OULUN AMMATTIKORKEAKOULU

Tapio Palomaa

SIJAINTIPERUSTEISEN PROGRESSIIVISEN WEB-SOVELLUKSEN KEHITTÄMINEN

SIJAINTIPERUSTEISEN PROGRESSIIVISEN WEB-SOVELLUKSEN KEHITTÄMINEN

Tapio Palomaa
Opinnäytetyö
Syksy 2020
Tietojenkäsittelyn tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma

Tekijä(t): Tapio Palomaa
Opinnäytetyön nimi: Sijaintiperusteisen progressiivisen web-sovelluksen kehittäminen
Työn ohjaaja: Liisa Auer
Työn valmistumislukukausi ja -vuosi: syksy 2020 Sivumäärä: 50

Verkkoselainten valmistajat ovat viime vuosina kehittäneet tuotteitaan siten, että web-sovellukset voivat hyödyntää ominaisuuksia, jotka ennen olivat vain natiivien mobiilisovellusten ulottuvilla. Esimerkiksi laitteen sijaintitieto tai mahdollisuus käyttää laitteen kameraa kuuluvat nykyään mobiililaitteiden verkkoselainten perusominaisuuksiin.

Nykyaikainen web-sovellus on mahdollista asentaa laitteen aloitusnäytölle, josta se voidaan avata koko ruudun näkymässä natiivien mobiilisovellusten tapaan. Web-sovellukset voivat lähettää push-ilmoituksia ja ajaa ohjelmakoodia taustalla selaimen ollessa suljettuna. Lisäksi web-kehittäjät voivat säädellä aiempaa yksityiskohtaisemmin selaimen välimuistin käyttöä, mikä mahdollistaa valittujen ominaisuuksien käytön myös ilman verkkoyhteyttä. Näitä uusia ominaisuuksia hyödyntäviä web-sovelluksia kutsutaan progressiivisiksi web-sovelluksiksi.

Opinnäytetyön kehittämistehtävänä oli kehittää mobiililaitteille suunnattu sijaintiperusteinen progressiivinen web-sovellus. Käyttäjän ympäristöstä luotiin kolmiulotteinen karttanäkymä verkkoselaimen Geolocation-rajapinnan, OpenStreetMapin kuvailudataa tarjoavan OverPass API -palvelun sekä 3D-grafiikan esittämiseen tarkoitetun three.js-kirjaston avulla. Käyttäjä voi selata sovelluksessa ympäristöön lisättyjä kuva- ja tekstisisältöjä koskettamalla karttanäkymän vuorovaikutteisia kohteita. Sovellukseen luotiin manifest- ja Service Worker -tiedostot, jotka mahdollistavat progressiivisten web-sovellusten ominaisuuksien lisäämisen. Progressiivisten web-sovellusten ominaisuuksien toteutumista arvioitiin Google Lighthouse -palvelun avulla.

Kehittämistehtävässä hyödynnettiin kahta progressiivisten web-sovellusten ominaisuutta: sovellus voidaan asentaa laitteen aloitusnäytölle ja osa toiminnosta on saatavilla myös ilman verkkoyhteyttä. Aloitusnäytölle asennettu sovellus voidaan avata koko ruudun näkymässä. Tällöin selaimen käyttöliittymä ja sen omat kosketuseleisiin perustuvat toiminnot eivät häiritse kolmiulotteisen karttanäkymän käyttöä. Käyttäjän sijainnin perusteella haetut alueet tallennetaan latauksen yhteydessä selaimen välimuistiin, mikä mahdollistaa sovelluksen käytön ilman verkkoyhteyttä kyseisten alueiden osalta. Ennen progressiivisten web-sovellusten ominaisuuksien toteutusta sovellus läpäisi viisi kohtaa Google Lighthouse -palvelun progressiivisten web-sovellusten ominaisuuksia mittaavasta osiosta. Niiden lisäämisen jälkeen sovellus läpäisi kyseisen osion kaikki 13 kohtaa.

Asiasanat: progressiiviset verkkosovellukset, paikannus, verkko-ohjelmointi, verkkosivustot, verkkosovellukset

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Business Information Systems

Author(s): Tapio Palomaa
Title of thesis: Developing a geolocation based progressive web application
Supervisor(s): Liisa Auer
Term and year when the thesis was submitted: fall 2020 Number of pages: 50

Modern web browsers make it possible for web applications to implement features that were previously available only in native mobile applications. For example, access to device camera and the device's geolocation are now basic features in all browsers.

It is now possible to install a web application to the mobile device home screen. The application can then be opened in various screen modes, including a full screen mode. Modern web applications can send push notifications and synchronize data in the background when the browser is not running. Web developers also have fine-grained access to the browser cache, which makes it possible to store parts of the application to be available even in offline mode. The web applications that use some of the features above are called progressive web applications.

The goal for this thesis was to develop a geolocation based progressive web application for mobile devices. A three-dimensional map was created using the browser's Geolocation API, OpenStreetMap data from Overpass API and three.js library for rendering 3D graphics in the browser. The user can browse text and image content by selecting interactive objects and markers on the map. The application includes a manifest file and a Service Worker file that make it possible to add progressive web application features to an application. The application was audited with Google Lighthouse.

Two progressive web application features were chosen for implementation: installation to home screen and selected offline features. The user can install the application on their home screen. From the home screen the application will open in a full screen mode, hiding the browser's user interface and preventing the browser's gesture-based navigation from interfering with map navigation. The data downloaded from Overpass API is stored in the browser cache, which makes it possible to visit previously downloaded areas in offline mode. Before the addition of progressive web application features the application passed five tests in the Google Lighthouse section for progressive web applications. After implementing the features, the application passed all 13 tests in the section.

Keywords: progressive web applications, geolocation, web programming, web applications, web-sites

SISÄLLYS

1 JOHDANTO

1.1 Opinnäytetyön taustaa

Mobiililaitteille suunnatun sovelluksen kehittämiseen on tarjolla useita teknologioita. Vaihtoehdot voidaan jakaa karkeasti mobiili-, hybridi- ja web-sovelluksiin. Valinta eri vaihtoehtojen välillä riippuu asiakkaan tarpeista, sovelluksen ominaisuuksista ja siitä, mille kohderyhmille sovellus on suunnattu.

Web- eli verkkosovellus on ohjelmisto, joka jaellaan internetin välityksellä ja jonka käyttöliittymä toimii käyttäjän laitteen verkkoselaimessa. Sisältö haetaan verkkoyhteyden välityksellä internet-palvelimelta. Hybridisovellus ohjelmoidaan käyttämällä web-teknologioita kuten HTML, CSS ja JavaScript, mutta lopputuotos voidaan asentaa mobiililaitteelle käyttöjärjestelmän sovelluskaupasta. Hybridisovelluksen ohjelmakoodia ajetaan käyttöjärjestelmässä verkkoselaimen korvaavassa web view -näkymässä, ja sama ohjelmakoodi soveltuu ajettavaksi eri käyttöjärjestelmissä. Natiivit mobiilisovellukset puolestaan toteutetaan tietylle käyttöjärjestelmälle tarkoitetulla ohjelmointikielellä ja käyttöjärjestelmän omia työkaluja hyödyntäen. Ne ovat suorituskyvyltään usein hybridisovelluksia jonkin verran tehokkaampia, mutta kullekin käyttöjärjestelmälle on toteutettava oma versionsa sovelluksesta. (Patro 2019.)

Mobiilisovellusten suosio ja tarjolla olevien sovellusten määrä on kasvanut yhtä jalkaa mobiililaitteiden yleistymisen ja etenkin älypuhelinten määrän kasvun kanssa. Osa mobiilisovellusten suosiosta perustuu niiden saumattomaan ja viimeisteltyyn käyttökokemukseen. (Hume 2017, luku 1.) Kehittyneet graafiset ominaisuudet, käyttäjän sijainnin hyödyntäminen, push-ilmoitukset, offline-saatavuus sekä muut ainoastaan natiivisovellusten käytössä olleet ominaisuudet ovat kasvattaneet mobiilisovellusten suosiota viimeisen vuosikymmenen aikana (Ater 2017, luku 1).

Älypuhelinten verkkoyhteys ja laitteisiin asennetut verkkoselaimet mahdollistavat natiivisovellusten lisäksi myös internet-sivustojen ja web-sovellusten käytön. Verkkosivustojen ja -sovellusten latausajat kuitenkin vaihtelevat käytössä olevasta verkosta ja käyttäjän sijainnista riippuen, minkä vuoksi verkkosovellukset eivät ole tavallisesti pystyneet kilpailemaan käyttökokemuksessa natiivisovellusten kanssa. Älypuhelimelle asennettu natiivisovellus käynnistyy nopeasti, ja usein kaikki tarvittava

tieto on asennettu laitteeseen sovelluksen latauksen yhteydessä. Suurin osa verkkosivuista ja -sovelluksista ei sen sijaan toimi laisinkaan ilman verkkoyhteyttä. (Hume 2017, luku 1.)

Viime vuosien perusteella mobiilisovellusten asema on kuitenkin muuttumassa. Käyttäjät asentavat yhä vähemmän natiivisovelluksia ja käyttävät vain osaa asentamistaan sovelluksista säännöllisesti. Applen App Storessa ja Googlen Play-kaupassa on tarjolla miljoonia sovelluksia (Hume 2017, luku 1), ja uuden natiivisovelluksen markkinointi ja sen saaminen käyttäjien ulottuville käy vuosi vuodelta kalliimmaksi. Mobiilisovelluksen asentaminen laitteeseen vaatii usean vaiheen läpikäymistä, ja sovellusten tuottajien näkökulmasta liian moni lataus jää tekemättä. (Ater 2017, luku 1.)

Digitaalista palvelua tarjoavan tahon pitäisikin pystyä arvioimaan tapauskohtaisesti, millaista tuotetta lähdetään toteuttamaan. Asiakkaan näkökulmasta voi tuntua houkuttelevalta kehittää suunniteltu palvelu natiivisovelluksena – kaikillahan on nykyään älypuhelin ja kaikilla menestyneillä yrityksillä oma tuotteensa sovelluskaupoissa. Natiivisovellus mielletään ehkä arvokkaammaksi tuotteeksi kuin verkkosivusto tai -palvelu ja sen ajatellaan vahvistavan asiakkaan brändiä ja liiketoimintaa.

Natiivisovelluksen kehittäminen kestää kuitenkin tavallisesti verkkopalveluita kauemmin ja kustannuksissa on huomioitava sovelluksen tarjoaminen eri käyttöjärjestelmille. Joidenkin arvioiden mukaan natiivin mobiilisovelluksen kehittäminen on jopa kaksi kertaa web-sovellusta kalliimpaa. Palvelun tuottajalle web-sovelluskehittäjien rekrytointi voi olla tänä päivänä natiiviosaajia helpompaa, sillä JavaScript on yksi suosituimmista ohjelmointikielistä. Lisäksi JavaScriptin osaajien työkalupakkiin kuuluvat luontevasti muutkin web-teknologiat. (Gustafson 2018.)

Digitaalisten palveluiden tuottajilla on nykyään aikaisempaa enemmän valinnanvaraa. Suurimmat selainvalmistajat ovat viime aikoina kehittäneet tuotteitaan siten, että web-sovellukset voivat muistuttaa ominaisuuksiltaan yhä enemmän natiivisovelluksia. Nämä uudenlaiset web-sovellukset, joille on vakiintunut nimitys progressiivinen web-sovellus (Progressive Web App, PWA), perustuvat samoihin web-teknologioihin kuin perinteiset verkkosivut ja -palvelut, mutta ne tarjoavat aikaisempaa paremman käyttökokemuksen. (Hume 2017, luku 1.)

Progressiiviset web-sovellukset mahdollistavat esimerkiksi verkkosovelluksen asentamisen laitteen aloitusnäkymään, valittujen ominaisuuksien toteuttamisen offline-tilassa sekä push-ilmoituk-

set ja taustasynkronoinnin (Gustafson 2017). Progressiivisen web-sovelluksen etuja natiivisovellukseen verrattuna ovat mahdollisuus käyttää sovellusta ilman asennusta, jokaisella latauskerralla päivittyvät uudet sisällöt ja ominaisuudet sekä näkyvyys internetin hakukoneissa ja linkkeinä muissa verkkopalveluissa. Hyvin rakennettu progressiivinen web-sovellus toimii ongelmitta monissa eri laitteissa ja käyttöjärjestelmissä. (Gustafson 2018.)

Ohjelmistokehittäjien keskuudessa onkin keskusteltu yhä kiivaammin siitä, pitäisikö uusia palveluja kehitettäessä keskittyä ensisijaisesti web-sovellusten vai natiivien mobiilisovellusten tuottamiseen. Valinnasta progressiivisen web-sovelluksen ja natiivisovelluksen välillä ei ole kuitenkaan mielekästä kiistellä yleisenä kysymyksenä. Kunkin projektin kohdalla on syytä arvioida erikseen, millainen ohjelmisto sopii parhaiten hankkeen tavoitteisiin ja budjettiin. Nykypäivänä on kuitenkin järkevää arvioida aluksi, mihin web-teknologiat pystyvät ja riittäisivätkö progressiivisen web-sovelluksen ominaisuudet täyttämään projektin vaatimukset. (Gustafson 2018.)

Idea käsillä olevan opinnäytetyön aiheeseen ja kehittämistehtävään syntyi edellä kuvatusta kehityksestä, joka näyttäisi laajentavan verkkosivustojen ja web-sovellusten soveltamisalaa suhteessa natiiveihin mobiilisovelluksiin. Kehittämistehtävän taustalla on halu tutustua aikaisempaa tarkemmin progressiivisiin web-sovelluksiin. Tavoitteena on hankkia sellaista asiantuntemusta, joka auttaa arvioimaan tapauskohtaisesti, riittävätkö nykyaikaisten web-teknologioiden tarjoamat ominaisuudet täyttämään suunnitellulle projektille asetetut tavoitteet ja olisiko projekti mahdollista toteuttaa kevyemmin ja joustavammin web-sovelluksena.

1.2 Kehittämistehtävän tavoitteet ja tekninen toteutus

Opinnäytetyön kehittämistehtävänä on sijaintiperusteisen progressiivisen web-sovelluksen kehittäminen mobiililaitteille. Tavoitteena on tuottaa runko palvelulle, jossa voidaan yhdistää dataa tai muuta sisältöä käyttäjää ympäröiviin todellisen maailman kohteisiin kuten rakennuksiin, puistoihin ja muihin julkisiin kohteisiin. Kehittämistehtävässä luodaan esimerkkisovellus, jossa palvelun runkoon yhdistetään dataa tunnetuista Oulun kaupungin keskusta-alueen nähtävyyksistä. Kehittämistehtävän yhteydessä tutkitaan, mitkä ovat tämänhetkiset web-standardit liittyen käyttäjän sijaintitiedon hyödyntämiseen, millaisia esimerkkejä löytyy käyttäjän sijaintitietoa hyödyntävistä sovelluksista ja mitkä niiden ominaisuuksista ovat toteutettavissa web-sovelluksissa.

Sijaintiperusteisen sovelluksen perustoimintojen lisäksi tavoitteena on soveltaa kehittämistehtävässä joitakin progressiivisiin web-sovelluksiin liitettyjä ominaisuuksia. Koska sijaintiperusteinen palvelu on perusluonteensa vuoksi suunnattu mobiililaitteille ja sijaintiperusteiset sovellukset olivat aikaisemmin nimenomaan natiiveja mobiilisovelluksia, soveltuu progressiivisen web-sovelluksen ominaisuuksien hyödyntäminen hyvin osaksi kehittämistehtävää. Kehittämistehtävässä tutkitaan, millaisia mahdollisuuksia web-sivuston muuttaminen progressiiviseksi web-sovellukseksi tarjoaa ja mitä progressiivisten web-sovellusten ominaisuuksia olisi perusteltua ottaa käyttöön tämän opinnäytetyön kehittämistehtävän yhteydessä.

Käyttäjää ympäröivä kaupunkinäkymä mallinnetaan sovelluksessa yksinkertaisella 3D-karttanäkymällä, ja käyttäjä voi saada tietoa ympäristön nähtävyyksistä valitsemalla ne karttanäkymästä. Sovelluksen käyttöliittymä muistuttaa Pokemon Go -pelistä tuttua näkymää, jossa käyttäjän sijainti on merkitty yläviistosta kuvattuun karttanäkymään. Käyttäjä voi valita interaktiivisia kohteita koskettamalla niitä kartalla. Tämän opinnäytetyön puitteissa ei luoda erillisiä 3D-malleja, vaan karttanäkymä, käyttäjän ”avatar” ja interaktiiviset kohteet luodaan ohjelmallisesti sovelluksessa. Asiakkaalle tarjottavassa tuotteessa voitaisiin lisätä ohjelmallisesti luotuun karttanäkymään myös varta vasten tuotettuja 3D-malleja.

Kehittämistehtävän ohjelmakoodi koostuu kahdesta kokonaisuudesta. Sijaintiperusteisen sovelluksen rungoksi luodaan JavaScript-kirjasto, joka luo 3D-näkymän käyttäjän ympäristöstä. Kirjasto tarjoaa ohjelmointirajapinnan, jonka avulla näkymä voidaan lisätä verkkosivulle ja joka mahdollistaa interaktiivisten kohteiden lisäämisen karttanäkymään. Kirjasto on uudelleenkäytettävissä erilaisiin tarkoituksiin, ja sen oheen rakennettava käyttöliittymä ja toiminnallisuus voi vaihdella kirjastoa hyödyntävän sovelluksen mukaan.

Käyttäjän ympäristön mallinnus toteutetaan nykyaikaisten selainten mahdollistaman Geolocation-rajapinnan, OpenStreetMapin dataa tarjoavan Overpass-rajapinnan sekä three.js-kirjaston avulla. Selainten Geolocation-rajapinta mahdollistaa käyttäjän laitteen sijaintitiedon hyödyntämisen. Overpass-rajapinnasta puolestaan voidaan hakea karttadataa ja ympäristön kuvailutietoja käyttäjän sijaintiin perustuvalla aluerajauksella. Three.js on JavaScript-kirjasto ja -ohjelmointirajapinta, joka hyödyntää selainten WebGL-rajapintaa ja yksinkertaistaa 3D-grafiikan renderöintiä verkkoselaimessa.

Toinen kehittämistehtävään kuuluva kokonaisuus on verkkosovellus, joka hyödyntää edellä mainittua JavaScript-kirjastoa. Sovellus toteutetaan hyödyntäen HTML-, CSS- ja JavaScript-teknologioita. Sovelluksen kehys ja käyttöliittymäkomponentit luodaan HTML:n ja CSS-tyylien avulla. JavaScriptin avulla lisätään verkkosivulle karttanäkymä ja siihen kuuluvat interaktiiviset elementit ja niihin liittyvä data. Kehittämistehtävän kannalta tärkeintä kuitenkin on, että progressiivisen web-sovelluksen ominaisuudet rakennetaan tähän kokonaisuuteen kuuluvan JavaScript-koodin avulla.

Nykypäivänä ovat suosittuja monet JavaScript-ohjelmointikehykset kuten React tai Angular. Opinnäytetyön kehittämistehtävä toteutetaan kuitenkin ilman ohjelmointikehyksiä, sillä niissä progressiivisen web-sovelluksen ominaisuudet liitetään sovellukseen usein kullekin ohjelmointikehykselle luodun erityisen kirjaston avulla. Kehittämistehtävän tarkoituksena on sen sijaan tutustua progressiivisen web-sovelluksen toteutukseen kirjoittamalla itse siihen tarvittava JavaScript-koodi.

Progressiivisen web-sovelluksen ominaisuuksien lisääminen sovellukseen edellyttää manifest- ja Service Worker -tiedoston luomista. Manifest-tiedosto sisältää sovelluksen kuvauksen ja tarvittavat tiedot sovelluksen aloitusnäkymään asentamista varten. Service Worker on puolestaan JavaScript-tiedosto, joka mahdollistaa suurimman osan progressiivisiin web-sovelluksiin liitetyistä ominaisuuksista kuten offline-ominaisuudet, push-ilmoitukset sekä taustasynkronoinnin. Kehittämistehtävässä tuotettua sovellusta auditoidaan Google Lighthouse -palvelussa. Google Lighthouse -palvelu arvioi verkkosivustoa tai -sovellusta useasta eri näkökulmasta, muun muassa suorituskyvyn ja progressiivisten web-sovellusten ominaisuuksien kannalta.

Kehittämistehtävän tietoperustassa korostuvat sijaintitiedon hyödyntämistä ja progressiivisia web-sovelluksia käsittelevät teokset sekä verkkomateriaalit. Sijaintiedon hyödyntämistä selaimissa käsittelee esimerkiksi Anthony Holdener teoksessaan *HTML Geolocation*. David Humen teos *Progressive Web Apps* sekä Tal Aterin *Building Progressive Web Apps* käsittelevät progressiivisten web-sovellusten ominaisuuksia sekä niiden käytännön toteutusta JavaScriptillä. Progressiivisen web-sovelluksen käsitteen kehittänyt Googlen ohjelmistokehittäjä Alex Russell sekä Microsoftilla web-standardien parissa työskentelevä Aaron Gustafson ovat kirjoittaneet useita progressiivisia web-sovelluksia käsitteleviä verkkoartikkeleita.

Google Developers -sivustolla on omistettu kokonainen osio progressiivisille web-sovelluksille, ja sekä progressiivisiin web-sovelluksiin että sijaintitiedon hyödyntämiseen liittyvää materiaalia löytyy

viljalti MDN Web Docs -sivustolta ja World Wide Web Consortiumin standardeista. OpenStreetMapia, Overpass-rajapintaa ja Three.js-kirjastoa koskevat dokumentaatiot löytyvät verkosta. Three.js-kirjastoa koskevaa kirjallisuutta on myös saatavilla, mutta tämän opinnäytetyön kehittämistehtävän kannalta web-dokumentaatio lienee sen osalta riittävä tietolähde.

2 KÄYTTÄJÄN YMPÄRISTÖN MALLINNUS

2.1 Sijaintitieto ja sijaintiperusteiset palvelut

Sijaintiperusteisiksi palveluiksi kutsutaan palveluita, jotka tavalla tai toisella hyödyntävät käyttäjän tai laitteen sijaintitietoa. Käyttäjän sijainti vaikuttaa tällöin olennaisesti palvelun sisältöön ja käyttökokemukseen. Sijaintiperusteisia palveluita käytetään tyypillisesti mobiililaitteilla, ja tarkoituksena on yksilöidä palvelun sisältöä käyttäjän sijainnin mukaan. Esimerkkejä sijaintiperusteisista palveluista ovat navigointipalvelut, turvapalvelut (esimerkiksi perheenjäsenten sijainnin seuraaminen), kanssakäyminen muiden lähistöllä liikkuvien käyttäjien kanssa tai sijaintiperusteinen markkinointi. Myös useat sosiaalisen median palvelut hyödyntävät käyttäjien sijaintia. (Holdener 2011, luku 1.) Nämä sovellukset perustuvat tavallisesti kaksisuuntaiseen tiedonkulkuun; käyttäjä toisaalta jakaa oman sijaintinsa ja jotain siihen liittyvää tietoa palvelussa, toisaalta saa käyttöönsä muiden käyttäjien jakamaa tietoa (Holdener 2011, luku 6).

Tieto käyttäjän sijainnista voi perustua useaan eri lähteeseen. Älypuhelimet ja muut GPS-yhteensopivat laitteet voivat hyödyntää kansainvälistä satelliittijärjestelmää. Mikä tahansa internetiin yhdistetty laite voidaan paikantaa tietyllä tarkkuudella sen IP- tai MAC-osoitteen perusteella. Matkapuhelimen paikannus puolestaan on mahdollista myös matkapuhelinverkon tukiasemien sijaintien avulla. Lisäksi sijaintitieto esimerkiksi postinumeron tai osoitteen tarkkuudella voidaan saada suoraan käyttäjältä. (Holdener 2011, luku 2.)

Satelliittipaikannus perustuu maapalloa kiertävien GPS-satelliittien lähettämiin signaaleihin. Laite, joka kykenee tulkitsemaan satelliittien lähettämän signaalin, voi selvittää sijaintinsa kolmiomittauksen avulla. Matkapuhelinverkon tukiasemiin perustuva paikannus noudattaa samantapaisia laskelmia kuin GPS-satelliittipaikannus. Laskelmien tarkkuus riippuu alueella olevien tukiasemien määrästä, joten matkapuhelinverkkoon perustuva paikannus on yleensä tarkimmillaan kaupunkialueilla. (Holdener 2011, luku 1.)

Kaikille internetiin liitetyille laitteille annetaan IP-osoite, jonka avulla laite voi lähettää ja vastaanottaa dataa tietoverkossa. IP-osoitteet on jaettu palveluntarjoajille alueellisesti, joten käyttäjän sijainti on helppo todentaa ainakin kaupungin tarkkuudella. Usein IP-osoitteen avulla ei kuitenkaan päästä

käsiksi suoraan käyttäjän laitteeseen, vaan verkon järjestelyistä riippuen selvitetty sijainti voi olla jopa kilometrien päässä itse käyttäjästä. (Holdener 2011, luku 1.)

On tärkeää ymmärtää, että kaikista lähteistä saadun sijaintitiedon tarkkuus vaihtelee jatkuvasti. Sijainnin määrittämiseen vaadittava data muuttuu, jolloin laskutoimituksen tulos heittelee. Erilaiset häiriöt signaaleissa tai laitteissa, sääolot ja muut tekijät voivat vaikuttaa paikannuksen tuloksiin. Suurin vaikutus sijaintitiedon tarkkuuteen on kuitenkin datan lähteellä. Tarkimmat tulokset saadaan käyttämällä GPS-satelliittipaikannusta, heikoimmat puolestaan IP-osoitteen perusteella. Tämän vuoksi on tärkeää seurata paikannuksen tuloksiin liitettyjä tietoja tulosten tarkkuudesta. (Holdener 2011, luku 2.)

Sijaintitietoa hyödyntävät palvelut ovat arkipäiväistyneet viimeisen vuosikymmenen aikana älypuhelinten yleistymisen ja tarkemman paikannuksen myötä. Lisäksi ominaisuudet, jotka aikaisemmin olivat saatavilla vain mobiilisovelluksissa, ovat nyt myös verkkosivujen ja web-sovellusten käytössä HTML5:n ja selainten Geolocation-rajapinnan myötä. On myös odotettavissa, että paikannuksen tarkkuus kasvaa edelleen tulevaisuudessa. (Holdener 2011, luku 6.)

2.2 Käyttäjän sijaintitiedon hyödyntäminen selaimen Geolocation-rajapinnan avulla

Verkkoselainten tarjoama Geolocation-rajapinta tarjoaa korkean tason ohjelmointirajapinnan laitteen sijaintitiedon hyödyntämiseen. Geolocation-rajapinnan toteuttavissa selaimissa on käytettävissä Geolocation-olio, joka sisältää kaikki rajapintaan liittyvät toiminnot. Rajapinta ei ota kantaa siihen, mistä lähteestä sijaintitieto on saatu, vaan selain käyttää laitteessa saatavilla olevaa tietoa. Vaihtelevien paikannusmenetelmien ja tulosten mahdollisen epätarkkuuden vuoksi onkin huomioitava, ettei saatavilla oleva sijaintitieto ole aina varmuudella laitteen tarkka tai edes todellinen sijainti. (Holdener 2011, luku 3.)

Edellä mainittu Geolocation-olio sisältää kolme julkista metodia: clearWatch, getCurrentPosition sekä watchPosition. Kaksi viimeisintä liittyvät laitteen sijaintitiedon selvittämiseen, ja molempien parametreiksi annetaan takaisinkutsufunktio sijaintitiedon käsittelyä varten ja haluttaessa takaisinkutsufunktio virhetilanteen varalta sekä Geolocation-rajapinnan määrittelyolio. Määrityksiin kuuluvat muuttujat enableHighAccuracy, maximumAge sekä timeout. (Holdener 2011, luku 3.)

Totuusarvomuuttuja enableHighAccuracy määrittää, pyritäänkö sijaintiedossa mahdollisimman suureen tarkkuuteen. Muuttujan asettaminen arvoon true voi pidentää sijaintitiedon haun kestoa ja lisätä sovelluksen virrankulutusta. Kokonaisluku maximumAge määrittää, kuinka pitkään käytetään välimuistiin tallennettua sijaintitietoa ja kokonaisluku timeout määrittää, kuinka kauan odotetaan vastausta ennen virheilmoitusta. Välimuistiin tallennettua tietoa voidaan käyttää erityisesti, kun sovelluksen kannalta ei ole välttämätöntä seurata jatkuvasti laitteen sijaintia. Sijaintitiedon haku välimuistista vähentää kutsuja Geolocation-rajapintaan ja vaikuttaa sovelluksen suorituskykyyn. (Holdener 2011, luku 3.)

Geolocation-rajapinnasta saatava tieto on talletettu Position-olion muuttujiin. Näistä tärkeimmät ovat desimaaliluvut *latitude* (suom. leveyspiiri) ja *longitude* (suom. pituuspiiri), jotka kertovat laitteen sijainnin World Geodetic System -koordinaatistossa (WGS 84, suom. Maailman geodeettinen järjestelmä). (Holdener 2011, luku 3.) WGS 84 on 1960-luvulla kehitetty ja edelleen päivitettävä järjestelmä, joiden avulla leveys- ja pituuspiireihin perustuva koordinaatisto ja maapallon hieman navoilta kasaan painunut muoto voidaan yhdistää toimivaksi kokonaisuudeksi. (Holdener 2011, luku 2.)

Geolocation-rajapinnasta saatu paikannus perustuu siis maantieteelliseen koordinaatistoon, jossa tietty piste ilmaistaan pituus- ja leveyspiirien avulla. Leveyspiirit kiertävät maapalloa vaakasuuntaisesti, ja sijainti leveyspiirillä ilmaistaan astelukuna väliltä -90–90 astetta. Leveyspiiri 0 sijaitsee päiväntasaajalla, ja asteikon ääripäissä sijaitsevat pohjois- ja etelänapa. Pituuspiirit kiertävät maapalloa pystysuunnassa, ja sijainti pituuspiirillä ilmaistaan astelukuna väliltä -180–180 astetta. Pituuspiiri 0 sijaitsee sopimuksenvaraisesti Englannin Greenwichissä. Sama koordinaatti voidaan ilmaista joko desimaalilukuna tai asteena, minuutteina ja sekunteina. (Holdener 2011, luku 2.)

Sijaintitietoon voidaan ajatella leveys- ja pituuspiirin lisäksi kuuluvan kohteen korkeus sekä sen nopeus ja suunta. Nopeuden ja suunnan avulla voidaan saada tieto laitteen tai käyttäjän tämänhetkisestä rintama- tai kulkusuunnasta, kuljetusta matkasta tai reitistä. (Holdener 2011, luku 2.) Leveys- ja pituuspiirin lisäksi Geolocation-rajapinnan Position-olio voikin sisältää tiedon laitteen korkeudesta, laitteen kulkusuunnasta sekä nopeudesta. Lisäksi olio sisältää tiedon sijaintitiedon tarkkuudesta metreinä. (Holdener 2011, luku 3.)

Geolocation-rajapinnan määritelmään kuuluu, ettei verkkosovelluksille saa lähettää tietoa käyttäjän sijainnista ilman käyttäjän hyväksyntää. Yksityisyyteen liittyvät kysymykset kuuluvatkin väistämättä

käyttäjän sijaintiedon hyödyntämiseen ja etenkin tietojen tallentamiseen. Mihin käyttäjän sijaintitieto tallennetaan? Kenen vastuulla on palvelimien tietoturva, ja kuinka käyttäjien tiedot on suojattu ulkopuolisilta? (Holdener 2011, luku 6.) Käytännössä käyttäjän suostumuksen saaminen on selaimissa toteutettu sivun latauksen yhteydessä esiin ponnahtavalla kyselyllä. Käyttäjä voi tällöin hyväksyä sijaintinsa käytön kertaluontoisesti tai pysyvästi kyseiselle verkkopalvelulle. Suostumuksen voi myöhemmin määrittää uudelleen selaimen asetuksista. Sovelluksen kehittäjän näkökulmasta on varauduttava siihen, että käyttäjä ei myönnä lupaa käyttää sijaintiaan. (Holdener 2011, luku 3.)

2.3 Interaktiivinen malli käyttäjän ympäristöstä

2.3.1 Ympäristön kuvailutiedon hakeminen Overpass-rajapinnasta

Kehittämistehtävässä rakennettiin yksinkertainen 3D-malli käyttäjän ympäristöstä. Ympäristön mallintamiseen kuului kolme vaihetta: käyttäjän sijainnin selvittäminen, ympäristön kuvailutietojen lataaminen käyttäjän sijainnin perusteella sekä karttanäkymän rakentaminen 3D-objektien avulla.

Käyttäjän sijaintitiedon hakemista varten luotiin JavaScript-moduuli Geolocation.js, jonka tehtävänä on pyytää käyttäjän laitteelta sijaintitieto ja päivittää tieto laitteen sijainnin muuttuessa. Moduuli sisältää Geolocation-funktion, jota ei kuitenkaan viedä moduulista sellaisenaan. Sen sijaan Geolocation-moduulista viedään singleton-olio, joka saadaan, kun *export default* -lauseessa kutsutaan moduulin sisältämää Geolocation-funktiota. Tällä varmistettiin, että Geolocation-moduulin palauttamasta oliosta on sovelluksen käytössä vain yksi kopio ja että koko sovelluksella on käytössään aina sama ajantasainen sijaintitieto.

Kun Geolocation-moduuli alustetaan, haetaan ensimmäinen sijaintitieto, joka tallennetaan moduuliin eräänlaiseksi nollapisteeksi. Tätä tietoa hyödynnetään myöhemmin, kun käyttäjän sijainti ja ympäristö esitetään 3D-objekteina karttanäkymässä. Alustusvaiheessa käynnistetään myös sijaintitiedon päivitys kutsumalla selaimen Geolocation-rajapinnan watchPosition-metodia. Metodille annetussa takaisinkutsufunktiossa tallennetaan saatu tulos ajantasaiseksi sijaintitiedoksi moduuliin.

Ennen sijaintitiedon hakua varmistetaan, että käyttäjän selaimesta löytyy Geolocation-rajapinnan tarjoama *geolocation*-olio. Geolocation-moduulin alustusfunktio palauttaa JavaScriptin Promise-

olion, joka hylätään, mikäli selaimessa ei havaita *geolocation*-oliota. Virhetilanteen varalta lisättiin myös sijaintietoa hakeviin funktioihin takaisinkutsufunktio, joka luo virheestä sovelluksen laajuisen virhetapahtuman ja kirjoittaa virheen selaimen konsoliin.

Geolocation-moduuliin tallennetun sijaintitiedon perusteella voidaan laatia kysely Overpass-rajapintaan käyttäjän ympäristön kuvailutietojen hakemiseksi. Overpass-rajapinta on palvelu, joka mahdollistaa OpenStreetMapin karttadatan hakemisen http-kutsujen avulla (OpenStreetMap 2019b). OpenStreetMap on yhteisöllinen projekti, jossa kerätään vapaaehtoisvoimin maantieteellistä dataa ja luodaan karttanäkymiä (OpenStreetMap 2017). OpenStreetMap on avointa dataa, joka on lisensoitu OpenStreetMap-säätiön omalla lisenssillä. OpenStreetMapin dataa voidaan kopioida ja hyödyntää vapaasti, kun lähde mainitaan käytön yhteydessä. (OpenStreetMap 2019a.)

Overpass-rajapinnasta voidaan hakea dataa käyttämällä XML-muotoista kyselyä tai hyödyntämällä Overpassin omaa kyselykieltä (Overpass QL) (OpenStreetMap 2019b). Kehittämistehtävässä käytettiin Overpassin omaa kyselykieltä. Overpass QL:n avulla voidaan rajata hakutuloksia useilla eri tavoilla. Kehittämistehtävässä tulokset rajattiin alueellisesti käyttäjän sijainnin perusteella sekä OpenStreetMapin dataan sisältyvien avain-arvo -parien avulla. Hakutulokset rajattiin vain seuraavanlaisiin kohteisiin: rakennukset, tiet, vesialueet, erilaiset puistot ja urheilukentät. Overpass-kysely rakennettiin osaksi HTTP GET-pyyntöä, ja kyselyn avulla asetettiin palautettavan datan muodoksi JSON-olio.

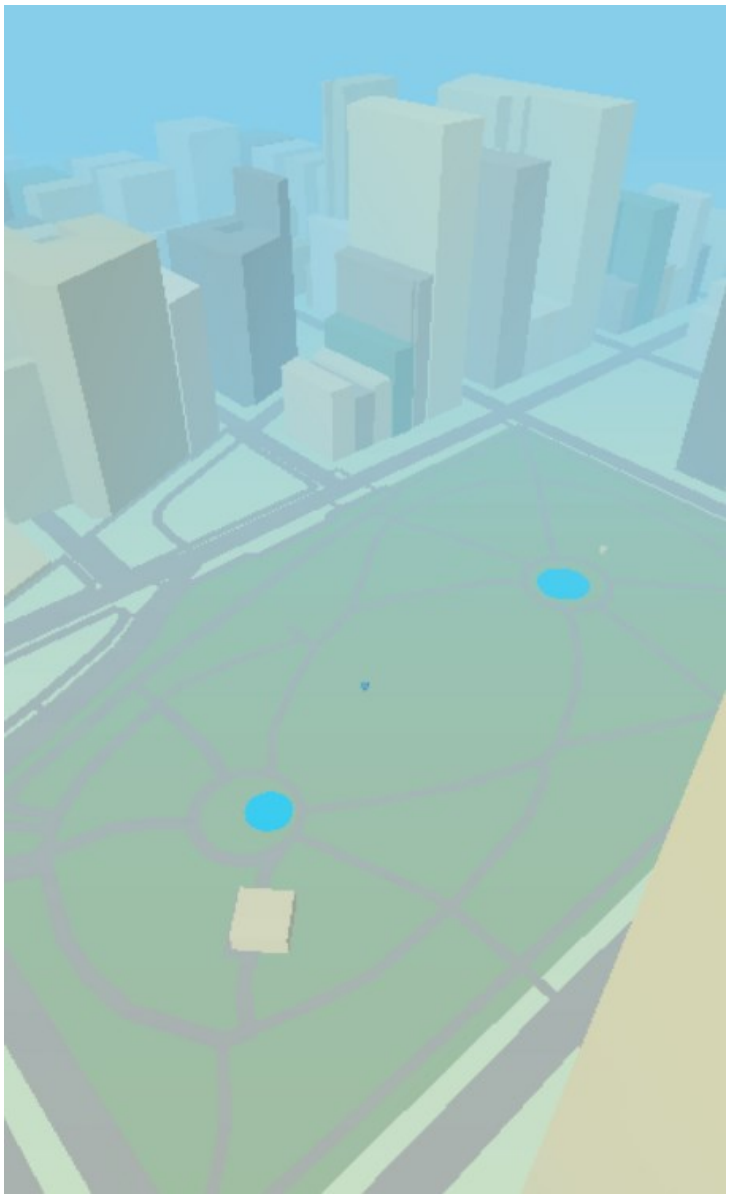
Alueellisesti rajatun kyselyn kannalta oli olennaista, että Overpass-rajapinnasta voitiin hakea lisää dataa laitteen sijaintitiedon muuttuessa. Overpass-rajapinnasta haetun datan määrä oli verrattain suurta, minkä vuoksi oli tärkeää välttää jo kertaalleen haetun datan hakemista uudelleen. Tämä toteutettiin jakamalla käyttäjän ympäristö matemaattisesti suorakulmion muotoisiin alueisiin. Overpass-rajapinnasta haettujen alueiden sijaintitiedot tallennetaan sovelluksen muistiin, ja kahden sekunnin välein verrataan käyttäjää ympäröivää suorakulmion muotoista aluetta aiemmin haettujen alueiden muodostamaan alaan. Mikäli tarkistuspisteet ovat aiemmin haetun alueen ulkopuolella, laaditaan uusi kysely Overpass-rajapintaan lähetettäväksi.

2.3.2 Käyttäjän ympäristön mallintaminen kuvailutiedon perusteella

Kolmiulotteisen karttanäkymän kannalta merkittävin vaihe oli Overpass-rajapinnasta saadun datan muuntaminen muotoon, jota voitiin hyödyntää three.js-kirjaston 3D-objektien luomisessa. Three.js on JavaScript kirjasto, joka hyödyntää selainten WebGL-rajapintaa ja yksinkertaistaa 3D-grafiikan renderöintiä verkkoselaimessa. WebGL on JavaScript-rajapinta, joka mahdollistaa interaktiivisen 3D-grafiikan hyödyntämisen verkkoselaimessa. WebGL-rajapinta on selainrajapintojen joukossa poikkeuksellisen matalan tason rajapinta, sillä se on yhteydessä laitteen näytönohjaimeen. Tämä mahdollistaa 3D-grafiikan esittämisen selaimessa, mutta tekee 3D-näkymien toteuttamisesta verrattain monimutkaista. (Caballero 2011.)

WebGL-rajapintaa hyödynnetäänkin useimmiten jonkin sitä varten laaditun JavaScript-kirjaston avulla. Esimerkiksi three.js-kirjasto tarjoaa useita perusmuotoja luokkina, jotka mahdollistavat 3D-objektien luonnin muutamalla rivillä ohjelmakoodia. Samanlaisen objektin luominen suoraan WebGL-rajapintaa käyttäen voi vaatia moninkertaisen määrän ohjelmakoodia. Three.js-kirjaston avulla voidaan selaimen 3D-näkymään ladata myös valmiita 3D-objekteja tekstuureineen. (Caballero 2011.)

Overpass-rajapinnasta haettuihin olioihin sisältyy avain-arvo -pareja, joiden perusteella tulokset voitiin lajitella esimerkiksi rakennuksiin, puistoihin ja kulkuväyliin. 3D-karttanäkymän rakennukset päätettiin toteuttaa piirtämällä dataan sisältyvien sijaintitietojen perusteella rakennuksesta kaksiulotteinen tasomuoto – siis vastaava, joka rakennuksella on kaksiulotteisella kartalla. Tämän jälkeen kaksiulotteisesta muodosta voitiin pursottaa 3D-objekti käyttämällä three.js-kirjaston ExtrudeGeometry-luokkaa. Muut karttanäkymän kohteet kuten tiet ja puistoalueet päätettiin toteuttaa 3D-näkymään sijoitettuina tasomuotoina (katso kuvio 1). Overpass-olioihin sisältyviä pituus- ja leveyspiirejä ei voitu kuitenkaan käyttää sellaisenaan kaksiulotteisessa karttanäkymässä, vaan ennen muotojen luomista koordinaattiparit piti muuntaa sopiviksi three.js-kirjaston luoman 3D-näkymän koordinaatistoon.



KUVIO 1. Kehittämistehtävän sovelluksen karttanäkymä New Yorkin Madison Square Parkista. Näkymässä hyödynnettiin OpenStreetMapin käyttäjien lisäämiä tietoja rakennusten korkeudesta.

Overpass-rajapinnasta haettu data sisältää taulukon OpenStreetMap-olioista, jotka vastaavat kyselyn rajausta. Karttanäkymää varten haetut oliot sisältävät listan solmuista (*node*). Solmut ovat yksilöllisiä maantieteellisiä pisteitä, jotka sisältävät sijaintitiedon WGS84-koordinaattiparina. Esimerkiksi rakennuksissa nämä solmut osoittavat rakennuksen kulmien sijainnit, mikä mahdollistaa kaksiulotteisen rakennuksen muodon piirtämisen solmujen avulla.

3D-karttanäkymän pohjana oli siis kaksiulotteinen taso, jolle kohteet sijoitettiin. WGS84-koordinaatit ilmoittavat OpenStreetMap-solmujen sijainnin maapallon muotoa mukailevalla geoidilla, joten

niitä ei voida sellaisenaan käyttää kaksiulotteiselle tasolle muodostettavan karttanäkymän piirtämiseen. Pituuspiirien välinen etäisyys kilometreinä on päiväntasaajalla noin 111 kilometriä, mutta leveyspiirillä 45 astetta pohjoista tai eteläistä leveyttä enää noin 80 kilometriä. Täsmälleen pohjois- tai etelänavalla pituuspiirien välinen etäisyys supistuu nollaan (katso kuvio 2). (NOAA 2019.) Jos pituuspiirien välisen etäisyyden välistä muutosta ei huomioida kaksiulotteisen karttanäkymän luomisessa, vääristyy näkymä merkittävästi.

KUVIO 2. Pituuspiirien välinen etäisyys muuttuu siirryttäessä päiväntasaajalta kohti napoja (Hellerick 2020).

Maantieteellisten etäisyyksien laskemiseen on kehitetty useita matemaattisia kaavoja, joiden monimutkaisuus ja laskelman tarkkuus kulkevat usein käsi kädessä (Veness 2019). Kehittämistehtävässä käytettiin pituuspiirien välisen etäisyyden laskemisessa yksinkertaista laskutapaa, jossa hyödynnetään laskettavien pisteiden leveyspiirien keskiarvoa ja trigonometrisia funktioita (katso kuvio 3). Geolocation-moduuliin nollapisteeksi tallennettu käyttäjän sijaintitieto on laskutoimituksen lähtöpisteenä. Kaikkien OpenStreetMap-solmujen sijainti 3D-karttanäkymässä laskettiin suhteessa tähän nollapisteeseen.

```
function longitudeToDistance(origin, coords) {
    const averageLatitude = (coords.latitude + origin.latitude) / 2;
    const deltaLongitude = coords.longitude - origin.longitude;
    return deltaLongitude * Math.cos(degreesToRadians(averageLatitude)) * ENV.DEGREE_IN_METERS;
}
```

KUVIO 3. Pituuspiirien aste-eron muuntaminen etäisyydeksi

Yksinkertaisen laskutavan epätarkkuus paljastuu, mikäli käyttäjän sijainti muuttuu käytön aikana merkittävästi pohjois-etelä -suunnassa, jolloin karttanäkymän 3D-objektit alkavat vääristyä. Alueellisesti rajatussa sovelluksessa tällä ei kuitenkaan ole merkitystä. Vääristymää voidaan tarvittaessa korjata käyttämällä tarkempaa laskutapaa tai päivittämällä laskennassa käytettyä nollapistettä käyttäjän siirtyessä huomattavia matkoja pohjois-etelä -suunnassa.

Kun WGS84-koordinaattiparit oli projisoitu kaksiulotteiselle tasolle, voitiin saatujen koordinaattipisteiden perusteella rakentaa karttanäkymä three.js-kirjaston avulla luotuun näkymään. Kehittämistehtävässä kartan kohteet rakennettiin käyttämällä three.js-kirjaston tarjoamia peruskappaleita ja -tasoja. Sovelluksen käyttökokemuksen kannalta osoittautui ongelmalliseksi, että Overpass-rajapinnasta saatujen tulosten suodattaminen, tulosten järjestäminen ryhmiin, WGS84-koordinaattien projisoiminen kaksiulotteiseen koordinaatistoon ja lopulta 3D-objektien luominen kestivät joidenkin alueiden kohdalla yhteensä jopa satoja millisekunteja. Tänä aikana karttanäkymän päivittäminen oli pysähdyksissä.

Ratkaisuna ongelmaan päätettiin käyttää JavaScriptin Web Worker -rajapintaa, jonka avulla voidaan ajaa ohjelmakoodia useammassa säikeessä. Selaimessa JavaScript-koodia ajetaan yhdessä säikeessä, minkä vuoksi pitkäkestoisten toimintojen aikana muun ohjelman suoritus on pysähdyksissä. Web Worker -rajapinnan avulla pitkäkestoiset toiminnot ja laskenta voidaan siirtää uuteen säikeeseen, mikä mahdollistaa web-sovelluksen toiminnan jatkamisen ohjelmakoodin ajon aikana. (Green 2012, luku 1).

Web Worker -rajapinta on käytettävissä kaikissa nykyaikaisissa selaimissa, myös mobiililaitteissa (Green 2012, luku 1; Can I use… 2019). Web Worker -rajapinta soveltuu erityisesti monimutkaisten laskutoimitusten suorittamiseen, suurten taulukoiden tai JSON-datan prosessointiin tai esimerkiksi kuvien, videon ja äänitiedostojen käsittelyyn. Web Workerin ohjelmakoodilla ei kuitenkaan ole yhteyttä selaimen *window*-, *document*- tai *parent*-oliohin, joten sillä ei voida vaikuttaa sovelluksen käyttöliittymään. (Green 2012, luku1.)

Web Worker on yhteydessä selaimen säikeessä ajettavaan JavaScript-koodiin viestien välityksellä. Web Workeriin voidaan luoda takaisinkutsufunktiot *onmessage* ja *onerror*-tapahtumille. (Green 2012, luku 2.) Kehittämistehtävässä Workerin säikeeseen kirjoitettiin http-kutsu Overpass-rajapintaan sekä rajapinnasta saadun vastauksen suodattaminen ja lajittelu. Worker-ohjelmakoodi akti-

voituu, kun selaimen säikeestä kutsutaan Worker-olion postMessage-metodia. Worker-ohjelmakoodi saa postMessage-kutsun parametrina viestin, joka sisältää Overpass-rajapinnan url-osoitteen Overpass QL -kyselyineen sekä tulosten lajitteluun tarvittavat tiedot. Tarvittavat toimenpiteet suoritettuaan Web Worker palauttaa tuloksen selaimen säikeelle postMessage-metodin avulla (katso kuvio 4).

```
self.onmessage = function(msg) {
    let {id, payload} = msg.data;
    let {url, filters} = payload;

    fetch(url).then((res) => {
        if (res.status === 200) {
            return res.json();
        } else {
            throw new Error('Overpass API call failed, ' + res.status);
        }
    })
    .then((result) => {
        return filterOverpassData(result.elements, filters);
    })
    .then(result => {
        self.postMessage({
            id,
            error: null,
            payload: {
                result,
                url
            }
        });
    })
    .catch(error => {
        self.postMessage({
            id,
            error: error.message,
            payload: null
        });
    });
```

KUVIO 4. Osa Web Workerin takaisinkutsufunktiosta *onmessage*-tapahtumalle

Web Workeria hyödynnettäessäkin kehittämistehtävän 3D-objektien luonti jää selaimen pääsäikeen varaan. Three.js-kirjaston 3D-objektit eivät sovellu siirrettäväksi selaimen säikeen ja Web Workerin välillä osana tavallista postMessage-viestiä. Viesteissä lähetettävä data serialisoidaan ja kootaan uudelleen viestin vastaanottavassa päässä (MDN 2019). Tähän prosessiin 3D-objektit ovat liian raskaita ja monimutkaisia, ja ne aiheuttavat virheilmoituksen lähetystä yritettäessä. Uusimmissa selaimissa on vaihtoehtoisia tapoja lähettää dataa pääsäikeen ja Web Workerin välillä

(MDN 2019; Green 2012, luku 2), mutta kehittämistehtävän yhteydessä tyydyttiin tutustumaan tavanomaisen Web Workerin toimintaan ja hyväksyttiin 3D-objektien luonnin aiheuttama pieni viive näkymän päivittymisessä.

2.3.3 Interaktiivisen sisällön toteuttaminen

Sijaintiperusteisten sovellusten keskeinen tarkoitus on, että käyttäjälle voidaan tarjota sisältöä ja toimintoja riippuen hänen sijainnistaan. Sijaintitietoa voidaan hyödyntää monissa erilaisissa palveluissa kuten sosiaalisen median sovelluksissa tai verkkokaupoissa. Kehittämistehtävän sijaintiperusteista sovellusta lähimpänä ovat kuitenkin sovellukset, joissa käyttäjän sijaintia seurataan ja päivitetään aktiivisesti ja joissa merkittävä osa sovelluksen toiminnoista kytkeytyy käyttäjän sijaintitietoon.

Esimerkkejä tällaisista sovelluksista ovat reitti- tai navigointipalvelut, kanssakäyminen muiden lähistöllä liikkuvien käyttäjien kanssa ja etenkin sijaintiperusteiset lisätyn todellisuuden sovellukset (Holdener 2011, luku 1 ja luku 6). Paikkatietoa ja mobiililaitteen muita sensoreita hyödyntäviä tunnettuja sovelluksia ovat esimerkiksi viihdepelit Pokemon GO sekä velhomaailmaan sijoittuva Harry Potter: Wizards Unite.

Kehittämistehtävässä päätettiin toteuttaa 3D-karttanäkymään interaktiivisia kohteita, joita koskettamalla käyttäjä voi saada lisätietoa valitusta kohteesta. Sijaintiperusteisen sovelluksen runkona olevaan JavaScript-kirjastoon luotiin mahdollisuus lisätä taulukkona tiedot karttanäkymän interaktiivisista kohteista.

Interaktiiviset kohteet lisätään sovellukseen karttanäkymän luontivaiheessa osana karttanäkymän luontifunktion asetusoliota (katso kuvio 5). Kirjasto mahdollistaa interaktiivisten kohteiden lisäämisen kahdella eri tavalla. Asetusolion *interactiveElements*-parametri sisältää taulukon datasta, joka halutaan lisätä olemassa oleviin OpenStreetMap-kohteisiin. Taulukon jokaisen olion tulee sisältää parametri *id*, joka kytkee olion sisältämän datan OpenStreetMap-kohteeseen, jolla on sama id. Tätä taulukkoa käytettiin kehittämistehtävässä lisäämään karttanäkymään Oulun keskusta-alueen tunnettuja kohteita kuten Kauppahalli, kaupunginkirjasto ja -teatteri, Oulun tuomiokirkko sekä Oulun linna (katso kuvio 6).

```
const map = jsGo({
    debug: false,
    useOSMBuildingHeight: false,
    buildingHeight: 2,
    interactiveElements: data,
    markers
});
```

KUVIO 5. Karttanäkymän luontifunktio ja sille annettu asetusolio.

Asetusolion parametri *markers* sen sijaan sisältää taulukon kohteista, jotka halutaan lisätä kartalle interaktiivisina merkkeinä. *Markers*-taulukon olioiden tulee sisältää parametri *location*, joka määrää merkin sijainnin karttanäkymässä. *Markers*-taulukon avulla sovellukseen voidaan lisätä kohteita, joita ei löydy Overpass-rajapinnasta haetusta datasta. Kehittämistehtävässä lisättiin tällä tavoin kartalle Oulun keskusta-alueen patsaita kuten Toripoliisi ja Franz Mikael Franzénin patsas (katso kuvio 6).

KUVIO 6. Vuorovaikutteiset kohteet on värjätty punaisella. Rakennus on Oulun Tuomiokirkko, johon lisätty sisältö on liitetty Overpass-rajapinnasta haettuun kuvailutietoon. Etualalla puolestaan on Frans Mikael Franzénin patsas, joka lisättiin markers-taulukkoon karttanäkymän luontivaiheessa.

Sijaintiperusteisen sovelluksen kirjastoon rakennettiin moduuli EventCenter, jonka tehtävänä on hallita määrättyjen tapahtumien käsittelyä, mukaan luettuna interaktiivisten kohteiden valinta karttanäkymässä. Kirjastoa hyödyntävä sovellus voi lisätä tapahtumakäsittelijöitä haluamilleen tapahtumille takaisinkutsufunktioina. Se, kuinka interaktiivisen kohteen valintaan liittyvä tapahtuma käsitellään ja näytetään sovelluksen käyttöliittymässä, on täysin kirjastoa hyödyntävän sovelluksen varassa.

Kehittämistehtävässä interaktiivisiin kohteisiin liittyvät teksti- ja kuvasisällöt näytettiin karttanäkymän päälle avautuvassa modaali-ikkunassa (katso kuvio 7). Sekä kohteisiin liittyvät sisällöt että niiden näyttäminen käyttöliittymässä toteutettiin hyvin yksinkertaisesti ja esimerkinomaisesti. Nykyisillä web-teknologoilla voidaan tietenkin toteuttaa monipuolisia ja teknisesti vaativia sisältöjä videoista erilaisiin interaktiivisiin tai pelillisiin sisältöihin ja lisätyn todellisuuden näkymiin.

KUVIO 7. Vuorovaikutteiseen kohteeseen liittyvä sisältö avattuna modaali-ikkunassa. Esimerkin kuva ja tekstisisällöt poimittiin Wikipedian artikkelista (Wikipedia 2019, viitattu 4.10.2020).

3 PROGRESSIIVISEN WEB-SOVELLUKSEN TOTEUTTAMINEN

3.1 Progressiivisen web-sovelluksen ominaisuuksia

Googlen insinööri Alex Russell ja suunnittelija Frances Berriman ottivat vuonna 2015 käyttöön termin progressiivinen web-sovellus (Progressive Web App, PWA). He kuvaavat käsitteellä sellaisia verkkosivuja ja web-sovelluksia, jotka käyttävät uusimpien verkkoselainten tarjoamia mahdollisuuksia hyödyntää natiivien mobiilisovellusten ominaisuuksia verkkosivuilla ja -palveluissa. Samalla he edellyttävät, että progressiiviset web-sovellukset noudattavat webin perusperiaatteita; niiden tulee olla linkitettävissä internetissä ja niiden tulee noudattaa HTML:n ja CSS:n semanttisia periaatteita sekä ilmaiseksi hyödynnettävissä olevia standardeja. (Russell 2015.)

Myös niin sanottujen hybridisovellusten kehityksessä käytetään web-teknologioita. Paikka eri käyttöjärjestelmien sovelluskaupoissa kuitenkin edellyttää, että hybridisovelluksissa luovutaan joistakin webin perusperiaatteista. Progressiiviset web-sovellukset puolestaan pyrkivät yhdistämään natiivi- ja web-sovellusten parhaita puolia toimivaksi kokonaisuudeksi. Russellin ja Berrimanin käsite on vakiintunut yleiseen käyttöön viimeisten vuosien aikana, ja sen sisältö nojaa edelleen pitkälti heidän alkuperäiseen kuvaukseensa. (Russell 2015.)

Progressiivisten web-sovellusten käyttöliittymät ovat responsiivisia, mikä mahdollistaa natiivisovellusten kaltaiset käyttöliittymät eri laitteissa (Russell 2015). Progressiivinen web-sovellus on lisäksi mahdollista asentaa laitteen aloitusnäkymään. Verkkosovelluksen yhteyteen liitetään tiedot sovelluksen ikoneista, taustakuvista, väreistä ja ruudun orientaatiosta. Tällöin sovellus käynnistyy aloitusnäkymästä yhdellä kosketuksella puhelimeen asennettujen sovellusten tavoin. (Hume 2017, luku 1.) Näin avattuna sovellus voidaan näyttää ilman selaimen kehyksiä, mikä vahvistaa entisestään natiivisovelluksen kaltaista käyttökokemusta (Russell 2015).

Progressiivisen web-sovelluksen erottaa perinteisestä verkkosivustosta myös mahdollisuus suorittaa ohjelmakoodia ja sovelluksen toimintoja taustalla, vaikka selaimen välilehti olisi suljettuna. Nykyaikaiset selaimet mahdollistavat push-ilmoitusten ja taustasynkronoinnin hyödyntämisen verkkopalveluissa. (Ater 2017, luku 1.)

Koska progressiiviset web-sovellukset ovat teknisesti ajatellen perinteisiä verkkosivustoja, ne ovat mobiilisovellukseen verrattuna helposti saavutettavissa. Ne ovat internetin hakukoneiden löydettävissä sekä linkitettävissä toisille verkkosivuille tai sosiaalisen median julkaisuihin. Sovelluksen käyttö ei edellytä sen lisäämistä laitteen aloitusnäytölle, joten käyttäjä voi palvelun löydettyään aloittaa sen käytön asentamatta mitään laitteelleen. (Hume 2017, luku 1.)

Perinteisen verkkosivuston luonne mahdollistaa sen, että sovelluksen sisällöt ovat aina ajan tasalla ja että uusimmat ominaisuudet ovat käytössä jokaisen latauskerran jälkeen (Russell 2015). Progressiivisten web-sovellusten vahvuudeksi onkin koettu juuri mahdollisuus yhdistää verkkosivuille tyypillinen matala käyttökynnys natiivisovelluksista tuttuun sujuvaan käyttökokemukseen (Ater 2017, luku 1).

Yksi merkittävimmistä progressiivisten web-sovellusten ominaisuuksista on tapa, jolla uudet selaimet mahdollistavat verkkosivun resurssien tallentamisen selaimen välimuistiin. Sovelluksen kehittäjät pystyvät hallitsemaan välimuistia huomattavasti aikaisempaa yksityiskohtaisemmin, mikä mahdollistaa esimerkiksi valittujen ominaisuuksien käytön ilman verkkoyhteyttä. (Hume 2017, luku 1.) Kun offline-ominaisuuksiin lisätään mahdollisuus käyttää taustasynkronointia ja push-ilmoituksia, alkaa verkkosovelluksen käyttövarmuus muistuttaa natiivisovelluksia. Oikein rakennetussa web-sovelluksessa käyttäjä voi huoletta kirjoittaa viestin offline-tilassa ja luottaa siihen, että se toimitetaan vastaanottajalle, kun verkkoyhteys on jälleen saatavilla. (Ater 2017, luku 1.)

Välimuistin hyödyntäminen mahdollistaa myös nopeat latausajat. Sovellukseen saadaan nähtäville sisältöä hitaallakin verkkoyhteydellä silmänräpäyksessä. (Ater 2017, luku 1.) Sivuston staattisten osien tallentaminen välimuistiin tarkoittaa, että ne ovat ladattavissa paikallisesti laitteelta sovelluksen tulevilla käyttökerroilla. Käyttäjälle pystytään näyttämään välittömästi mielekästä sisältöä, mikä saa sovelluksen käytön tuntumaan sujuvammalta, vaikka dynaamisen sisällön lataaminen olisikin vielä käynnissä. (Hume 2017, luku 1.)

Monia progressiivisten web-sovellusten ominaisuuksia tuetaan vain eri selainvalmistajien viimeisimmissä selainversioissa, ja esimerkiksi iOS-käyttöjärjestelmässä osa selaimista ei tue laisinkaan kyseisiä ominaisuuksia. Koska pohjimmiltaan kyse on kuitenkin tavallisista verkkosovelluksista, on progressiivisten web-sovellusten tarkoitus toimia myös sellaisissa selaimissa, jotka eivät tue mainittuja uusia ominaisuuksia. Sovellus rakennetaan siten, että uudet ominaisuudet parantavat käyt-

tökokemusta, mikäli selain tukee niitä. Perustoiminnot ovat käytettävissä myös vanhemmilla verkkoselaimilla. Siten progressiivisen web-sovelluksen ominaisuuksia voidaan myös helposti liittää jo olemassa oleviin verkkosivuihin tai -palveluihin osa-alue kerrallaan. (Hume 2017, luku 1; Gustafson 2017.)

Kehittämistehtävässä päätettiin hyödyntää kahta merkittävää progressiivisten web-sovellusten ominaisuutta: mahdollisuutta asentaa sovellus mobiililaitteen aloitusnäytölle sekä mahdollisuutta hallita yksityiskohtaisesti selaimen välimuistiin talletettavia sisältöjä latausaikojen nopeuttamiseksi ja käyttökokemuksen parantamiseksi. Kehittämistehtävässä ei ole sellaista päivittyvää sisältöä, jonka vuoksi olisi mielekästä toteuttaa push-ilmoitusten lähettäminen sovelluksesta. Käyttäjä ei pysty tallentamaan tai lähettämään sovelluksessa mitään sisältöä tai dataa, joten taustasynkronointikaan ei ole tarpeen.

3.2 Progressiivisen web-sovelluksen ominaisuuksien lisääminen

3.2.1 Sovelluksen auditointi Google Lighthouse -työkalulla

Google Lighthouse on avoimen lähdekoodin työkalu, jonka tarkoituksena on auttaa web-kehittäjiä arvioimaan verkkosivustojen ja -sovellusten laatua eri näkökulmista. Lighthouse on automatisoitu työkalu, joka arvioi esimerkiksi sovelluksen suorituskykyä, saavutettavuutta ja hakukoneoptimointia. Kehittämistehtävän kannalta olennaista on, että Lighthousen avulla voidaan arvioida, kuinka hyvin sovellus toteuttaa progressiivisen web-sovelluksen ominaisuuksia. (Google Developers 2019.)

Lighthouse-auditointi voidaan suorittaa usealla eri tavalla. Google Chrome -selaimessa auditointi on saatavilla sekä kehittäjätyökaluissa että erillisenä selainlaajennuksena. Lisäksi verkkosivuja tai -sovelluksia voi halutessaan auditoida Node-ympäristössä ajettavalla komentorivityökalulla. (Google Developers 2019.) Kehittämistehtävässä päätettiin käyttää selaimen kehittäjätyökaluissa saatavilla olevaa työkalua.

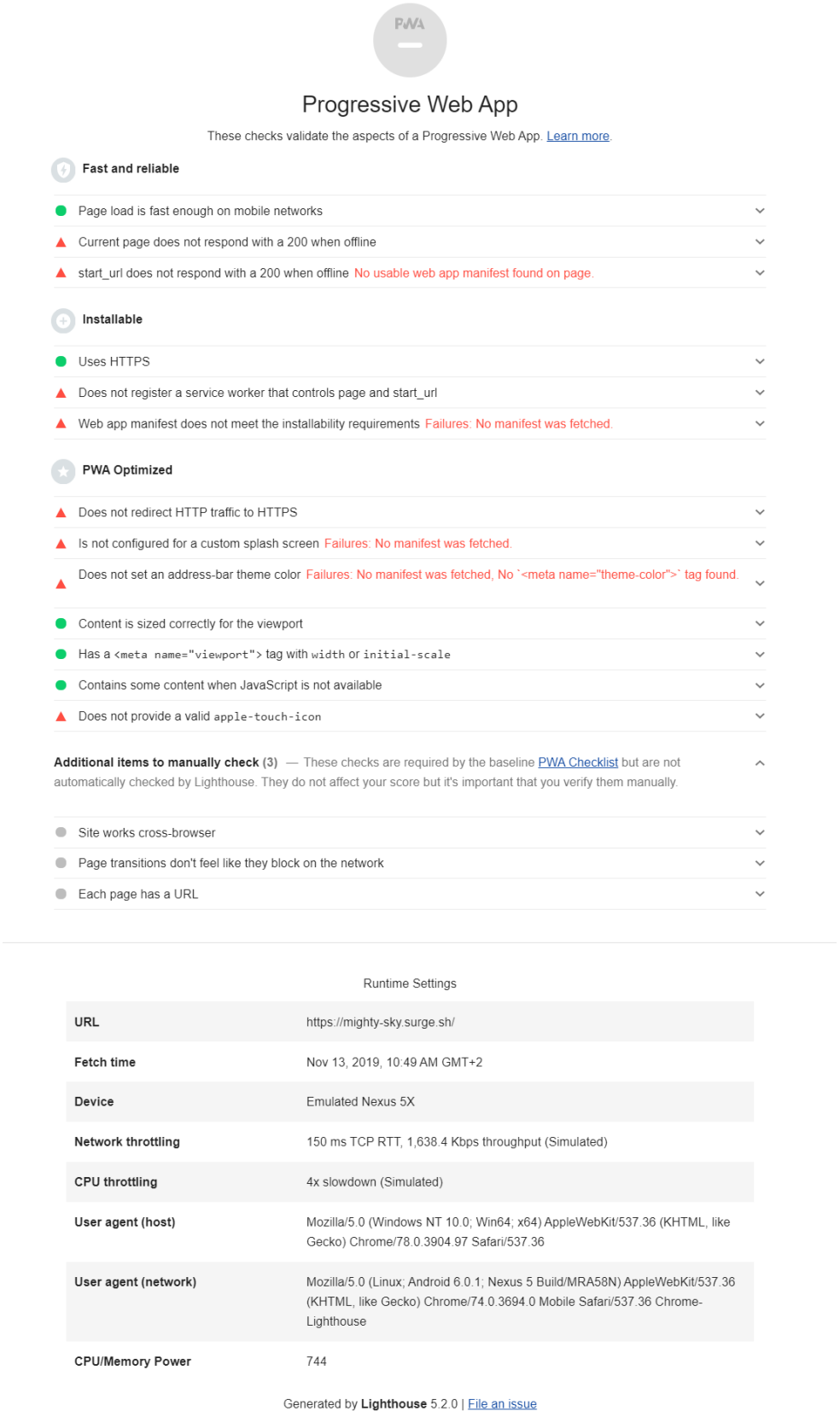
Lighthouse-työkalun progressiivisen web-sovelluksen auditointi sisältää arviot manifest-tiedoston sisällöstä, kotinäkymästä avautuvan verkkosovelluksen ulkoasusta, sovelluksen yleisestä responsiivisuudesta, HTTPS-protokollan hyödyntämisestä, sovelluksen suorituskyvystä vaihtelevissa

verkko-olosuhteissa sekä offline-ominaisuuksista verkkoyhteyden puuttuessa kokonaan. Työkalun avulla voidaan arvioida toteutettuja ominaisuuksia ja löytää lisäkehitystä vaativia kohteita. Auditoinnin eri osioista löytyvät hyperlinkit kutakin arviota koskevaan dokumentaatioon, jonka avulla on mahdollista korjata auditoinnissa havaitut puutteet. (Google Developers 2019.)

Kehittämistehtävässä Lighthouse-työkalua voitiin hyödyntää useammalla eri tavalla. Ensinnäkin auditoinnin eri osa-alueet ovat hyvä muistutus siitä, mitä progressiivisen web-sovelluksen käsitteellä käytännössä tarkoitetaan. Toisaalta Lighthouse-auditoinnin avulla oli helppo tarkistaa, toimivatko sovellukseen lisätyt progressiivisen web-sovelluksen ominaisuudet odotetusti. Lopuksi olisi mahdollista käydä läpi auditoinnissa toteutumatta jääneet ominaisuudet ja pohtia, olisivatko ne sovelluksen kannalta olennaisia ominaisuuksia ja millaisia teknisiä muutoksia vaadittaisiin niiden toteuttamiseksi.

Google Lighthouse -auditointi toteutettiin sovellukselle ensimmäisen kerran silloin, kun sovelluksen perustoiminnot olivat valmiina. Tällöin ei sovellukseen ollut vielä tietoisesti lisätty lainkaan progressiivisen web-sovelluksen ominaisuuksia. Sovelluksen kehitysversio oli ladattu Surge-palveluun, joka tarjoaa ilmaisen alustan julkaista selainpuolen sovelluksia. Auditoinnin kohdelaitteeksi valittiin mobiililaite, ja auditoinnissa simuloitiin hidasta 4G-verkkoyhteyttä.

Kolmestatoista tarkistetusta ominaisuudesta sovellus sai viisi hyväksyttyä tulosta (katso kuvio 8). Kaksi hyväksyttyä tulosta liittyi https-yhteyteen ja sovelluksen latausaikaan. Surge-palvelu tarjoaa verkkosivut ja -sovellukset https-yhteydellä, mikä on yksi progressiivisen web sovelluksen vaatimuksista. Myös käyttäjän sijaintitiedon hyödyntäminen edellyttää https-yhteyttä. Lisäksi sovelluksen latausaika simuloidulla 4G-yhteydellä täytti auditoinnin vaatimukset. Kaksi muuta hyväksyttyä tulosta liittyivät sovelluksen mobiilinäkymään. Sovelluksen html-koodi sisälsi mobiilinäkymän optimointiin tarkoitetun meta-tagin, ja sovelluksen sisältö oli oikeassa koossa suhteessa selainikkunaan.



KUVIO 8. Google Lighthouse -auditoinnin tulokset ennen progressiivisen web-sovelluksen ominaisuuksien lisäämistä.

Merkittävä osa hylätyistä auditointikohteista liittyi siihen, ettei sovellukseen ollut vielä lisätty manifest- ja Service Worker -tiedostoja. Manifest-tiedosto sisältää sovelluksen kuvauksen ja tarvittavat tiedot sovelluksen aloitusnäkymään asentamista varten. Service Worker on puolestaan JavaScript-tiedosto, joka mahdollistaa suurimman osan progressiivisiin web-sovelluksiin liitetyistä ominaisuuksista kuten offline-ominaisuudet, push-ilmoitukset sekä taustasynkronoinnin. Kehittämistehtävän sovelluksen muuttaminen progressiiviseksi web-sovellukseksi edellytti näiden kahden tiedoston lisäämistä ja progressiivisen web-sovelluksen ominaisuuksien toteuttamista niiden avulla.

3.2.2 Suorituskyky ja offline-ominaisuudet

Tämän opinnäytetyön kehittämistehtävä on web-sovellus, jossa käyttäjän ympäristöstä luodaan kolmiulotteinen karttanäkymä. Karttanäkymän luomiseen tarvittava data haetaan OpenStreetMap -projektin Overpass-rajapinnasta. Käyttäjän sijainnista riippuen saatetaan verkkopyynnöissä ladata suuriakin määriä ympäristön kuvailutietoa. Vaikka ladattu data käsiteltiin Web Worker -tiedostossa selaimen JavaScript-säikeen keskeytymättä, syntyi Overpass-rajapintaan lähetetyn pyynnön vastausta odotellessa viive, jonka käyttäjä väistämättä huomasi ennen karttanäkymän päivittymistä.

Tämän kaltaiset viiveet ovat tyypillisiä web-sovelluksille, mikä erottaa ne natiivisovelluksista, joissa merkittävä osa resursseista on tallennettu mobiililaitteen sisäiseen muistiin. Verkkoyhteyden välityksellä haettavat kuvat, ohjelmakoodi ja muu data ovat käytettävissä vasta, kun ne on ladattu kokonaan palvelimelta selaimeen. Kehittämistehtävässä hyödynnettiin ympäristön kuvailutietojen lataamisessa vapaaehtoisvoimin ylläpidettävää Overpass-palvelinta. Kyseisen palvelimen vasteajat olivat tavallisesti useita sekunteja, toisinaan jopa yli kymmenen sekuntia, jona aikana lataamattomat alueet näkyivät käyttäjän ympäristössä tyhjinä. Mikäli laitteessa ei ollut toimivaa verkkoyhteyttä, kutsut Overpass-rajapintaan epäonnistuivat, eikä karttanäkymään voitu lisätä uusia alueita.

Kehittämistehtävässä haluttiin kokeilla, kuinka progressiivisen web-sovelluksen ominaisuuksia voidaan hyödyntää sovelluksen latausaikojen ja käyttökokemuksen parantamiseksi. Kuinka karttanäkymään tarvittava data ja interaktiivisten kohteiden esittelyyn tarvittavat kuvatiedostot voitaisiin saada käyttöön nopeasti riippumatta palvelimen vasteajoista? Voitaisiinko progressiivisen web-sovelluksen offline-ominaisuuksia hyödyntää siten, että joitakin toimintoja voitaisiin tarjota käyttäjälle myös kokonaan ilman verkkoyhteyttä?

Progressiivisten web-sovelluksien ominaisuuksista tähän tarkoitukseen soveltuu CacheStorage-rajapinta eli mahdollisuus vaikuttaa yksityiskohtaisesti selaimen välimuistiin tallennettaviin tiedostoihin. Rajapinta mahdollistaa useiden nimettyjen välimuistien luomisen ja tiedon tallentamisen, hakemisen ja poistamisen välimuisteista. Service Worker -tiedoston avulla sovelluksen kehittäjä voi vaikuttaa ohjelmallisesti siihen, mitä selaimen välimuistiin tallennetaan ja millaisissa tilanteissa hyödynnetään välimuistiin tallennettuja tietoja. (Ater 2017, luku 3.)

Perinteinen tapa hyödyntää selaimen välimuistia on, että selain saa palvelimelta palautetun http-vastauksen mukana tiedon siitä, kuinka kauan kutakin resurssia voidaan käyttää selaimen välimuistista. Kyseisen ajan päätyttyä selain hakee jälleen tuoreen resurssin palvelimelta. Tämä voi toisinaan johtaa tilanteisiin, jossa esimerkiksi sivuston tyylitiedostosta käytetään vanhentunutta versiota, koska päivitetty tiedosto haetaan palvelimelta vasta määrätyn ajan kuluttua. Service Worker -tiedoston avulla kehittäjä voi laatia palvelimesta riippumattomia sääntöjä välimuistin käyttöön. (Hume 2017, luku 3.)

Kehittämistehtävässä hyödynnettiin CacheStorage-rajapintaa kahdessa eri tarkoituksessa. Ensinnäkin haluttiin tallentaa ensimmäisellä latauskerralla selaimen välimuistiin sovelluksen rungon muodostavat resurssit. Sovelluksen runko (engl. *application shell*) koostuu niistä HTML-, CSS- ja JavaScript-tiedostoista, joiden avulla on mahdollista näyttää sovelluksen käyttöliittymän peruselementit ilman dynaamista sisältöä. Näkymään voivat kuulua esimerkiksi ylä- ja alatunnisteet navigaatioineen. (Hume 2017, luku 2.)

Service Worker -tiedostoon voidaan myös lisätä tapahtumakäsittelijöitä sovelluksesta lähteville http-pyynnöille. Kehittämistehtävän kannalta mielenkiintoista on, että tämä mahdollistaa välimuistiin tallentamisen dynaamisesti http-pyynnön sisällöstä riippuen. (Hume 2017, luku 3.) Kehittämistehtävässä haluttiin tallentaa käyttäjän ympäristön kuvailutiedot selaimen välimuistiin, jolloin samoja alueita ei tarvitsisi hakea yhä uudelleen hitaalta Overpass-palvelimelta. Kuvailutietojen tallentaminen välimuistiin mahdollistaisi lisäksi kolmiulotteisen karttanäkymän näyttämisen ilman verkkoyhteyttä aiemmin vierailtujen alueiden osalta.

Välimuistin hyödyntämiseen on olemassa useita eri strategioita. Keskeinen kysymys on, käytetäänkö ensisijaisesti välimuistiin tallennettua dataa vai suositaanko verkon välityksellä haettuja ajan tasalla olevia resursseja. Jos pidetään tärkeänä, että sovellus tai sen osat toimivat myös ilman

verkkoyhteyttä, voidaan käyttää strategiaa, jossa resurssit haetaan aina ensin välimuistista. Jos kyseistä resurssia ei löydy välimuistista, se voidaan yrittää hakea verkkoyhteydellä palvelimelta. Tämä strategia soveltuu hyvin sovelluksen runkoon kuuluville resursseille, mutta sitä ei voida käyttää säännöllisesti päivittyville sisällöille kuten sosiaalisen median julkaisuille tai blogikirjoituksille. (Google Developers 2019b.)

Kehittämistehtävässä haluttiin varmistaa sovelluksen mahdollisimman sujuva toiminta ja natiivisovellusta muistuttava käyttökokemus, joten päätettiin hyödyntää strategiaa, jossa välimuistia pidetään ensisijaisena datan lähteenä. OpenStreetMapin dataan ei ole säännöllisesti odotettavissa merkittäviä päivityksiä, ja sekä ohjelmakoodin että interaktiivisiin karttakohteisiin liittyvä datan ja kuvaresurssien voidaan katsoa kuuluvan tiettyyn sovelluksen versioon. Nämä resurssit voidaan tyhjentää välimuistista tarpeen tullen uuden Service Worker -tiedoston asennuksen yhteydessä.

Kehittämistehtävässä Service Worker rekisteröitiin index.js-tiedostossa (katso kuvio 9). Ennen rekisteröintiä on syytä tarkistaa, tukeeko käytössä oleva selain Service Workereita. Mikäli tukea ei ole, jäävät progressiivisten web-sovellusten ominaisuudet hyödyntämättä. Kuten aiemmin on todettu, tämä ei saa estää sivuston tai sovelluksen käyttöä. Progressiivisten web-sovellusten on tarkoitus toimia vanhemmilla selaimilla normaalisti tavanomaisten verkkosivujen ja -sovellusten tapaan.

```
if ('serviceWorker' in navigator) {
    navigator.serviceWorker.register('/sw.js').then(() => {
        console.log('[index.js] Service Worker registered!');
    });
}
```

KUVIO 9. Service Workerin rekisteröinti sovelluksen index.js-tiedostossa.

Service Worker -tiedostoa ja CacheStorage-rajapintaa käytettäessä on tärkeää tuntea Service Workerin elinkaari. Kun Service Worker on rekisteröity, se siirtyy asennusvaiheeseen (*installing*). Onnistuneen asennuksen jälkeen Service Worker siirtyy tilaan *installed* eli asennettu. Seuraavaan vaiheeseen vaikuttaa, onko selaimessa jo asennettuna aikaisempi Service Worker samalta sivustolta. Mikäli selaimessa on aiempi Service Worker asennettuna, uusi Service Worker siirtyy odotustilaan (*waiting*) ja aktivoituu vasta, kun kaikki kyseisen sivuston avatut selainikkunat ja -välilehdet on suljettu. Jos aiempaa Service Workeria ei ole, uusi siirtyy välittömästi *activating*-tilan kautta

aktiiviseksi (*active*) Service Workeriksi. *Installed*- ja *activated*-vaiheisiin liittyvät *install*- ja *activate*-tapahtumat, joihin voidaan yhdistää tapahtumakäsittelijä. (Ater 2017, luku 4; Archibald 2019.)

Uusi aktiivinen Service Worker ei ota heti ladattua sivua hallintaansa. Oletuksena Service Worker vaikuttaa vain sivustoihin, jotka on ladattu aktiivisen Service Workerin kautta. Kehittäjä voi kuitenkin pakottaa ennen Service Workerin aktivoitumista avatut sivustot sen hallintaan kutsumalla selaimen Clients-rajapinnan *claim*-metodia. Tällöin kehittäjän on tiedostettava, että nämä sivustot on voitu avata ilman Service Workeria tai vanhan version ollessa voimassa. (Archibald 2019.)

Sovelluksen rungon eli *application shell*in muodostavat resurssit tallennetaan välimuistiin Service Workerin *install*-tapahtuman yhteydessä. Tapahtumakäsittelijässä voidaan käyttää tapahtuman *waitUntil*-metodia, joka estää Service Workerin asennusta etenemästä loppuun, kunnes *waitUntil*-metodista on palautettu selvitetty Promise. Mikäli asennustapahtumaan liitetyn tapahtumakäsittelijän ohjelmakoodi aiheuttaa virheen, ei kyseistä Service Workeria asenneta lainkaan. (Ater 2017, luku 4; Archibald 2019.)

Kehittämistehtävässä sovelluksen rungon muodostaneet resurssit lisättiin *preCache*-nimiseen taulukkomuuttujaan, joka lisättiin välimuistiin *install*-tapahtuman tapahtumakäsittelijässä (katso kuvio 10). Yhteys selaimen CacheStorage-rajapintaan saatiin globaalin *caches*-muuttujan avulla. Uusi välimuisti nimeltään ”static” luotiin CacheStoragen *open*-metodilla, joka palauttaa parametriksi annetun nimen perusteella löytyvän välimuistin tai luo uuden, mikäli nimellä ei ole aiemmin luotu välimuistia.

Yksittäiseen välimuistiin voidaan lisätä resursseja taulukkomuodossa välimuistin sisältämän *addAll*-metodin avulla. Kehittämistehtävässä resurssit lisätään välimuistiin asennustapahtuman *waitUntil*-metodin sisällä. Tämä varmistaa, että Service Worker siirtyy *installed*-vaiheeseen vain, jos *preCache*-taulukon sisältämien resurssien tallentaminen välimuistiin onnistuu ilman virheitä. Onnistuneen asennuksen jälkeen *preCache*-taulukon sisältämät resurssit ovat varmuudella käytettävissä välimuistista ilman verkkoyhteyttä.

```
self.addEventListener('install', (event) => {
    event.waitUntil(
        caches.open('static').then((cache) => {
            cache.addAll(preCache);
        })
    );
});
```

KUVIO 10. Sovelluksen rungon muodostavat resurssit ovat preCache-taulukossa, joka lisätään static-nimiseen välimuistiin.

Aktiivinen Service Worker mahdollistaa tapahtumakäsittelijän lisäämisen *fetch*-tapahtumalle. *Fetch*-tapahtuma käynnistyy, kun selaimesta lähetetään http-pyyntö. Välimuistin käytön kannalta *fetch*-tapahtuman tapahtumakäsittelijä on tärkeä, sillä se määrää, millaista strategiaa noudatetaan välimuistiin talletettujen resurssien käytössä. *Fetch*-tapahtumakäsittelijässä on myös mahdollista tallentaa välimuistiin sovelluksen käytön aikana noudettuja resursseja sekä dynaamisia http-pyyntöjä, joiden sisältö ei ole vielä tiedossa Service Workerin asennusvaiheessa. (Ater 2017, luku 4; Archibald 2019.)

Kehittämistehtävässä http-pyynnön tapahtumakäsittelijää hyödynnettiin palauttamalla haetut resurssit ensisijaisesti selaimen välimuistista. Mikäli pyydettyä resurssia ei ollut tallennettuna välimuistiin, se haettiin verkosta, palautettiin selaimelle ja tallennettiin samalla ”dynamic”-nimiseen välimuistiin (katso kuvio 11). Poikkeuksena kehittämistehtävässä oli Overpass-rajapintaan lähetetty *kill_my_queries*-pyyntö, jota ei koskaan tallennettu välimuistiin. Overpass-palvelimelle on asetettu rajoituksia samasta ip-osoitteesta lähetettyjen pyyntöjen määrälle. Toisinaan epäonnistuneet pyynnöt jäivät elämään palvelimelle, jolloin ne piti erikseen poistaa lähettämällä palvelimelle *kill_my_queries*-pyyntö.

```
self.addEventListener('fetch', (event) => {
    if (event.request.url.includes('kill_my_queries')) {
        event.respondWith(
            fetch(event.request)
        );
    } else {
        event.respondWith(
            caches.match(event.request).then((response) => {
                if (response) {
                    return response;
                }
                return fetch(event.request).then((response) => {
                    if (!response || response.status !== 200) {
                        return response;
                    }
                    const responseToCache = response.clone();
                    caches.open('dynamic').then((cache) => {
                        cache.put(event.request, responseToCache);
                    });
                    return response;
                });
            })
        );
    }
});
```

KUVIO 11. Fetch-tapahtuman tapahtumakäsittelijässä haetaan palautettava resurssi ensisijaisesti selaimen välimuistista.

Strategia, jossa kaikki haetut resurssit tallennettiin välimuistiin, mahdollisti niiden välittömän latauksen myöhemmillä käyttökerroilla. Välimuistiin tallennettujen resurssien osalta sovellus oli nyt käytettävissä myös offline-tilassa. Tällainen aggressiivinen välimuistin käyttö voi aiheuttaa myös haasteita. Selainten välimuistin määrä vaihtelee laite- ja selainkohtaisesti ja voi muuttua vapaan levytilan määrän muuttuessa. Mikäli välimuistiin tallennetaan paljon resursseja, voi sovelluksen käyttöön varattu välimuisti täyttyä. Selaimet tyhjentävät tällöin automaattisesti sovelluksen välimuisteja. Tämä ei varsinaisesti aiheuta ongelmia, sillä tarvittaessa resurssit haetaan uudelleen palvelimelta ja tallennetaan uudelleen välimuistiin. Kehittäjän on kuitenkin hyvä olla tietoinen sovelluksen käyttämän välimuistin määrästä ja välttää tarpeettomasti täyttämästä käyttäjän laitteen tallennustilaa. (Ater 2017, luku 4; Archibald 2019.)

Kun kaikki resurssit haetaan ensisijaisesti välimuistista, tulee kehittäjän itse huolehtia välimuistin tyhjentämisestä tilanteessa, jossa halutaan tarjota käyttäjälle resurssien päivitettyjä versioita pal-

velimelta. Esimerkiksi tyylitiedostojen päivittyessä on tiedostojen vanhat versiot poistettava välimuistista, jotta sovellus pystyy hakemaan uudet tyylitiedostot palvelimelta. Tällöin lähestytään tilannetta, jossa verkkosovelluksesta tarjotaan päivitettyjä versioita tietyin aikavälein sen sijaan, että käyttäjä saisi jokaisella latauskerralla päivitetyt resurssit käyttöönsä. (Ater 2017, luku 4; Archibald 2019.)

Selain asentaa Service Workerin uudelleen aina, kun Service Worker -tiedoston sisältö on muuttunut. Yksi tapa versioida verkkosovellus on nimetä välimuistit uudelleen Service Workerin uusissa versioissa (static_v1, static_v2 jne.). Tällöin lakataan käyttämästä vanhan Service Workerin hyödyntämiä välimuisteja. Hyvä käytäntö on poistaa vanhat välimuistit ohjelmallisesti uuden Service Workerin aktivoituessa, jolloin edellinen Service Worker on jo poissa käytöstä. Tätä varten *caches*-muuttuja tarjoaa *delete*-metodin (katso kuvio 12). (Ater 2017, luku 4; Archibald 2019.)

```
self.addEventListener("activate", function(event) {
    event.waitUntil(
      caches.keys().then(function(cacheNames) {
        return Promise.all(
          cacheNames.map(function(cacheName) {
            if (CACHE_NAME !== cacheName &&  cacheName.startsWith("static")) {
              return caches.delete(cacheName);
            }
          })
        );
      })
    );
});
```

KUVIO 12. Esimerkki static-välimuistin vanhojen versioiden poistamisesta. CACHE_NAME-vakiomuuttujaan on tallennettu välimuistin uusimman version nimi.

3.2.3 Verkkosovelluksen asentaminen aloitusnäytölle

Eräs natiivien mobiilisovellusten suosiota kasvattanut tekijä on ollut niiden mobiililaitteille optimoidut, visuaalisesti houkuttelevat käyttöliittymät sekä mobiililaitteen käyttöjärjestelmän kanssa saumattomasti limittyvä käyttökokemus. Verkkosovellusten käyttöliittymäsuunnittelussa on jo pitkään huomioitu myös mobiililaitteet ja mobiilikäyttöliittymä asetetaan yhä useammin ensisijaiseksi suhteessa muihin laitteisiin. Progressiivisen web-sovelluksen ominaisuuksien avulla voidaan verkkosovelluksissa tarjota vielä aikaisempaa sulavampi ja visuaalisesti houkuttelevampi käyttökokemus. (Hume 2017, luku 5.)

Kehittämistehtävässä sovelluksen perustoiminnallisuus muistuttaa jo sinällään joitakin käytössä olevia mobiilisovelluksia. 3D-objektien hyödyntämiseen ja kolmiulotteiseen karttanäkymään sekä karttanäkymän pyörittämiseen ja zoomaukseen kosketuseleiden avulla otettiin vaikutteita esimerkiksi aiemmin mainituista Pokemon GO - ja Harry Potter: Wizards Unite -viihdepeleistä.

Mobiililaitteen selaimessa ajettuna sovelluksen näkymään kuitenkin lisätään selaimen osoitepalkki ja mahdollisia muita valikkoja. Lisäksi selaimessa voi olla selaimen käyttöliittymään kuuluvia ominaisuuksia, jotka häiritsevät kosketuseleiden avulla toimivaa karttanäkymää. Tällaisia voivat olla esimerkiksi sivun lataaminen uudelleen ruudun yläreunasta alas vedettäessä tai selaimen osoitepalkin katoaminen sivustoa ylös vieritettäessä, mikä aiheuttaa myös sovelluksen näkymän uudelleensijoittumisen näytöllä. Tämän vuoksi päätettiin kehittämistehtävässä hyödyntää mahdollisuutta asentaa sovellus mobiililaitteen aloitusnäkymään ja avata sovellus koko ruudun näkymässä.

Progressiivisen web-sovelluksen visuaalisen ulkoasun kannalta keskeisessä roolissa on sovellukseen lisättävä manifest-tiedosto. Manifest-tiedosto mahdollistaa verkkosovelluksen asentamisen mobiililaitteen aloitusnäytölle ja antaa kehittäjälle mahdollisuuden muokata sovelluksen latausruutua sekä värimaailmaa. Tätä varten luodaan JSON-muodossa oleva tekstitiedosto, johon viitataan HTML-koodin head-tagin sisään kirjoitettavassa link-tagissa. (Hume 2017, luku 5.)

Verkkosovelluksen liittäminen aloitusnäytölle on kehittäjän kannalta toteutettavissa helposti. Prosessia on mahdollista ohjata ja sen eri vaiheisiin voidaan päästä käsiksi JavaScript-koodilla. Nykyaikaiset selaimet kuitenkin kysyvät tiettyjen ehtojen täyttyessä automaattisesti käyttäjältä, haluaako tämä lisätä kyseisen verkkosovelluksen laitteensa aloitusnäytölle. (Hume 2017, luku 5; LePage 2019.) Aloitusnäytölle asennettu verkkosovellus voidaan avata sille määritellystä pikakuvakkeesta, ja se on näkyvissä mobiililaitteiden käyttöliittymässä itsenäisenä käynnissä olevana sovelluksena. On tärkeää huomata, että sovelluksen lisääminen laitteen aloitusnäyttöön ei tarkoita, että sovelluksen tiedostoja tallennettaisiin laitteen muistiin tai että sovellus olisi automaattisesti käytettävissä ilman verkkoyhteyttä (MDN 2019b).

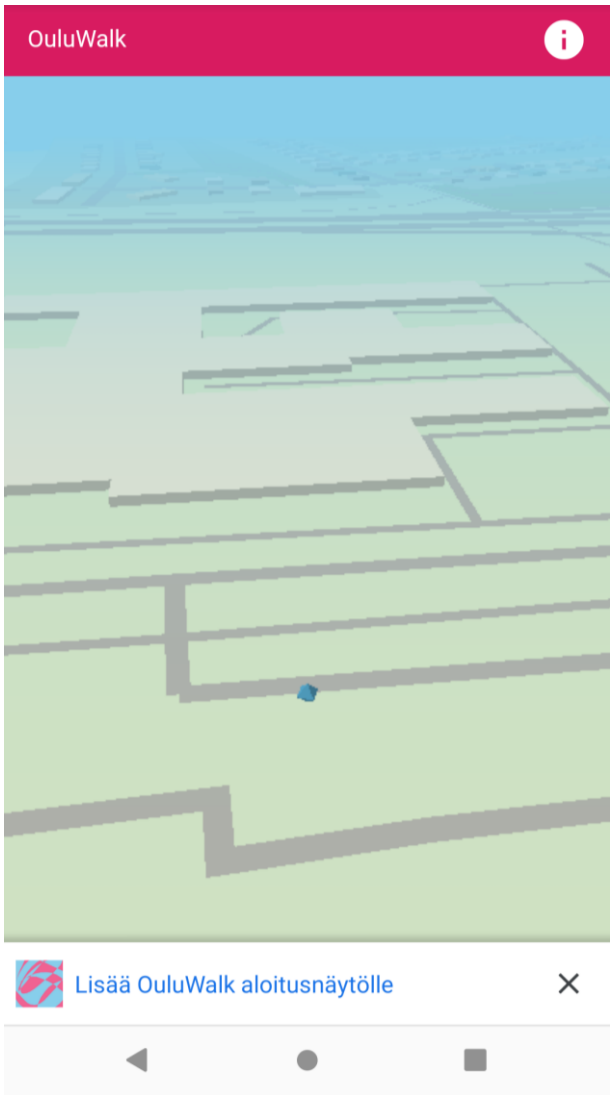
Jotta verkkosovellus voidaan asentaa laitteen aloitusnäytölle, tarvitaan manifest-tiedoston lisäksi Service Worker -tiedosto. Manifest-tiedostossa täytyy olla määriteltynä sovelluksen nimi, valikoima tiedostoja käytettäväksi sovelluksen kuvakkeina sekä sovellukselle valittu taustaväri, sovelluksen url-osoite ja asetus, joka määrittää, millaisessa näkymässä sovellus avataan aloitusnäkymästä. Service Worker -tiedoston asentaminen puolestaan edellyttää https-yhteyttä, ja Service Workerissa

täytyy olla tapahtumakäsittelijä selaimen *fetch*-tapahtumalle. Tällä pyritään varmistamaan, että aloitusnäytölle asennettavassa verkkosovelluksessa on toteutettu manifest.json-tiedoston lisäksi myös progressiivisen web-sovelluksen suorituskykyyn ja offline-ominaisuuksiin liittyviä ominaisuuksia. (LePage 2018.)

Kehittämistehtävässä sovellukseen lisättiin manifest-tiedosto, jossa määriteltiin sovelluksen nimi *name/short_name*, ladattava url-osoite *start_url* sekä sovelluksen värimaailma *background_color* ja *theme_color*. *Icons*-taulukossa määriteltiin sovelluksen käytössä olevat eri kokoiset kuvaketiedostot ja kenttä *display* määrittelee, kuinka aloitusnäytöstä avattu sovellus näytetään käyttäjän laitteessa (katso kuvio 13). Näillä perustiedoilla saatiin selain tarjoamaan mahdollisuutta asentaa sovellus laitteen aloitusnäytölle (katso kuvio 14).

```
{
    "name"          : "OuluWalk",
    "short_name"    : "OuluWalk",
    "start_url"     : "/index.html",
    "scope"         : ".",
    "display"       : "fullscreen",
    "background_color" : "#87CEEB",
    "theme_color"   : "#f06292",
    "description"   : "Tutustu Oulun kaupunkikuvan tunnettuihin kohteisiin.",
    "dir"           : "ltr",
    "lang"          : "fi-FI",
    "orientation"   : "portrait-primary",
    "icons"         : [
        {
            "src"   : "/assets/images/icons/icon-48x48.png",
            "type"  : "image/png",
            "sizes" : "48x48"
        },
        {
            "src"   : "/assets/images/icons/icon-128x128.png",
            "type"  : "image/png",
            "sizes" : "128x128"
        },
        {
            "src"   : "/assets/images/icons/icon-192x192.png",
            "type"  : "image/png",
            "sizes" : "192x192"
        },
        {
            "src"   : "/assets/images/icons/icon-512x512.png",
            "type"  : "image/png",
            "sizes" : "512x512"
        }
    ]
}
```

KUVIO 13. Kehittämistehtävän sovellukseen lisätty manifest-tiedosto.



KUVIO 14. Selain ehdottaa käyttäjälle sovelluksen asentamista laitteen aloitusnäytölle.

Aloitusnäkymän kuvakkeesta avatun sovelluksen ulkoasun ja käyttökokemuksen kannalta olennaisia asetuksia ovat manifest-tiedoston kohdat *display* ja *orientation*. *Display*-asetuksen arvo määrää, kuinka mobiililaitteen käyttöjärjestelmä esittää sovelluksen. Vaihtoehdot kattavat erilaiset näkymät verkkoselainnäkymästä (*browser*) koko ruudun näkymään (*fullscreen*). Ääripäiden väliin jäävät *standalone*-asetus, jossa sovellus esitetään kuten laitteeseen asetetut natiivisovellukset. Tämä näkymä saattaa sisältää erilaisia natiiveja käyttöliittymäelementtejä kuten tilarivi tai paluunäppäin. *Minimal-ui* -asetus muistuttaa koko ruudun näkymää, mutta voi sisältää erilaisia navigointiin liittyviä käyttöliittymäelementtejä ja esimerkiksi sovelluksen verkko-osoitteen. (Hume 2017, luku 5.)

Kehittämistehtävän sovelluksessa käyttäjä ei siirry sivulta toiselle, joten siinä ei ole tarvetta näkymään lisätyille navigaatioelementeille. Sen sijaan haluttiin varmistaa, että interaktiivinen karttanäkymä saa käyttöönsä mahdollisimman suuren tilan. Tämän vuoksi *display*-asetukseksi valittiin *fullscreen* eli koko ruudun näyttö.

Orientation-asetuksella voidaan määrittää, kuinka sovellus reagoi mobiililaitteen näytön asentoon. Mahdollisia arvoja ovat *any*, *natural*, *portrait* ja *landscape.* Asetuksella *any* sovellus reagoi vapaasti laitteen eri asentoihin ja vaihtaa näkymää tarvittaessa pysty- ja vaakasuunnan välillä. *Natural*-asetus valitsee ruudun asennon laitteen käyttöjärjestelmän oletussuunnan perusteella. *Portrait*- ja *landscape*-asetuksilla ruutu voidaan lukita joko pysty- tai vaakasuuntaan. Näistä asetuksista on lisäksi valittavissa *primary* ja *secondary*-versiot, joista esimerkiksi *portrait-secondary* tarkoittaa älypuhelimessa tavallisesti pystysuuntaista näyttöä ylösalaisin käännettynä. (Lamouri, M., Cáceres, M. & Herman, J. 2019; Ater 2017, luku 9.)

Kehittämistehtävän sovellus on suunnattu älypuhelimille tai vastaaville mobiililaitteille, ja sovelluksen suunta haluttiin lukita näytössä pystysuuntaan. 3D-karttanäkymä on suunniteltu pystysuuntaiselle näytölle, ja se vääristyy jonkin verran vaakasuuntaan käännettäessä. *Orientation*-asetukseksi valittiin *portrait-primary*, joka esimerkiksi älypuhelimessa tarkoittaa tavallisesti pystyasennossa olevaa näyttöä, jossa laitteen tai käyttöjärjestelmän navigaatio asettuu ruudun alareunaan. Lukitsemalla näyttö yhteen suuntaan voidaan varmistaa, ettei karttanäkymä käänny ylösalaisin kesken käytön.

4 TULOKSET JA JOHTOPÄÄTÖKSET

Kehittämistehtävän tavoitteena oli rakentaa runko sijaintiperusteiselle web-sovellukselle, jossa käyttäjän ympäristöstä luotuun kolmiulotteiseen näkymään voidaan lisätä interaktiivisia kohteita. Karttanäkymä rakennetiin hyödyntämällä laitteen sijaintitietoa sekä selaimen WebGL-rajapintaa, joka mahdollistaa 3D-grafiikan piirtämisen verkkoselaimessa. Sijaintitiedon hyödyntäminen ja 3D-grafiikka ovat esimerkkejä teknologioista, joihin on aiemmin totuttu natiiveissa mobiilisovelluksissa. Kehittämistehtävän sovelluksen käyttökokemusta haluttiin viedä yhä lähemmäs natiivisovellusten kokemusta lisäämällä siihen progressiivisen web-sovelluksen ominaisuuksia.

Selaimen Geolocation-rajapinta tarjoaa hyvin yksinkertaisen tavan hyödyntää laitteen sijaintitietoa. Yksinkertaisten metodien ja asetusolioiden avulla kehittäjä saa helposti käyttöönsä valmiiksi muotoiltua dataa laitteen sijainnista, suunnasta ja jopa nopeudesta. Geolocation-rajapinta on korkean tason abstraktio, joka hakee sijaintitiedon parhaista tarjolla olevista lähteistä, ja nykyaikaiset selaimet pyytävät automaattisesti käyttäjän suostumusta sijaintitiedon hyödyntämiseen.

Sovellusten kehittäjien ja suunnittelijoiden pohdittavaksi jää lähinnä se, mihin käyttäjien sijaintitietoa on järkevää hyödyntää ja millaisia yksityisyyteen liittyviä näkökohtia tulee huomioida sijaintietoja hyödynnettäessä ja tallennettaessa. On myös järkevää pohtia, kannattaako verkkoselaimen tarjoaman sijaintitiedon varaan rakentaa sovelluksia, jotka edellyttävät sijaintitiedon suurta tarkkuutta, sillä tarkkuus vaihtelee laitekohtaisesti. Tarkkuus on luonnollisesti paras mobiililaitteissa ja älypuhelimissa, jotka hyödyntävät paikannuksessa satelliittijärjestelmiä, mutta myös niiden välillä on paljon eroja.

Sijaintitiedon hyödyntämisen osalta kehittämistehtävässä jäi puutteelliseksi mahdollisten virheilmoitusten näyttäminen käyttäjälle. Käyttäjä voi kieltää sijaintitiedon hyödyntämisen, laitteen paikannus voi olla pois päältä käyttöjärjestelmän asetuksista tai paikannuksessa voi tapahtua virhe. Näissä tilanteissa käyttäjän pitäisi saada tieto siitä, miksi sovellus ei toimi halutulla tavalla. Kehittämistehtävässä karttanäkymä jää tyhjäksi, mutta käyttäjä ei saa tietoa, mistä tämä johtuu.

WebGL-rajapinnan käyttö on mielekästä sitä varten luotujen JavaScript-kirjastojen avulla. Kehittämistehtävässä käytetty three.js-kirjasto on selkeä, ja se tarjoaa valmiita olioita ja metodeja sekä yksinkertaisten 3D-kappaleiden että valmiiden 3D-mallien hyödyntämiseen. Kehittämistehtävässä

3D-näkymään yhdistettiin kosketuseleillä tapahtuva navigaatio sekä kosketukseen reagoivia interaktiivisia kohteita. Näin luotiin verkkosovellukseen käyttöliittymä, joka muistutti vastaavanlaiseen karttanäkymään perustuvia natiivisovelluksia. Yksinkertaisia peruskappaleita hyödynnettäessä selaimen suorituskyky riitti erinomaisesti kolmiulotteisen karttanäkymän esittämiseen.

Perustoiminnallisuuden toteuttamisen jälkeen kehittämistehtävään lisättiin progressiivisen web-sovelluksen ominaisuuksia. Tavoitteena oli edelleen kuroa umpeen natiivisovelluksen ja selaimessa ajettavan web-sovelluksen välisiä eroja käyttökokemuksessa. Manifest-tiedoston ja muutaman kuvakkeille varatun kuvatiedoston avulla saatiin sovellus asennettua mobiililaitteen aloitusnäyttöön. Aloitusnäytön kuvakkeesta avattaessa kolmiulotteinen karttanäkymä avattiin koko ruudun näytölle ilman mitään viitteitä selainnäkymästä. Karttanäkymälle ja siinä tapahtuvalle navigaatiolle jäi aiempaa suurempi tila näytöllä, eivätkä selaimen omat kosketuseleisiin perustuvat toiminnot häirinneet käyttökokemusta.

Service Workerin rekisteröinnin avulla voitiin sovelluksen runko sekä käytön aikana ladatut resurssit tallentaa selaimen välimuistiin, mikä mahdollisti ennen kaikkea ajoittain heikosti toimineeseen Overpass-palvelimeen liittyvän viiveen vähentämisen. Samalla saatiin sovellus toimimaan aikaisemmin ladattujen resurssien ja karttadatan osalta kokonaan ilman verkkoyhteyttä. Aiemmin vierailtujen alueiden osalta käyttökokemus muuttui merkittävästi, kun niiden lisääminen näkymään kesti muutamien sekuntien sijasta joitakin kymmeniä millisekunteja.

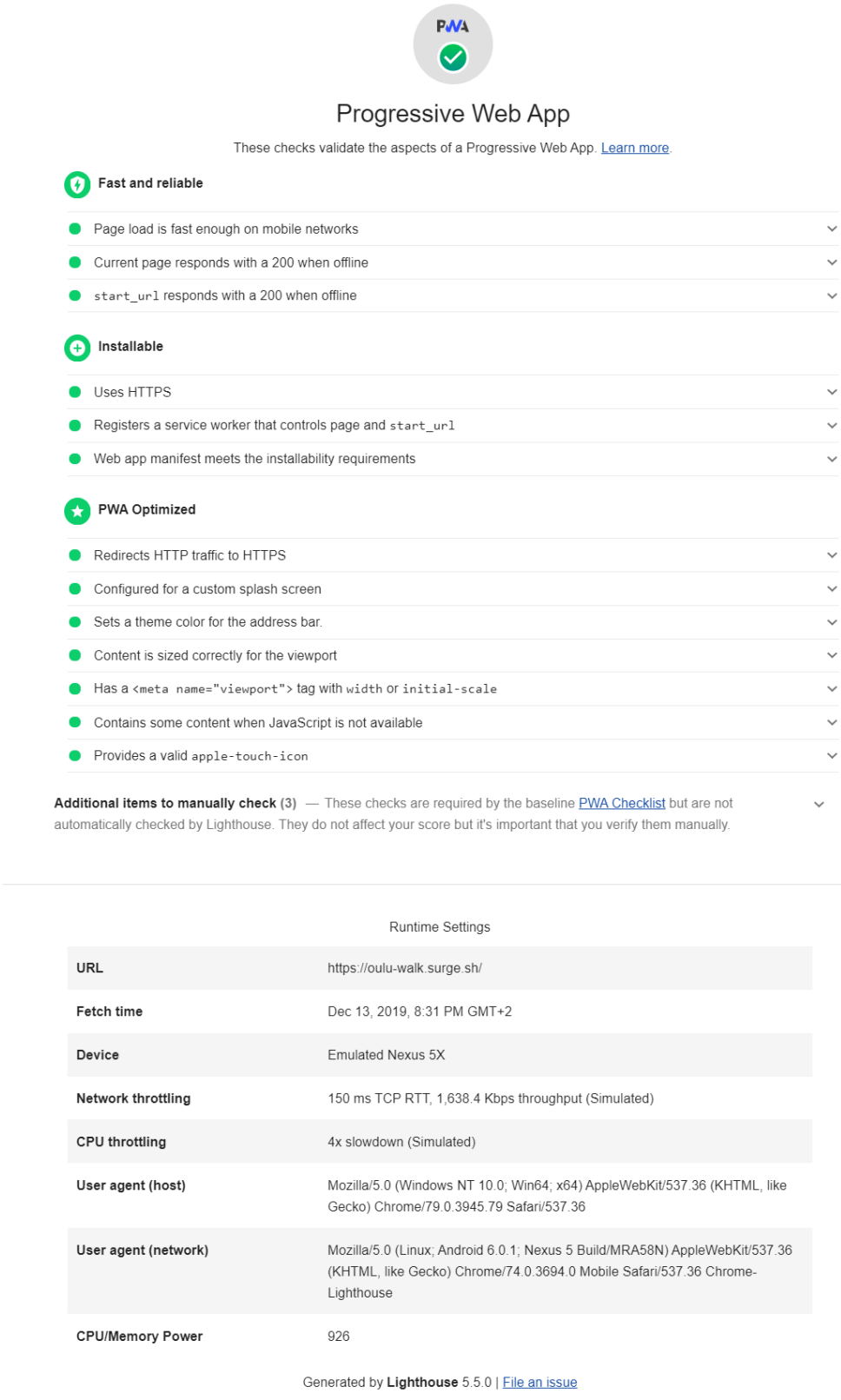
Progressiivisen web-sovelluksen ominaisuuksien toteuttamisen jälkeen kehittämistehtävä auditoitiin uudelleen Googlen Lighthouse -työkalulla. Manifest-tiedoston ja Service Workerin lisäämisen jälkeen oletuksena oli, että tulos parantuisi merkittävästi aikaisemmasta auditoinnista. Kehittämistehtävässä toteutettu sovellus saikin nyt hyväksytyn tuloksen kaikista kolmestatoista auditointikohteesta (katso kuvio 15).

Osiossa *Fast and reliable* sovellus hylättiin aiemmin offline-ominaisuuksien testissä. Service Workerin rekisteröinnin ja välimuistiin tallennettujen resurssien avulla sovellus vastasi http koodilla 200 (OK) nyt myös ilman-verkkoyhteyttä. Kohteiden ilmestyminen karttanäkymään toki edellyttää, että käyttäjä on vieraillut samalla alueella aikaisemmin ja ladannut ympäristön kuvailutiedot toimivalla verkkoyhteydellä. Samoin uusien alueiden lataaminen vaatii luonnollisesti verkkoyhteyttä.

Installable-osiossa tarkastellaan niitä edellytyksiä, jotka mahdollistavat sovelluksen asentamisen laitteen aloitusnäytölle. Service Workerin lisääminen ja oikein konfiguroitu manifest-tiedosto mahdollistivat myös tässä osiossa kaikkiin auditointikohteisiin hyväksytyn tuloksen. *PWA Optimized* -osion hyväksytyt auditointikohteet liittyivät muun muassa sovelluksen responsiivisuuteen, teemavärien ja ikonien asetuksiin sekä sovelluksen tarjoamiseen vain https-yhteydellä.

Kehittämistehtävässä ei toteutettu progressiivisten web-sovellusten joitakin ominaisuuksia kuten push-ilmoituksia tai taustasynkronointia. Tämä oli kuitenkin perusteltua, sillä kehittämistehtävässä ei ollut säännöllisesti päivittyvää sisältöä eikä käyttäjällä ollut mahdollisuutta tallentaa tai lähettää dataa sovelluksessa. Push-ilmoitukset ja taustasynkronointi olisivat vaatineet uusien ominaisuuksien lisäämistä kehittämistehtävään, ja niiden lisääminen olisi kasvattanut kehittämistehtävän liian laajaksi.

Monet kehittämistehtävässä hyödynnetyistä teknologioista ovat täysipainoisesti hyödynnettävissä vain viimeaikaisilla selaimilla, ja eri selainvalmistajat ja käyttöjärjestelmät tukevat eri tavoin progressiivisten web-sovellusten ominaisuuksia. Progressiivista web-sovellusta kehitettäessä tuleekin pitää mielessä, että pohjimmiltaan kyse on perinteisestä verkkosivusta tai -sovelluksesta. Tavoitteena on, että sovelluksen tai sivuston perustoiminnot ovat käytettävissä myös vanhemmilla selaimilla ja että progressiivisen web-sovelluksen ominaisuuksia hyödynnetään, mikäli selain tukee niitä. Tämä mahdollistaa uusien ominaisuuksien lisäämisen helposti myös aiemmin rakennettuihin sivustoihin ja sovelluksiin.



KUVIO 15. Google Lighthouse -auditoinnin tulokset manifest-tiedoston ja Service Workerin toteutamisen jälkeen.

5 POHDINTA

Kehittämistehtävässä toteutettiin sijaintiperusteinen web-sovellus, johon lisättiin progressiivisen web-sovelluksen ominaisuuksia. Aihe oli ajankohtainen, sillä suhtautuminen natiiveihin mobiilisovelluksiin vaikuttaa muuttuneen viimeisten vuosien aikana. Samaan aikaan uudet web-teknologiat ja viimeisimmät verkkoselaimet mahdollistavat aiemmin ainoastaan natiivisovelluksissa tavattujen ominaisuuksien lisäämisen web-sovelluksiin. Selaimessa voidaan nyt hyödyntää käyttäjän sijaintia ja mobiililaitteiden laitteita kuten kameraa ja mikrofonia. 2D- ja 3D-grafiikan esittäminen selaimessa on vaivatonta erilaisten JavaScript-kirjastojen avulla. Progressiivisen web-sovelluksen ominaisuudet mahdollistavat sovelluksen asentamisen laitteen aloitusnäytölle ja valittujen toimintojen käyttämisen ilman verkkoyhteyttä.

Kaikki selaimet ja käyttöjärjestelmät eivät vielä tue progressiivisten web-sovellusten viimeisimpiä ominaisuuksia. Tämä teki kehittämistehtävästä entistä mielenkiintoisemman, sillä se osoitti, että aihe oli osuvasti valittu ja ikään kuin teknologisen kehityksen aallonharjalla. Samalla täytyy tiedostaa, että kokeelliset web-teknologiat ja tässä opinnäytetyössä käytetyt selainrajapinnat voivat muuttua merkittävästi lähivuosina. Tämä tosin koskee teknologista kehitystä laajemminkin. Kehittämistehtävässä toteutetun sovelluksen kannalta on mielenkiintoista, kuinka immersiivinen web eli lisätyn todellisuuden ja virtuaalitodellisuuden sisällöt kehittyvät verkkoselaimissa lähitulevaisuudessa. Joitakin sijaintiperusteisia lisätyn todellisuuden sovelluksia on nähty natiivisovellusten joukossa, ja myös kehittämistehtävässä toteutettua sovellusta olisi mielenkiintoista kehittää siihen suuntaan.

Progressiivisten web-sovellusten periaatteista ja teknologioista on saatavilla hyvin tietoa. Opinnäytetyön alkuvaiheessa haaveenani oli, että kehittämistehtävän toteuttaminen auttaisi jatkossa arvioimaan tapauskohtaisesti nykyaikaisen web-sovelluksen etuja ja haasteita suhteessa natiiveihin mobiilisovelluksiin. Tämän suhteen ei yhden opinnäytetyön ja kehittämistehtävän aikana kertynyt kokemus nähdäkseni ole riittävä. Natiivi- ja web-sovellusten vertailu on monisyinen kysymys ja teknologisten näkökulmien lisäksi siihen liittyy paljon liiketaloudellisia ja markkinointiin liittyviä näkökohtia, joiden läpikäyminen olisi vähintäänkin yhden opinnäytetyön laajuinen työ.

Kehittämistehtävän kannalta progressiivisten web-sovellusten lisäämistä sijaintiperusteiseen sovellukseen voidaan pitää kuitenkin onnistuneena. 3D-karttanäkymän ja siihen pohjautuvan navigaation rakentaminen selaimelle onnistui hyvin. Kehittämistehtävässä tehty työ voisi olla perustana

mielenkiintoisille, aikaisemmasta poikkeaville sijaintiperusteisille verkkosovelluksille. Sovelluksen käyttökokemuksen kannalta manifest-tiedoston ja Service Workerin mahdollistamat ominaisuudet kuten sovelluksen avaaminen aloitusnäytöstä koko ruudun näkymään ja karttadatan tallentaminen välimuistiin olivat keskeisessä asemassa. Siten voidaan pitää perusteltuna progressiivisten web-sovellusten ominaisuuksien yhdistämistä kehittämistehtävään.

LÄHTEET

Archibald, J. 2019. The Service Worker Lifecycle. Web Fundamentals, Google Developers. Viitattu 13.12.2019, https://developers.google.com/web/fundamentals/primers/service-workers/lifecycle.

Ater, T. 2017. Building Progressive Web Apps. Sebastopol: O'Reilly Media.

Caballero, L. 2011. An Introduction to WebGL – Part 1. Dev.Opera. Opera Software AS. Viitattu 6.11.2019, https://dev.opera.com/articles/introduction-to-webgl-part-1.

Can I use… 2019. Web Workers. Can I use… Viitattu 8.11.2019, https://caniuse.com/#feat=web-workers.

Google Developers 2019. Lighthouse. Tools for Web Developers, Google Developers. Viitattu 13.11.2019, https://developers.google.com/web/tools/lighthouse.

Google Developers 2019b. Caching Files with Service Worker. Progressive Web Apps Training, Google Developers. Viitattu 12.12.2019, https://developers.google.com/web/ilt/pwa/caching-files-with-service-worker.

Green, I. 2012. Web Workers. Sebastopol: O'Reilly Media.

Gustafson, A. 2017. Yes, That Web Project Should Be a PWA. A List Apart. Viitattu 3.6.2019, http://alistapart.com/article/yes-that-web-project-should-be-a-pwa.

Gustafson, A. 2018. Native And PWA: Choices, Not Challengers! Smashing Magazine. Viitattu 3.6.2019, https://www.smashingmagazine.com/2018/02/native-and-pwa-choices-not-challengers.

Hellerick 2020. Division of the Earth into Gauss-Krueger zones – Globe (CC BY-SA 3.0), Wikipedia. Viitattu 26.10.2020, https://en.wikipedia.org/wiki/Longitude#/media/File:Division_of_the_Earth_into_Gauss-Krueger_zones_-_Globe.svg.

Holdener, A. T. 2011. HTML5 Geolocation. Sebastopol: O'Reilly Media.

Hume, D. 2017. Progressive Web Apps. Sebastopol: O'Reilly Media.

Lamouri, M., Cáceres, M. & Herman, J. 2019. The Screen Orientation API. World Wide Web Consortium. Viitattu 9.12.2019, https://www.w3.org/TR/screen-orientation.

LePage, P. 2018. Discover what it takes to be installable. web.dev, Google Developers. Viitattu 13.11.2019, https://web.dev/discover-installable.

LePage, P. 2019. Changes to Add to Home Screen Behavior. Google Developers. Viitattu 13.11.2019, https://developers.google.com/web/updates/2018/06/a2hs-updates.

MDN, 2019. Using Web Workers. MDN web docs. Viitattu 8.11.2019, https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers.

MDN 2019b. Add to Home screen. MDN web docs. Viitattu 13.11.2019, https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Add_to_home_screen.

NOAA 2019. Latitude/Longitude Distance Calculator. National Hurricane Center and Central Pacific Hurricane Center. National Oceanic and Atmospheric Administration. Viitattu 7.11.2019. https://www.nhc.noaa.gov/gccalc.shtml.

OpenStreetMap 2017. About OpenStreetMap. OpenStreetMap Wiki. Viitattu 6.11.2019, https://wiki.openstreetmap.org/wiki/About_OpenStreetMap.

OpenStreetMap 2019a. Copyright and License. Viitattu 6.11.2019, https://www.openstreetmap.org/copyright.

OpenStreetMap 2019b. Overpass_API. OpenStreetMap Wiki. Viitattu 6.11.2019, https://wiki.openstreetmap.org/wiki/Overpass_API.

Patro, N. 2018. Choose the best – Native App vs Hybrid App. Medium. Viitattu 13.12.2019, https://codeburst.io/native-app-or-hybrid-app-ca08e460df9.

Russell, A. 2015. Progressive Web Apps: Escaping Tabs Without Losing Our Soul. Medium. Viitattu 3.6.2019, https://medium.com/@slightlylate/progressive-apps-escaping-tabs-without-losing-our-soul-3b93a8561955.

Veness, C. 2019. Calculate distance, bearing and more between Latitude/Longitude points. Movable Type Scripts. Viitattu 7.11.2019, https://www.movable-type.co.uk/scripts/latlong.html.

Wikipedia 2019. Karjasillan kirkko. Wikipedia, Vapaa tietosanakirja. Viitattu 4.10.2020, https://fi.wikipedia.org/wiki/Karjasillan_kirkko.