Opinnäytetyö (AMK)

Tietojenkäsittely

2020

Johannes Luukkonen

MUSTA SURMA EUROOPASSA VUOSINA 1346–1353 ESITETTYNÄ REACT-SOVELLUKSELLA

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely

2020 | 52 sivua

Johannes Luukkonen

MUSTA SURMA EUROOPASSA VUOSINA 1346–1353 ESITETTYNÄ REACT-SOVELLUKSELLA

Historiallisen ajan tuhoisin rutto oli vuosina 1346–1353 riehunut kulkutauti, musta surma, joka huipentui Euroopassa vuosina 1347–1351. Tauti tappoi Euroopan väestöstä 30 prosentista 60 prosenttia – erään arvion mukaan 25 miljoona ihmistä – aiheuttaen mullistuksia keskiaikaisessa yhteiskunnassa.

Opinnäytetyön tarkoitus oli luoda vuorovaikutteinen karttasovellus kulkutaudin leviämisestä Euroopassa. Toteutuksessa käytettiin JavaScript-ohjelmointikielen React-kirjastoa, karttasovelluksiin kehitettyä ja Reactille muunnettua ohjelmointipakettia, siihen tehtyjä lisäosia sekä yleishyödyllistä React-projektialustusohjelmointipakettia. Mustan surman kulun lisäksi sovelluksessa esitellään sen aiheuttamaa hävitystä akateemispainotteiseen historialliseen tietoon viitaten. Sovelluksessa käytetty data on kerätty näistä historiallisista lähteistä.

Lopputulos on kattava sovellus, jolla käyttäjä pystyy rajaamaan taudin vuosittaista etenemistä ja maantieteellisiin kategorioihin eriteltyjä kauppareittejä ja kaupunkikohteita. Laaja lähdeluettelo takaa suuren määrän historiallista tietoa. Käyttäjä pystyy hallitsemaan sovellusta muuttamalla näytetyn tiedon määrää ja ohjelma on siten vuorovaikutteinen alusta.

Opinnäytetyön valmistumishetkellä (4.12.2020) sovellus sijaitsee osoitteessa https://sedrik1.github.io/musta-surma/ ja sovelluksen lähdekoodi osoitteessa https://github.com/sedrik1/musta-surma.

ASIASANAT:

React, React.js, karttasovellus, musta surma, vuorovaikutteinen oppiminen

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Bachelor's degree of Business Information Technologies

2020 | 52 pages

Johannes Luukkonen

THE BLACK DEATH IN EUROPE DURING THE YEARS 1346–1353 PRESENTED WITH A REACT APPLICATION

Known as The Black Death, the plague that cost Europe the lives of 30 to 60 percent of its population – an estimated 25 million people – is the most destructive plague in recorded history. The plague wreaked havoc during the years 1346–1353 and brought considerable ruin to Europe especially in 1347–1351, causing many changes to the medieval society.

The aim of the thesis was to create a map application which would track the spread of the pestilence in Europe. Implementation was done by using a JavaScript programming language library called React, a code package specifically converted to React, which is made for the creation of map applications, the package's third-party add-ons and a React project initialiser, which creates a generic starting platform.

In addition to recording the progress of The Black Death, the application presents the devastation left in its wake by utilising mainly academic sources. All of the application's data is gathered from these sources.

The finished product is a substantial application, in which the user can limit the disease's yearly spread and geographically organised trade routes and city locations. The ample source section provides a large amount of historical information. A user can engage with the application and control the displayed information, thus making it an interactive platform.

At the time of completion of the thesis (4.12.2020), the application is located at https://sedrik1.github.io/musta-surma/ and the source code of the application is at https://github.com/sedrik1/musta-surma.

KEYWORDS:

React, React.js, Map application, The Black Death, Interactive learning

SISÄLTÖ

KUVAT

KOMENTOESIMERKIT

KOODIESIMERKIT

KÄYTETYT LYHENTEET

CSS	porrastetut tyylitaulukot, merkintäkieliasiakirjojen, kuten hypertekstin merkintäkielen, ulkonäköä muokkaava tyylikieli, Cascading Style Sheets (Collins English Dictionary – Complete and Unabridged 2014)
DOM	asiakirjaoliomalli, joka on ohjelmointirajapinta hypertekstin merkintäkielille; ohjelmointirajapinta määrittää loogisen rakenteen asiakirjoille; Document Object Model (World Wide Web Consortium 1998)
HTML	hypertekstin merkintäkieli on verkossa nähtävän sisällön merkitsemiseen kehitetty kuvauskieli; Hypertext Markup Language (Dictionary of Unfamiliar Words by Diagram Group 2008)
JS	etupäässä – muttei ainoastaan – verkkokehityksessä käytetty komentosarjakieli, JavaScript (The Free Dictionary 2020)

1 JOHDANTO

Vuorovaikutteinen oppiminen edistää tiedon sisäistämistä ja soveltamista. Valitun aiheen tosiasioiden muistaminen on helpompaa ja tiedon hyödyntäminen käytännön tarkoituksissa tehostuu. Vuorovaikutuksessa oleva oppija kytkeytyy paremmin käsiteltyyn aiheeseen ja siten omaksuu kohtaamansa tiedon huomattavasti paremmin, kuin ollessaan passiivinen kuuntelija tai katsoja. Ero vuorovaikutteisen ja vuorovaikuttoman välillä ei ole mitätön; vuorovaikutuksessa oleva tarvitsee vähemmän aikaa asioiden muistamiseen ja tiedon soveltamiseen. (Gibbons & Evans 2007.)

Hakukoneella tekemieni kyselyiden perusteella aiheesta ei ole tehty yhtäkään laajaa sovellusta. Muutamia kuriositeettiomaisia on, mutta ne eivät käsittele taudin aiheuttamia seurauksia yksityiskohtaisesti ja havainnoivat sen levittäytymistä summittaisesti. Yksityiskohtaisinkin esimerkki esittää taudin leviämisen suhteellisen tarkasti, mutta kauppareitejä on vain muutama ja kaupunkikohtaisia tietoja on vain muutama (Maps.com). Kyseessä on täyttämätön rako niin sovelluskehityksen kuin myös tehokkaan opetustyökalun näkökulmasta.

Kulkutaudit ovat olleet osa ihmiskuntaa läpi sen historian. Koronapandemian myötä muinaiset kulkutaudit ovat varmasti monen mielenkiinnon kohteena. Sovellus tarjoaa hyvän mahdollisuuden tutustua menneinä aikoina tapahtuneeseen samankaltaiseen vitsaukseen. Nykyhetken rajoituksien takia on otollista, että sovelluksen lopullinen sijoituskohde on internet. Täten esimerkiksi karanteenissakin olevalla on esteetön pääsy palveluun.

Länsi-Aasiasta lähtöisin oleva kulkutauti levittäytyi kauppareittejä pitkin koko Eurooppaan. Vain Suomi ja Islanti säästyivät rutolta, mutta taudin laajuudesta kielii, että se ylettyi silloisesta Konstantinopolista Pohjolaan, kutakuinkin Norjan keskiosaan, saakka. (Nykänen 2010.) Historialliset lähteet ovat mittavat. Suhteellisen pienistä maa-alueista Euroopassa ja koko aihetta laajemmin käsittelevistä teksteistä saatu tieto on monipuolista ja varsinkin historiaan nojaavassa sovelluksessa välttämätöntä. Rajattua aluetta käsittelevä tieto tarjoaa yksityiskohtaisempia huomioita ihmisten käyttäytymiseen ja reagointia tautiin. Asiaa yleisemmin tarkastelevat tekstit antavat laajemman, mutta yksioikoisemman kuvan, kuin jotain Euroopan tiettyä osaa käsittelevä teksti.

Opinnäytetyön tarkoitus on luoda karttasovellus vuosina 1346–1353 Euroopassa riehuneen kulkutaudin, musta surma -nimisen ruttopandemian, ympärille. Sovelluksessa

esitellään taudin leviämistä, sen aiheuttamia tuhoja ja keskiaikaisia kauppareittejä. Samalla työ vastaa kysymykseen, miten vuorovaikutteisen karttasovelluksen rakentaminen toteutuu suunnittelu- ja ohjelmointitasolla. Opinnäytetyö on luonteeltaan konstruktiivinen.

Teoriaosuus käsittää luvut kaksi, kolme ja neljä. Toisessa luvussa kerrotaan käytetyistä tekniikoista niiden sovelluksen kannalta oleellisimmilta osilta. Kolmannessa luvussa sovellukselle suunnitellaan ilme ja tarkastellaan erilaisten muotojen käyttämistä tiedon esittämistä varten kartalla ja tekstinä esitettävän tiedon mahdollisuuksista. Neljännessä luvussa annetaan suppea seloste historiallisesta tiedosta ruttobakteeriin liittyen ja mustan surman historiasta.

Käytännön osuus on sovellussuunnitelman täytäntöönpanon dokumentointia. Ohjelmointiin liittyvät asiat selostetaan viidennessä luvussa. Sovelluskehitystä varten tarvittavat paketit asennetaan, luodaan hakemistorakenne, joka palvelee sovelluksen tarkoitusperiä ja jaotellaan tiedostot hakemistoihin. Myös sovellussuunnitelmassa mainittuja komponentteja tarkastellaan koodin kannalta ja selostetaan niiden käytännön tarkoitus. Komponentteja käytettäessä tuodaan ilmi, miten jotain tietoa esitetään ja mikä on käytännöllinen tapa suuren tietomäärän esittämistä varten.

Yhteenvedossa käsitellään opinnäytetyön lopputulemaa, onnistumista, hyötyä ja mahdollisia jatkokehityshankkeita sovellukseen liittyen. Lopputuloksen tarkastelulla tarkoitetaan sovelluksen kokonaistilaa eli toiminnallisuutta, käytettävyyttä, eheyttä ja sisältöä.

2 KÄYTETYT TEKNIIKAT

2.1 React

Opinnäytetyössä käytetyt tekniikat ovat JS-pohjaisia ja toimivat tukeakseen Reactin toimintaa. Tekniikat esitellään lyhyesti, ottamalla huomioon niiden oleellisimmat toiminnallisuudet ja hyödyt opinnäytetyössä tarvittavia käyttötarkoituksia varten. Mainituista toiminnallisuuksista ja ominaisuuksista esitellään projektikohtaisia käytännön esimerkkejä.

React, myös React.js ja ReactJS, on Facebookin kehittämä JS-ohjelmointikielen kirjasto (engl. library) – kokoelma esimerkiksi eri toiminnallisuuksia tarjoavia funktioita, luokkia tai aliohjelmia (Wozniewicz 2019) – käyttöliittymäkehitystä varten (Facebook 2020).

Reactin hyödyntäminen tekee verkkosovelluksessa käytetyn logiikan ja sen esityksen kytkemisestä toisiinsa todella luontevaa ja kehitysprosessista nopeaa. JS:n käyttäminen tavallisten HTML-elementtien tapaisesti tarkoittaa, että verkkosivulla näytetty tieto voi lähtökohtaisesti hyödyntää kaikkia ohjelmointikielen tarjoamia ominaisuuksia. Tavallisesti HTML- ja JS-tiedostot olisivat erillään, mutta Reactin johdosta yksi tiedosto pystyy hallinnoimaan omaa rakennettaan täysin itsenäisesti. (Hunt 2013.)

Tiedon päivittäminenkin toimii vaivattomammin Reactin ominaisuuksien johdosta. Tieto esitetään render-funktiossa ja koska kaikki esitetty tieto on käytännössä JS:ää, funktiota uudestaan kutsuttaessa ohjelman täytyy vain verrata uutta tietoa vanhaan; muuttunut tieto ottaa vanhan paikan, mutta muuttumaton pysyy ennallaan. Päivittäminen on tehty mahdollisimman kitsaaksi, jotta resursseja säästetään. (Hunt 2013.)

2.1.1 Reactin komponentit

Komponentti (engl. component) on käyttöliittymän pala. Käytännössä tämä tarkoittaa sivun ulkoasun muodustuvan monesta erillisestä osasta eli komponentista, jotka yhdessä luovat kokonaisuuden. Esimerkiksi sivun yläpalkki voi olla komponentti, joka pitää sisällään navigaatiokomponenttia, rekisteröitymislomakekomponenttia ja hakukenttäkomponenttia. Näidenkin komponenttien sisällä voisi olla lisää komponentteja, kuten haun suodattamista ja lomakkeen osioita varten. (Abramov 2015.)

Komponentti toimii siis ulkoisesti HTML-elementin tapaisesti, mutta käytännössä se on funktio (engl. function), jonka toiminnallisuutta ohjaavat sille annetut, mutta eivät välttämättömät, ominaisuudet (engl. props), komponentin omat muuttujat (engl. variables) ja funktiot. Ominaisuusargumentti on olio (engl. object), joka voi sisältää tietoa eli ominaisuuksia (engl. attributes) ja toiminnallisuutta eli metodeja (engl. methods). Metodit ovat oliokohtaisia funktioita. (Facebook 2020a.)

Komponentit jaotellaan kahteen ryhmään riippuen, onko niillä tilaa (engl. state) vai käytetäänkö niitä vain informaation esittämistä varten. Tilalliset (engl. stateful) ja tilattomat (engl. stateless) komponentit voivat olla luokalla (engl. class) toteutettuja tai Reactin versiopäivityksen 16.8 jälkeen funktiopohjaisia. (Abramov 2019.) Funktiopohjaiset komponentit ovat nimensä mukaisesti tavallisia JS-funktioita (Facebook 2020a).

2.1.2 Reactin elinkaarimetodit

Elinkaarimetodit ovat komponentin kolmen eri päävaiheen aikana suoritettavia funktioita. Vaiheet ovat seuraavat:

- komponentin luominen ja kiinnittäminen (engl. mounting) DOM:iin
- komponentin päivittäminen ja
- DOM:ista poistaminen (engl. unmounting).

Vaiheiden avulla jokaisen komponentin toimintaa pystyy hallitsemaan ennalta-arvattavissa vaiheissa, koska komponentit toimivat määrätyssä järjestyksessä ja niillä kaikilla on elinkaaren vaiheet. (Facebook 2020b.)

2.2 Leaflet ja React-Leaflet

Leaflet on avoimen lähdekoodin JS-kirjasto karttasovelluksien kehittämistä varten (Agafonkin 2020). Sovelluksessa käytetään React-Leafletiä, joka muuntaa Leafletin toiminnallisuuden React-komponenteiksi hyödyntäen Reactin elinkaarimetodeja (Le Cam 2020b).

Kirjastolla on laaja ohjelmointirajapinta, jonka yksi oleellisimmista osista projektin kannalta on – itse kartan lisäksi – geometristen muotojen sijoittaminen karttaan koordinaattitietojen avulla. Näitä muotoja voi muokata värin ja koon mukaan, muodostaa niistä

ryhmäkokonaisuuksia sekä antaa kullekin muodolle tai ryhmälle muotoja ponnahdusikkunan esimerkiksi lisätiedon esittämiseksi. (Agafonkin 2020.)

Leaflet tarjoaa jokseenkin niukasti käyttöliittymäelementtejä, mutta kolmansien osapuolten tekemät lisäominaisuudet täydentävät näitä puutteita (Le Cam 2020a). Sovelluksessa tullaan hyödyntämään muutamia kolmansien osapuolten lisäominaisuuksista.

2.3 Node.js

Node.js on avoimen lähdekoodin suoritusaikainen ympäristö (engl. runtime environment) palvelimella suoritettavaa JS:iä varten (Node.js 2020). Node.js mahdollistaa JS:n käytön niin selain- kuin palvelinpuolella, mikä tekee kummankin puolen kehittämisestä vaivatonta, koska kieltä ei tarvitse vaihtaa. Vaikka React ei vaadikaan omaa toimintaansa varten Node.js:ää, sitä hyödynnetään käytännöllisyytensä takia. Se on käyttövalmis kehityspalvelin ja mahdollistaa npm-pakettien vaivattoman asentamisen komentokehotetta käyttämällä.

2.4 npm ja npx

npm – kaikki kirjaimet pienellä – on pakettihallintaohjelma JS:lle ja oletusvalinta pakettihallintaan Node.js:lle (Node.js 2020). npm mahdollistaa paketeiksi kutsuttujen, pienten JS-ohjelmien asentamisen npm:n oman komentokehotteen kautta. Pakettien avulla oman ohjelman toiminnallisuutta pystyy laajentamaan valmiin koodin avulla. (npm.)

npx on npm:n 5.2-versiopäivityksessä esitelty pakettisuoritustyökalu, joka helpottaa npm-paketeissa olevien komentojen suorittamista ja riippuvuuspakettien asentamista automatisoimalla toiminnon (Pelu 2020).

3 SOVELLUKSEN SUUNNITTELU

3.1 Sovelluksen ulkoinen ilme

3.1.1 Käyttöliittymäosat

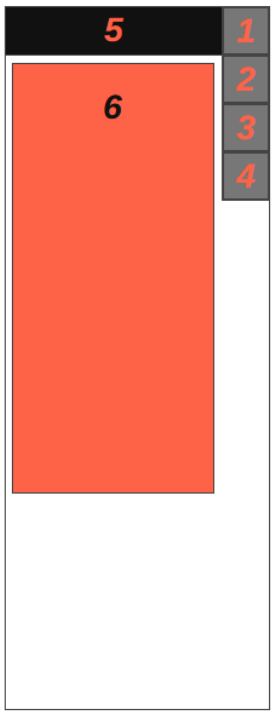
Karttasovellus koostuu kolmesta käyttöliittymäosasta: kartasta, sivupalkista ja tarkkuussäätimestä. Tarkkuussäädin on yksinkertainen, karttaa loitontava ja tarkentava yhteen- lasku- ja vähennyslaskumerkeistä muodostuva suorakulmio. React-Leaflet hoitaa automaattisesti sen liittämisen karttaan ja toiminnan, eikä siten vaadi lisäkehitystä.

React-Leafletissa ei ole sisäänrakennettua sivupalkkia, mutta tämän puutteen pystyy paikkaamaan kolmannen osapuolen tekemällä lisäosalla. Vain tyyliä ja projektikohtaisia muutoksia, esimerkiksi värien ja sivupalkin alaosioiden nimien muuttaminen, luukun ottamatta react-leaflet-sidetabs-npm-paketti tarjoaa hyvän käyttöliittymäosan.

Sivupalkki käsittää neljä alaosiota, jotka ovat seuraavat:

- karttataso-ohjain, josta voi valita, mitä elementtejä – esimerkiksi kauppareitit – kartalla näkyy; Leaflet tarjoaa sisäänrakennetun taso-ohjaimen, mutta se on toiminnallisuudeltaan liian yksinkertainen projektin tarpeita varten
- tieto-osio, joka jotain kaupunkikohdetta klikatessa näyttää sivupalkissa lisätietoa tautiin liittyen ja tiedon lähteet
- karttatietoa jossa on luettelo ja selitykset kartalla näkyvien elementtien väreistä ja lähteet kauppareiteille sekä taudin leviämiselle
- sovelluksen kaikki lähteet aakkosjärjestyksessä.

Neliöt yhdestä neljään ovat alaosioiden valintapainikkeita. Avatun osion nimi näkyy kohdassa viisi ja sisältö kohdassa kuusi (kuva 1).

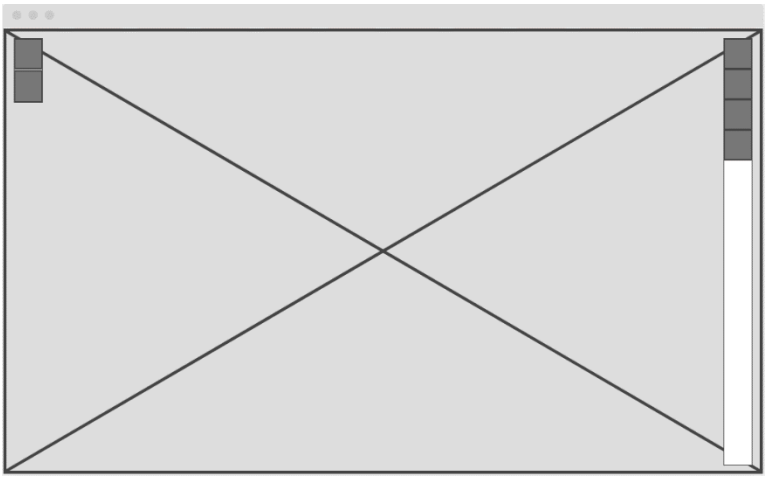


Kuva 1. Sivupalkin eri osat numeroituna.

Kartta, kuten tarkkuussäädinkin, on React-Leafletiin valmiiksi tehty ominaisuus. Ainoat säätöä vaativat ominaisuudet ovat kartan käyttämä kuvakokoelma, joka muodostaa käyttöliittymässä näkyvän kuvan kartasta sekä ensitarkennus Eurooppaan ja sopiva loitonnus. Kartta esittää kaiken visuaalisen tiedon, sivupalkki ja ponnahdusikkunat tekstitiedon. Pelkistetysti sovellus näyttää suorakulmioiden asetelmalta.

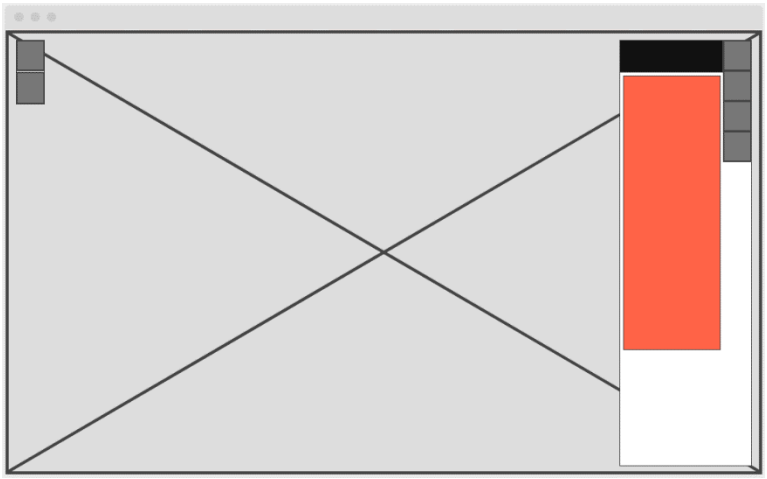
Sivukaaviolla sivun elementtien sommittelu tekee varsinaisesta työstä jouhevampaa, kun sovelluksen ilme on jo suunniteltu. Sivupalkki sijaitsee suljettuna oikeassa laidassa alaosiopainikkeineen. Vasemmassa yläkulmassa on tarkkuussäädin tarkennus- ja

loitonnuspainikkeineen. Suuri, taustalla oleva, ruksilla neljään osaan jaettu nelikulmio esittää karttaa (kuva 2).



Kuva 2. Sivukaavio sovelluksen sommittelusta.

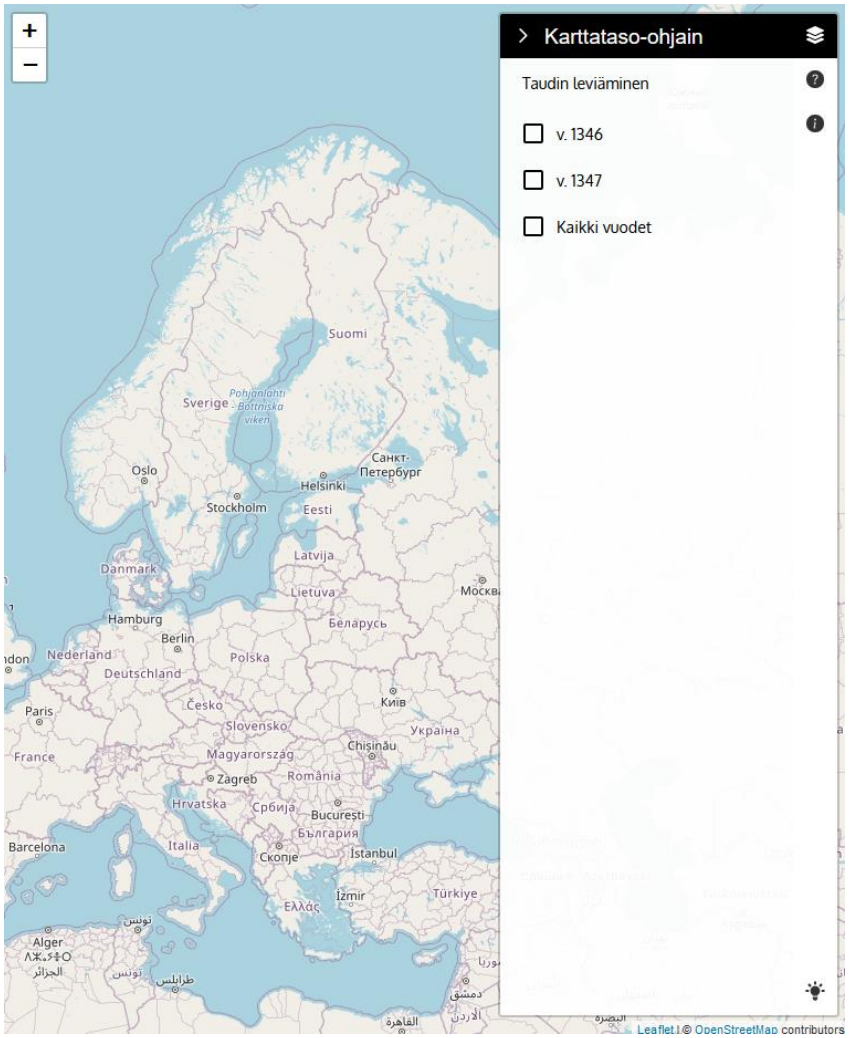
Avatun sivupalkin vasemmalle puolelle avautuu osio, jossa mustassa yläpalkissa näkyy valitun alaosion nimi. Oranssi neliskulma esittää avatun alaosion sisältöä (kuva 3).



Kuva 3. Sivukaavio sovelluksen sommittelusta sivupalkki avattuna.

Mittasuhteet saattavat muuttua jonkin verran oikean sovelluksen elementtien suhteen, mutta ne täsmäävät mallin määräämää sommittelua yleisilmeeltään melkein tarkalleen.

Taustalla on kartta, oikealla avattu sivupalkki alaosio-otsikoineen ja sisältöineen. Vasemmassa yläkulmassa on tarkennussäädin toimintonappeineen (kuva 4).



Kuva 4. Sivukaavion pohjalta tehty alustava esimerkki sovelluksen sommittelusta. Huomaa, että näkymä on tiivistettyssä muodossa.

Sivupalkin alaosioiden määrä saattaa muuttua, riippuen valintaominaisuuksista ja eri toiminnallisuuksien pilkkomisesta pienempiin osiin. Esimerkkinä eräs mahdollinen tapaus voi olla karttataso-ohjaimen valintojen jakaminen vaikkapa kahteen alaosioon.

3.2 Tiedon esittäminen

Sovelluksessa on huomattava määrä käyttäjälle välillisesti näkymätöntä dataa, lähinnä koordinaattitaulukkoja ja komponenttien välillä kulkevaa, sovelluksen toimintaa

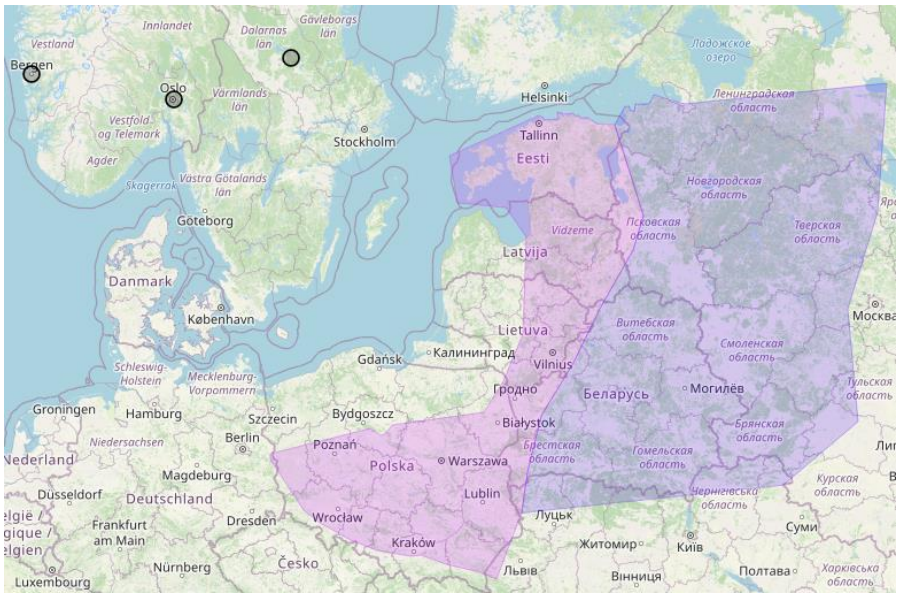
määrittävää tietoa. Tämä tieto määrittää, mitä käyttöliittymässä näkyy käyttäjän antaman syötteen perusteella.

Näkyvä tieto on tekstiä ja koordinaattien perusteella karttaan piirrettyjä muotoja.

Visuaalisesti esitetään seuraavasti:

- kulkutaudin leviäminen suurina, erivärisinä monikulmioina
- kauppareitit erimuotoisina viivoina; maa- ja merireitit erivärisinä
- kaupungit ympyröinä
- muut mahdolliset elementit.

Kaupunkien sijainteihin sopii CircleMarker-komponentti, joka FeatureGroup-komponentin alikomponenttina sopii ryhmäkohtaiseen esittämiseen. Polygon-komponentilla on käytännöllistä maalata kartalle mustan surman eteneminen (kuva 5).



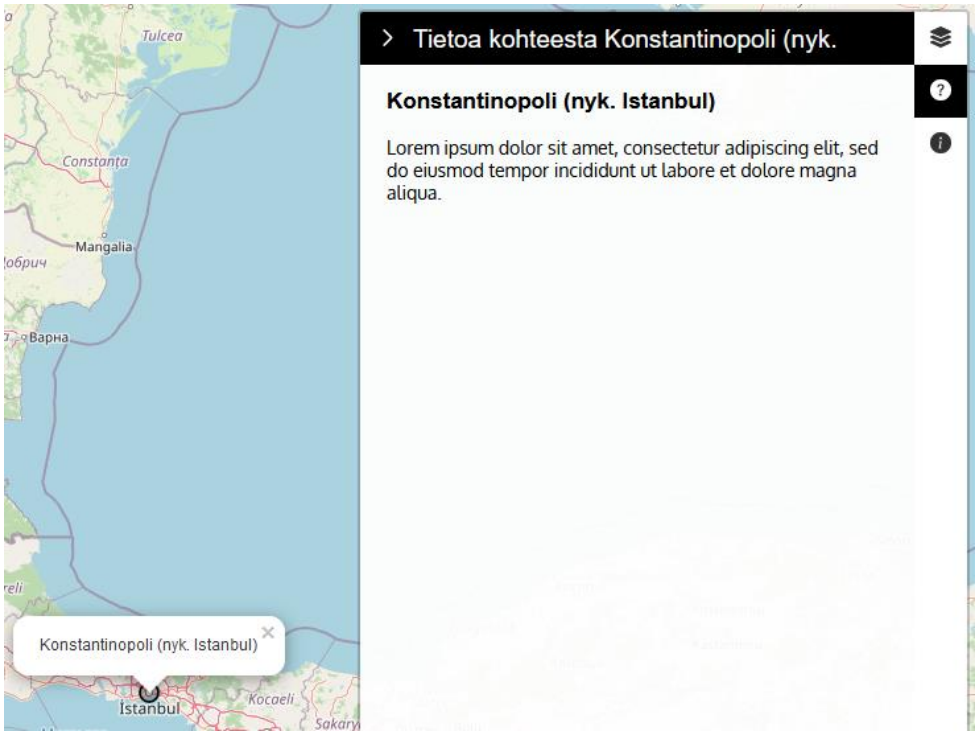
Kuva 5. Visuaalisten elementtien alustavia esimerkkejä. Purppuralla ja liilalla taudin leviäminen muutaman vuoden ajalta. Kolme kaupunkia mustilla ympyröillä.

Suorien viivojen tekeminen onnistuu ilman kolmansien osapuolten tekemiä lisäosia, koska suoran piirtäminen onnistuu Polyline-komponentilla React-Leafletissä. Kuitenkin kaarevien viivojen piirtämistä varten tarvitaan kolmannen osapuolen tekemä lisäosa react-leaflet-curve. Epäselvempiä reittejä, kuten kauppalaivojen kulkemia reittejä, varten kaarevat viivat soveltuvat paremmin kuin suorat viivat. Aika ajoin myös maareittejä

varten on hyvä käyttää kaarevia viivoja, jos kyseessä on vaikkapa hankalasti kuljettavan maaston läpi kulkeva väylä (kuva 6).

Kuva 6. Alustavia esimerkkejä kaariviivoilla tehdyistä maa- ja merikauppareiteistä omine väreineen.

Popup-komponentti eli ponnahdusikkuna antaa oivan mahdollisuuden esittää jotain kriittistä tietoa, kuten sijainnin tai kulkureitin nimeä. Kulkutaudin etenemistä esittävissä monikulmioissa olevissa ponnahdusikkunoissa olisi mahdollista esimerkiksi kertoa erittäin suppeasti ruton leviämislaajuudesta yhden vuoden aikana (kuva 7).



Kuva 7. Esimerkki toiminnallisuudesta kaupunkia klikatessa. Sivupalkki avautuu Tietoa-osiosta ja esittää lisätietoa kaupungista. Ponnahdusikkunassa kaupungin nimi.

Tekstinä esitetään tarkemmat kuvaukset taudista, sen aiheuttamat seuraukset, muut mahdolliset lieveilmiöt ja visuaalisen tiedon selitteet eli ponnahdusikkunat.

4 HISTORIALLINEN TIETO

4.1 Yersinia pestis -bakteerin aiheuttama rutto

Vaikka musta surma onkin tuhoisin tunnettu ruttoepidemia, sitä ennen on ollut muitakin Yersinia pestis -bakteerin aiheuttamia kulkutauteja. Bakteeri on ollut piikki ihmisen lihassa jo tuhansien vuosien ajan; ensimmäinen tunnettu tapaus ilmeni 3000 eaa. (Kjaer 2019.) Taudinaiheuttaja on siis ehtinyt riehua tavalla tai toisella pronssikaudesta nykyaikaan saakka eri puolilla maapalloa (Frith 2012). Esimerkki lähihistoriassa tapahtuneesta ruttoepidemiasta on syyskuussa 2014 Madagascarilla puhjennut tautiaalto (WHO 2014).

Tautiin sairastuneen kehoon kasvoi paiseita, erityisesti nivusten ja kainaloiden alueelle (Frith 2012). Ruttotautinen joutui kärsimään erittäin kivuliaista oireista, sillä he päätyivät muun muassa kuumeen, kouristuksien, oksentelun ja keuhko-oireiden runtelemiksi. Äärreisverenkierrossa bakteeri voi aiheuttaa laikkuja ihon alle. (Nykänen 2010.)

4.2 Musta surman kulkeutuminen Eurooppaan

Vuosina 1346–1353 riehunut ruttopandemia tunnetaan nykyään nimellä musta surma, mutta aikalaiset eivät kutsuneet tautia sillä nimellä. Sanomattoman kurjuuden koettelemat nimittivät tautia nimillä, kuten ”mortalitas” ja ”epidemia”. Kulkutaudin säälimättömästä ankaruudesta kertoo nimen yhteydessä käytetty lisänimi ”mitä ankarin”. Nykyaikana käytössä olevalle nimelle juontaa juurensa saksalaisen lääkärin, J.F.C. Hecklerin vuonna 1832 julkaiseman esseen ”Der Schwarze Tod” rubriikista. Kuitenkin aikaisempia havaintoja nimestä löytyy: Englannissa melkein kymmenen vuotta ennen Hecklerin esseetä käytettiin taudista nimeä ”The Black Death”. Huomattavasti ennen kumpaakaan kahta edellä mainittua esimerkkiä, kutsuttiin 1500-luvun Ruotsissa ruttoa ”Swarta döden” -nimellä. (Heikura 2003.)

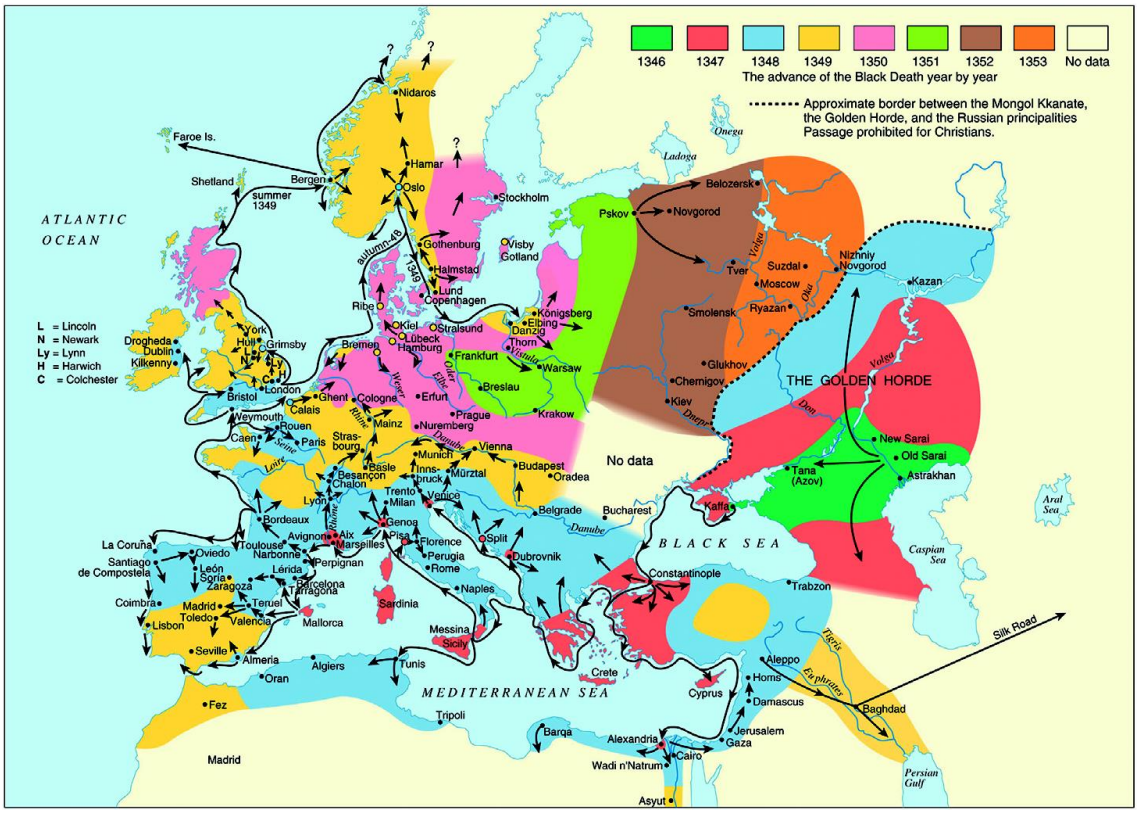
Euroopan läpi pyyhkäisseen ruton alkuperä on Keski-Aasiaan aroilla, joilla tauti kyti jyrsijöissä. 1300-luvulla rutto levisi Kiinaan ja Intiaan, sieltä todennäköisesti Turkestaniin ja vielä eteenpäin Uzbekistaniin, Persiaan ja lopuksi Krimin niemimaalle. (Heikura 2003.)

Eräs taudin leviämistä edesauttanut asia olivat Kaukoidässä tapahtuneet luonnonilmiöt. Tulvat ja maanjäristykset tuhosivat jyrsijöiden pesät, aiheuttaen niille pakottavan tarpeen

siirtyä uusille asuinalueille. Tämä muuttoliike johti jyrsijät ihmisten asuttamille seuduille ja siten näiden alueiden jyrsijöistä tuli myöskin rutonkantajia. (Heikura 2003.)

Musta surma kulki Silkkitietä pitkin mongolien maailmanvallan seuraajavaltion, Kultaisen ordan alueelle Etelä-Venäjälle (kuva 8). Genovalaisilla oli tähän aikaan vuonna 1346 Krimin niemimaalla Kaffan, nykyisessä Feodosijan, kaupungissa tukikohta tuotantotehtaalla ja kauppapaikalla varustettuna. Ympäröivä alue kuitenkin kuului Kultaiselle ordalle ja vuonna 1343 Kultaisen ordan armeija ajoi italialaiset kauppiaat pois tukikohdastaan Tanasta, joka nykyään tunnetaan Azovin kaupunkina. Henkeään varjellakseen kauppiaat turvautuivat maanmiestensä genovalaisten apuun ja pääsivät suojaan Kaffan tukikohtaan. (Benedictow & ym. 2016.)

Kaffa joutui itsekin myöhemmin piirityksen kohteeksi, ja sitä piirittävä armeija päätti hyötykäyttää ruttoon menehtyneiden ruumit sotilastarkoitusta varten: vainajat heitätettiin katapultein muurin yli. Seurauksena kaupunki saastui ja sen asukkaat sairastuivat. (Snell 2019.)



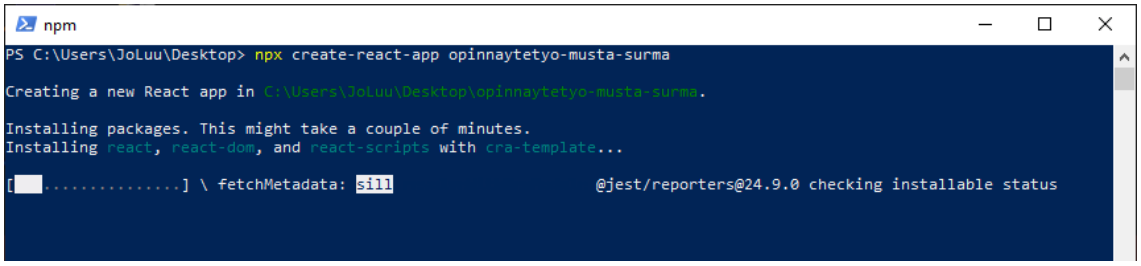
Kuva 8. Eräs sovelluksen merkittävimmistä tiedonlähteistä. Mustan surman eteneminen, Kultaisen ordan hallitsemat alueet ja senaikaisia – ylimalkaisesti merkittyjä – maa- ja merikauppareittejä (Benedictow & ym. 2016).

Vuonna 1347 genovalaiset laivat lähtivät mitä suurimmalla todennäköisyydellä kohti Konstantinopolia, vieden mukanaan kulkutaudin ja siten saattaen alulle Eurooppaa koetelleen pandemia-aallon (Benedictow & ym. 2016).

5 SOVELLUKSEN SUUNNITELMAN TOTEUTTAMINEN

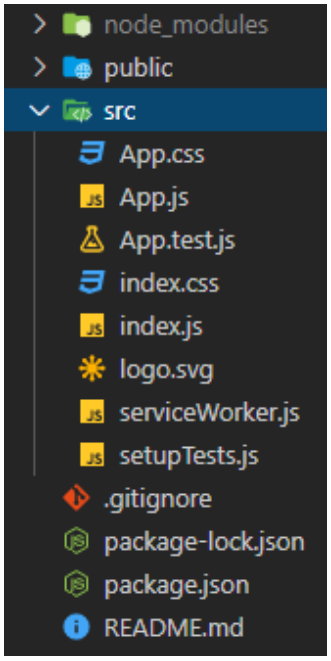
5.1 npm-pakettien esiasennukset

Node.js:n lisäksi tarvitaan npm-paketti create-react-app, jonka avulla suoritetaan alustus projektille. npx:ää hyödyntämällä alustus hoituu käden käänteessä: create-react-app (komentoesimerkki 1) luo React-projektia varten tarvittavat tiedostot, niille sisällön ja hakemistorakenteen rungon (Abramov 2016).



Komentoesimerkki 1. npx-komennolla create-react-app-nimisen npm-paketin suorittaminen luo valmiin pohjan React-sovellukselle.

Paketin luoman hakemiston avulla työnteko nopeutuu, koska välttämättömät mutta standardinomaiset toimet saa automatisoitua (kuva 9).



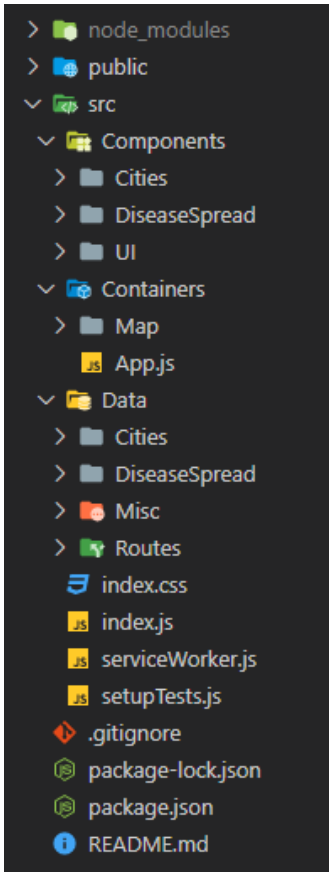
Kuva 9. create-react-app-npm-paketin luoma hakemistorakenne ja esimerkkitiedostot.

Kaikki ohjelmointiin liittyvä tulee tapahtumaan src-hakemiston sisällä. Ulkoiset komentosarjatiedostot, joita ei voi hakea npm:n kautta, ja tyylittelytiedostot tullaan lisäämään public-hakemistossa olevaan index.html-tiedostoon.

Jäljellä olevien npm-pakettien react-leaflet, react-leaflet-curve ja react-leaflet-sidetabs asentaminen toimii samalla tavalla kuin create-react-app-paketin asentaminen sillä muutoksella, että asentamiseen käytetään npm-komentoa.

5.2 Projektin hakemistorakenne

Ylähakemiston rakenne noudattaa tiedostojen tarkoituksia. Esimerkiksi kaikkien vain dataa sisältävien tiedostojen niputtaminen yhteen hakemistoon ja mahdollisiin alihakemistoihin selkeyttää projektissa navigoimista ja tiedostojen löytämistä. Components-hakemistossa tietoa esittävät komponentit, Containers-hakemistossa loogisia komponentteja ja Data-hakemistossa tietoa, esimerkiksi koordinaatteja, sisältäviä tiedostoja. UI-hakemistossa on sivupalkki ja sen alikomponentit (kuva 10).



Kuva 10. Hakemistorakenne erittelee tiedostot niiden ominaisuuksien mukaan.

Heti alussa määritelty rakenne tekee ohjelmistokehityksestä huomattavasti vaivattomampaa:

- tiedon ylläpitäminen on helppoa, koska sen etsiminen rajoittuu yhteen paikkaan.
- uuden tiedon, vaikkapa koordinaattitiedon, lisääminen johonkin kohteeseen vaikuttaa kaikkiin tietoa käyttäviin sijainteihin. Jokainen tietoa hyödyntävä tiedosto saa automaattisesti, ilman yksittäistä tiedostomuokkaamista, uuden tiedon.
- mahdollinen projektiin tutustumaton ohjelmoija ei joudu viettämään yhtä paljon aikaa tiedostojen etsimiseen, kuin jos ne olisivat hajautetusti ja järjestelemättä.

5.3 Kartta ominaisuuksineen

Karttaa varten tarvitaan Leafletin toimintaa määrittävä JS-tiedosto ja tyyliä määrittävä CSS-tiedosto. Nämä tiedostot saa hankittua unpkg-sisällönjakeluverkko avulla (koodiesimerkki 1).

```
<link
    rel="stylesheet"
    href="https://unpkg.com/leaflet@1.6.0/dist/leaflet.css"
    integrity="sha512-xwE/Az9zrjBIphAcBb3F6JVqxf46+CDLwfLMHloNu6KEQCAWi6HcDUbeOfBIptF7tcCzusKFjFw2yuvEpDL9wQ=="
/>
<script
    src="https://unpkg.com/leaflet@1.6.0/dist/leaflet.js"
    integrity="sha512-gZwIG9x3wUXg2hdXF6+rVkLF/0Vi9U8D2Ntg4Ga5I5BZpVkVxlJWbSQtXPSiUTtC0TjtGOmxa1AJPuV0CPthew=="
></script>
```

Koodiesimerkki 1. Leaflet-kirjaston CSS- ja JS-tiedostojen tuonti unpkg-sisällönjakeluverkkoa käyttäen, liittäen ne head-elementtin.

Koodiresurssit sijoitetaan index.html-tiedoston head-elementtiin, joka sisältää HTML-tiedoston metadatan eli esimerkiksi tyyli- ja komentosarjatiedostojen sijainnit, tiedoston otsikon ja kuvauksen sekä käytetyn koodisivun.

Pääkomponentti on MapContainer-niminen luokka, joka pitää sisällään sovelluksen loogisen rakenteen. Index.js-niminen tiedosto on ylin JS-tiedostojen hierarkiassa. Se hahmottaa (engl. render) MapContainer-komponentin ja kaikki sen alikomponentit. Index.html-tiedostossa oleva id-ominaisuudella varustettu HTML-elementti toimii Reactin hahmotuskohteena (koodiesimerkki 2).

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import Map from './Containers/Map/Map';
import * as serviceWorker from './serviceWorker';

ReactDOM.render(
  <React.StrictMode>
    <Map />
  </React.StrictMode>,
  document.getElementById('root')
);

serviceWorker.unregister();
```

Koodiesimerkki 2. Index.js-tiedosto toimii komponenttien hahmottamista varten.

Jotta suunnitelmassa linjattuja karttaominaisuuksia voidaan käyttää, pitää react-leaflet-paketista tuoda paljon komponentteja. Osaa tarvitaan kartan esittämiseen, toisia muotojen luomiseen ja osaa kartan toimintaa varten. Kaarisulkujen sisällä olevat nimet ovat komponenttien nimiä, mutta ne voivat olla myös muuttujia tai peitenimiä. Kaikki tuonti

tiedostoon toimii import-from-tyyppisesti, mutta kaarisulkuja ei aina tarvita. Myöhemmin funktiopohjaisen komponentin koodiesimerkissä komponentti viedään oletusarvona, jolloin tuonti tiedostoon vaatii vain nimen (koodiesimerkki 3).

```
import {
    FeatureGroup,
    LayersControl,
    Map,
    Popup,
    TileLayer,
    Polyline,
} from 'react-leaflet';
```

Koodiesimerkki 3. react-leaflet-npm-paketista tuodut komponentit sijoitetaan MapContainer-komponenttiin.

Kartan alustamiseen tarvitaan Map- ja TileLayer-komponentteja. Map-komponentille määritetään tarkennuskohde antamalla center-ominaisuudelle koordinaatit taulukkona (engl. array) ja tarkennustaso zoom-ominaisuudella. TileLayer sisältää itse karttakuvat, joka annetaan yhtenä kokonaisuutena url-ominaisuudessa linkin muodossa (koodiesimerkki 4).

```
<Map className="map-style" center={[56.1, 23.106111]} zoom={4}>
    <TileLayer
        attribution='
        &copy; <a href="https://stadiamaps.com/">Stadia Maps</a>,
        &copy; <a href="https://openmaptiles.org/">OpenMapTiles</a>
        &copy; <a href="http://openstreetmap.org">OpenStreetMap</a> contributors'
        url="https://tiles.stadiamaps.com/tiles/alidade_smooth/{z}/{x}/{y}{r}.png"
    />
</Map>
```

Koodiesimerkki 4. Map- ja TileLayer-komponentit MapContainerissa.

Jäljellä olevat react-leaflet-paketista tuodut komponentit ja react-leaflet-curvesta tuotu Curve-komponentti tarvitaan silmukoilla luotavien kokonaisuuksien esittämiseen eli muotojen luomista varten. Lopulta kaikki esitettävät muodot sijoitetaan LayersControllin sisälle. LayersControl on varsinaisesti tarkoitettu hallinnoimaan kartan eri tasoja, mutta Polyline ja Curve vaativat sijoittamista sen sisälle, jotta nämä komponentit voidaan hahmottaa kartalle. FeatureGroup ja Polygon voitaisiin sijoittaa suoraan Map-komponentin sisälle, mutta yhdenmukaisuuden vuoksi nekin sijoitetaan LayersControliin.

MapContainer sisältää sovelluksen ainoan tilan eli state-olion. Olio määrittelee sivupalkin toimintaa, eli onko se auki vai kiinni, ja mikä alaosio on valittu ja kaiken tiedon, joka esitetään kartalla ja sivupalkissa. DisplayElements-olio pitää sisällään kaikki kartalla esitettävät muodot jaoteltuna ylempiin kenttiin, joilla voi määritellä kokonaisen alaolion näkyvyyden, itse alaoliot ja viimeisenä on kenttä Kultaisen ordan rajan esittämistä varten. DisplayElementsin viereiset kentät infoboxLocationista selectediin määrittävät sivupalkkissa esitettävien tekstitietojen hallintaa (koodiesimerkki 5).

```
class MapContainer extends Component {
    state = {
        infoboxLocation: '',
        infoboxLocationInfo: '',
        reference: [],
        collapsed: false,
        selected: 'layers',
        displayElements: {
            displayRoutes: false,
            displayAllTerrain: true,
            displayDiseaseSpread: false,
            displayGoldenHorde: false,
            cities: {
                displayMediterranean: false,
                displayMainlandWest: false,
                displayMainlandEast: false,
                displayNorthEurope: false,
                displayBritain: false,
            },
            routes: {
                MediterraneanRoutes: false,
                WestEuropeRoutes: false,
                EastEuropeRoutes: false,
                NorthEuropeRoutes: false,
                BritainRoutes: false,
                SpainRoutes: false,
            },
            disease: {
                displayDiseaseSpread_1346: false,
                displayDiseaseSpread_1347: false,
                displayDiseaseSpread_1348: false,
                displayDiseaseSpread_1349: false,
                displayDiseaseSpread_1350: false,
                displayDiseaseSpread_1351: false,
                displayDiseaseSpread_1352: false,
                displayDiseaseSpread_1353: false,
            },
        },
    };
```

Koodiesimerkki 5. MapContainer-komponentin state-olio.

State-olion sisällä on huomattava määrä kenttiä: Disease-alaoliossa vuodet, routes-ala-oliossa kauppareittejä, cities-alaoliossa kaupunkeja ja kaikkia näitä ylemmällä tasolla kentät määrittämään, näytetäänkö kaikkien alaosioiden sisällöt vai ei.

Tilan määrittäminen on huomattavasti helpompaa, kun kaikki siihen liittyvä tieto on yhdessä komponentissa. Tietoa voi kätevästi jakaa eteenpäin ja päivittää alikomponenteista käsin. Tällä tavalla sovelluksessa on vain yksi ”oikea” tila, eikä ongelmia väärien tilatietojen kanssa tule.

5.4 Visuaalisen tiedon esittäminen

Suuri osa Reactin toiminnallisuudesta tulee kätevästi esiteltyä yhden komponentin kautta. Luonnollisesti tämänkin jälkeen jää osia, jotka vaativat omat selityksensä. Kulkutaudin etenemisessä käytettyjä ja esiteltyjä funktioita, tiedontallennustapoja ja komponentteja tullaan hyödyntämään monessa eri paikassa.

5.4.1 Mustan surman eteneminen

Yksi vuosi taudin etenemisessä muodostaa luontaisesti käytännöllisen tavan määrittää komponentit koko. Sama logiikka pätee kaikkiin vuosia esittäviin komponentteihin.

Taudin etenemistä koskeva tieto keskitetään yhteen tiedostoon, DiseaseSpreadInfo.js, josta tuodaan kutakin vuotta varten yksi olio. Oliossa on väri vuoden erottamiseksi muista vuosista, vuosi merkkijonona (engl. string), jota hyödynnetään taso-ohjaimessa ja koordinaatit monikulmion muotoa varten. Suuren pituuden takia esimerkissä esitetään vain alkuosio vuoden 1346 koordinaattitaulukosta, joka on matriisi eli kaksiulotteinen taulukko. Kaksiulotteisuus tarkoittaa, että taulukon sisällä on toinen taulukko. Taulukoista tulee helposti yli satojen rivien pituisia, monimatriisisia kokonaisuuksia (koodiesimerkki 6).

```
const coordinates_1346 = [
    [
        [47.25211, 39.192151],
        [47.404391, 39.357795],
        [47.881355, 41.613548],
        [49.02076, 42.576846],
        [50.241935, 44.714153],
        [50.270032, 45.439724],
        [50.192726, 45.945425],
        [49.853923, 46.209269],
        [49.075665, 46.539074],
        [48.786962, 47.275638],
        [48.5257, 48.25363],
```

Koodiesimerkki 6. Rajattu osuus vuoden 1346 koordinaateista.

Tieto eritellään vuosiksi ja lopulta kaikki muuttujat viedään tiedostosta. DiseaseYears on oletusvalinta, kun tiedosto tuodaan johonkin toiseen tiedostoon (koodiesimerkki 7).

Vuosimerkkijonoa käytetään tiedon esittämisen automatisoimisessakin. Yksinkertaistettuna automatisointi tarkoittaa tässä tilanteessa silmukan (engl. loop) luomista, joka käy jokaisen sille annetun arvon läpi. Silmukkoja käyttämällä ohjelmointimäärä vähenee huomattavasti.

```
const DiseaseYears = [
    { coordinates: coordinates_1346, colour: '#ff0000', date: 'v. 1346' },
    { coordinates: coordinates_1347, colour: '#f79f66', date: 'v. 1347' },
    { coordinates: coordinates_1348, colour: '#fc5d23', date: 'v. 1348' },
    { coordinates: coordinates_1349, colour: '#fc2356', date: 'v. 1349' },
    { coordinates: coordinates_1350, colour: '#fc23c2', date: 'v. 1350' },
    { coordinates: coordinates_1351, colour: '#c923fc', date: 'v. 1351' },
    { coordinates: coordinates_1352, colour: '#5d23fc', date: 'v. 1352' },
    { coordinates: coordinates_1353, colour: '#2356fc', date: 'v. 1353' },
    { coordinates: [], colour: '', date: 'Kaikki vuodet' },
];

const v1346 = DiseaseYears[0];
const v1347 = DiseaseYears[1];
const v1348 = DiseaseYears[2];
const v1349 = DiseaseYears[3];
const v1350 = DiseaseYears[4];
const v1351 = DiseaseYears[5];
const v1352 = DiseaseYears[6];
const v1353 = DiseaseYears[7];

export {
    DiseaseYears as default,
    v1346,
    v1347,
    v1348,
    v1349,
    v1350,
    v1351,
    v1352,
    v1353,
};
```

Koodiesimerkki 7. DiseaseYears-muuttuja sisältää kaikki vuositiedot koottuna.

Vuosikohtaiset tiedot jaotellaan taulukon hakemistonumeroin. Lopulta kaikki tieto viedään tiedostosta. Jäljelle jää funktiopohjaisen komponentin luominen. Siihen tuodaan Polygon- ja Popup-komponentit, jossa Polygon-komponentin color-ominaisuudeksi määritetään tuodun vuosikohtaisen tiedon väriarvo ja positions-ominaisuudelle koordinaattitiedot. Popup-komponentille annetaan esitettäväksi vuosiluku. Funktiokomponentti palauttaa monikulmion, jossa on ponnahdusikkuna. Komponentti viedään oletusarvona, eli sitä ei tarvitse sijoittaa kaarisulkuihin tuontivaiheessa (koodiesimerkki 8).

```
import React from 'react';
import { Polygon, Popup } from 'react-leaflet';
import { v1346 } from '../../Data/DiseaseSpread/DiseaseSpreadInfo'

const V1346 = () => {
    return(
        <Polygon color={v1346.colour} positions={v1346.coordinates}>
            <Popup>{v1346.date}</Popup>
        </Polygon>
    );
};

export default V1346;
```

Koodiesimerkki 8. Vuoden 1346 funktiokomponentti.

Komponentti on valmis vietäväksi MapContainer-komponenttiin, jossa pitää tehdä joitakin alustuksia ennen, kuin komponentin palauttama monikulmio voidaan lopultakin esittää kartalla. Esittämistä varten tilassa oleva Boolen arvo, displayDiseaseSpread_1346, määrittää, näkyykö monikulmio kartalla vai ei. Boolen arvo on totuusarvo, eli arvo on totta (engl. true) tai ei-totta (engl. false).

Boolen arvoa voidaan muuttaa sivupalkissa olevan valintaruudun avulla. Valintaruutu kutsuu funktiota, handleLayerControl, johon se syöttää argumenttina halutun tason arvon, event.target.value, merkkijonona ja muuttaa tilaa määrätyllä tavalla (koodiesimerkki 9).

```
onClick={event => {
    handleLayerControl(
        event.target.value
    );
}}
```

Koodiesimerkki 9. Elementissä oleva funktio, joka suoritetaan valintalaatikkoa klikkaamalla.

SetState-funktio päivittää tilan sille annetulla arvolla. Jos setStatelle ei antaisi spread-operaattorilla (engl. operator), eli kolmella pisteellä alustettuja, jo tilassa olevia arvoja, tila ei päivittyisi oikein. Spread-operaattori kentän edessä kopioi arvon, mutta vain jos kyseessä on taulukko tai olio. Tilaolion muut tiedot halutaan säilyttää sellaisinaan pois lukien arvo, joka nimenomaan halutaan muuttaa (koodiesimerkki 10).

```
handleLayerControl(layerName) {
    switch (layerName) {
        case 'Disease_v. 1346':
            this.setState({
                displayElements: {
                    ...this.state.displayElements,
                    disease: {
                        ...this.state.displayElements.disease,
                        displayDiseaseSpread_1346: !this.state
                            .displayElements.disease
                            .displayDiseaseSpread_1346,
                    },
                },
            });
            break;
```

Koodiesimerkki 10. Karttataso-ohjaimen funktio, jossa on eri tasovaihtoehtoja elementtejä, kuten vuotta 1346, varten.

Map-komponentin sisällä oleva years-muuttuja pitää sisällään olioissa kaikki vuosien funktiokomponentit ja MapContainer-komponentin tilassa määritetyn Boolen arvon kullekin vuodelle (koodiesimerkki 11).

```
let years = [
    {
        year: <V1346 key="V1346" />,
        value: this.state.displayElements.disease
            .displayDiseaseSpread_1346,
    },
```

Koodiesimerkki 11. Years-muuttuja on taulukko olioita, missä yksi olio sisältää yhden vuoden tiedot.

Map-funktio käy jokaisen taulukon arvon läpi, tarkistaa kunkin arvon annetun ehdon mukaan ja palauttaa uuden taulukon ehdon läpäisseillä tuloksilla muuttamatta alkuperäistä taulukkoa.

Muuttujien välissä näkyvä ?:-yhdistelmä on kolmivaiheinen operaattori (engl. ternary operator), joka toimii samalla periaatteella, kuin ehtolauseet (engl. conditional expressions). Huutomerkin käyttäminen muuttujan edessä tarkoittaa negaatiota (engl. not), eli esimerkiksi !steep olisi ei-jyrkkä.

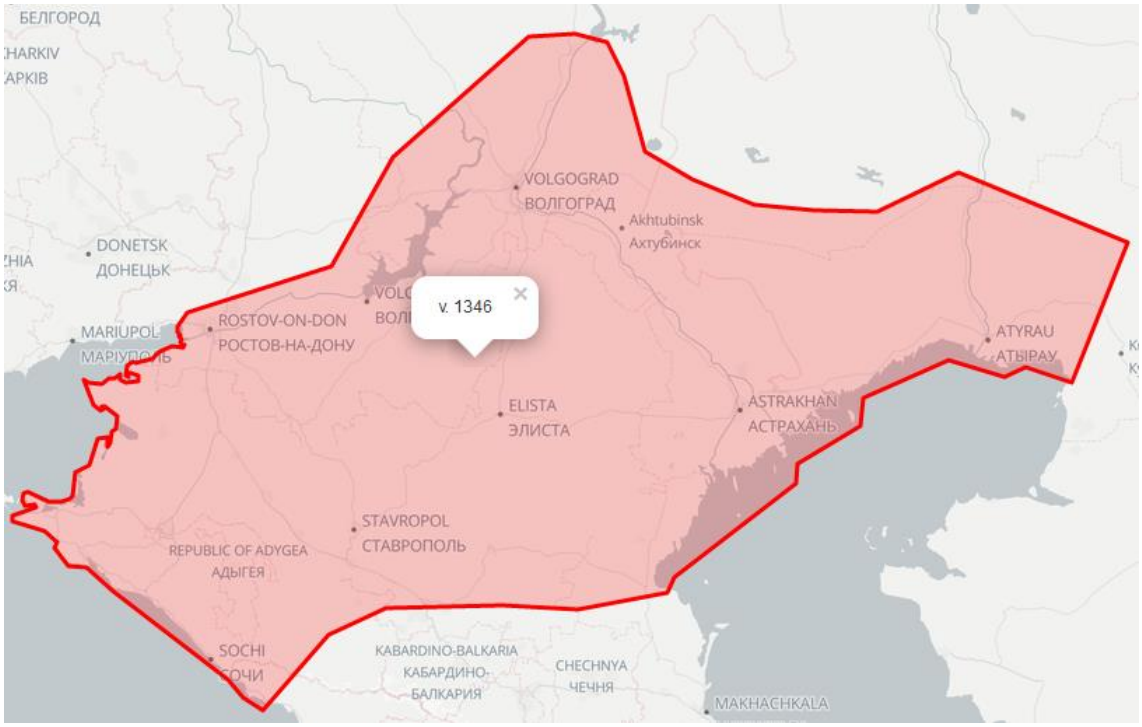
Kolmivaihe eriteltynä:

- ehto tarkistaa onko tilaoliossa oleva kaikkien vuosien Boolen arvo totta vai ei. Jos ei niin tarkastetaan, onko yksittäisen vuoden Boolen arvo, eli value-muuttuja, totta. Ehdon täyttyessä vuosi näytetään.
- kaikkien vuosien Boolen arvon ollessa totta jokainen vuosi näytetään huolimatta niiden omista Boolen arvoista.
- jos kumpikaan ehdoista ei täyty, ei näytetä mitään (koodiesimerkki 12).

```
{!this.state.displayElements.displayDiseaseSpread
    ? years.map(({ year, value }) =>
          !value ? false : year
    )
    : this.state.displayElements.displayDiseaseSpread
    ? years.map(({ year }) => year)
    : false}
```

Koodiesimerkki 12. Map-funktio, jolla vuosien esittäminen tapahtuu.

Polygonin sulkiessa itsensä, eli palatessaan lähtöpisteen koordinaatteihin, se värjää rajojen sisään jääneen alueen DiseaseSpread-muuttujassa olleen värikentän arvon mukaan (kuva 11).



Kuva 11. V1346-funktiokomponentin tuottama lopputulos ponnahdusikkunan kera.

Vaikka vuoden 1346 koordinaattitiedot koostuvat vain yhdestä matriisista, voivat tiedot sisältää periaatteessa niin monta matriisia, kuin vain tarve vaatii. Ainoa ehto on, että jokainen matriisi on erikseen sijoitettu yhden, kaikki matriisit sisältävän taulukon sisälle.

5.4.2 Kaupungit

Kaupunkien esittämiseen ei tarvita lainkaan yhtä paljon dataa, kuin mitä vuodet vaativat. Kuitenkin datan keskittämisperiaate on sama, eli kaupunkitietojen vienti ja tuonti noudattaa vuosien esittämissä käytettyjen komponenttien linjaa. Kaupungit on jaettu maantieteellisiin ryhmiin. Yksi maantieteellinen kokonaisuus on taulukko, jossa on olioita. Olio pitää sisällään kaupungin nimen, sijainnin, historiallisen tiedon viitteet taulukossa ja kaupunkikohtaista tietoa, joka esitetään sivupalkissa (koodiesimerkki 13).

```
{
  reference: [A, CBC, TK],
  location: "San Gimignano",
  coordinates: [43.468, 11.042],
  info: `Musta surma aiheutti Euroopassa järisyttäv
  Kunnan asukasluku oli vuoden 2018 lopulla 8149 he
  Yksinkertaisella prosenttilaskulla, 8149 / 100 *
  Vuonna 2019 Suomessa elävänä syntyneitä oli 45 61
},
```

Koodiesimerkki 13. Esimerkki San Gimignanon kaupungista Länsi-Eurooppa-taulukossa.

Kaupunkia esittävän komponentin luominen toimii muuten samalla tavalla kuin vuosikomponentti. Poikkeuksena on, että funktiokomponenttiin viedään argumentilla yksi funktio ominaisuudeksi. Funktio on handleInfoboxClick ja se saa argumenttina olion, jossa on kaupungin nimi, kaupunkitieto ja kaupunkitiedon historialliset lähteet (koodiesimerkki 14). Funktion toiminta käydään läpi sivupalkkiluvussa.

```
import React from 'react';  8.3K (gzipped: 3.3K)
import { Popup, CircleMarker } from 'react-leaflet';  12.4K (gzipped: 3.9K)
import { WestEurope as data } from '../../../Data/Cities/LocationData';

const MainlandWest = props => {
    return data.map(({ reference, info, location, coordinates }, index) => {
        return (
            <CircleMarker
                key={index}
                onClick={() =>
                    props.handleOverlayClick({ location, info, reference })
                }
                center={coordinates}
                radius={7}
            >
                <Popup>{location}</Popup>
            </CircleMarker>
        );
    });
};

export default MainlandWest;
```

Koodiesimerkki 14. Kaupungeista vastaava funktiokomponentti argumentteineen.

Kaupungit luodaan CircleMarker-komponentilla, tuodaan MapContainer-komponenttiin ja lisätään omaan taulukkoonsa samalla tavalla, kuin vuosien komponentit. Tällä tavalla muodostetaan yksi maantieteellinen kokonaisuus (koodiesimerkki 15).

```
let cities = [
    {
        City: (
            <MainlandEast
                handleOverlayClick={locationData =>
                    this.handleInfoboxClick(locationData)
                }
                key="MainlandEast"
            />
        ),
        value: this.state.displayElements.cities.displayMainlandEast
    },
```

Koodiesimerkki 15. Cities-taulukko ja sen sisällä Itä-Euroopan kaupungit MainlandEast-komponentissa.

Kaupungit on kuitenkin vielä liitettävä FeatureGroup-komponenttiin, joten map-funktio ei näytä aivan samalta kuin vuosien vastaavassa toteutuksessa. Ehtolauselogiikka säilyy (koodiesimerkki 16).

```
{!this.state.displayElements.displayAllTerrain
    ? cities.map(({ City, value, index }) =>
            !value ? (
                false
            ) : (
                <FeatureGroup key={index} color="#000">
                    {City}
                </FeatureGroup>
            )
    )
    : this.state.displayElements.displayAllTerrain
    ? cities.map(({ City, index }) => (
            <FeatureGroup key={index} color="#000">
                {City}
            </FeatureGroup>
    ))
    : false}
```

Koodiesimerkki 16. Kaupungit FeatureGroup-komponenteissa ehtolausekokonaisuudessaan.

CircleMarkerilla tehtyjä kaupunkikohteita ei varsinaisesti tarvitsisi laittaa FeatureGroupiin, koska niistä voisi muodostaa kokonaisuuden ohjelmallisestikin. Kuitenkin FeatureGroup-komponentti on tarkoitettu ryhmien esittämistä varten, eikä sen sivuuttaminen palvelisi React-Leafletin komponenttien hyödyntämistä.

5.4.3 Kauppareitit

Polyline-komponentti toimii samalla periaatteella, kuten Polygon: joukko matriisissa olevia koordinaatteja luo muodon (koodiesimerkki 17).

```
let routesLand = [];
let routesVariable = [];

this.landExclusiveRoutes.forEach(element => {
    let assignLandElement;
    assignLandElement = element.map(({ coordinates, route }, index) => {
        return (
            <Polyline
                key={index}
                positions={coordinates}
                color={this.landRouteStyle.color}
            >
                <Popup>{route}</Popup>
            </Polyline>
        );
    });
    routesLand.push(assignLandElement);
});
```

Koodiesimerkki 17. Silmukka silmukan sisällä Polyline-komponenttia käyttäville maareiteille.

Ensimmäinen funktio, forEach, on samantapainen kuin map, mutta se ei palauta taulukkoa toisin kuin map. Tässä tilanteessa sen ei tarvitsekaan palauttaa mitään suorituksen jälkeen eikä sellaista toiminnalisuutta edes haluttaisi, koska palautetulla taulukolla ei tehtäisi mitään. ForEachin on vain mahdollistettava map-funktion käyttäminen jokaisen taulukon elementin kohdalla.

Polylinella on huomattavasti nopeampaa tehdä reittejä, ja varsinkin maareittejä, koska kulkuväylät ovat lähestulkoon aina suoraviivaisia. Joskus on kahden kaupungin välille tehtävä lovi, jottei reittiä merkittäisi kulkevan niemen tai salmen läpi (kuva 12).

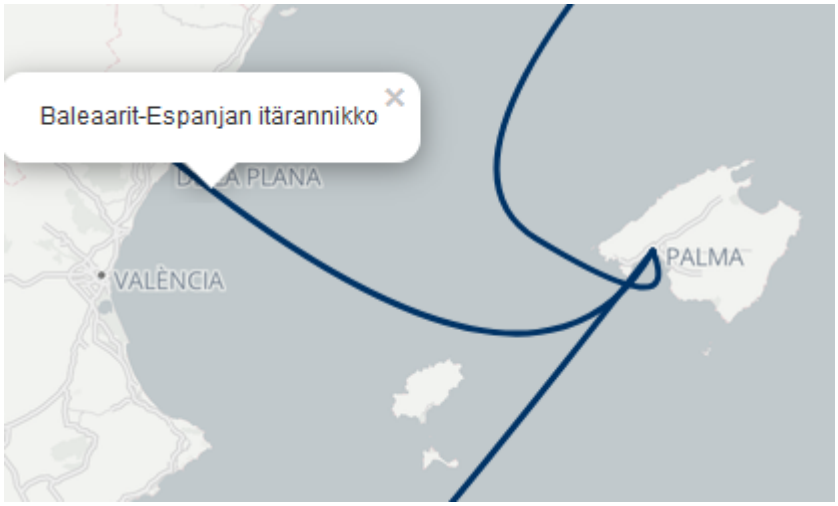
Kuva 12. Polylinella tehtyjä maareittejä Espanjassa sekä Ranskan ja Espanjan raja-mailla.

Curve sen sijaan mahdollistaa käyrien piirtämisen, joka teknilliseltä kannalta tulee ilmi taulukon muuttuneessa muodossa. Antamalla paketissa määrättyjä kirjaimia arvoina merkkijonomuodossa, voidaan määrittää tiettyjä pisteitä, joiden perusteella viiva kaartuu koordinaattien määrittämään suuntaan. Curve-komponenttia käyttävällä reitillä on myös kauppareittiarvo, eli onko kyseessä maalla vai merellä kulkeva väylä (koodiesimerkki 18).

```
{
    route: 'Baleaarit-Espanjan itärannikko',
    coordinates: [
        'M',
        [39.569444, 2.65],
        'Q',
        [38.7, 2],
        [39.983056, -0.033056],
    ],
    by: 'sea',
},
```

Koodiesimerkki 18. Coordinates-arvo on taulukko, joka tässä tapauksessa koostuu yhdestä M- ja kahdesta Q-taulukkoarvosta. Se on myös merireitti by-arvon perusteella.

Baleaarit-Espanjan itärannikko -esimerkissä M-arvo merkitsee lähtökoordinaatteja ja ensimmäinen Q-arvo taittopistettä ja toinen taittopisteen jälkeistä suuntaa. Kaarevilla viivoilla on mahdollista tehdä erittäin monimutkaisiakin muotoja, jotka sopivat merenkulun esittämiseen. Varsinkin satamasta lähtevä viiva näyttää luontaiselta laivareitiltä kaarevana (kuva 13).



Kuva 13. Käyrä kartalle piirrettynä.

Silmukkaperiaate pätee kauppareittienkin luomiseen, mutta by-arvolle tehdään tarkistus värien varalta: if-else-ehtolause määrittää merireitit sinisellä ja maareitit punaisella. Kuitenkin niiden esittäminen kartalla vaatii kahden muuttujan palauttamisen yhden sijasta, koska vain maareiteistä koostuvat oliot eivät sisällä by-arvoa (koodiesimerkki 19).

```
{
    value: this.state.displayElements.routes.BritainRoutes,
    routes: [routesVariable[4], routesLand[4]],
},
```

Koodiesimerkki 19. Britteinsaarten reitit. Routes-kenttä on taulukko, jossa on kaksi muuttujaa.

RoutesVariable-muuttuja pitää sisällään Curve-komponentit ja routesLand vain maareitit. Ne palautetaan yhtenä kokonaisuutena taulukkona, valiten oikeat arvot kummankin taulukon sisältä. Myös Kultaisen ordan raja on tehty Curve-komponentilla.

5.5 Tekstitiedon esittäminen sivupalkissa

Ponnahdusikkunan toiminta on jo esitelty monikulmion yhteydessä eikä siitä ole variaatioita. Jäljellä on vain sivupalkissa esitettävä tekstitieto. Sivupalkki on npm-paketti ja sen mukana tulee kaksi valmista funktiota: onClose ja onOpen. OnClose sulkee palkin, onOpen avaa sen klikatun alaosion kohdalta id-argumentin avulla (koodiesimerkki 20).

```
onClose() {
    this.setState({ collapsed: true });
}

onOpen(id) {
    this.setState({ collapsed: false, selected: id });
}
```

Koodiesimerkki 20. Sivupalkin tilaa määrittävät funktiot MapContainerissa.

Sivupalkin sisältämä tieto tulee olemaan huomattava, joten se on järkevää siirtää omaksi komponentikseen. Sen hallinnointia varten on komponentille annettava ominaisuuksia, kuten onClose- ja onOpen-funktiot sekä MapContainerin tila, josta otetaan SidebarContainerin sisällä tarvittavat arvot sivupalkin toimintaa varten. HandleSource-funktio avaa Lähteet-osion. Funktio sijoitetaan kaupunkikohteen tietojen lopussa näkyviin lähdelinkkeihin (koodiesimerkki 21).

```
<SidebarContainer
    state={this.state}
    onClose={() => this.onClose()}
    onOpen={child => this.onOpen(child)}
    handleLayerControl={layerName =>
        this.handleLayerControl(layerName)
    }
    handleSource={() => this.handleSource()}
/>
```

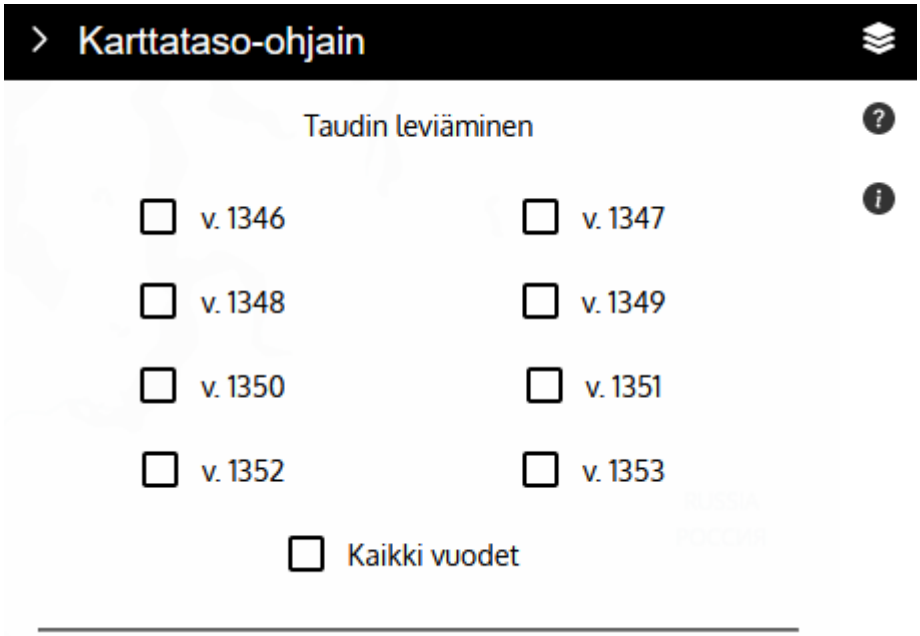
Koodiesimerkki 21. Sivupalkin komponentti ja sille annetut ominaisuudet.

SidebarContainer-komponentti on funktiokomponentti, joka tulee antamaan varsinaiselle sivupalkille arvot. Collapsed-ominaisuus määrittää, onko palkki kiinni vai auki ja selected-ominaisuus on alaosion valitsemista varten (koodiesimerkki 22).

```
<Sidebar
    id="sidebar"
    position="right"
    collapsed={state.collapsed}
    closeIcon={<HiOutlineChevronRight />}
    selected={state.selected}
    onOpen={onOpen.bind(this)}
    onClose={onClose.bind(this)}
>
```

Koodiesimerkki 22. Sidebar-komponentti SidebarContainer-komponentin sisällä. Osa MapContainer-komponentista annetuista ominaisuuksista määritetään Sidebarin ominaisuuksiksi.

Valintaruudut saa tehtyä samalla silmukkaa käyttävällä periaatteella, kuin muutkin rekursiota hyödyntävät elementit. Map-funktion tuotos on luettelo kulkutautileviämisestä vuosittain (kuva 14).



Kuva 14. Sivupalkin sisältö karttataso-ohjaimessa, Taudin leviäminen -osion alla.

Kaupunkeja käsittelevässä osiossa kaupungeista vastaavaan funktiokomponenttiin annettiin argumenttina funktio. Kun kaupunkia klikkaa kartalla, MapContainer-komponentissa olevaan funktioon tulee olio. Tilaolio päivitetään avaamaan sivupalkista kaupunkikohtainen osio ja määritetään viitteet, kaupungin nimi palkkiin ja palkin sisältämä tieto kohteesta (koodiesimerkki 23).

```
handleInfoboxClick({ location, info, reference }) {
    this.setState({ selected: 'locationInfo' });
    if (this.state.infoboxLocation !== location) {
        this.setState({
            infoboxLocation: location,
            infoboxLocationInfo: info,
            reference: reference,
            collapsed: false,
        });
    }
}
```

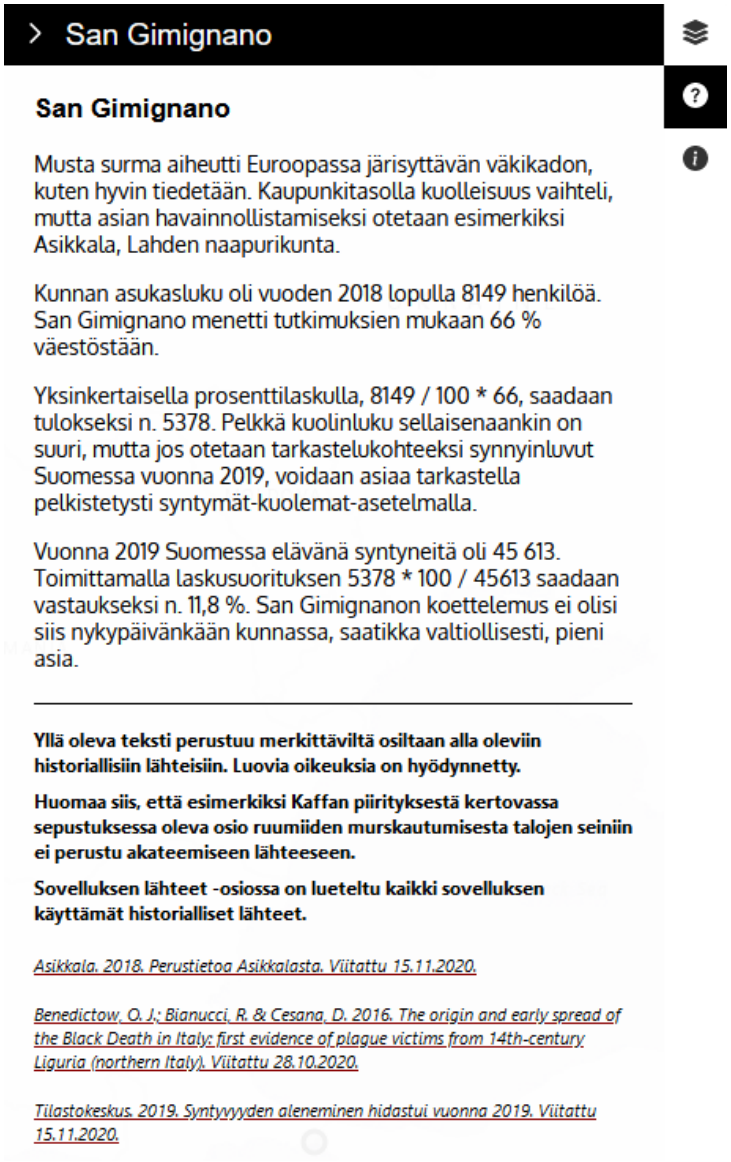
Koodiesimerkki 23. Sivupalkki avautuu ja päivittää alaosiossa näkyvän kaupunkitiedon taulukon tiedoilla.

Sivupalkin kaupunkiosio saa uudet tiedot SidebarContainerille annettujen ominaisuuksien kautta (koodiesimerkki 24).

```
<Tab
    id="locationInfo"
    header={
        state.infoboxLocation.length === 0
            ? 'Tietoa'
            : /*prettier-ignore*/
            (typeof state.infoboxLocation.split(' ')[1] !== 'undefined' && !state.infoboxLocation.split(' ')[1].startsWith('('))
            ? state.infoboxLocation
            : state.infoboxLocation.split(' ')[0]
    }
    icon={<BsQuestionCircleFill />}
>
    <div id="paragraph-parent">
        <h3>
            {state.infoboxLocation.length !== 0
                ? state.infoboxLocation
                : false}
        </h3>
        {state.infoboxLocationInfo.length !== 0 ? (
            createParagraphs(state.infoboxLocationInfo)
        ) : (
            <p>···
        )}
    </div>
    {state.infoboxLocationInfo.length !== 0 ? (
        <div className="source">
            <strong>···
            {state.reference.map(({ reference, link }) => {
                return (
                    <a
                        rel="noopener noreferrer"
                        target="blank"
                        href={link}
                        key={reference}
                        title="Avaa linkin ja lähdeosion"
                        onClick={() => handleSource()}
                    >
                        {reference}
                    </a>
                );
            })}
        </div>
    ) : (
        false
    )}
</Tab>
```

Koodiesimerkki 24. Sivupalkin kaupunkikohtainen osio.

Jos kaupungin nimi on vaihtunut vuosien saatossa, niin ennen historiallista aihetta käsittelevää tekstiä kerrotaan kaupungin entinen ja sitten vasta nykyinen nimi, tähän tapaan: ”Konstantinopoli (nyk. Istanbul)”. Jos nimi ei ole muuttunut, ei tehdä mitään (kuva 15).



Kuva 15. Kaupungista kertova alaosio lähteineen.

Muiden alaosioiden tekeminen noudattaa samaa kaavaa: taulukko olioita, joissa on esitettävä tieto. Esimerkiksi lähdeosiossa taulukko on SourcesAndAdditionalReadingContent-niminen muuttuja (koodiesimerkki 25).

```
return (
    <>
        <p>
            Alla lähteet ja mieltä kiinnittävää aineistoa.
            <br />
            Linkit avautuvat uuteen välilehteen.
        </p>
        <ul className="more-ul">
            {SourcesAndAdditionalReadingContent.map(
                ({ link, author, contentName }) => {
                    return (
                        <li key={author}>
                            <a
                                rel="noopener noreferrer"
                                target="blank"
                                href={link}
                            >
                                <b>{author}</b>
                                <br />
                                {contentName}
                            </a>
                        </li>
                    );
                }
            )}
        </ul>
    </>
);
```

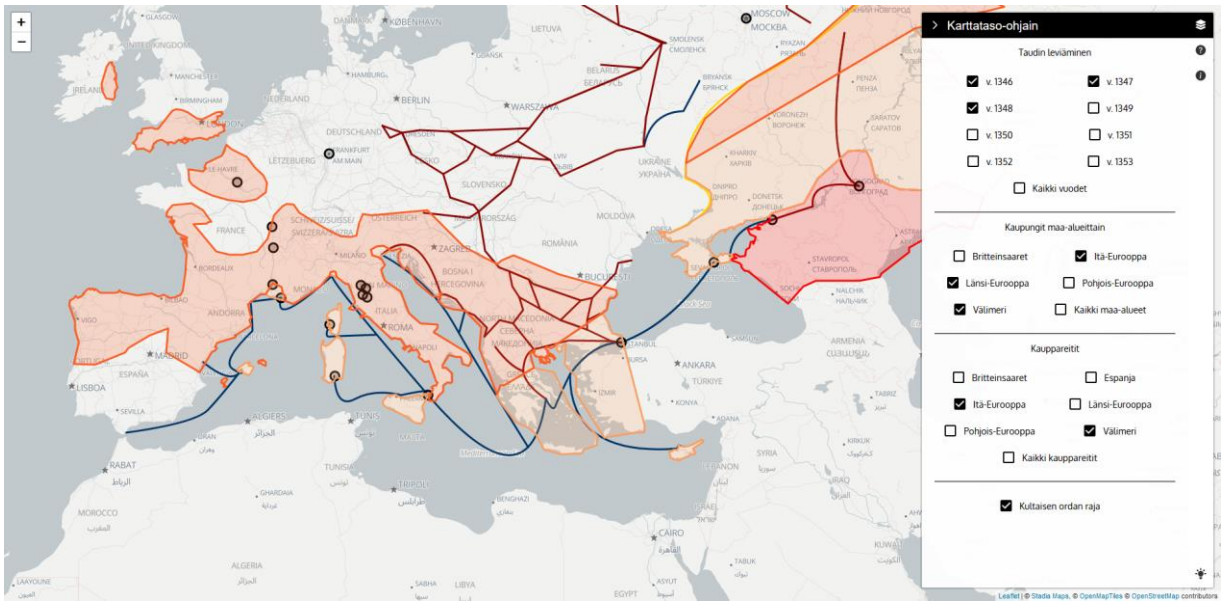
Koodiesimerkki 25. Lähteet-osion komponentti.

Luonnollisesti ainoa ero osioiden välillä on taulukoissa olevien olioiden sisältö ja sisällöstä käytettävät muuttujanimet funktioissa.

6 YHTEENVETO JA PÄÄTELMÄT

Opinnäytetyön lopputuloksena mustan surman leviäminen on kartoitettu melko tarkasti, suuri määrä kauppareittejä antaa kuvan keskiajan globalisaatiosta, ja kaupungeista kertovat tiedot tuovat tämän julman kulkutaudin konkreettisemmaksi käyttäjille.

Ilmaisena ja verkossa saatavilla sovellus toivottavasti vetää puoleensa edes jonkin verran historiasta lumoutuneita. Parhaimmassa tapauksessa se innostaa aikaisemmin aiheesta vähemmän välittäneitä perehtymään menneisyyteen. Vuorovaikutteisena sovelluksena käyttäjällä on vapaat kädet säännöstellä kartalla näkyvää tietoa ja siten tehostaa tiedon sisäistämistä (kuva 16).



Kuva 16. Monta eri komponenttia käytössä samanaikaisesti.

Suorituksena opinnäytetyö oli erittäin hyödyllinen niin henkilökohtaisesti kuin ammatillisestikin. Sovelluksen suunnittelu ja suunnitelman toteuttaminen käytännössä antoi hyvän läpileikkauksen kummastakin toimesta. Varsinkin ohjelmointiosuuden suhteen ratkaisujen ja optimointien hoksaaminen oli erittäin mielekäs kokonaisuus.

Ohjelmoinnin kannalta se on avointa lähdekoodia, ja toiveenani onkin, että ehkä osa siitä pääsee hyödynnetyksi johonkin toiseen projektiin. Esimerkiksi koordinaattitiedot voisivat olla tällaisia koodiosia, jotka kiinnostaisivat tekniikanharrastajia.

Korona-aikana valtion määräämät rajoitukset ja suositukset ja taudin aiheuttamat yhteiskunnalliset mullistukset luovat mielenkiintoisen yhteyden menneiden aikojen ja nykypäivän välille. Huomiona on, ettei mikään varsinaisesti muutu, eikä mitään uutta ole Auringon alla; lintu- ja sikainfluenssa, kolera ja espanjantauti ovat kaikki kulkutauteina nälvineet ihmiskuntaa viimeisen sadan vuoden aikana, jokainen niistä aiheuttaen valtiollisia rajoitustoimia ja laajaa murhetta. Kulkutaudeista seuraavaa hämmennys, suru, vastatoimet ja muut reaktiot pätevät riippumatta ajasta, ja se tekee sovelluksesta ajankohtaisen portin menneeseen.

Sovellus täyttää asetetut määreet ja on täysin käypä sellaisenaan. Jäljempänä oleva luettelo nimenomaisesti ei sisällä keskeneräisen ohjelman puutteiden paikkausta, vaan mahdollisia lisäyksiä sovellukseen. Jatkokehitys on ilmeinen ja avoin kysymys. Avoimena lähdekoodina kuka tahansa voi tehdä oman versionsa sovellukseni pohjalta, mutta nostan muutamia huomioita mahdollisista lisäyksistä.

Parannuksiin lukeutuvat seuraavat:

- Kauppareittien terävöittäminen tärkeyden ja määrän suhteen. Mitättömämpien kauppareittien kartoittaminen toisi lisäarvoa, mutta tällaisen erotuksen lisääminen merkitsisi muutoksia jo sovelluksessa oleviin reitteihin. Ohjelmointimuutoksiakin tarvittaisiin silkkojen koordinaattilisäyksien suhteen, sillä kauppareittien tärkeyttä pitäisi korostaa jollain tavalla. Varsinkin Curve-komponentilla tehdyt reitit ovat joskus epätavallisen kaartavia tai tarpeettoman syviä käännösten kohdalla. Näiden muuttaminen vaatisi vain lisää aikaa.
- Lisää kaupunkikohteita. Yksinkertainen, vain tiedon lisäystä vaativa toimi. Tämä on jatkokehityksen suhteen todennäköisesti kaikista helpoin lisäys.
- Lisää tietoa ponnahdusikkunoihin. Ei järin suuri lisäys, mutta sen tarkoituksenmukaisuus on kyseenalaista. Kenties tietoa vuosittaiseen tautietenemiseen liittyen tai jotain vastaavaa.
- Taudin etenemisen terävöittäminen. Jotkut mustan surman monikulmioiden rajat ovat ylimalkaisesti tehtyjä. Esimerkiksi osa saarista ei ole omana koordinaattitaulukkonaan, vaan viereisen mantereen kylkiäisenä. Myös jotkut tautirajat voisivat olla tarkemmin määriteltyjä, kuten vuoden 1353 rajat. Täysin dataa vailla oleva osa nykyisen Moldovan, Romanian ja Ukrainan alueilla voisi olla täytettävissä. Tämä on tosin silkkaa spekulaatiota, sillä historiallinen tieto ei välttämättä pysty täyttämään tätä vajetta taudin etenemisestä.
- Koodin optimointi. On ilmeistä, että koodia voisi muuttaa siten, että nykyistä suurempi osa siitä olisi automatisoitua. Tämä olettamus perustuu komponenttien yhtäläisyyteen.
- Ohjelmointivirheiden (engl. software bug) ja virheiden korjaus. Ilmiselviä ohjelmointivirheitä tuskin on, mutta joitakin siivilän läpi päässeitä virheitä varmasti on. Sovelluksessa Kirjoitusvirheitä ja epäjohdonmukaisia tai vaikeasti ymmärrettäviä lausemuotoja

Jatkokehitysmahdollisuuksiin lukeutuvat seuraavat:

- Sovelluksen kääntäminen eri kielille. Käännösten suhteen sinänsä yksinkertainen toimi, mutta sovelluksen toiminnan kannalta se vaatisi paljon muutoksia. Reactille on kehitetty käännöslisäosia, jotka tekevät toimesta jokseenkin vähemmän tuskallisen, mutta koko toiminnon toteutus veisi paljon aikaa jo silkan tekstimääränkin takia.
- JavaScriptin muuntaminen TypeScriptiksi. Menemättä teknillisyyksiin, TypeScript tekee JavaScriptistä ennakoitavampaa ja selkeämpää muun muassa vaatimalla ohjelmoijaa määrittämään koodissa muuttujien tyypit (engl. types), eli onko kyseessä esimerkiksi merkkijono tai taulukko.
- Esitetyn datan siirtäminen ohjelmointirajapintaan tai palvelimelle. Sovelluksessa käytetään tuhansien rivien verran tietoa, varsinkin koordinaattien muodossa. Siirtäminen ulkoiseen sijaintiin vaatisi huomattavia muutoksia ohjelmaan sekä ponnistuksia tallennettavan paikan valmistelemiseen. Esimerkiksi Node.js-palvelimen pystyttämiseen tarvittaisiin tietojen kyselyä varten polut ja karttasovelluksessa logiikka datan käsittelyä varten. Lisäksi palvelin pitäisi sijoittaa verkkoon datan käytännöllisen noutamisen varmistamiseksi.

Sovelluksen muuntaminen toisen kulkutaudin tarpeisiin on jatkokehityksestä erillinen idea. Opinnäytetyöni ohjaajan esittämä mahdollisuus hyödyntää sovellusta jonkin muun kulkutaudin esittämiseen on täysin toteutettavissa. Informaatioaikana puhjenneista pandemioista tallennettu tieto on kattavaa, ja sellaisen taudin esittäminen sovelluksella olisi huomattava toimi. Todennäköisesti tällainen muunnos vaatisi lisätoiminnallisuuksien kehittämistä silkan datan määrän, mutta myös sen tarkkuuden takia.

Opinnäytetyön valmistumishetkellä (4.12.2020) sovellus sijaitsee osoitteessa https://sedrik1.github.io/musta-surma/ ja sovelluksen lähdekoodi osoitteessa https://github.com/sedrik1/musta-surma.

LÄHTEET

Abramov, D. 2019. React v16.8: The One With Hooks. Facebook. Viitattu 4.11.2020 https://reactjs.org/blog/2019/02/06/react-v16.8.0.html.

Abramov, D. 2017. What's New in Create React App. Facebook. Viitattu 4.11.2020 https://reactjs.org/blog/2017/05/18/whats-new-in-create-react-app.html.

Abramov, D. 2016. Create Apps with No Configuration. Facebook. Viitattu 4.11.2020 https://reactjs.org/blog/2016/07/22/create-apps-with-no-configuration.html.

Abramov, D. 2015. React Components, Elements, and Instances. Facebook. Viitattu 4.11.2020 https://reactjs.org/blog/2015/12/18/react-components-elements-and-instances.html.

Agafonkin, V. 2020. Viitattu 29.10.2020 https://leafletjs.com/reference-1.7.1.html.

Benedictow, O. J.; Bianucci, R. & Cesana, D. 2016. The origin and early spread of the Black Death in Italy: first evidence of plague victims from 14th-century Liguria (northern Italy). Viitattu 28.10.2020 https://www.jstage.jst.go.jp/article/ase/advpub/0/advpub_161011/_pdf.

Collins English Dictionary – Complete and Unabridged 2014. Cascading style sheets. 12. painos. Viitattu 29.10.2020. Saatavissa myös https://www.thefreedictionary.com/cascading+style+sheet.

Dictionary of Unfamiliar Words by Diagram Group 2008. Hypertext Markup Language (HTML). Viitattu 29.10.2020. Saatavissa myös https://www.thefreedictionary.com/Hypertext+Markup+Language+(HTML).

Facebook 2020. React. Viitattu 24.9.2020 https://reactjs.org/.

Facebook 2020a. Components and Props. Viitattu 24.9.2020 https://reactjs.org/docs/components-and-props.html.

Facebook 2020b. Lifecycle Methods. Viitattu 30.10.2020 https://reactjs.org/docs/glossary.html#lifecycle-methods.

Frith, J. 2012. The History of Plague – Part 1. The Three Great Pandemics. Journal of Military and Verans' Health, 20(2). Viitattu 28.10.2020 https://jmvh.org/article/the-history-of-plague-part-1-the-three-great-pandemics/.

Gibbons, N. J. & Evans, J. 2007. The interactivity effect in multimedia learning. Computers & Education, 49(4), 1147-1160. Viitattu 17.11.2020 https://www.sciencedirect.com/science/article/abs/pii/S0360131506000285.

Heikura, P. T. 2003. Musta surma. Tieteessä Tapahtuu, 21(8). Viitattu 27.10.2020 https://journal.fi/tt/article/view/57247.

Hunt, P. 2013. Why did we build React? Facebook. Viitattu 4.11.2020 https://reactjs.org/blog/2013/06/05/why-react.html.

Kjaer, C. 2019. Tutkijat paljastavat: Täältä musta surma alkoi. Tieteen Kuvalehti. Viitattu 28.10.2020 https://tieku.fi/laaketiede/sairaudet/tutkijat-paljastavat-taalta-musta-surma-alkoi.

Le Cam, P. 2020a. Viitattu 13.10.2020 https://react-leaflet.js.org/docs/en/plugins.

Le Cam, P. 2020b. Viitattu 29.10.2020 https://react-leaflet.js.org/.

Maps.com. The Black Death. Viitattu 17.11.2020 https://edarchive.maps.com/home/student-dashboard/my-books/discovering-past-history-world-early-ages/black-death-interactive/.

Marchán, K. 2017. Introducing npx: an npm package runner. Medium. Viitattu 27.9.2020 https://medium.com/@maybekatz/introducing-npx-an-npm-package-runner-55f7d4bd282b.

Node.js 2020. About Node.js®. Viitattu 27.9.2020 https://nodejs.org/en/about/#about-node-js.

Node.js 2011. What is npm? Viitattu 27.6.2020 https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/.

npm. About npm. Viitattu 29.10.2020 https://docs.npmjs.com/about-npm.

Nykänen, R. 2010. Elämä tautisessa kaupungissa – Mustan surman vaikutus toscanalaiskaupunkien arkeen. Historian Pro Gradu -tutkielma. Humanistinen tiedekunta. Tampere: Tampereen yliopisto. Viitattu 28.10.2020 https://trepo.tuni.fi/bitstream/handle/10024/81816/gradu04473.pdf.

Pelu, C. 2020. npm vs npx – What's the Difference? freeCodeCamp. Viitattu 26.9.2020 https://www.freecodecamp.org/news/npm-vs-npx-whats-the-difference/.

Snell, M. 2019. The Arrival and Spread of the Black Plague in Europe. ThoughtCo. Viitattu 14.10.2020 https://www.thoughtco.com/spread-of-the-black-death-through-europe-4123214.

The Free Dictionary 2020. JS. Viitattu 29.10.2020 https://acronyms.thefreedictionary.com/JS.

WHO 2014. Plague in Madagascar. Viitattu 28.10.2020 https://www.who.int/csr/disease/plague/madagascar-outbreak/en/.

World Wide Web Consortium 1998. Level 1 Document Object Model Specification. W3C Working Draft, 1. versio. Viitattu 30.10.2020 https://www.w3.org/TR/WD-DOM/cover.html.

Wozniewicz, B. 2019. The Difference Between a Framework and a Library. freeCodeCamp. Viitattu 27.9.2020 https://www.freecodecamp.org/news/the-difference-between-a-framework-and-a-library-bd133054023f/.