PAIKKATIETORAJAPINTOJEN JA KARTTAKIRJASTOJEN HYÖDYNTÄMINEN REACTJS-SOVELLUKSISSA

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus, Hämeenlinnan korkeakoulukeskus
kevät, 2021

Nestori Metsäranta

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus Tiivistelmä

Hämeenlinnan korkeakoulukeskus

Tekijä	Nestori Metsäranta	Vuosi 2021
Työn nimi	Paikkatietorajapintojen ja karttakirjastojen hyödyntäminen ReactJS-sovelluksissa	
Ohjaajat	Lasse Seppänen	

TIIVISTELMÄ

Opinnäytetyön aiheena oli tutkia, miten paikkatietorajapintoja ja karttakirjastoja hyödynnetään käytännössä ReactJS-sovelluksissa. Tausta työlle nousi kasvavasta tarpeesta dokumentoida avoimen tiedon käyttötarkoituksia jokapäiväisessä elämässä.

Opinnäytetyö on toiminnallinen, ja sen aikana kehitettiin esimerkkisovelluksia hyödyntäen ReactJS:ää ja erilaisia karttakirjastoja, kuten Mapbox GL:lää ja Leafletia. Esimerkkisovellusten jälkeen tarkasteltiin, miten WFS-rajapintaa hyödynnetään käytännössä. Opinnäytetyön teoreettisessa osuudessa määritellään työn kannalta keskeiset käsitteet, kuten edellä mainitut WFS- ja WMS-rajapinnat, karttakirjastot ja ReactJS. Opinnäyteyön teoriapohja keskittyykin pitkälti rajapintatekniikoiden ja karttakirjastojen esittelyyn ja niiden taustoihin, kun taas käytännön osuudessa esitellään, mitkä karttakirjastot soveltuvat parhaiten rajapintojen hyödyntämiseen.

Opinnäytetyö tehtiin tukemaan kirjoittajan omaa ammatillista kehittymistä, ja sen tuloksena esimerkkisovellusten lisäksi oli ratkaisuehdotus tulevien paikkatietorajapintasovelluksien kehittämiseen karttakirjastojen osalta. Todettiin, että React-Leaflet oli kaikin puolin helppokäyttöisin ja parhaiten dokumentoitu karttakirjasto opinnäytetyössä esitellyistä ratkaisuista.

Avainsanat	karttakirjasto, Paikkatietorajapinta, WFS, WMS, ReactJS, Leaflet, Mapbox GL, OpenLayers
Sivut	31 sivua ja liitteitä 1 sivu

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Hämeenlinna University Centre

Abstract

Author Nestori Metsäranta Year 2021

Subject Utilization of spatial data interfaces and map libraries in ReactJS-applications

Supervisors Lasse Seppänen

ABSTRACT

The topic of the thesis was to study how spatial data interfaces and map libraries are utilized in practice in ReactJS applications. Background for the work arose from the growing need to document the uses of open information in everyday life.

The thesis is functional, during which example applications were developed using ReactJS and various map libraries such as Mapbox GL and Leaflet. After the example applications were finished, the practical application of the WFS interface was examined. The theoretical part of the thesis defines the key concepts for the work, such as the above-mentioned WFS and WMS interfaces, map libraries and ReactJS. The theoretical basis of the thesis largely focuses on the introduction of interface technologies and map libraries and their backgrounds, while the practical part presents which map libraries are best suited for the utilization of spatial data interfaces.

The thesis was done to support the author's own professional development, and in addition to the example applications, the result of the thesis was a solution proposal for the development of future spatial data interface applications concerning map libraries. It was found that React-Leaflet was the easiest and best documented map library of the solutions presented in the thesis.

Keywords map library, spatial data interface, WFS, WMS, ReactJS, Leaflet, Mapbox GL, OpenLayers

Pages 31 pages and appendices 1 page

Sanasto

INSPIRE	Infrastructure for Spatial Information in the European Community
ISO	International Organization for Standardization
OGC	Open Geospatial Consortium
OSM	Open Street Map
PWA	Progressive Web Application
WFS	Web Feature Service
WMS	Web Map Service
WMTS	Web Map Tile Service

Sisällys

Kuvat, komennot ja ohjelmakoodit

Liitteet

1 Johdanto

Lähivuosina on kiinnitetty enemmän huomiota internetistä saatavan tiedon laatuun ja sen avoimuuteen. Avoin tieto on tärkeää, sillä sen pohjalta pystytään tekemään laadukkaampia päätöksiä, jotka koskettavat meitä kaikkia. Joet, järvet, vuoret ja laaksot eivät tunne valtioiden rajoja, joten tiedon tulee olla yhteensopivaa, laadukasta ja kaikkien saatavilla, organisaatiosta ja maasta riippumatta. (Maanmittauslaitos, n.d.)

Tätä tärkeää asiaa ajaa EU-tasolla Euroopan komission direktiivi INSPIRE. Lyhykäisyydessään se ohjaa EU:n jäsenmaita rakentamaan yhteisiä ja yhtenäisiä paikkatietoinfrastruktuureita. Direktiivi on tullut voimaan jo 2007, ja sen toimeenpanon on tarkoitus kestää vuoden 2021 joulukuulle. (Maanmittauslaitos, n.d.)

Yhtenäisiin paikkatietoinfrastruktuureihin liittyy standardeja, jotta niitä voidaan kehittää yhteisesti. Näistä työn kannalta tärkeimmät ovat WFS- ja WMS-rajapinnat, joiden kautta mm. ohjelmistokehittäjät pystyvät rakentamaan ja kehittämään sovelluksia, jotka palvelevat avoimen tiedon arvoja.

Opinnäytetyön tarkoituksena onkin ruohonjuuritasolla selvittää, miten avointa dataa hyödynnetään käytännössä, moderneissa ReactJS-sovelluksissa. Toiminnallisessa osuudessa pureudutaan ensin paikkatietorajapinta-sovelluksen kivijalkaan, eli karttakirjastoihin. Tämän jälkeen tutkitaan, miten avoin tieto saadaan näkymään kartalle. Työtä lähdetään ratkomaan seuraavien kysymyksien avulla:

1. Mitä paikkatietorajapinnat ovat?

2. Mikä karttakirjasto olisi yhteensopivin Reactin kanssa?

3. Miten karttakirjastoja ja paikkatietorajapintoja hyödynnetään käytännössä ReactJS-sovelluksissa?

2 Avoimet paikkatietorajapinnat ja standardit

Tässä luvussa tutustutaan opinnäytetyön kannalta keskeisimpiin käsitteisiin, mm. WFS- ja WMS-paikkatietorajapintoihin. Tietoperustassa avataan myös muita tärkeitä käsitteitä, kuten INSPIRE-direktiivi ja OGC-yhteisö.

2.1 WFS

Web Feature Service eli WFS tarjoaa rajapinnan, jonka kautta maantieteellisiä tietoja luodaan, muokataan ja vaihdetaan internetissä. WFS tarjoaa suoran ja tarkan pääsyn maantieteellisiin tietoihin ja niiden ominaisuuksiin ilman välikäsiä. (OGC, n.d.)

WFS on kansainvälinen standardi, joka määrittelee hakutoiminnot, kyselytoiminnot, lukitustoiminnot, tapahtumatoiminnot ja toiminnot tallennettujen, parametrisoitujen kyselylausekkeiden hallitsemiseksi. Hakutoiminnot mahdollistavat sen, että palvelu pystyy kuulustelemalla määrittelemään palvelun ominaisuudet ja hakemaan sovelluksen mallin, joka määrittelee ne toiminnot, joita kyseinen palvelu tarjoaa. (OGC, n.d.)

Kyselytoiminnot mahdollistavat toiminnon ominaisuuksien tai arvojen hakemisen taustalla olevasta tietovarastosta rajoitusten, jotka asiakasohjelma määrittelee, pohjalta. Lukitustoiminnot puolestaan sallivat yksinoikeuden käyttää poisto- ja/tai muokkaustoimintoja toiminnoille. (OGC, n.d.)

Tapahtumatoiminnot mahdollistavat taustalla olevasta tietovarastosta toimintojen tai tietojen poistamisen, korvaamisen, luomisen ja muuttamisen. Tallennettujen kyselytoimintojen avulla käyttäjät pystyvät luomaan, pudottamaan, listaamaan ja kuvailemaan parametrisoituja kyselylausekkeita, jotka palvelin on tallentanut ja joita pystytään käyttämään uudestaan toistuvasti eri parametriarvoilla. (OGC, n.d.)

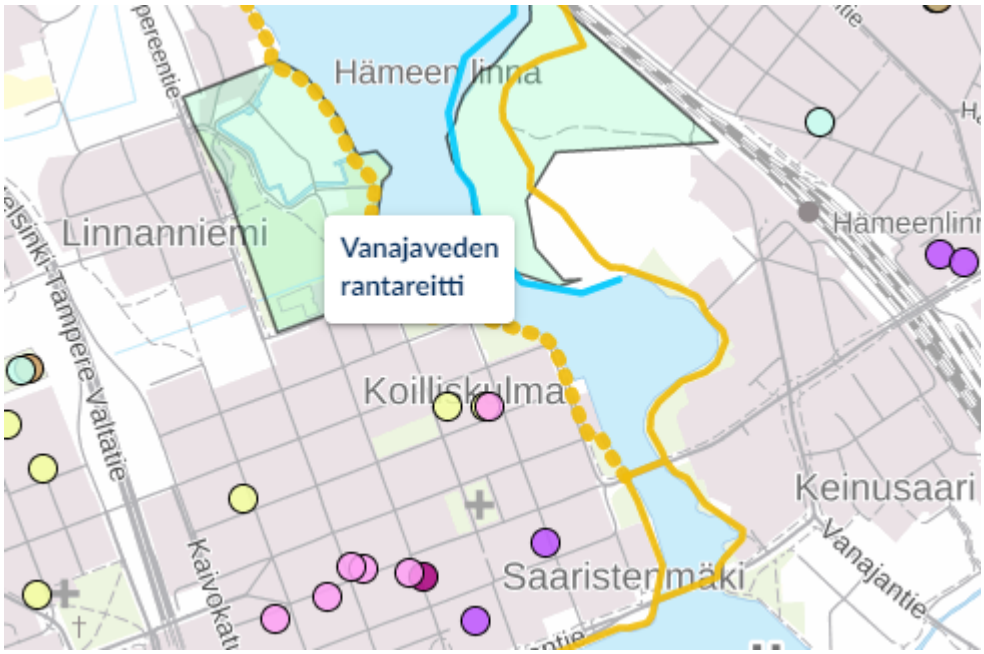
WFS-standardi määrittelee yksitoista toimintoa, jotka listattu alla:

GetCapability (hakutoiminto)
DescribeFeatureType (hakutoiminto)
GetPropertyValue (kyselytoiminto)
GetFeature (kyselytoiminto)
GetFeatureWithLock (kysely- ja lukitustoiminto)
LockFeature (lukitustoiminto)
Transaction (tapahtumatoiminto)
CreateStoredQuery (tallennettu kyselytoiminto)
DropStoredQuery (tallennettu kyselytoiminto)
ListStoredQueries (tallennettu kyselytoiminto)
DescribeStoredQueries (tallennettu kyselytoiminto)

ISO 19119 -standardissa määriteltyjen palveluiden luokittelussa WFS on ensisijaisesti toimintojen käyttöpalvelu, mutta se sisältää myös elementtejä toimintojen tyyppipalvelusta, koordinaattimuunnos- ja muuttopalvelusta sekä maantieteellisen tallennusmuodon muunnospalvelusta. (OGC, n.d.)

ISO 19119, tai tarkemmin ISO 19119:2016 on maantieteellisen tiedon standardi, joka määrittelee tarkennukset alustoille luoduista palveluista, jotta ne voidaan erottaa toistaan.
ISO 19119 määrittelee myös, miten maantieteellisiä palveluita luokitellaan erilaisista näkökulmista. (ISO 19119, 2016)

Kuva 1 Kuvankaappaus lipas.fi-sivustolta (lipas.fi, n.d.)



Esimerkkinä kuvankaappaus (Kuva 1) lipas.fi-sivustolta, joka näyttää kaikki Suomen julkiset liikuntapaikat ja virkistyskohteet kartalla (Jyväskylän yliopisto, n.d.). Kävelypolut (merkitty kuvassa keltaisella), puistot ja virkistäytymisalueet (merkitty vihreällä), ja kuntosalit yms. muut liikuntapaikat (merkitty ”palloilla”) tulevat LIPAS-tietokannan WFS- ja WMS-palveluista.

2.2 WMS

Web Map Service eli WMS on karttapalvelu, joka tarjoaa yksinkertaisen HTTP-rajapinnan georekisteröityjen karttakuvien eli laattojen hakemiseen yhdestä tai useammasta paikkatieto-tietokannasta. WMS-pyyntö määrittelee maantieteelliset kerrokset ja alueet, joihin käyttäjä haluaa keskittyä ja hakee ne tietokannasta. Tuloksena on yksi tai monta georekisteröityä karttalaattaa, joiden tiedostomuoto on JPEG, PNG yms. (OGC, n.d.)

2.3 WMTS

Web Map Tile Service eli WMTS on hyvin samankaltainen kuin WMS, mutta se on tehokkaampi, sillä toisin kun WMS, se käyttää valmiiksi renderöityjä karttakuvia eli laattoja näyttääkseen käyttäjälle halutun tiedon. Tämä vähentää huomattavasti palvelimella olevan suoritustehon käyttöä. (OGC, n.d.)

2.4 INSPIRE

INSPIRE on EU-direktiivi, joka tulee sanoista Infrastructure for Spatial Information in the European Community. Se tähtää mahdollistamaan paikkatietojen yhteensopivuuden yli valtioiden ja organisaatioden rajojen. Tietojen tulee olla yhteen toimivia, laadukkaita ja helposti saatavilla, jotta mm. kriisitilanteissa voidaan perustaa päätöksiä niiden varaan. (Maanmittauslaitos, n.d.)

INSPIRE perustuu paikkatietoinfrastruktuureihin, jotka Euroopan unionin jäsenvaltiot ovat perustaneet ja ylläpitäneet. Direktiivissä käsitellään 34 ympäristösovelluksiin tarvittavaa paikkatietoteemaa. INSPIRE tuli voimaan 15.toukokuuta 2007, ja se laitetaan täytäntöön eri vaiheissa, viimeinen täytäntöönpano on vuoteen 2021 mennessä. (European Commission, n.d.)

2.5 OGC

OGC eli Open Geospatial Consortium on maailmanlaajuinen yhteisö, joka koostuu yhtiöistä, valtion virastoista ja yliopistoista. Se keskittyy luomaan ja ylläpitämään avoimia ja ilmaisia paikkatietoon ja paikkatietorajapintoihin liittyviä standardeja, kuten WFS ja WMS. (OGC, n.d.)

3 ReactJS-viitekehys

Tämän opinnäytetyön toiminnallinen osuus, eli esimerkkisovellusten kehittäminen, toteutetaan käyttämällä Reactia. Ennen kuin puhutaan Reactista, avataan teoriaosuudessa hieman perusasioita muista sovelluskehitykseen liittyvistä ympäristöistä.

3.1 HTML, CSS ja JavaScript

HTML eli HyperText Markup Language on keskeinen verkkosivujen rakennuspalikka, joka määrittelee rakenteen verkkosisällölle. CSS tuo verkkosivulle tyylin, ja JavaScript toiminnallisuuden. (Mozilla, n.d.)

HTML on kieli, joka koostuu elementeistä, joita pystyy antamaan tekstille, muuttaen sen tarkoitusta. Esimerkkejä elementeistä:

<title>, kertoo verkkosivun otsikon.

<body>, pääelementti, joihin suurin osa muista elementeistä menee.

<img>, sisältää kuvan. (Mozilla, n.d.)

CSS eli Cascading Stylesheets on kieli, joka määrittelee HTML-dokumentin tyylin, ja määrittelee miten HTML-elementit tulisi näyttää (W3Schools, n.d.). CSS:llä pystyy esimerkiksi muuttamaan sisällön fonttia, väriä, kokoa tai vaikka lisäämään animaatioita (Mozilla, n.d.).

JavaScript on ohjelmointikieli, jonka avulla verkkosivulle pystyy tekemään monimutkaisia toimintoja, kuten luoda dynaamisesti päivittyvää sisältöä, ohjata multimediaa tai vaikka animoida kuvia (Mozilla, n.d.).

3.2 ReactJS

ReactJS on avoimen lähdekoodin JavaScript-kirjasto, jolla rakennetaan vuorovaikutteisia käyttöliittymiä (Reactjs, n.d.). Se on alun perin Facebookin julkaisema, ja sitä on käytetty mm. Instagramin rakentamiseen (Hemel, 2013). MVC-arkkitehtuurissa (Model View Controller) se on osa View-tasoa, eli se on vastuussa siitä, mitä sovellus näyttää ja tuntuu. (Sufiyan, 2020)

ReactJS on lähiaikoina noussut yhdeksi suosituimmaksi front-end-viitekehykseksi sen ominaisuuksien vuoksi. Yksi syistä on se, että ReactJS on koettu ”helpoksi” kieleksi, ja se vaatii toimintojen toteuttamiseksi vähemmän koodia kuin esim. JavaScript. ReactJS:ssä on myös matala oppimiskäyrä, koska se on pääasiassa HTML:llän ja JavaScriptin konseptien sekoitus lisätyillä ominaisuuksilla. (Sufiyan, 2020)

Toinen pääsyy on se, että ReactJS on tehokas ja nopea, koska se käyttää virtuaalista DOMia (Document Object Model). Tämä tarkoittaa sitä, että ReactJS vertaa komponenttien tilaa, ja päivittää ne ainoastaan, jos niissä on tapahtunut muutoksia, toisin kuin tavanomaiset web-sovellukset, jotka lataavat kaikki komponentit uudestaan, oli muutoksia tai ei. (Sufiyan, 2020)

ReactiJS:ssä keskeisessä osassa on uudelleenkäytettävät komponentit, joita voidaan ajatella ReactJS-rakennuspalikoina. Uudelleenkäytettävät palikat vähentävät huomattavasti sovelluskehitykseen uppoavaa aikaa. (Sufiyan, 2020)

3.3 PWA

PWA eli Progressive Web Application on ohjelmistotekniikan muoto, jossa mobiilissa verkkoselaimessa toimiva verkkosivu tai sovellus pyrkii yhdistelemään natiivin mobiilisovelluksen ja responsiivisen verkkosivun parhaat ominaisuudet (itewiki, n.d.). PWA tarkoittaa sovellusta, jota ei ladata sovelluskaupasta, vaan se toimii missä tahansa laitteessa, jossa on verkkoselain (Alajoki, 2020).

PWA pystyy hyödyntämään laitteen sisäänrakennettuja ominaisuuksia, kuten kameraa tai mikrofonia (itewiki, n.d.). PWA ei rajoitu käyttöjärjestelmään tai kieleen, vaan se toimii kaikilla

alustoilla, vähentäen näin sovelluskehityksen kuluja huomattavasti, kun kahden tai useamman järjestelmän kehityksen sijaan kehitetään vain yksi (Richard, LePage, 2020).

Yksinkertaisuudessaan PWA:t ovat normaaleja verkkosovelluksia, jotka käyttävät moderneja, uusien selainten tuomia ominaisuuksia täyteen potentiaalinsa. PWA:t ovat nopeita, kyvykkäitä ja luotettavia, ja sen takia ne ovat suosittuja nykyisillä kehitysmarkkinoilla. (Richard, LePage, 2020).

4 Karttakirjastot

Opinnäytetyön toiminnallinen osuus koodattaisiin ReactJS:llä, joten tarkoituksena oli löytää ratkaisuja, jotka sopivat ReactJS:n kanssa yhteen. Nopeasi kävi ilmi, että karttakirjastojen kentällä on neljä nimeä, jotka nousivat kerta toisensa jälkeen pintaan: Leaflet, Mapbox GL, OpenLayers ja Google Maps. Google Mapsin tarjoama karttakirjasto jäi nopeasti pois opinnäytetyössä tehtävästä vertailusta, sillä se olisi vaatinut rahallista sitoutumista. (Google, n.d.)

4.1 Leaflet

Leaflet on alun perin Vladimir Agafonkinin kehittämä, vuonna 2011 julkaistu avoimen lähdekoodin JavaScript-kirjasto, jolla rakennetaan sovelluksia, jotka käyttävät karttaa. Monet yritykset, kuten Facebook, flickr ja GitHub käyttävät sitä omissa toiminnallisuuksissaan. (Agafonikin, 2021)

Leaflet tukee suoraan WMS-tasoja, GeoJSON-tasoja ja Vektori-tasoja, mutta ei suoraan WFS-tasoja. Toki, Leafletiin on olemassa lukuisia lisäosia, joilla haluttuja ominaisuuksia voidaan lisätä. (leafletjs.com, n.d.)

JavaScript-kirjastona Leaflet on verraten kevyt kokonaisuus, Leafletin oman arvion mukaan noin 39 KB, kiitos sen modulaarisen ja nykyaikaisen rakenteen. Se on rakennettu toimimaan niin mobiilissa kuin työpöytäkäytössäkin. (leafletjs.com, n.d.)

Leafletille löytyy ReactJS-kirjasto, nimeltään React-Leaflet. Se ei korvaa Leafletia täysin, vaan se sitoo Leafletin kirjaston ominaisuuksia ReactJS-komponenteiksi. React-Leafletia kehitetään aktiivisesti monen kehittäjän toimesta, suurimmaksi osaksi vapaaehtoisvoimin. (react-leaflet.js.org, n.d.)

4.2 Mapbox GL

Mapbox GL JS on JavaScript-kirjasto, jolla luodaan sovelluksia, jotka hyödyntävät karttaa. Mapbox GL:n avulla pystyy luomaan täysin omaan tarpeeseen räätälöityjä karttoja, jotka ovat tehokkaita

vektorityylisiä karttoja. Mapbox GL pitää myös sisällään oman työkalupakin, jolla voi ilman koodia muokata karttoja haluamikseen. GL tulee sanoista graphics library. (docs.mapbox.com, n.d.)

Mapbox GL on muita tässä opinnäytetyössä esiintyviä karttakirjastoja laajempi kokonaisuus. Sen edut tulevat esille erityisesti silloin, kun kehitetään suurempia GIS- eli Geographical Information System -järjestelmiä (suomeksi paikkatietojärjestelmiä). (geoapify.com, n.d.)

Mapbox GL:ään on Leafletin tapaan olemassa ReactJS-kirjasto, joka sitoo Mapbox GL:n ominaisuuksia ReactJS-komponenteiksi. Se käyttää laajasti hyödykseen ReactJS:n ominaisuuksia, kuten koukkuja (hook), DOM:ia ja tiloja (state). (npmjs.com, n.d.) Koukkuja hyödynnetään ReactJS:ssä, kun halutaan uudelleenkäyttää jotakin logiikkaa tai tilaa, ilman että muutetaan komponenttien hierarkiaa. (reactjs.org, n.d.)

4.3 OpenLayers

OpenLayers on JavaScript-kirjasto, jolla voidaan näyttää dynaaminen kartta millä tahansa verkkosivulla. Se on täysin ilmainen, avoimen lähdekoodin projekti. Sen tavoite on helpottaa maantieteellisten järjestelmien kehitystä ja käyttöä. (openlayers.com, n.d.)

Monet muut karttakirjastot käyttävät koordinaattiformaattinaan Latitudi-Longitudi-muotoa, mutta OpenLayers käyttää päinvastaista, Longitudi-Latitudi-muotoa. GIS-kehittäjien kannalta ensimmäinen vaihtoehto on mielekkäämpi. (Stepanov, 2020)

OpenLayersin ominaisuuksiin luetaan sen vahvuus hyödyntää karttojen laattoja suoraan lähteestä, oli kyse sitten MapBoxin, OpenStreetMapin tai vaikka Bingin XYZ-karttalähteistä. OpenLayers tukee myöskin ”suoraan laatikosta” niin työpöytä- kuin mobiilikäyttöä. Siinä on itsessään myös suora tuki vektoritasoille kuten GeoJSON, GML, KML ja TopoJSON. (Stepanov, 2020)

4.4 OpenStreetMap

OpenStreetMap on projekti, joka luo ja jakaa avointa maantieteellistä tietoa karttojen muodossa. Se on ilmainen, ja sitä rakentaa suuri joukko vapaaehtoisia. Tarve projektille on syntynyt siitä, ettei esim. Google Mapsin tieto ole ilmaista tai aina saatavilla. (wiki.openstreetmap.org, n.d.)

OpenStreetMap on edelleen kehitteillä oleva projekti, ja kuka vain voi auttaa sen laajentamisessa. OpenStreetMapin käyttäjäkunta oli elokussa 2021 n. kahdeksan miljoonaa, ja sen aktiivisia kehittäjiä oli huhtikuussa 2021 n. 50 000. (wiki.openstreetmap.org, n.d.)

Steve Coast perusti OpenStreetMapin aikoinaan vuonna 2004, tavoitteenaan kartoittaa Yhdistyneet Kuningaskunnat. Vuonna 2006 perustettiin OpenStreetMap Foundation, jonka tehtävänä oli tukea kasvua, kehitystä ja tiedon jakamista. (wiki.openstreetmap.org, n.d.)

5 Esimerkkisovellusten tavoite, tarkoitus ja menetelmät

Opinnäytetyön tarkoituksena on kolmen esimerkkisovelluksen avulla havainnoida ja pohtia, miten karttakirjastoja ja paikkatietorajapintoja hyödynnetään käytännössä ReactJS-sovelluksissa. Samalla pohditaan myös, mikä kirjastoista olisi ihanteellisin, jos tavoitteena olisi näyttää ja visualisoida tietoa kartalla mm. WFS-palvelun avulla.

Käytännössä sovellukset ovat siis PWA-mallin mukaisia, yksinkertaisia sovelluksia, jotka:

1. Näyttävät kartan
2. Karttaa pystyy liikuttamaan ja raahaamaan
3. Kartasta voi zoomata kauemmas ja lähemmäs

Yksinkertaisten sovellusten avulla on tarkoitus päästä syvemmälle eri karttakirjastojen tarjoamiin vaihtoehtoihin ja vertailla niitä. Vertailussa otetaan huomioon koko sovelluksen kehitysprosessi ja sen apuna toimivat seuraavat kysymykset:

1. Kuinka helppoa karttakirjaston käyttöönotto on?
2. Kuinka selkeää ja helposti ymmärrettävää karttakirjaston oma syntaksi on?
3. Kuinka laajasti karttakirjastosta on löydettävissä dokumentaatiota tai muuta, kehitystä helpottavaa tietoa?

Esimerkkisovellusten teon aikana ja sen jälkeen syntyy käsitys siitä, mikä karttakirjasto olisi ihanteellisin paikkatietorajapinta-implementointia tai sen esittelyä ja tutkintaa varten. Valitusta karttakirjastosta ja paikkatietorajapinnasta kirjoitettava käytännön esittely tulee esimerkkisovellusten kehityksen jälkeen.

5.1 Menetelmät

Opinnäytetyö on luonteeltaan toiminnallinen opinnäytetyö, ja se suoritetaan vesiputousmallia parhaan mukaan noudattaen. Vesiputousmallin vahvuuksiin kuuluu se, että sitä on helppo ymmärtää ja seurata, koska se asettaa projektille selkeät välivaiheet ja päättymispisteet. Mallin heikkoudet tulee sen joustamattomuudesta; muutoksia on vaikea saada läpi, jos prosessi on jo mennyt eteenpäin. Myöskin, kun asioita ja prosesseja ei voida suorittaa samaan aikaan, vähentää se tehokkuutta. (Lewis, 2019)

5.2 Vesiputousmalli

Vesiputousmalli on suoraviivainen, vaihe vaiheelta etenevä ohjelmistotuotantoprosessi, jossa kehitysvaiheet soljuvat alaspäin kuten vesiputouksessa. Se koostuu seitsemästä vaiheesta, jotka ovat usein irtonaisia toisistaan:

1. Vaatimukset. Aluksi pohditaan, mitä sovellus tulee mahdollisesti vaatimaan, mitkä projektin päättymispäivät ovat ja paljon aikaa on käytettävissä.
2. Analyysi. Käydään tarkemmin läpi käyttötapauksia ja liiketoimintalogiikkaa, jotka ohjaavat tuotantoa.
3. Suunnittelu. Mikä ohjelmointikieli on käytössä, tarvitaanko palvelinrautaa? Mistä saadaan tieto, arkkitehtuuri ja palvelut?
4. Toteutus. Itse tekeminen, käytetään aiemmin pohdittuja kohtia käytännössä. Kehitetään sovellusta mahdollisesti pienissä osissa.
5. Testaus. Tarkastetaan sovelluksen laatu, tehdään muutoksia, jotta saadaan sovellus toimimaan halutulla tavalla.
6. Tuotanto. Sovellus on valmis käytettäväksi.
7. Ylläpito. Sovellusta päivitetään suunnitelman mukaisesti ja sitä korjataan teknologian mennessä eteenpäin. (Lewis, 2019)

Vesiputousmalli sopii opinnäytetyön toiminnallisen osuuden kehitysmalliksi, sillä opinnäytetyön aikana kehitettävät sovellukset tulisivat olemaan yksinkertaisia ja vaatimuksiltaan pieniä. Vesiputousmallia ei noudateta sanatarkasti ja kohta kohdalta, vaan sitä hyödynnetään parhaaksi nähdyllä tavalla. Esimerkiksi ylläpitoa ei sovelluksille lähdetä suunnittelemaan, sillä sovellukset eivät tule kuluttajakäyttöön, vaan niiden tarkoitus on tukea opinnäytetyön kirjoittajan ammatillista kehitystä ja osaamista.

6 Karttasovellusten kehittäminen ja vertailu

Seuraavissa luvuissa käydään ensin läpi sovellusten tekoa eri karttakirjastojen kanssa. Karttakirjastoiksi valikoituivat Leaflet, OpenLayers ja Mapbox GL. Google tarjoaa myös rajapinnan, jota hyödyntää omissa sovelluksissa, mutta se on osittain maksullinen, jonka takia Googlen tarjoamaa vaihtoehtoa ei tässä opinnäytetyössä hyödynnetä. Tavoitteena on siis kehittää näillä kolmella karttakirjastolla kolme erillistä sovellusta, jotka näyttävät liikuteltavan ja zoomattavan kartan. Koodi tulee olemaan yksinkertaista, koska tavoitteena ei ole tehdä kiillotettua versiota tai täydellistä julkaisukelpoista sovellusta, vaan havainnoida ja vertailla kirjastoja. Sovellukset aloitetaan kaikki ReactJS-peruspohjasta, joka saadaan luotua alla näkyvällä komennolla (Komento 1).

Komento 1 Npx-komento, jolla saadaan luotua uusi ReactJS-sovelluksen runko

```
npx create-react-app <apin nimi tähän>
```

Ensimmäisten testisovellusten jälkeen valitaan intuitiivisimman oloisin karttakirjasto ja havainnoidaan sen avulla, miten ReactJS-sovelluksella haetaan tietoa esim. WFS-palvelusta. Intuitiivisin karttakirjasto valitaan sillä perusteella, millä karttakirjastolla oli paras kehityskokemus, ja kuinka kyseiselle karttakirjastolle löytyy dokumentaatiota ja tukea verkosta.

Kaikkien sovelluksien lähdekoodit löytyvät osoitteesta https://github.com/NestoriMe?tab=repositories, jossa ne ovat julkisesti esillä vähintään kaksi vuotta opinnäytetyön julkaisusta.

6.1 ReactJS ja Leaflet

Sovelluksen kehittämisen aloittaminen on Leafletin kanssa suoraviivaista, sillä Leaflet tarjoaa npm-paketin, jonka pystyy suoraan lisäämään omaan projektiin. ReactJS-runkoon tulee lisätä alla olevat komennot (Komennot 2, 3, 4), jotta saadaan Leafletin käytön kannalta kriittisimmät ominaisuudet käyttöön.

Komento 2 Npm-komento, jolla saadaan asennettua leaflet

```
npm install leaflet
```

Komento 3 Npm-komento, jolla saadaan asennettua react-leaflet

```
npm install react-leaflet
```

Komento 4 Npm-komento, jolla saadaan asennettua react-dom leaflet

```
npm install react react-dom leaflet
```

Ohjelmakoodi 1 App-funktio, jonka sisällä alustetaan kartta

```
function App() {
  return (
    <MapContainer center={[60.181788, 24.927551]} zoom={13}>
      <TileLayer
        url="https://tiles.hel.ninja/styles/hel-osm-bright/{z}/{x}/{y}@2x@fi.png"
        attribution='&copy; <a href="https://www.openstreetmap.org/copyright">OpenS
treetMap</a> contributors'
      />
    </MapContainer>
  );
```

Seuraamalla React-Leafletin ja Leafletin dokumentaatiota päästään haluttuun lopputulokseen suhteellisen nopeasti (Ohjelmakoodi 1), ja kartta saadaan näkyviin. React-Leafletin MapContainer pitää sisällään kartan kannalta tärkeimmät osiot, joista ensimmäisenä mainitaan center-attribuutti, johon voi syöttää koordinaatit, joihin kartta kohdistuu ladatessa. Zoom-attribuutissa määritetään, kuinka lähellä kartta on tarkennettuna perustilassa.

TileLayer pitää sisällään ominaisuuden, joka lataa kartan kuvat ”laattoina” sovelluksen MapContainer-näkymään. Laattoja pystyy lataamaan erilaisilta sivustoilta, jotka ovat yhteyksissä OpenStreetMapin lähdekoodiin. Tässä sovelluksessa käytettiin oletuslaattojen sijaan Helsingin kaupungin tarjoamaa laattakokoelmaa.

React-Leaflet sovelluksessa piti myös .css-tiedostoon määrittää kartan korkeus ja leveys, joita ilman kartta ei näkynyt oikein. Tämä ei selvinnyt virallisesta dokumentaatiosta, vaan asia täytyi selvittää muuta kautta, mm. toisia tutoriaaleja tutkimalla.

6.2 ReactJS ja Mapbox GL

Mapbox GL:lää käyttävässä sovelluksessa täytyy ensimmäisenä luoda kartalle ”token” Mapbox GL:n sivuilla. Tämä ”token” on ikään kuin lupa käyttää Mapboxin luomaa karttaa. Sivusto vaatii rekisteröinnin, mutta on ilmainen. Mapbox GL:n käyttöä varten tarvitsee sovellukseen lisätä muutama paketti, samalla tavalla kuin Leafletissä (Komento 5).

Komento 5 npm-komento, jolla saadaan asennettua react-mapbox

```
npm install react-map-gl
```

Ohjelmakoodi 2 Map-luokka, jonka sisällä alustetaan kartta

```
constructor() {
    super()
    this.state = {
      viewport: {
        width: '100vw',
        height: '100vh',
        latitude: 60.171609,
        longitude: 24.927127,
        zoom: 13
      }
 }
    this.handleViewportChange = this.handleViewportChange.bind(this)
```

Toisin kuin Leafletissa, Mapbox GL:ssä luodaan ”rakentaja” (Ohjelmakoodi 2), jossa kartta alustetaan, eli annetaan sille leveys, korkeus, zoom-taso ja koordinaatit, joihin kartta kohdistuu perustilassa. Viewport on kehys, jolla lisätään ominaisuuksia karttaan.

Ohjelmakoodi 3 handleViewportChange eli miten karttaa liikutetaan ja miten se näkyy

```
handleViewportChange(viewport) {
    this.setState(prevState => ({
      viewport: {...prevState.viewport, ...viewport}
    }))
  }
  render() {
    return (
      <ReactMapGl
        {...this.state.viewport}
        onViewportChange={viewport => this.setState({viewport})}
        mapboxApiAccessToken={mapboxToken}
        mapStyle="mapbox://styles/mapbox/streets-v10"
      />
    )
  }
```

Yläpuolella näkyvässä koodissa (Ohjelmakoodi 3) viewportille rakennetaan käsittelijä, joka mahdollistaa kartan liikuttelun. Ilman käsittelijää kartta on paikallaan, eikä se liiku mihinkään.

Luokan Map lopussa kartta palautetaan ja renderöidään sovelluksessa. ReactMapGl sisällä määritellään viewportista ominaisuudet, joita karttaan tuodaan ladatessa, ja siinä asetetaan kartta siihen tilaan, johon se on rakentajassa määritelty. Kartalle annetaan sitten tieto siitä, että sovelluksella on ”lupa” Mapbox GL:ltä käyttää luotua karttaa ApiAccessToken-kohdassa. Lopuksi sovellus hakee kartalle tyylin suoraan Mapbox GL:ltä, tässä tapauksessa tyylin nimeltä ”streets-v10”.

6.3 ReactJS ja OpenLayers

OpenLayersin kanssa sovelluksen rakentaminen tuotti päänvaivaa. ReactJS-tuki oli vähäistä, ja iso osa ohjeista ja dokumentaatiosta on tarkoitettu JavaScriptillä kirjoitettavaksi, joka ei palvele tämän opinnäytetyön tarkoitusperiä. Medium.com:sta löytyi kuitenkin Matthew Brownin kirjoittama artikkeli, jossa hän oli törmännyt samaan ongelmaan. Tämän sovelluksen kehitysprosessi siis seuraakin tarkasti Matthewn luomaa artikkelia, vaikkakin Matthew käyttää myös paljon JavaScriptiä sovelluksen kehittämiseen. Kehittäminen alkaa jälleen npm-paketilla (Komento 6), jolla saadaan OpenLayersin kattava valikoima ominaisuuksia käyttöön. (Brown, 2020)

Komento 6 Npm-komento, jolla saadaan asennettua OpenLayers

```
npm install ol --save
```

Ohjelmakoodi 4 const App OpenLayer-sovelluksessa

```
const App = () => {
  const [center] = useState([24.928837, 60.176975]);
  const [zoom] = useState(13);
return (
  <div>
    <Map center={fromLonLat(center)} zoom={zoom}>
      <Layers>
        <TileLayer
          source={osm()}
          zIndex={0}
        />
      </Layers>
   </Map>
    </div>
  );
}
```

Sovellus rakennettiin monesta eri tiedostosta, mutta tärkeimpänä esiteltävänä pääsovellus-tiedosto (Ohjelmakoodi 4), karttatiedosto ja laattojen asettelijatiedosto. Tuttuun tapaan asetetaan keskipiste, johon kartta ladatessa kohdistaa ja annetaan sille koordinaatit. Kartan laatat tulevat OpenStreetMapin kautta samalla tavalla kuin Leafletissa.

Ohjelmakoodi 5 useEffect const Mapin sisällä OpenLayers-sovelluksessa

```
const Map = ({ children, zoom, center }) => {
  const mapRef = useRef();
  const [map, setMap] = useState(null);
  // on component mount
  useEffect(() => {
    let options = {
      view: new ol.View({ zoom, center }),
      layers: [],
      controls: [],
      overlays: []
    };
    let mapObject = new ol.Map(options);
    mapObject.setTarget(mapRef.current);
    setMap(mapObject);
    return () => mapObject.setTarget(undefined);
  }, []);
```

Karttaa käsittelevässä tiedostossa on muutama OpenLayersin APIn ympärille rakentuvaa komponenttia, joilla karttaa hallitaan. Edellä näytetyssä koodissa (Ohjelmakoodi 5) alustetaan kartta ja tallennetaan se nykyiseen muotoonsa. Koodin seassa on paljon puhdasta JavaScriptiä, ja vähemmän ReactJS:lle tuttuja komponentteja.

Ohjelmakoodi 6 const TileLayer OpenLayers-sovelluksessa

```
const TileLayer = ({ source, zIndex = 0 }) => {
  const { map } = useContext(MapContext);
  useEffect(() => {
    if (!map) return;

    let tileLayer = new OLTileLayer({
      source,
      zIndex,
    });
    map.addLayer(tileLayer);
    tileLayer.setZIndex(zIndex);
    return () => {
      if (map) {
        map.removeLayer(tileLayer);
      }
    };
  }, [map]);
  return null;
};
```

Yllä olevassa koodinpätkässä (Ohjelmakoodi 6) luodaan uusi OpenLayersin TileLayer, eli toisin kuin esim. Leafletissa, jossa TileLayer on suoraan ReactJS-komponenttina, pitää se OpenLayers-sovelluksessa ensin alustaa JavaScriptillä. Luonnin jälkeen se lisätään ”kerroksena” karttaan. Mediumin artikkelissa lisätään paljon muitakin ominaisuuksia karttaan, mutta tässä kohtaa kartta toimii juuri kuten tässä kohtaa tavoiteltiinkin.

7 ReactJS ja WFS

Leaflet-, Mapbox GL-, ja OpenLayers karttakirjastoilla tehtyjen testisovellusten jälkeen huomattiin, että helppokäyttöisyydessä Leaflet ja Mapbox GL ovat hurjasti OpenLayersia edellä. Erityisesti Leaflet-karttakirjasto sopeutui käyttöön nopeasti ja sille löytyi paljon sitä tukevaa dokumentaatiota, jota seuraamalla pääsee käyttäjä kuin käyttäjä syvälle karttakirjastojen maailmaan.

Seuraavaksi opinnäytetyössä oli tarkoituksena tarkastella, miten ReactJS-sovelluksissa hyödynnetään WFS/WMS-toimintoja, eli miten rajapintoja kutsutaan. Ennen koodiin sukellusta tutustutaan kuitenkin QGIS-ohjelmiston ja LIPAS-rajapinnan avulla WFS- ja WMS-tasoihin, sillä ennen kuin voidaan kutsua rajapintoja, täytyy tietää mitä kutsutaan.

Rajapintojen WFS/WMS-toimintoja tarkastellaan jo olemassa olevan lähdekoodiesimerkin ja muiden, osaksi valmiiden ratkaisuiden kautta. Rajapintoihin perehdyttäessä tutkitaan Jyväskylän yliopiston kehittämää LIPAS-tietokantaa, ja lähdekoodiesimerkkinä toimii maantieteellisten tietojärjestelmien foorumeilta löytyvä keskustelu, jossa havainnoidaan Espanjan sairaaloita.

7.1 QGIS ja rajapinnan tutkiminen

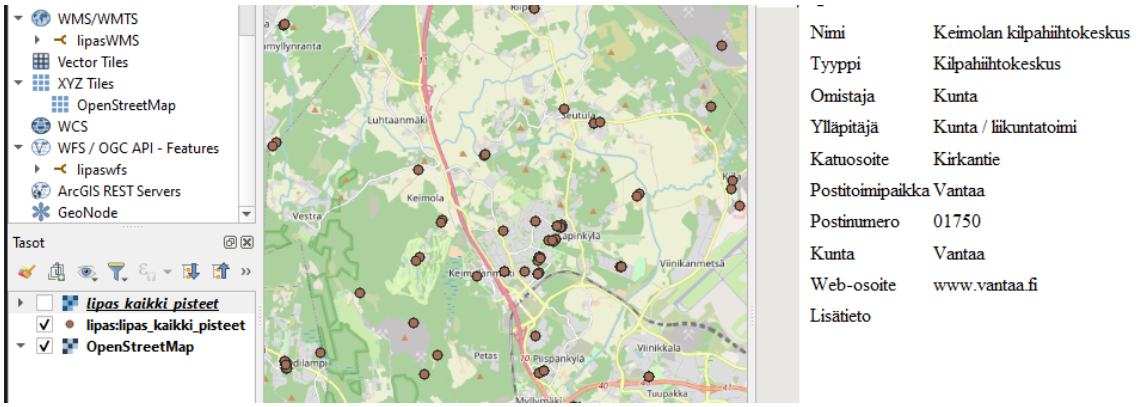
QGIS on paikkatieto-ohjelmisto, jonka avulla tietoa voidaan näyttää, selata, muokata ja analysoida. Sen voi ladata ilmaiseksi verkosta, ja se tai muut paikkatieto-ohjelmat (ArcGIS, MapInfo) ovat käytännössä välttämättömiä, kun kehitetään sovelluksia, jotka näyttävät tietoa kartalla.

Tässä opinnäytetyössä WFS-rajapintaa tutkiessa QGIS toimii erinomaisesti työkaluna, jonka avulla pystytään selvittämään, mitä rajapinnasta voidaan hakea, ja esimerkiksi millä nimellä WFS-tasot kulkevat. QGIS:illä testailu antaa myös hyvän vertailukohteen siihen miltä datan pitäisi näyttää, jos on esimerkiksi kehittämässä omaa sovellusta.

Ensimmäisenä QGIS:iin ladataan XYZ-taustakartta, ja siinä onkin suora tuki OpenStreetMapille, joten kartta saadaan näkyviin nopeasti. QGIS:issä on valmiudet tuoda tasoja monesta eri lähteestä,

mukaan lukien vektori- ja rasteritasoja, PostGIS-tasoja, WMTS-tasoja ja paljon muita. LIPAS-tarjoaa WFS- ja WMS-tasot, joten niihin muodostetaan seuraavaksi yhteys. Yhteyden muodostamiseen tarvitaan Jyväkylän yliopiston verkkosivuilta löytyvät URL-osoitteet, jossa rajapinnat sijaitsevat.

Kuva 2 Kuvankaappaus QGIS-ohjelmasta



Vasemmassa alakulmassa (Kuva 2) näkyy valittuna OSM:n taustakartta, ja sitä ylempänä on lippaan WFS-taso. Ylimpänä on WMS-taso, joka ei kuvankaappauksen ottohetkellä ollut aktiivinen. Jokaisen kartalla näkyvän pisteen tietoja voi tarkastella, ja kuvan oikealla puolella näkyy auki avattuna yhden pisteen tiedot.

Yhteyden onnistuneen muodostamisen jälkeen QGIS listaa kaikki mahdolliset tasot, jotka voidaan näyttää kartalla, ja sitä kautta kutsua jatkossa koodissa. Syy miksi tämä on tärkeää, selviää hieman tuonnempana. Yhteyden muodostamisen jälkeen tässä esimerkissä valittiin ”lipas_kaikki_pisteet”-taso molemmissa palveluissa, joka nimensä mukaan hakee kaikki mahdolliset tietolähteet, jossa geometrian tyyli on juuri tuo piste/pallo.

7.2 WFS-pyyntö koodissa

QGIS:n avulla saatiin selville, mitä tasoja LIPAS-tietokannan WFS- ja WMS-rajapinta pitävät sisällään. Tämä on tärkeä tieto, sillä vaikka Jyväskylän yliopisto sivuillaan tätä tietoa esitteleekin ilman tarvetta paikkatieto-ohjelmistolle, eivät kaikki avointa tietoa tarjoavat järjestöt dokumentoi rajapintojaan näin mallikkaasti.

ReactJS:n puolella koodi tarvitsee käytännössä kolme asiaa: URLin tietokantaan, WFS-pyynnön parametrit ja funktion, jolla tieto haetaan tietokannasta.

Ohjelmakoodi 7 WFS-osoite ja parametrit

```
const WFSosoite = "http://lipas.cc.jyu.fi/geoserver/ows?";

const Pyyntö = {
    outputFormat: 'application/json',
    maxFeatures: 250,
    request: 'GetFeature',
    service: 'WFS',
    typeName: 'lipas:lipas_kaikki_pisteet',
    version: '2.0.0'
};
```

Osoite (Ohjelmakoodi 7) ohjaa LIPAS-tietokannan WFS-rajapintaan, joka sijaitsee Jyväskylän yliopiston operoimalla GeoServerillä. Pyynnössä (Ohjelmakoodi 7) määritellään, missä muodossa tieto haetaan, tässä tapauksessa se haetaan JSON-formaatissa. Kohdassa maxFeatures määritellään, kuinka monta pistettä tietokannasta haetaan. Jyväskylän yliopisto ohjaa käyttäjiä asettamaan rajan haulle, jotta turhan suuret testipyynnöt eivät kuormittaisi palvelimia liikaa. Request-kohdassa määritellään, mitä toimintoa halutaan käyttää kyselyn suorittamiseksi. Toiminnot ovat laajemmin lueteltuna ja esiteltynä teoriapohjassa, mutta tässä käytetään GetFeature-kyselytoimintoa. Service- ja version-kohdissa määritellään, mitä rajapintaa käytetään ja mikä sen versio on.

Kohdassa typeName määritellään, mitä lukuisista eri tasoista halutaan tällä kertaa hakea. Aikaisemmin tehtyjen QGIS-testien perusteella tiedetään, että tietokanta pitää sisällään ”lipas:lipas_kaikki_pisteet” -nimisen WFS-tason.

Ohjelmakoodi 8 Haku-funktio

```
  componentDidMount() {
        axios.get(WFSosoite, { params: Pyyntö })
            .then(({ data }) => this.setState({ data }))
            .catch(error => Promise.reject(error));
    }
```

Funktiossa componentDidMount (Ohjelmakoodi 8) annetaan axios-lisäosalle parametreina WFS-osoite ja Pyyntö-kohdassa määritellyt tarvittavat tiedot. Axios käsittelee pyynnön http:n kautta ja tekee itse haun. Sen jälkeen tieto sidotaan data-muuttujaan ja käsitellään mahdollinen virhekoodi.

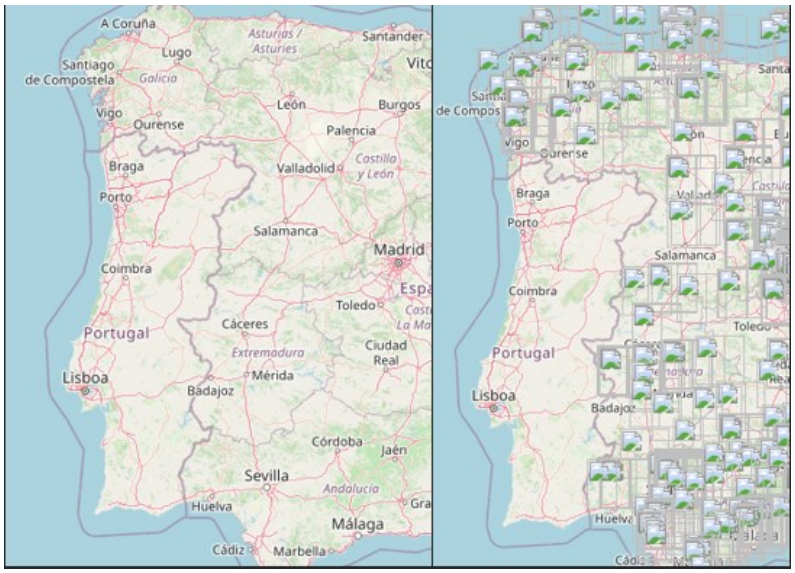
Kuva 3 Haun tuottama lopputulos

Ylempänä (Kuva 3) on onnistuneen haun näkymä konsolissa. WFS-rajapinnasta haettava tieto esiintyy usein ylempänä näkyvänä FeatureCollectionina, jonka sisällä on suuri määrä taulukoita (array), joiden sisällä on tiedot jokaisesta pisteestä. Yllä on porauduttu yhteen tällaiseen pisteeseen, ja kuten kuvasta näkyy, on yhteen pisteeseen sidottu paljon tietoa; löytyy osoitteet, ylläpitäjät ja tiedot, mitä pisteessä sijaitsee (palloseinä).

7.3 Esimerkki lähdekoodista

Seuraavassa on esitelty yksi lähestymistapa siihen, miten sovellus lataa kartan, ja nappia painettaessa näyttää kartalla kaikki Espanjan julkiset sairaalat. Koodia on muunneltu sen verran, että se on käännetty suomeksi ja siitä on poistettu epäoleellisia osioita.

Kuva 4 Sovellus perustilassa, ja sovellus ”Etsi sairaalat”-napin jälkeen



Ylempänä (Kuva 4) on esitelty lähdekoodiesimerkin toimintaa, kun sovelluksessa muodostetaan tietokantaan yhteys ja haetaan kartalle kaikki sairaalat. Pisteille ei ole määritelty sen kummoisemmin mitään grafiikkaa, joten sovellus näyttää sen sijaan tyhjän neliön. Tämä tekee kuvasta hieman vaikealukuisen, mutta ajatuksena oli demonstroida WFS-pyyntöä, jättäen vähemmän huomiota sovelluksen rujoudelle.

Ohjelmakoodi 9 haeSairaalat-funktio

```
  haeSairaalat = async () => {
    let sairaalat = await axios.get(
      "https://cartovis-server.herokuapp.com/hospitales"
    );

    let reference = React.createRef();

    this.setState({
      geoJSON: sairaalat.data,
      reference: reference,
     });
  };
```

Sovellus aloittaa datan hakemisella (Ohjelmakoodi 9), jossa hyödynnetään URL:in kautta löytyvää FeatureCollectionia, jossa on tiedot kaikista Espanjan sairaaloista. FeatureCollectionissa on enemmänkin tietoa sairaaloista, mm. osoitteet ja fax numerot, mutta havainnollistamisen vuoksi käytetään ainoastaan paikkatietoa. URL-osoite ei ole perinteinen WFS-osoite, mutta se toimii samalla periaatteella, eli axios-lisäosan avulla haetaan sivustolta tiedot, jotka kiinnitetään ReactJS:ssä geoJSON muuttujaan ”sairaalat.data”. Muuttuja on tilassa, jossa sille määritetään myös viittaus ReactJS:n ohjelmointityylin mukaisesti.

Ohjelmakoodi 10 Lähdekoodiesimerkin render-osio

```
       <MapContainer center={[40, -7]} zoom="6" style={this.style}>
        {this.state.geoJSON && (
          <GeoJSON
            data={this.state.geoJSON} />
        )}
        <TileLayer
          attribution='&copy; <a
href="http://osm.org/copyright">OpenStreetMap</a> contributors'
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" />
      </MapContainer>
      <div className="otherLayers">
          <h4>Espanjan sairaalat</h4>
          <button className="btn btn-primary" onClick={this.haeSairaalat}>
            Etsi sairaalat
          </button>
        </div></>
```

Ylhäältä alas, tuttuun tapaan kohdistetaan kartta tiettyihin koordinaatteihin ja asetetaan zoom-taso (Ohjelmakoodi 10). Tämän jälkeen asetetaan GeoJSON-kerros valmiiksi kartalle. Kartalla ei

aluksi näy mitään tietoja, mutta ”haeSairaalat”-napin painamisen jälkeen populoidaan GeoJSON-kerros sairaalat.datalla.

Samalla tavalla kuin ensimmäisessä React-Leaflet sovelluksessa, paketoidaan sovelluksen osat MapContaineriin ja TileLayeriin. Erona se, että tässä esimerkissä tärkeään rooliin nousee myös GeoJSON-kerros, jonka avulla tiedot saadaan visualisoitua.

8 Johtopäätökset ja pohdinta

Opinnäytetyön tarkoituksena oli tutkimuskysymysten avulla selvittää paikkatietorajapintojen käyttöä nykyaikaisessa ReactJS-sovelluksessa. Ensin tarkoituksena oli selvittää, mitä paikkatietorajapinnat ylipäätään ovat, ja siihen vastaus löytyikin pitkälti teoriapohjasta, ja tietoon syvennyttiin käytännön esimerkkien johdolla.

Toteutusvaiheessa kuitenkin huomattiin, etteivät paikkatietorajapintoja hyödyntävät yhteisöt jaa tietoa samaan tapaan kuin ehkä muut kehittäjäyhteisöt. Esimerkit mm. WFS/WMS-ominaisuuksien lisäämisestä omaan koodiin olivat vaikeasti saatavilla, vaikka kaiken tiedon tulisi standardien mukaan olla avointa ja helposti saatavilla. Dokumentaatiolla pääsee toki pitkälle, mutta käytännön esimerkkejä oli hankala löytää.

Karttakirjastoista löytyvä dokumentoinnin määrä oli ilahduttavaa. Pääsääntöisesti jokaisella karttakirjasto-ratkaisulla oli mielekästä lähteä kehittämään sovellusta, joka näyttäisi yksinkertaisen kartan. Kolmesta ratkaisusta helppokäyttöisin ja intuitiivisin oli kuitenkin React-Leaflet. Sen syntaksi päihitti toiset ohjelmakirjastot helppolukuisuudellaan ja ymmärrettävyydellään. Myös dokumentaation ja esimerkkien määrä suosi React-Leafletia muita enemmän.

React-Mapbox GL:ssä hyvää oli Mapbox GL:n tarjoama dokumentaatio, mutta muuten ohjeet mm. WFS-ominaisuuksien lisäämisestä jäivät vähäisiksi. Tämä on yllättävää, ottaen huomioon, että Mapbox GL:n käyttöaste on kolmesta opinnäytetyössä käytettävästä karttakirjastosta mahdollisesti suurin.

React-OpenLayers osoittautui pettymykseksi. ReactJS tuki oli vähäistä, eikä ole mielekästä kirjoittaa puhdasta JavaScriptiä työssä, jossa keskitytään ReactJS:ään. Tämä oli sääli, sillä OpenLayersin JavaScriptille tarkoitetut esimerkit ja dokumentaatio olivat kattavia ja hyvin kuvattuja esimerkkikoodi-pätkillä. Tähän varmasti osasyynä on se, ettei OpenLayers ole läheskään yhtä suosittu kuten esimerkiksi Leaftet. Googlaus ”OpenLayers StackOverflow” tuottaa ainoastaan 52 700 tulosta, kun taas vastaavasti ”Leaflet StackOverflow” tuottaa yli kuusi miljoonaa tulosta

Karttakirjasto-vertailun jälkeen vastattiin kolmanteen tutkimuskysymykseen, ja tutkittiin lähdekoodiesimerkin avulla, miten WFS-dataa hyödynnetään käytännössä ReactJS-pohjaisessa

sovelluksessa. Esimerkkinä toimi yksinkertainen sovellus, joka nappia painamalla näytti kaikki Espanjan sairaalat kartalla. Esimerkin kautta oli helppo visualisoida, mistä koko opinnäytetyössä on kyse – avoimen datan esittäminen kartalla.

Jatkosuunnitelmat opinnäytetyölle olisivat paremman lähdekoodin löytäminen, mahdollisesti oman esimerkin rakentaminen ja sitä kautta laajempi sovelluskehitys. Tämä tietysti vaatisi mahdollisen asiakkaan, ja olisi huomattavasti suurempi kokonaisuus niin työltään kuin budjetiltaan, kuin mitä opinnäytetyön raameihin perinteisesti kuuluu. Aiheesta on verraten vähän tehty opinnäytetöitä, joten sivistynyt arvaus olisi, että tällaisen sovelluksen kehittämiseen liittyvälle projektille olisi kysyntää.

9 Yhteenveto

Opinnäytetyön tavoitteena oli tutkia, kuinka karttakirjastoja ja paikkatietorajapintoja hyödynnetään käytännössä ReactJS-sovelluksissa. Mielestäni tavoitteisiin päästiin ja kuten jo pohdinnassa huomattiin, tutkimuskysymyksiin vastattiin. Työ vaati paljon sinnikkyyttä ja uuden omaksumista, ja koko prosessi oli ajoittain raskas, sillä ainoastaan ReactJS oli jotenkin tuttu ympäristö. Kaikki muu, karttakirjastoista ja WFS/ WMS-ominaisuuksista lähtien oli täysin tuntematonta reviiriä. Myöskin ohjelmointitaitoni ovat aloittelijan tasolla, joten tekemistä totisesti riitti.

Ohjelmoinnissa tutkitaan usein jo luotuja ratkaisuja, ja hyödynnetään mahdollisesti jo kehitettyjä, toimivia osia omaan sovellukseen. Tämänkin työn kohdalla karttakirjastosovelluksien tekeminen oli mielekkäintä ja opetti minulle paljon ReactJS:stä, karttakirjastoista ja ylipäätään ohjelmoinnista. Dokumentaation avulla pääsi pitkälle, eikä mikään päihitä sitä tunnetta, kun saa koodin toimimaan pienen painin jälkeen.

Paikkatietorajapintojen kohdalla tulikin sitten työn suurin kompastuskivi. Kuten aiemminkin mainittiin, paikkatietorajapinta-yhteisö tuntuu aika pieneltä piiriltä ammattilaisia ja asiantuntijoita, ja tietoa on hyvinkin rajallisesti saatavilla. Karttakirjastoista kuten Leaflet, Mapbox GL ja OpenLayers tietoa löytyy mittavasti eri lähteistä, mutta harvasta paikasta löytyy tietoa, kuinka esim. WFS-rajapinnasta haetaan tietoa sovelluksen sisällä. Vaikeuksista huolimatta koin, että opin tästäkin aihealueesta paljon, ja olisikin kiinnostavaa tulevaisuudessa kehittää tätä aihetta eteenpäin, esimerkiksi työpaikan projekteissa.

Lähteet

Agafonikin, V. (2021). *Leaflet.* Retrieved November 14, 2021 from https://leafletjs.com/

Agafonikin, V. (2021). *Leaflet - Plugins.* Retrieved November 14, 2021, from https://leafletjs.com/plugins.html

Alajoki, J. (2020). *Mikä on progessiivinen verkkosovellus (PWA) ja mitä etuja se tarjoaa?* https://www.evermade.fi/fi/artikkeli/mika-on-progressiivinen-verkkosovellus-pwa-edut/

Brown, M. (2020). *How to Use OpenLayers Maps in React.* Retrieved October 15, 2021, from https://medium.com/swlh/how-to-incorporate-openlayers-maps-into-react-65b411985744

European Comission. (n.d.). *INSPIRE Knowledge base.* Retrieved February 11, 2021, from https://inspire.ec.europa.eu/about-inspire/563

Geoapify. (2020). *Map libraries comparison: Leaflet vs Mapbox GL vs OpenLayers - trends and statistics.* Retrieved November 14, 2021, from https://www.geoapify.com/map-libraries-comparison-leaflet-vs-mapbox-gl-vs-openlayers-trends-and-statistics

Google. (n.d.). *Google Maps Platform – Pricing*. Retrieved November 13, 2021, from https://mapsplatform.google.com/pricing/

Hemel, Z. (2013). *Facebook's React JavaScript User Interfaces Library Receives Mixed Reviews.* https://www.infoq.com/news/2013/06/facebook-react/

ISO 19119:2016. (2016). *Geographic information — Services.* https://www.iso.org/standard/59221.html

itewiki. (n.d.). *Progressive web application (PWA)/Progressiivinen verkkosovellus.* Retrieved February 12, 2021, from https://www.itewiki.fi/opas/progressive-web-application-pwa-progressiivinen-verkkosovellus/

Jyväskylän yliopisto. (n.d.). *LIPAS – liikuntapaikat.* Retrieved February 14, 2021, from https://lipas.fi/etusivu

Lewis, S. (2019). *Definition: Waterfall model.* Retrieved November 25, 2021, from https://searchsoftwarequality.techtarget.com/definition/waterfall-model

LIPAS. (n.d.). *Kuvankaappaus Lipas.fi-sivustolta* [kuva]. Retrieved February 14, 2021, from https://lipas.fi/liikuntapaikat

Maanmittauslaitos. (n.d.). *Mikä INSPIRE?* Retrieved February 8, 2021, from https://www.maanmittauslaitos.fi/kartat-ja-paikkatieto/paikkatietojen-yhteentoimivuus/inspire/mika-inspire

Mapbox. (n.d.). *Mapbox GL JS.* Retrieved November 14, 2021, from https://docs.mapbox.com/mapbox-gl-js/guides/

Mozilla. (n.d.). *Learn to style HTML using CSS.* Retrieved February 9, 2021, from https://developer.mozilla.org/en-US/docs/Learn/CSS

Mozilla. (n.d.). *HTML: HyperText Markup Language.* Retrieved February 9, 2021, from https://developer.mozilla.org/en-US/docs/Web/HTML

Mozilla. (n.d.). *What is JavaScript?* Retrieved February 9, 2021, from https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

NPM. (n.d.). *React-mapbox-gl.* Retrieved November 14, 2021, from https://www.npmjs.com/package/react-mapbox-gl

OGC. (n.d.). *OGC Requests Web Map Tile Service (WMTS) Reference Implementations.* Retrieved February 4, 2021, from https://www.ogc.org/node/1395

OGC. (n.d.). *Open Geospatial Consortium.* Retrieved February 4, 2021, from https://www.ogc.org/

OGC. (n.d.). *Web Feature Service.* Retrieved February 4, 2021, from https://www.ogc.org/standards/wfs

OGC. (n.d.). *Web Map Service.* Retrieved February 4, 2021, from https://www.ogc.org/standards/wms

OpenLayers. (n.d.). *OpenLayers – Overview.* Retrieved November 14, 2021, from https://openlayers.org/

OpenStreetMap. (n.d.). *OpenStreetMap.* Retrieved November 20, 2021, from https://wiki.openstreetmap.org/wiki/Main_Page

Reactjs. (n.d.) *Introducing Hooks.* Retrieved December 9, 2021, from https://reactjs.org/docs/hooks-intro.html

React-Leaflet. (n.d.). *Introduction.* Retrieved November 14, 2021, from https://react-leaflet.js.org/docs/start-introduction/

Richard, S. & LePage, P. (2020). *What are Progressive Web Apps?* Retrieved February 2, 2021, from https://web.dev/what-are-pwas/

Stepanov, R. (2020). *Openlayers vs Leaflet: What Makes One Better Than the Other.* Retrieved November 23, 2021, from https://mapsvg.com/blog/openlayers-vs-leaflet

Sufiyan, T. (2021). *What is React: Definition, Why ReactJS, its Features and Installation.* Retrieved February 11, 2021, from https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs

W3Schools. (n.d.). *CSS Tutorial.* Retrieved February 9, 2021, from https://www.w3schools.com/css/

OpenStreetMap. (n.d.). *About OpenStreetMap.* Retrieved November 23, 2021, from https://wiki.openstreetmap.org/wiki/About_OpenStreetMap

OpenStreetMap. (n.d.). *History of OpenStreetMap.* Retrieved November 23, 2021, from https://wiki.openstreetmap.org/wiki/History_of_OpenStreetMap

Liite 1: Aineistonhallintasuunnitelma

Kehitysprojektin aikana pidetään päiväkirjaa (aineisto), johon kerätään teknistä tietoa projektista. Tämä tieto analysoidaan opinnäytetyötä varten. Päiväkirjaa säilytetään tekijän tietokoneen C-asemalla, ja siitä tehdään säännöllisesti varmuuskopioita OneDriveen. Päiväkirjaa säilytetään C-asemalla ainakin vuoden verran opinnäytetyön valmistumisesta. Kehitysprojektin aikana luodut sovellukset löytyvät opinnäytetyön tekijän GitHubista vähintään 2 vuotta opinnäytetyön valmistumispäivämäärän jälkeen, osoitteesta https://github.com/NestoriMe?tab=repositories

Opinnäytetyön tekemisessä ei ole ollut tarpeen käsitellä luottamuksellisia tietoja kuten esimerkiksi henkilötietoja tai muuta salassa pidettävää tietoa. Opinnäytetyö ei ole sisältänyt aineistoa, jota olisi tarpeen erikseen säilyttää tai tuhota.