude);
7    });
8  } else {
9    // Virhekäsittely.
10   setError('Geolocation not supported by browser');
11 }
```

Ohjelma 6.6. *Laitteen sijainnin hakeminen*

Käyttäjän laitteen sijainti haetaan sovelluksen etusivulla ohjelman 6.6 mukaisesti. Selain ilmoittaa käyttäjälle, jos verkkosivusto yrittää hakea sijaintitietoja, ja pyytää käyttäjältä luvan sijaintitietojen jakamiseen verkkosivustolle. WatchPosition-metodia käyttämällä käyttäjän sijainti päivittyy, jos käyttäjän koordinaatit muuttuvat.

6.6 QR-koodin lukeminen

Sovelluksen vaatimusten mukaan huoltokäynnin työtehtävään kuuluvan laitteen voi identifioida siinä olevan QR-koodin avulla. QR-koodin lukemista varten tarvitaan pääsy mobiililaitteen kameraan, joka on mahdollista selaimen MediaDevices-rajapinnan ansiosta. QR-koodin tunnistetaan kameran ottamasta kuvadatasta jsQR-kirjaston avulla, joka palauttaa sille annetusta datasta löytämänsä QR-koodin [84].

```
return (
  <React.Fragment>
    <video id="video" ref={videoRef} />
    <canvas id="canvas" ref={canvasRef} hidden />
    <div>{error}</div>
  </React.Fragment>
);
```

Ohjelma 6.7. *CameraContainer-komponentin renderöinti.*

QR-koodin lukemista varten web-sovelluksen CameraContainer-komponenttiin luodaan video- sekä canvas-elementti ohjelmakoodin 6.7 mukaisesti. Video-elementissä toistetaan kameran ottamaa videodataa, jotta käyttäjä voi kohdistaa kameransa QR-koodia kohti. Canvas-elementti on piilotettu käyttöliittymästä, sillä sitä hyödynnetään taustalla kuvadatan välittämisessä jsQR-kirjastolle.

```
const cameraSetup = async () => {
  try {
    // Pyydetään pääsy videolaitteeseen.
    const stream = await navigator.mediaDevices.getUserMedia({
      video: true,
    });
    // Asetetaan lähde video-elementille ja käynnistetään toisto.
    videoRef.current.srcObject = stream;
    videoRef.current.play();
  } catch (error) {
    // Virhekäsittely.
    setError(true);
  }
};
```

Ohjelma 6.8. *CameraSetup-funktio*

Laitteen kameran kuvavirran lukeminen aloitetaan ohjelmakoodin 6.8 avulla. Ensin haetaan laitteessa oleva videomedialaite MediaDevices-rajapinnan getUserMedia-metodin avulla, joka pyytää myös käyttäjän suostumuksen median käyttöön [50]. Kun sopiva medialaite löytyy, asetetaan se video-elementin lähdeobjektiksi sekä aloitetaan median tois-

to. Jos laitetta ei löydy tai median toistaminen epäonnistuu, error-muuttujaan tallennetaan tieto virheestä.

```
videoRef.current.addEventListener('canplay', () => {
  if (!streaming.current) {
    streaming.current = true;
    const width = 300;
    const widthRatio = videoRef.current.videoWidth / width;
    const imageHeight = videoRef.current.videoHeight / widthRatio;
    videoRef.current.setAttribute('width', width);
    videoRef.current.setAttribute('height', imageHeight);
    const interval = setInterval(tick, 500);
    setQrInterval(interval);
  }
});
```

Ohjelma 6.9. *Canplay-tapahtumakuuntelija*

Ohjelmassa 6.9 esitetty tapahtumakuuntelijan sisältö ajetaan, kun video-elementti on vastaanottanut tarpeeksi dataa toistaakseen videokuvaa [41]. Käyttäjälle näytettävän videokuvan leveys on ennalta määritetty 300 pikseliin, jotta se sopii mobiiliversion käyttöliittymään. Tapahtumakuuntelijassa asetetaan video-elementin korkeus, jotta esitettävän videon kuvasuhde pysyy oikeana. Tapahtumakuuntelijassa aloitetaan lopuksi kutsumaan 500 millisekunnin välein tick-funktiota, jossa tarkistetaan video-elementin sisältö QR-koodin varalta.

Ohjelmassa 6.10 on esitelty QR-koodin tarkastamisesta vastaava tick-takaisinkutsufunktion sisältö, joka ajetaan mikäli video-elementti on vastaanottanut tarpeeksi dataa. Jotta video-elementin data voidaan välittää jsQR-kirjastolle, video-elementin sen hetkisestä datasta piirretään kuva piilossa olevalle canvas-elementille. 2D-grafiikkaa voidaan piirtää canvas-elementille hyödyntämällä sen CanvasRenderingContext2D-kontekstia. Piirretty kuvadata annetaan jsQR-kirjaston funktiolle, joka antaa paluuarvona kuvasta löytyvän QR-koodin arvon. Jos QR-koodi löytyi, välitetään se onChange-takaisinkutsufunktiolle.

```
if (video.readyState === video.HAVE_ENOUGH_DATA) {
  var canvasData = canvas.getContext('2d');
  canvas.height = video.videoHeight;
  canvas.width = video.videoWidth;
  canvasData.drawImage(
    video, 0, 0, canvas.width, canvas.height,
  );
  var imageData = canvasData.getImageData(
    0, 0, canvas.width, canvas.height,
  );
  var code = jsQR(
    imageData.data,
    imageData.width,
    imageData.height,
    { inversionAttempts: 'dontInvert' },
  );
  if (code) {
    onChange(code.data);
  }
}
```

Ohjelma 6.10. *QR-koodin lukemisen toteutus tick-funktiossa.*

Kameran hakemiseen käytetty MediaDevices-rajapinnan getUserMedia-metodi on tuettu kaikissa selaimissa paitsi Internet Explorerissa [50]. Videokuvan esittämisessä hyödynnetyt HTMLMediaElement-rajapinnan play- ja readyState-ominaisuudet on tuettu kaikissa selaimissa [40]. Videokuvan tarkkailussa hyödynnetyt setInterval-metodi, HTMLCanvasElement-rajapinnan getContext-metodi ja CanvasRenderingContext2D-rajapinnan metodit on tuettu kaikissa selaimissa [14, 39, 83].

6.7 Push-ilmoitukset

Huoltajan mobiililaitteella tulee näkyä push-ilmoitus, jos huoltajalle määrätään uusi huoltokäynti. Tällöin huoltaja voi reagoida uuteen huoltokäyntiin lyhyellä viiveellä. Push-ilmoitusten lähettäminen ja vastaanottaminen on toteutettu sovelluksessa käyttämällä Firebase Cloud Messaging -palvelua (FCM), jolla voi lähettää joko notifikaatioviestejä tai dataviestejä [2]. Notifikaatioviestit käsitellään Firebase-ohjelmistokehityspaketin toimesta, kun taas dataviestien käsittely on asiakassovelluksen vastuulla [2]. Firebasen Cloud Messaging -ominaisuudet eivät ole tuettu Internet Explorerissa eikä Safarin työpöytä- tai mobiiliversiossa [69].

Firebasen tarvitsemat resurssit otetaan käyttöön firebase-messaging-sw.js-tiedoston avulla, jonka sisältö on esitetty ohjelmassa 6.11. Ohjelman sisältämä init.js-tiedosto on luo-

tu Firebase-komentorivityökalun avulla, ja se alustaa sekä kehitystyökalujen asetukset että Firebase-yhteyteen vaadittavat asetukset [48]. Firebase-messaging-sw.js-tiedoston avulla sovelluksessa otetaan käyttöön uusi palvelutyöläinen, joka hoitaa push-ilmoitusten vastaanottamisen ja näyttämisen.

```
1 importScripts('/__/firebase/7.24.0/firebase-app.js');
2 importScripts('/__/firebase/7.24.0/firebase-messaging.js');
3 importScripts('/__/firebase/init.js');
```

Ohjelma 6.11. *Firebase-messaging-sw.js-tiedoston sisältö.*

Firebase-käyttöönottoa varten sovellukseen on luotu Firebase-luokka ja FirebaseContext-konteksti, jotka on toteutettu Wieruchin React-firebase-authentication -projektia mukaillen [81]. Kyseinen konteksti otetaan sovelluksessa käyttöön index.js-tiedostossa ohjelman 6.12 mukaisesti. Kontekstien avulla React-sovelluksessa voidaan jakaa dataa kätevästi eri komponenteille.

```
1 ReactDOM.render(
2   <FirebaseContext.Provider value={new Firebase()}>
3     <App />
4   </FirebaseContext.Provider>,
5   document.getElementById('root'),
6 );
```

Ohjelma 6.12. *Firebase-kontekstin luonti index.js-tiedostossa.*

Firebase-luokan rakentajassa alustetaan Firebase-sovellusinstanssi antamalla sille vaaditut asetukset ohjelman 6.13 mukaisesti, ja luokalle määritellään myös getToken-metodi. Asetuksiin tarvittavat arvot saadaan Firebase-palvelusta, ja ne on tässä tapauksessa tallennettu projektin ympäristömuuttujiin. Asetuksiin lukeutuu mukaan push-ilmoitusten lähettämiseen tarvittava messagingSenderId-arvo [28].

```
class Firebase {
  constructor() {
    // Alustetaan Firebase-sovellusinstanssi.
    app.initializeApp({
      apiKey: process.env.REACT_APP_API_KEY,
      authDomain: process.env.REACT_APP_AUTH_DOMAIN,
      databaseURL: process.env.REACT_APP_DATABASE_URL,
      projectId: process.env.REACT_APP_PROJECT_ID,
      storageBucket: process.env.REACT_APP_STORAGE_BUCKET,
      messagingSenderId: process.env.REACT_APP_MESSAGING_SENDER_ID,
      appId: process.env.REACT_APP_APP_ID,
    });
    this.messaging = app.messaging();
  }
  // Tilataan push-ilmoitukset ja haetaan rekisteröintitunnus.
  getToken = () => this.messaging.getToken();
}
```

Ohjelma 6.13. *Firebase-luokan toteutus.*

FirebaseContext luo uuden React-kontekstin, ja sille tallennetaan uusi instanssi Firebase-luokasta. Samaa instanssia voidaan jakaa eri komponenteille kontekstin avulla. Kun web-sovellus käynnistetään, ajetaan Firebase-luokan getToken-metodi, joka on esitetty ohjelmassa 6.13. Metodin avulla selain pyytää käyttäjän suostumusta push-ilmoitusten näyttämiseen, ja suostumuksen myötä sovellus rekisteröidään push-ilmoitusten tilaajaksi [31]. GetToken-metodi palauttaa selainkohtaisen rekisteröintitunnuksen, jonka avulla push-ilmoituksia voidaan kohdentaa tietylle käyttäjälle [31]. Rekisteröintitunnus lähetetään backend-sovellukselle, joka tallentaa tunnuksen tietokantaan UserNotificationToken-tauluun. Taulussa on rekisteröintitunnuksen lisäksi käyttäjän ID, jotta ilmoitukset voidaan lähettää tietyn käyttäjän laitteille.

Push-ilmoitusten lähettämistä varten backend-projektiin on asennettu FirebaseAdmin-kehitystyökalu NuGet-pakettihallintaa käyttäen. Lisäksi projektiin tarvitaan yksityisavain, jonka voi luoda Firebase-palvelussa.

Kun työtehtävä asetetaan käyttäjälle backend-sovelluksessa, tietokannasta haetaan kaikki kyseiselle käyttäjälle tallennetut Firebase-rekisteröintitunnukset. Jos rekisteröintitunnuksia löytyy, jokaista tunnusta varten luodaan uusi viesti ohjelman 6.14 mukaisesti. Push-ilmoitusta varten luodaan ensin Message-objekti, jolle voidaan antaa Token-, Condition- tai Topic-arvo viestin kohdentamista varten [29]. Condition- tai Topic-arvoa käyttämällä viesti voidaan lähettää useammalle käyttäjälle, mutta tässä tapauksessa hyödynnetään Token-muuttujaa asettamalla siihen käyttäjän selainkohtainen rekisteröintitunnus. Message-objektia varten luodaan myös uusi Notification-objekti, jolle asetetaan Title- ja Body-arvot. Title on push-ilmoituksen otsikko ja Body-arvo sisältää ilmoituksen muun

tekstin, mutta Notification-objektille voidaan myös antaa kuvan polku ImageURL-kenttään [30]. Lopuksi viesti lähetetään Firebase-palveluun SendAllAsync-metodin avulla.

```
var messages = new List<Message>();
for (var i = 0; i < notificationTokens.Count; i++)
{
    // Luodaan uusi Firebase-viesti ja tallennetaan se listaan.
    var message = new Message()
    {
        Notification = new Notification
        {
            Title = message.title,
            Body = message.body
        },
        Token = token
    };
    messages.Add(message);
}
var messaging = FirebaseMessaging.DefaultInstance;
// Lähetetään listan viestit Firebase-palveluun.
var result = await messaging.SendAllAsync(messages);
```

Ohjelma 6.14. *Push-ilmoituksen lähettäminen.*

Tässä projektissa lähetettävät push-ilmoitukset ovat notifikaatioviestien tyyppisiä, joten käyttäjän selain hoitaa ilmoituksen näyttämisen. Jos push-ilmoitukset käsitellään sovelluksen puolella, ilmoituksen tiedot tulee laittaa Message-objektin data-kenttään avain-arvopareina. Data-objektin sisältö on vapaavalintainen, mutta se ei saa sisältää from-kenttää tai muita varattuja sanoja avaimina. Notifikaatioviestien mukana voidaan myös lähettää data-objekti, kuten ohjelmassa 6.15 on esitetty. Kyseisessä tapauksessa data-objekti voidaan käsitellä kun sovellus on aktiivisena, kun taas tausta-ajossa ainoastaan notifikaatio-objektin sisältö on käytettävissä. [2]

```
{
  "message": {
    "token": "abc123...",
    "notification": {
      "title": "Esimerkkiotsikko",
      "body": "Esimerkkiviesti"
    },
    "data": {
      "avain": "arvo"
    }
  }
}
```

Ohjelma 6.15. *Notifikaatioviesti data-objektilla.*

Push-ilmoitukselle voidaan myös määritellä alustakohtaista sisältöä android-, webpush- ja apns-objekteina. Android-objektissa annetaan Android-alustaa koskevat arvot, joita ovat muun muassa viestin prioriteetti ja elinaika. iOS-käyttöjärjestelmässä esitettävien ilmoitusten asetukset ovat apns-objektissa, jonka payload-arvossa voidaan antaa esimerkiksi ilmoitusääntä koskeva tieto [35]. Webpush kattaa verkkoselaimia koskevat asetukset, joihin sisältyy headers-, notification-, data- sekä fcm_options-arvot. [60]

6.8 NFC

Rajapintamäärittely NFC-tunnisteiden lukemista varten on luonnosvaiheessa. Määrittelyn mukaan tunnisteita voidaan lukea selaimen NDEFReader-objektilla, ja tunnisteille voidaan kirjoittaa NDEFWriter-objektilla. [15]

```
const reader = new NDEFReader();
function read() {
  return new Promise((resolve, reject) => {
    const controller = new AbortController();
    controller.signal.onabort = reject;
    reader.addEventListener("reading", event => {
      // Luetaan tunnisteen sisältö kerran.
      controller.abort();
      resolve(event);
    }, { once: true });
    reader.scan({ signal: controller.signal })
      .catch(err => reject(err));
  });
}
read().then(({ serialNumber }) => {
  // Tunnisteen sisällön käsittely.
});
```

Ohjelma 6.16. *Esimerkkitoteutus NFC-tagin lukijasta.*

Web NFC-rajapintamäärittelystä löytyy NFC-tunnisteen lukemisen koodiesimerkki, joka on ohjelman 6.16 mukainen. Kyseisessä ohjelmassa NFC-tunniste luetaan kertaalleen, jonka jälkeen lukeminen keskeytetään virran säästämiseksi. Web NFC-rajapinta ei ole virallisesti tuettu millään selaimella, mutta sitä voidaan testata lokaalisti Android-puhelimella Chromen kokeellisen version avulla [10, 77].

7 HAVAINNOT

Luvussa 4 listattujen kohdeyrityksen määrittelemien kriteereiden perusteella valittiin web-sovellukseen toteutettavat ominaisuudet. Näistä ominaisuuksista olennaisimmat tutkimuskysymystä ajatellen ovat seuraavat:

- Palvelutyöläinen: PWA-ominaisuuksien mahdollistaminen
- Asennettavuus: sovelluksen asentaminen laitteelle
- Offline-käyttö: sovelluksen käyttö verkkoyhteyden puuttuessa
- Taustasynkronointi: tiedon muokkauksen yrittäminen uudelleen taustalla
- Sijainti: käyttäjän ja huoltokäyntien sijainnin näyttäminen
- QR:n lukeminen: QR-koodien lukeminen laitteen kameralla
- Push-ilmoitukset: tärkeän tiedon näyttäminen käyttäjälle
- NFC: NFC-tunnisteiden lukeminen laitteen lukijalla.

Kyseisten ominaisuuksien toteutus on esitelty tarkemmin luvussa 6. Nämä ominaisuudet on koottu kuvaan 7.1, jossa on myös eritelty eri mobiiliselainten tarjoama tuki per ominaisuus. Kyseiset selaimet on markkinaosuuksiltaan suosituimpia mobiiliselaimia, joista suosituin on Chrome noin 61 prosentin osuudellaan [51]. Kirjoittamishetkellä kuvan 7.1 mobiiliselaimet sekä niiden markkinaosuudet pyöristettyinä ovat [51]:

- Chrome (Android): 61 %
- Safari (iOS): 26 %
- Samsung Internet (Android): 7 %
- Opera Mobile (Android): 2 %
- Firefox (Android): 1 %.

Kuvassa 7.1 mainitut mobiiliselaimet tukevat lähes kaikkia web-sovellukseen toteutettuja ominaisuuksia: palvelutyöläinen, asennettavuus, offline-käyttö, taustasynkronointi, sijainti sekä QR-koodien lukeminen. Ominaisuudet, jotka on tuettu vajaavaisesti, ovat NFC ja push-ilmoitukset. NFC-tunnisteiden lukemiseen tarvittava Web NFC-rajapinta on kirjoittamishetkellä luonnosvaiheessa, joten rajapinnalle ei löydy virallista tukea vielä miltään web-selaimelta. Firefox- ja Safari-selainten osalta tuen lisääminen näyttää heikolta, sillä viralliset vastaukset ovat olleet kielteisiä [78]. Syynä Web NFC -rajapinnan vastustukseen ovat rajapinnan mahdolliset tietoturvauhat, kuten haitallinen sisältö NFC-tunnisteessa

	Chrome (Android)	Safari (iOS)	Samsung Internet (Android)	Opera Mobile (Android)	Firefox (Android)
Palvelutyöläinen	Tuettu	Tuettu	Tuettu	Tuettu	Tuettu
Asennettavuus	Tuettu	Tuettu	Tuettu	Tuettu	Tuettu
Offline-käyttö	Tuettu	Tuettu	Tuettu	Tuettu	Tuettu
Taustasynkronointi	Tuettu	Tuettu	Tuettu	Tuettu	Tuettu
Sijainti	Tuettu	Tuettu	Tuettu	Tuettu	Tuettu
QR:n lukeminen	Tuettu	Tuettu	Tuettu	Tuettu	Tuettu
Push-ilmoitukset	Tuettu	Ei tuettu	Tuettu	Tuettu	Tuettu
NFC	Ei tuettu	Ei tuettu	Ei tuettu	Ei tuettu	Ei tuettu

Kuva 7.1. *Mobiiliselainten tuki web-sovelluksen ominaisuuksille.*

[76]. Tässä projektissa käyttötapaus NFC-tunnisteiden lukuominaisuudelle on huollettavien laitteiden tunnistaminen. Saman tarpeen täyttää myös paremmin tuettu QR-lukuominaisuus, joten NFC-tuen puute ei estä mobiilisovelluksen korvaamista progressiivisella web-sovelluksella.

Kirjoittamishetkellä push-ilmoitukset ei ole tuettu Safarin iOS-versiossa, kun taas NFC-ominaisuudelta puuttuu tuki kaikista mobiiliselaimista. Push-ilmoitusten puuttuessa käyttäjälle ei voida esittää kriittistä tietoa silloin, kun sovellus ei ole aktiivisena. Tällöin käyttäjälle ei voida ilmoittaa esimerkiksi uuden työtehtävän määräämisestä, vaan käyttäjän on avattava sovellus nähdäkseen viimeisimmät työtehtävänsä. iOS-Safarin markkinaosuus on 26 prosenttia, joten push-ilmoitusten tuki puuttuu keskimäärin neljäsosalta mobiililaitteiden käyttäjistä. Kyseisen web-selaimen käyttäjäkunnan osuus on suuri ja push-ilmoitusten hyöty merkittävä, joten progressiivisten web-sovelluksia ei voida pitää mobiilisovellusten varteenotettavina korvaajina. Safarin iOS-version tulisi tukea push-rajapintaa tai tarjota vastaavanlainen vaihtoehto, jotta mobiilisovellus voitaisiin korvata progressiivisella web-sovelluksella.

Mobiiliselainten lukumäärää tarkastellessa nousee esiin web-alustan haaste: selainten tarjoama tuki vaihtelee ominaisuuksittain, joten uusimpia web-alustan ominaisuuksia ei pystytä heti hyödyntämään sovelluksessa. Mikäli web-sovelluksen käyttäjäkunta on rajattu esimerkiksi tietyn yrityksen työntekijöihin, voidaan sovelluksen käyttö rajoittaa vain ajan tasalla oleviin web-selaimiin. Sen sijaan julkisessa jaossa olevan web-sovelluksen kohdalla web-selainten rajoittaminen sulkee osan käyttäjistä kokonaan pois palvelusta. Web-sovellusten kohdalla olisi tarpeen tilastoida sovelluksen käyttäjien web-selaimet ja arvioida tiedon perusteella, missä vaiheessa uusimpien ominaisuuksien hyödyntäminen

olisi varteenotettavaa.

Kohdeyrityksen asettamien kriteereiden lisäksi sovelluksen vaatimuksiin lukeutuvat mukaan myös Lighthouse-työkalun määrittelemät PWA-kriteerit, jotka on jaettu neljään kategoriaan:

- nopeus ja luotettavuus
- asennettavuus
- PWA-optimointi
- muut vaatimukset.

Toteutettu web-sovellus täyttää kyseiset vaatimukset PWA-auditoinnin perusteella, joka suoritettiin Chrome-selaimeen asennetulla Lighthouse-lisäosalla. PWA-auditointi testaa web-sovelluksen jokaisen kategorian vaatimusten osalta. Näin voidaan varmistua siitä, että web-sovellus latautuu tarpeeksi nopeasti myös hitaammilla mobiiliverkkoyhteyksillä ja että se tarjoaa sisältöä myös offline-tilassa. Lisäksi Lighthouse testaa, että web-sovellus voidaan asentaa laitteelle ja että se tarjoaa sisältöä, jonka ulkoasu on optimoitu myös pienemmille näytöille.

8 YHTEENVETO

Tässä työssä toteutettiin kuvitteellisen kohdeyrityksen asettamien vaatimusten pohjalta progressiivinen web-sovellus. Yrityksen asettamien vaatimusten lisäksi toteutettavassa prototyyppisovelluksessa otettiin huomioon progressiivisia web-sovelluksia koskevat vaatimukset ja määritelmät. Sovelluksen vaatimukset mukailivat perinteistä mobiilisovellusta ja sisälsivät muun muassa QR-koodin skannaamisen, push-notifikaatioiden hyödyntämisen sekä käyttäjän sijainnin lukemisen.

Vaatimusten pohjalta määriteltiin toteutettava prototyyppisovellus, jonka avulla kohdeyrityksen vastaanottotyöntekijät voivat tallentaa huoltokäyntejä ja määrätä niitä huoltohenkilöille. Huoltohenkilöt voivat tunnistamaa huollettavan laitteen sovelluksen avulla, sekä merkitä työtehtävän suoritetuksi huollon jälkeen. Vaatimusten perusteella päädyttiin Reactilla toteutettavaan web-sovellukseen, jossa hyödynnettiin muun muassa Workbox-kirjastoja palvelutyöläisen räätälöintiin. Lisäksi toteutettiin palvelinpuolen sovellus rajapintoineen .NET Corea hyödyntäen, ja prototyyppisovelluksen käyttäjä- ja huoltokäyntitietoja säilytettiin Microsoft SQL Server-tietokannassa. Firebase-palvelua hyödynnettiin palvelinpuolen sovelluksessa push-ilmoitusten lähettämiseen ja asiakaspään sovelluksessa niiden vastaanottamiseen.

Toteutettu prototyyppisovellus pystyi täyttämään kaikki progressiivisia web-sovelluksia koskevat vaatimukset. Kohdeyrityksen vaatimuksista ainoastaan NFC- ja push-ilmoitus jäivät vajaiksi. Huollettavien laitteiden tunnistamiseen tarkoitettu NFC-lukuominaisuus ei ole tuettu vielä millään mobiiliselaimella, mutta toinen tunnistamistapa eli QR-lukuominaisuus saatiin toteutettua. Toiseksi suosituin mobiiliselain Safari ei tue push-ilmoituksia; tällöin ei pystytä huomauttamaan käyttäjää uudesta työtehtävästä sovelluksen ollessa suljettuna. Ottaen huomioon Safari-mobiiliselaimen markkinaosuuden ja push-ilmoitusten tuoman hyödyn todettiin, että progressiiviset web-sovellukset eivät ole vielä varteenotettava vaihtoehto mobiilisovelluksille. Web-alustan haasteena ovat myös mobiiliselainten lukumäärä sekä niiden tarjoamien ominaisuuksien vaihtelevuus.

Prototyyppisovellus tarjosi katsauksen web-selainten tarjoamiin rajapintoihin ja ominaisuuksiin, ja samalla nousi esiin jatkokehitysideoita. Mikäli Safari-mobiiliselaimelle ei ole tulossa tukea push-ilmoituksille, olisi tarpeen kartoittaa vastaavan ominaisuuden toteuttavia vaihtoehtoja. Lisäksi mobiiliselainten laajan valikoiman ja tuen kirjavuuteen tarvittaisiin ratkaisu, jonka avulla progressiivisen web-sovelluksen kehittäjä voisi tulkita varteenotettavat ominaisuudet: tutkimalla sovelluksen käyttäjäkunnan mobiiliselainvalintoja sekä kyseisten selainten tarjoamaa tukea eri rajapinnoille voidaan asettaa raamit sovelluksen

mahdollisille ominaisuuksille.

LÄHTEET

[1] *2020 Developer Survey*. Stack Exchange Inc. 2020. URL: https://insights.stackoverflow.com/survey/2020 (viitattu 28. 05. 2020).

[2] *About FCM messages*. Google LLC. 1. syyskuuta 2020. URL: https://firebase.google.com/docs/cloud-messaging/concept-options (viitattu 23. 10. 2020).

[3] D. Abramov. *Redux Fundamentals, Part 1: Redux Overview*. 2020. URL: https://redux.js.org/tutorials/fundamentals/part-1-overview (viitattu 23. 11. 2020).

[4] V. Agafonkin. *Leaflet - a JavaScript library for interactive maps*. URL: https://leafletjs.com (viitattu 27. 08. 2020).

[5] *Android Studio features*. Google LLC. 25. elokuuta 2020. URL: https://developer.android.com/studio/features (viitattu 05. 11. 2020).

[6] J. Archibald. *Introducing Background Sync*. Google LLC. 14. tammikuuta 2019. URL: https://developers.google.com/web/updates/2015/12/background-sync (viitattu 01. 12. 2020).

[7] J. Archibald. *The Service Worker Lifecycle*. Google LLC. 8. huhtikuuta 2019. URL: https://developers.google.com/web/fundamentals/primers/service-workers/lifecycle (viitattu 15. 10. 2020).

[8] K. Basques. *Why HTTPS matters*. Google LLC. 7. huhtikuuta 2020. URL: https://web.dev/why-https-matters (viitattu 30. 06. 2020).

[9] K. Basques ja M. Gaunt. *Push notifications overview*. Google LLC. 10. marraskuuta 2020. URL: https://web.dev/push-notifications-overview (viitattu 12. 11. 2020).

[10] F. Beaufort. *Interact with NFC devices on Chrome for Android*. Google LLC. 4. kesäkuuta 2020. URL: https://web.dev/nfc (viitattu 15. 10. 2020).

[11] *Build Progressive Web Applications with ASP.NET Core Blazor WebAssembly*. Microsoft. 10. kesäkuuta 2020. URL: https://docs.microsoft.com/en-us/aspnet/core/blazor/progressive-web-app (viitattu 17. 11. 2020).

[12] *Can I use... IndexedDB?* URL: https://caniuse.com/indexeddb (viitattu 17. 12. 2020).

[13] *Can I use... Web App Manifest?* URL: https://caniuse.com/web-app-manifest (viitattu 17. 12. 2020).

[14] *CanvasRenderingContext2D*. Mozilla. 9. maaliskuuta 2019. URL: https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D (viitattu 15. 10. 2020).

[15] K. R. Christiance, Z. Kis ja F. Beaufort. *Web NFC*. Web NFC Community Group. 8. joulukuuta 2020. URL: https://w3c.github.io/web-nfc (viitattu 17. 12. 2020).

[16] *Content is not sized correctly for the viewport*. Google LLC. 19. syyskuuta 2019. URL: https://web.dev/content-width (viitattu 14. 08. 2020).

[17] *Current page does not respond with a 200 when offline*. Google LLC. 4. kesäkuuta 2020. URL: https://web.dev/works-offline (viitattu 11. 06. 2020).

[18] N. Dabit. *React Native in Action*. 1. painos. Manning Publications, 2019.

[19] *Desktop Browser Market Share Worldwide*. StatCounter. Tammikuu 2021. URL: https://gs.statcounter.com/browser-market-share/desktop/worldwide (viitattu 07. 01. 2020).

[20] *Does not have a <meta name="viewport"> tag with width or initial-scale*. Google LLC. 20. syyskuuta 2019. URL: https://web.dev/viewport (viitattu 14. 08. 2020).

[21] *Does not provide a valid apple-touch-icon*. Google LLC. 19. syyskuuta 2019. URL: https://web.dev/apple-touch-icon (viitattu 20. 08. 2020).

[22] *Does not provide fallback content when JavaScript is not available*. Google LLC. 19. syyskuuta 2019. URL: https://web.dev/without-javascript (viitattu 14. 08. 2020).

[23] *Does not register a service worker that controls page and start_url*. Google LLC. 10. kesäkuuta 2020. URL: https://web.dev/service-worker (viitattu 30. 06. 2020).

[24] *Does not set a theme color for the address bar*. Google LLC. 17. kesäkuuta 2020. URL: https://web.dev/themed-omnibox (viitattu 14. 08. 2020).

[25] *Does not use HTTPS*. Google LLC. 29. huhtikuuta 2020. URL: https://web.dev/is-on-https (viitattu 30. 06. 2020).

[26] *Each page has a URL*. Google LLC. 19. syyskuuta 2019. URL: https://web.dev/pwa-each-page-has-url (viitattu 20. 08. 2020).

[27] N. Ebel. *Mastering Kotlin : learn advanced Kotlin programming techniques to build apps for Android, iOS, and the web*. 1. painos. Birmingham, England ; Mumbai : Packt, 2019.

[28] *Firebase*. Google LLC. 1. lokakuuta 2020. URL: https://firebase.google.com/docs/reference/js/firebase (viitattu 25. 10. 2020).

[29] *FirebaseAdmin.Messaging.Message*. Google LLC. 9. syyskuuta 2020. URL: https://firebase.google.com/docs/reference/admin/dotnet/class/firebase-admin/messaging/message (viitattu 25. 10. 2020).

[30] *FirebaseAdmin.Messaging.Notification*. Google LLC. 9. syyskuuta 2020. URL: https://firebase.google.com/docs/reference/admin/dotnet/class/firebase-admin/messaging/notification (viitattu 25. 10. 2020).

[31] *firebase.messaging.Messaging*. Google LLC. 1. lokakuuta 2020. URL: https://firebase.google.com/docs/reference/js/firebase.messaging.Messaging (viitattu 25. 10. 2020).

[32] *First Meaningful Paint*. Google LLC. 5. marraskuuta 2019. URL: https://web.dev/first-meaningful-paint (viitattu 11. 06. 2020).

[33] M. Gaunt. *Browser Support*. Google LLC. 13. joulukuuta 2017. URL: https://github.com/GoogleChrome/workbox/wiki/Browser-Support (viitattu 15. 10. 2020).

[34] M. Gaunt. *Service Workers: an Introduction*. Google LLC. 9. elokuuta 2019. URL: https://developers.google.com/web/fundamentals/primers/service-workers (viitattu 07. 09. 2019).

[35] *Generating a Remote Notification*. Apple Inc. URL: https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server/generating_a_remote_notification (viitattu 29. 10. 2019).

[36] *Geolocation*. Mozilla. 2. lokakuuta 2020. URL: https://developer.mozilla.org/en-US/docs/Web/API/Geolocation (viitattu 04. 10. 2020).

[37] T. Hagos. *Learn Android Studio 3: Efficient Android App Development*. 1. painos. Berkeley, CA: Apress L. P, 2018.

[38] *How to make PWAs installable*. Mozilla. 25. elokuuta 2020. URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Installable_PWAs (viitattu 19. 11. 2020).

[39] *HTMLCanvasElement.getContext()*. Mozilla. 31. elokuuta 2020. URL: https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement/getContext (viitattu 15. 10. 2020).

[40] *HTMLMediaElement*. Mozilla. 28. syyskuuta 2020. URL: https://developer.mozilla.org/en-US/docs/Web/API/HTMLMediaElement (viitattu 15. 10. 2020).

[41] *HTMLMediaElement: canplay event*. Mozilla. 18. toukokuuta 2019. URL: https://developer.mozilla.org/en-US/docs/Web/API/HTMLMediaElement/canplay_event (viitattu 28. 05. 2020).

[42] *Introduction to progressive web apps*. Mozilla. 13. huhtikuuta 2020. URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction (viitattu 11. 06. 2020).

[43] *Introduction to Push Notifications*. Google LLC. 1. toukokuuta 2019. URL: https://developers.google.com/web/ilt/pwa/introduction-to-push-notifications (viitattu 15. 11. 2020).

[44] *Is not configured for a custom splash screen*. Google LLC. 19. syyskuuta 2019. URL: https://web.dev/splash-screen (viitattu 14. 08. 2020).

[45] *JavaScript*. Mozilla. 1. marraskuuta 2020. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (viitattu 29. 11. 2020).

[46] C. Keur ja A. Hillegass. *iOS Programming: The Big Nerd Ranch Guide, 7th Edition*. 1. painos. Big Nerd Ranch Guides, 2020.

[47] *Lighthouse*. Google LLC. 10. maaliskuuta 2020. URL: https://developers.google.com/web/tools/lighthouse (viitattu 11. 06. 2020).

[48] *Load Firebase SDKs from reserved URLs*. Google LLC. 16. lokakuuta 2020. URL: https://firebase.google.com/docs/hosting/reserved-urls (viitattu 25. 10. 2020).

[49] *Manifest doesn't have a maskable icon*. Google LLC. 6. toukokuuta 2020. URL: https://web.dev/maskable-icon-audit (viitattu 20. 08. 2020).

[50] *MediaDevices.getUserMedia()*. Mozilla. 4. toukokuuta 2020. URL: https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia (viitattu 04. 10. 2020).

[51] *Mobile Browser Market Share Worldwide*. StatCounter. Marraskuu 2020. URL: https://gs.statcounter.com/browser-market-share/mobile/worldwide (viitattu 17. 12. 2020).

[52] *Mobile Operating System Market Share Worldwide*. StatCounter. Marraskuu 2020. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide (viitattu 17. 12. 2020).

[53] F. Nah. A study on tolerable waiting time: how long are Web users willing to wait? *Behaviour & information technology* 23 (2004).

[54] *Notifications API*. Mozilla. 23. heinäkuuta 2020. URL: https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API (viitattu 12. 11. 2020).

[55] *Page load is not fast enough on mobile networks*. Google LLC. 10. kesäkuuta 2020. URL: https://web.dev/load-fast-enough-for-pwa (viitattu 11. 06. 2020).

[56] *Page transitions don't feel like they block on the network*. Google LLC. 19. syyskuuta 2019. URL: https://web.dev/pwa-page-transitions (viitattu 20. 08. 2020).

[57] S. Pieters. *HTML5 Differences from HTML4*. W3C. 9. joulukuuta 2014. URL: https://www.w3.org/TR/html5-diff/#new-apis (viitattu 29. 11. 2020).

[58] *Push API*. Mozilla. 16. huhtikuuta 2020. URL: https://developer.mozilla.org/en-US/docs/Web/API/Push_API (viitattu 12. 11. 2020).

[59] *Queue.ts*. Github, Inc. 10. syyskuuta 2020. URL: https://github.com/GoogleChrome/workbox/blob/d62c18598934e1d8f0e2bd2ff63eb0d7d3e6535b/packages/workbox-background-sync/src/Queue.ts (viitattu 18. 10. 2020).

[60] *REST Resource: projects.messages*. Google LLC. 21. toukokuuta 2020. URL: https://firebase.google.com/docs/reference/fcm/rest/v1/projects.messages (viitattu 29. 10. 2020).

[61] R. Roche. *Essential iOS build and release*. 1. painos. Beijing : O'Reilly, 2012.

[62] D. Rousset. *Green Energy Efficient Progressive Web Apps*. Microsoft. 30. marraskuuta 2020. URL: https://devblogs.microsoft.com/sustainable-software/green-energy-efficient-progressive-web-apps (viitattu 01. 12. 2020).

[63] *ServiceWorkerContainer*. Mozilla. 5. syyskuuta 2020. URL: https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorkerContainer (viitattu 05. 10. 2020).

[64] *ServiceWorkerRegistration*. Mozilla. 3. syyskuuta 2020. URL: https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorkerRegistration (viitattu 08. 10. 2020).

[65] D. Sheppard. *Beginning Progressive Web App Development Creating a Native App Experience on the Web*. 1. painos. Berkeley, CA : Apress : Imprint: Apress, 2017.

[66] *Site works cross-browser*. Google LLC. 19. syyskuuta 2019. URL: https://web.dev/pwa-cross-browser (viitattu 20. 08. 2020).

[67] P. Späth. *Learn Kotlin for Android Development The Next Generation Language for Modern Android Apps Programming*. 1. painos. Berkeley, CA : Apress : Imprint: Apress, 2019.

[68] *start_url does not respond with a 200 when offline*. Google LLC. 29. huhtikuuta 2020. URL: https://web.dev/offline-start-url (viitattu 11. 06. 2020).

[69] *Supported environments for the Firebase JavaScript SDK*. Google LLC. 10. marraskuuta 2020. URL: https://firebase.google.com/support/guides/environments_js-sdk (viitattu 01. 12. 2020).

[70] *SyncManager*. Mozilla. 27. elokuuta 2020. URL: https://developer.mozilla.org/en-US/docs/Web/API/SyncManager (viitattu 18. 10. 2020).

[71] *Time to Interactive*. Google LLC. 10. lokakuuta 2019. URL: https://web.dev/interactive (viitattu 11. 06. 2020).

[72] *Using Service Workers*. Mozilla. 4. elokuuta 2020. URL: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers (viitattu 05. 10. 2020).

[73] *Using Web Workers*. Mozilla. 11. tammikuuta 2020. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers (viitattu 19. 03. 2020).

[74] J. Wargo. *Learning Progressive Web Apps: Building Modern Web Apps Using Service Workers*. 1. painos. Addison-Wesley Professional, 2020.

[75] *Web app manifest does not meet the installability requirements*. Google LLC. 19. syyskuuta 2019. URL: https://web.dev/installable-manifest (viitattu 14. 08. 2020).

[76] *Web NFC*. Github, Inc. 6. tammikuuta 2020. URL: https://github.com/mozilla/standards-positions/issues/238 (viitattu 12. 01. 2021).

[77] *Web NFC API*. Mozilla. 8. heinäkuuta 2020. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_NFC_API (viitattu 18. 10. 2020).

[78] *Web NFC - Chrome Platform Status*. Google LLC. URL: https://www.chromestatus.com/feature/6261030015467520 (viitattu 12. 01. 2021).

[79] *Web workers*. Web Hypertext Application Technology Working Group. 10. kesäkuuta 2020. URL: https://html.spec.whatwg.org/multipage/workers.html (viitattu 11. 06. 2020).

[80] *Web Workers API*. Mozilla. 11. tammikuuta 2020. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API (viitattu 19. 03. 2020).

[81] R. Wieruch. *the-road-to-react-with-firebase / react-firebase-authentication*. 12. kesäkuuta 2020. URL: https://github.com/the-road-to-react-with-firebase/react-firebase-authentication (viitattu 25. 10. 2020).

[82] E. Windmill. *Flutter in action*. 1. painos. Shelter Island, NY : Manning Publications, 2020.

[83] *WindowOrWorkerGlobalScope.setInterval()*. Mozilla. 27. heinäkuuta 2020. URL: https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setInterval (viitattu 15. 10. 2020).

[84] C. Wolfe. *jsQR*. 11. joulukuuta 2015. URL: https://github.com/cozmo/jsQR (viitattu 10. 04. 2020).

[85] *Workbox Strategies*. Google LLC. 7. toukokuuta 2020. URL: https://developers.google.com/web/tools/workbox/modules/workbox-strategies (viitattu 22. 11. 2020).

[86] *workbox-background-sync.Queue*. Google LLC. 26. maaliskuuta 2020. URL: https://developers.google.com/web/tools/workbox/reference-docs/latest/module-workbox-background-sync.Queue (viitattu 10. 09. 2020).

[87] *workbox-core*. Google LLC. 26. maaliskuuta 2020. URL: https://developers.google.com/web/tools/workbox/reference-docs/latest/module-workbox-core (viitattu 15. 10. 2020).

[88] *WorkerGlobalScope.importScripts()*. Mozilla. 22. kesäkuuta 2020. URL: https://developer.mozilla.org/en-US/docs/Web/API/WorkerGlobalScope/importScripts (viitattu 15. 10. 2020).