Arttu Hartikainen

Koronadatan visualisointi Javascript-kirjaston avulla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

30.9.2021

Tiivistelmä

Tekijä:	Arttu Hartikainen
Otsikko:	Koronadatan visualisointi Javascript-kirjaston avulla
Sivumäärä:	37 sivua
Aika:	30.9.2021
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Mediatekniikka
Ohjaaja:	Lehtori Toni Spännäri

Opinnäytetyön tarkoituksena oli luoda web-sovellus, jonka avulla pystytään tarkastelemaan valitun maan koronatilastoja. Tavoitteena oli tehdä sovelluksesta selkeä ja helppokäyttöinen. Toisena tavoitteena työssä oli saada selkeä kuva visualisointiin käytettävistä tekniikoista ja työkaluista tulevaisuutta varten.

Monelle perustason käyttäjälle web-sivu ja web-sovellus saattavat tarkoittaa samaa asiaa, mutta todellisuudessa web-sovellukset ovat paljon monimutkaisempia. Web-sovellusten toimintatavat määräytyvät sovelluksessa käytettävien komponenttimallien ja arkkitehtuurin tyypin perusteella.

Dataa kerätään koko ajan kasvavissa määrin eri lähteistä, ja sitä opitaan hyödyntämään jatkuvasti enemmän. Datan avulla pystytään esimerkiksi saamaan vastauksia liiketoiminnassa tehtäviin päätöksiin.

Sovelluksen pohja toteutettiin käyttäen React-ohjelmistokehystä. Visualisoitava data haettiin COVID19API-rajapinnasta, jonka avulla voidaan hakea koko maailman tilastot yhdellä pyynnöllä. Varsinainen visualisointi toteutettiin Chart.js-kirjastoa käyttäen, jolla pystytään luomaan valmiita kaaviopohjia. Kaaviopohjaan tuotiin tilastot käytetystä rajapinnasta ja visualisoitiin ne kasvukäyräksi.

Työn lopputulokseksi saatiin ulkonäöllisesti hyvin yksinkertainen ja selkeä web-sovellus. Sovellus mahdollistaa perustason loppukäyttäjälle tarvittavan tiedon hakemisen nopeasti ja tieto on saatavilla nopealla vilkaisulla. Visualisoidut tilastot toteutettiin siten, että ne ovat helppolukuisia ja kaikkien ymmärrettävissä. Ilmaisen rajapinnan tuomien rajoitusten vuoksi sovelluksen sisältö ja visualisoitavat tilastot jäivät hieman suppeiksi. Maksullisen rajapinnan avulla sovellukseen voitaisiin tuoda lisäominaisuuksia, jotka mahdollistaisivat sovelluksen jatkokehityksen ja mahdollisen kaupallistamisen.

Avainsanat: data, datan visualisointi

Abstract

Author:	Arttu Hartikainen
Title:	Corona data visualization with Javascript library
Number of Pages:	37 pages
Date:	30 September 2021
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communications Technology
Professional Major:	Media technology
Supervisor:	Toni Spännäri, Senior Lecturer

The purpose of the thesis was to create a web application that can be used to view corona statistics of a selected country. The aim was to create a clear and easy-to-use application. Another goal of the thesis was to get clear picture about techniques and tools which can be used for data visualization.

The theoretical part of the study focuses on the theory of web application architecture, as well as data and its visualization in general. As a technical implementation, a web application was created, which utilizes the techniques and tools covered in the theoretical part.

The base of the application is implemented by using React framework. Visualized data is retrieved from COVID19API interface. The actual data visualization was made by using the Chart.js library, which can be used to create ready-made chart templates. Retrieved statistics were imported into chart template and visualized as a growth curve.

The result of the project was a very simple and clear web application. The visualized statistics are implemented in a way that they are easy to read and understand. Due to the limitations of the free interface, the content of the application remained somewhat limited.

Keywords: data, data visualization

Sisällys

Lyhenteet

AJAX: *Asychronous Javascript And XML.* Tekniikka, jolla siirretään dataa palvelimelta.

CDN: *Content Delivery Network.* Hajautettu verkko, joka koostuu välityspalvelimista.

CSS: *Cascading Style Sheets*. Verkkosivuille kehitetty tyylitiedosto, jota käytetään verkkosivujen ulkoasun määrittämiseen.

HTML: *Hypertext Markup Language*. Tekstin muotoiluun käytetty kieli, jolla määritellään, kuinka verkkosivun sisältö näkyy käyttäjälle.

JSON: *JavaScript Object Notation.* Tiedostomuoto, jota käytetään tiedonvälitykseen.

PHP: *PHP: Hypertext Preprocessor.* Ohjelmointikieli, jota käytetään nykyään yleisesti web-palvelimien luomiseen.

RSS: *Really Simple Syndication.* Verkkosyötemuoto, jota käytetään päivittyvän sisällön julkaisemiseen.

SCSS: *Syntactically Awesome Style Sheet.* Esikääntäjän kieli, joka käännetään CSS-muotoon.

SVG: *Scalable Vector Graphics.* Vektorikuvien kuvauskieli.

XML: *Extensible Markup Language.* Merkintämuoto, jota käytetään järjestelmien väliseen tiedonvälitykseen ja tiedostomuotona dokumenttien tallennuksessa.

1 Johdanto

Vuoden 2020 alkupuolella koko maailman tietoisuuteen nousi Kiinan Wuhanista lähtöisin oleva Covid-19-virus. Nopeasti viruksen ensiesiintymisen jälkeen alettiin raportoida suuria tartuntamääriä ympäri maailmaa, ja poiketen monista oletamuksista viruksesta kehittyi vain muutamassa kuukaudessa kansainvälinen pandemia. Heti viruksen ilmaantumisen jälkeen terveydenhuollon asiantuntijat ja analyytikot ryhtyivät tilastoimaan muun muassa tartuntamääriä ja kuolemia. Tilastoja päivitetään edelleen pandemian ollessa aktiivinen, ja tilastot ovat suurelta osin kaikkien saatavilla eri maiden tutkimuslaitosten jakamana avoimena datana.

Insinöörityöprojektin tavoitteena on luoda yksinkertainen ja selkeä web-sovellus, jolla pystytään tarkastelemaan eri maiden koronatilastoja. Työn yhteydessä toisena tavoitteena on saada selkeä kuva datan visualisointiin käytetyistä tekniikoista ja työkaluista, joita voidaan hyödyntää tulevaisuudessa.

Jo olemassa olevissa sovelluksissa ja verkkosivustoilla tulokset on esitetty helposti liian analyyttisesti ja monimutkaisesti, mikä tekee perustason loppukäyttäjälle tilastojen havainnoinnista vaikeaa ja epämiellyttävää. Lisäksi helposti havainnoitavat tilastot on usein sisällytetty esimerkiksi iltapäivälehtien verkkosivuille ja artikkeleihin, mikä hankaloittaa tiedon hakemista nopeasti. Sovelluksen avulla pyritään esittämään oleellisimmat koronatilastot perustason loppukäyttäjälle helposti luettavassa muodossa ja siten, että haettu tieto on suoraan saatavilla ja luettavissa nopealla vilkaisulla.

Sovellukseen luodaan osio, josta näkee tarkasteltavan maan yleisimmät tilastot yhteenvetona. Olennaisessa asemassa lopputuloksen kannalta on datan varsinainen visualisointi kaaviomuotoon. Sovelluksessa haetaan tilastot rajapinnasta ja visualisoidaan ne valitulla kirjastolla luotuun kaavioon. Lopuksi sovellukseen luodaan vielä valikot, joista pystytään valitsemaan kaaviossa visualisoitava maa sekä aikaväli.

Sovelluksen pohjana käytetään React-ohjelmistokehystä. Rajapinnaksi on sovellukseen valittu COVID19API, josta pystytään hakemaan helposti koko maailman koronatilastot yhdellä pyynnöllä. Rajapinnasta haetut tilastot visualisoidaan kaavioon, johon käytettävät työkalut kartoitetaan työn aikana. Projektin suunnitteluvaiheessa tutkitaan jo olemassa olevia kaavioita, joiden perusteella pyritään havainnoimaan, millaisia tilastoja on yleisesti visualisoitu ja millaisia menetelmiä ja esitystapoja tilastojen visualisointiin on järkevää käyttää.

2 Web-sovellusten arkkitehtuuri

Vuosien myötä ja teknologian kehittyessä perinteisten staattisten verkkosivujen rinnalle ovat tulleet web-sovellukset. Web-sovellus pystytään yksinkertaisesti määrittelemään siten, että se on ohjelma, jota käytetään verkkoselaimessa. Käytännössä web-sovellus on aivan kuten tavallinen työpöytäsovellus, mutta ainoana erona on, että se toimii internetissä ja siihen päästään käsiksi verkkoselaimen kautta (1; 2).

Perustason käyttäjälle perinteisten verkkosivujen ja web-sovellusten välinen ero saattaa olla tuntematonta aluetta, ja useimmiten molemmat näistä mielletään verkkosivustoiksi. Kehitystyössä näillä kahdella on kuitenkin selkeä ero. Perinteiset verkkosivustot ovat staattisia, mikä tarkoittaa sitä, että sisältö ei päivity dynaamisesti ja sisältö on kaikkien käyttäjien saavutettavissa. Verkkosivut ovat ainoastaan yhteen suuntaan informatiivisia, eikä käyttäjän ole mahdollista kommunikoida sivuston kanssa tai vaikuttaa sen sisältöön. Yleensä staattiset verkkosivut on toteutettu käyttäen HTML:ää, CSS:ää ja joissain tapauksissa myös Javascriptiä. (2; 3.) Esimerkkeinä tällaisista verkkosivuista voidaan mainita yksinkertaiset yritysten verkkosivut tai henkilökohtaiset portfoliot.

Kun verkkosivuille aletaan lisätä funktionaalisuutta ja interaktiivisia elementtejä, tulee verkkosivusta web-sovellus. Web-sovelluksissa käyttäjän on tietojen tarkastelun lisäksi mahdollista vaikuttaa sisältöön ja näin ollen käsitellä sivuston sisältämää dataa. Helppoja esimerkkejä web-sovellusten funktionaalisuudesta ja interaktiivisista elementeistä ovat lomakkeiden täyttäminen tai kirjautuminen

palveluun käyttäjänimen ja salasanan avulla. Web-sovellusten luominen on yleisesti monimutkaisempaa ja vaikeampaa kuin staattisten verkkosivujen. Täysikokoisten web-sovellusten luomiseen vaaditaan usein kokonainen työryhmä kokeneita kehittäjiä, kun taas perinteiset verkkosivut pystyy käytännössä luomaan kuka tahansa lyhyellä perehtymisellä. (2; 3.) Peruskäyttäjälle helposti ymmärrettäviä esimerkkejä web-sovelluksista ovat sähköpostipalvelut ja verkkopankit. Web-sovellusten luomisessa käytetään siihen tarkoitettuja ohjelmistokehyksiä parantamaan projektien hallittavuutta ja organisointia sekä nopeuttamaan kehitystyötä.

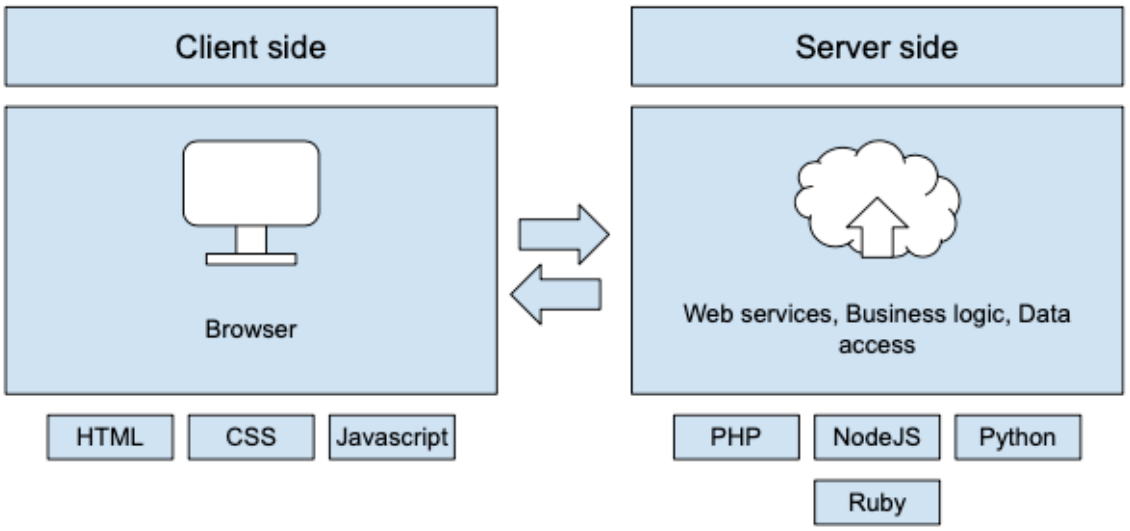
2.1 Komponentit

Web-sovellusten sisältämät komponentit pystytään jakamaan karkeasti kahteen ryhmään, käyttöliittymään ja käyttäjäkokemukseen liittyviin komponentteihin sekä rakenteellisiin komponentteihin (1). Toiminnan kannalta web-sovellukset eivät välttämättä vaadi kuin rakenteellisia komponentteja, mutta käyttöliittymään ja käyttäjäkokemukseen liittyvillä komponenteilla sovelluksesta saadaan käyttäjäystävällisempi.

Käyttöliittymällä tarkoitetaan tässä yhteydessä sovelluksen graafista ulkoasua ja sitä, millaiselta sovellus näyttää käyttäjän näytöllä. Käyttöliittymäsuunnittelussa pääpaino on luoda elementeistä visuaalisesti miellyttäviä ja saada ne vastaamaan sovelluksen ulkoasua tai tarkoitusta. Käyttäjäkokemuksella puolestaan pyritään luomaan sovelluksesta loogisesti järkevä ja helpottamaan käyttäjän tiedonhakua. Käyttöliittymästä ja käyttäjäkokemuksesta puhutaan usein yhdessä, koska esimerkiksi hyvä värisuunnittelu on myös osa hyvää käyttäjäkokemusta. (5.) Käyttöliittymään ja käyttäjäkokemukseen liittyvät komponentit eivät suoranaisesti vaikuta web-sovelluksen sisäisiin toimintoihin, mutta niiden avulla sovellus tuodaan lähemmäs loppukäyttäjää ja näin ollen ne ovat osa web-sovellusten arkkitehtuuria.

Web-sovellusten rakenteelliset komponentit ovat vastuussa sovelluksen varsinaisesta toiminnasta. Rakenteelliset komponentit muodostuvat kahdesta osa-

alueesta, selainpuolesta eli asiakkaan näkemästä puolesta, ja palvelinpuolesta. Selainpuoli vastaa siitä, mitä loppukäyttäjä näkee näytöllään, kun taas palvelinpuoli siitä, mitä pinnan alla tapahtuu (kuva 1). (1; 6.)



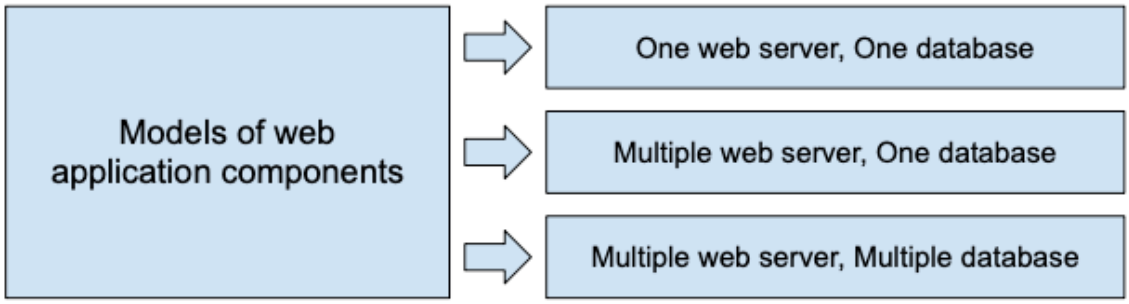
Kuva 1. Web-sovelluksen selainpuolen ja palvelinpuolen komponentit. Molemmat ovat rakenteellisia komponentteja. (7.)

Selainpuolen komponentti on yleensä rakennettu käyttäen HTML:ää, CSS:ää ja Javascriptiä. Koska tämä komponentti on käyttäjän verkkoselaimessa, laitteeseen tai käyttöjärjestelmään ei tarvitse tehdä muutoksia. Lisäksi se on käytettävissä kaikilla laitteilla, joista löytyy verkkoselain. Tämä komponentti kuvastaa niitä web-sovelluksen osia, joiden kanssa loppukäyttäjä on vuorovaikutuksessa. (1; 4; 6.)

Palvelinpuolen komponentti voidaan rakentaa usealla eri kielellä käyttäen yhtä tai useampaa ohjelmointikieltä, esimerkiksi PHP:tä, Pythonia, Rubya tai Javascriptiin pohjautuvaa NodeJS:ää (8). Tämä komponentti sisältää vähintään kaksi osa-aluetta, sovelluksen varsinaisen logiikan ja tietokannan. Ensimmäistä edellä mainituista voidaan yleisesti pitää koko sovelluksen aivoina tai ohjauskeskuksena, joka käsittelee käyttäjän tekemiä käskyjä. Jälkimmäinen osa puolestaan säilyttää käyttäjän syöttämää informaatiota, jota voidaan käyttää myöhemmin dynaamisen sisällön näyttämiseen. (1; 4; 6.)

2.2 Komponenttimallit

Kehitettäessä web-sovelluksia erilaisia komponenttimalleja on kolme, ja sovelluksen käyttämä malli määräytyy palvelimien ja tietokantojen määrän mukaan (kuva 2) (9). Kun sovellukseen lisätään palvelimia ja tietokantoja, vastaavasti myös sovelluksen suojaus ja luotettavuus paranee.



Kuva 2. Web-sovelluksen kaikki kolme komponenttimallia (7).

Ensimmäinen kolmesta komponenttimallista on yksi palvelin ja yksi tietokanta. Tämä malli on selkeästi yksinkertaisin näistä kolmesta, mutta myös vähiten luotettava. Jos sovelluksessa on vain yksi palvelin ja yksi tietokanta eikä mitään turvaamassa kaatumista, palvelimen kaatuessa koko sovellus kaatuu sen mukana. Tästä syystä kyseistä mallia harvoin käytetään lopullisissa web-sovelluksissa. Yleisimmin tämä malli on käytössä sovellusten testauksessa ja oppimistarkoituksissa. (1; 9.)

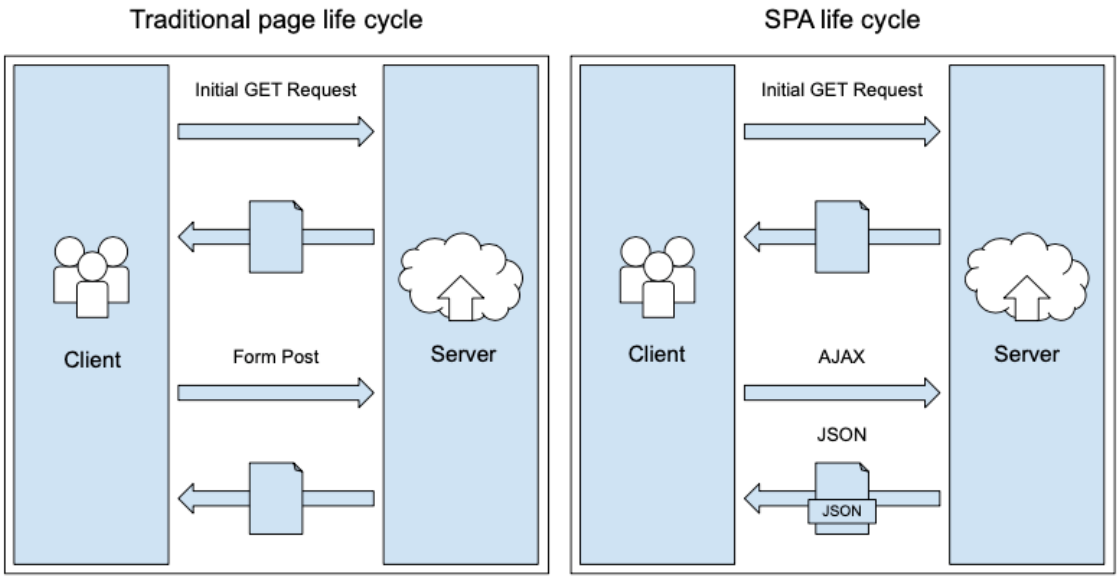
Toinen komponenttimalleista on useita palvelimia ja yksi tietokanta. Tässä vaihtoehdossa verkkopalvelin ei säilytä itsessään mitään dataa, vaan vastaanottaessaan informaatiota se prosessoidaan ja tallennetaan tietokantaan. Informaatiota käsitellään siis palvelimen ulkopuolelta. Tämän komponenttimallin tarkoitus on puhtaasti pyrkiä välttämään sovelluksen kaatuminen kokonaan. Kun yksi palvelimista kaatuu, ohjataan sovelluksen lähettämät pyynnöt toiminnassa oleville palvelimille. Tästä huolimatta, jos tietokanta kaatuu, sovellukselle käy samoin. (1; 9.)

Kolmas vaihtoehto komponenttimalleista on useita palvelimia ja useita tietokantoja, ja luonnollisesti tämä on näistä kolmesta tehokkain ja luotettavin, mutta vastaavasti myös monimutkaisin rakentaa. Tällä komponenttimallilla on kaksi erilaista toteutustapaa, joko säilyttää identtinen data kaikissa tietokannoissa tai jakaa sama määrä dataa eri tietokantoihin. Käytettäessä näistä ensimmäistä vaihtoehtoa toteutukseen riittää kaksi erillistä tietokantaa, mutta jälkimmäisen vaihtoehdon toteuttaminen saattaa vaatia useamman. Molempien vaihtoehtojen kanssa käytetään yleisesti tietokannan hallintajärjestelmän normalisointia. (1; 9.) Tietokannan hallintajärjestelmän normalisoinnilla tarkoitetaan tietokannan jakamista pienempiin osiin ja niiden yhdistämistä relaatioiden kautta, millä pyritään minimoimaan relaatioiden päällekkäisyydet (10).

2.3 Arkkitehtuurin tyypit

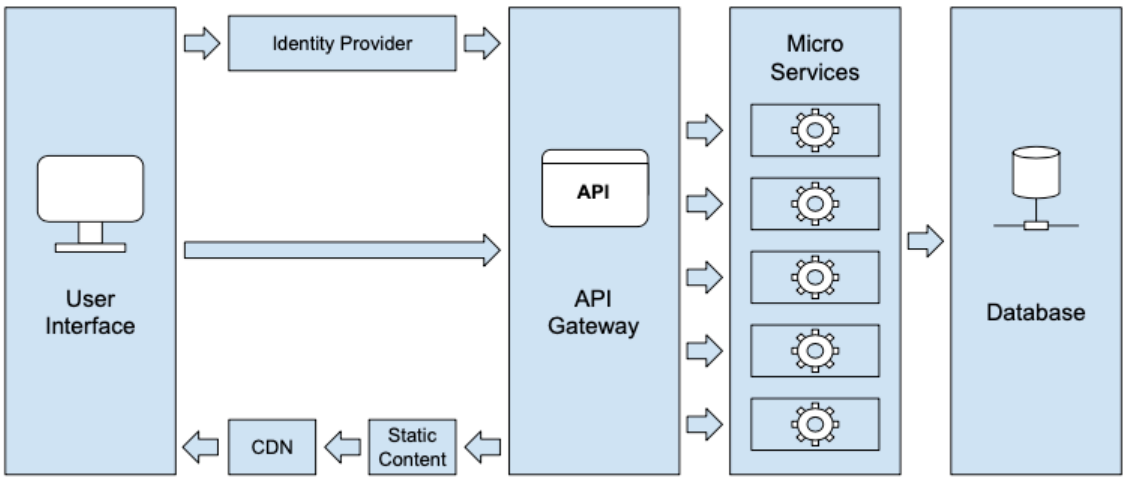
Arkkitehtuurien tyypit voidaan karkeasti jakaa kolmeen eniten käytössä olevaan ryhmään. Niiden lisäksi on kuitenkin olemassa muitakin, vähemmän käytössä olevia arkkitehtuurin tyyppejä, kuten esimerkiksi usean sivun sovellukset ja progressiiviset sovellukset. (6.)

Tämän hetken suosituin tyyppi web-sovellusten arkkitehtuurissa on yhden sivun sovellukset. Sen sijaan, että sivu ladattaisiin uudelleen joka kerta, kun käyttäjä haluaa näkyviin uutta sisältöä, sisältö päivitetään nykyiselle sivulle (kuva 3). Yhden sivun sovellukset minimoivat keskeytykset sovelluksen käytössä ja näin parantavat käyttäjäkokemusta. Tästä syystä ne toimivat kuten perinteiset tietokoneohjelmat, joiden ainoana erona on, että niitä käytetään verkkoselaimen kautta. (1; 6; 9.) Twitter ja Gmail ovat esimerkkejä yhden sivun sovelluksista.



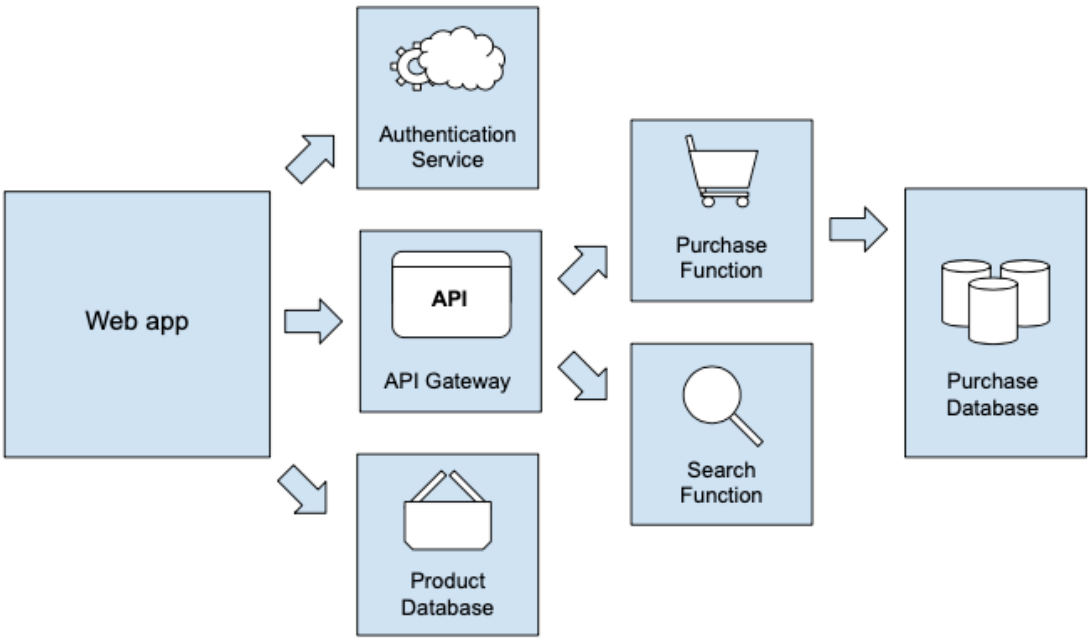
Kuva 3. Perinteisen verkkosivun ja yhden sivun web-sovelluksen syklin vertailu (7).

Toinen vaihtoehto arkkitehtuurien tyypeistä on mikropalvelut. Niiden periaatteena on, että sovellus jaetaan pienempiin, helposti hallittavissa oleviin paloihin, joilla luodaan sovellukseen yksittäisiä toimintoja (kuva 4). Mikropalveluiden käyttäminen arkkitehtuurina nopeuttaa ja yksinkertaistaa kehitystyötä, ja vastaavasti myös sovelluksen varsinaista käyttöönottoa. Komponentit eivät ole itsessään riippuvaisia toisistaan, ja niitä kehitettäessä ei ole välttämätöntä käyttää samaa ohjelmointikieltä. Tämäntyyppinen arkkitehtuuri on päinvastainen verrattuna perinteiseen sovelluskehitykseen, jossa koko sovellus luodaan yhtenä palana. (1; 9.) Esimerkkejä sovelluksista, jotka käyttävät mikropalveluita, ovat Netflix ja Spotify.



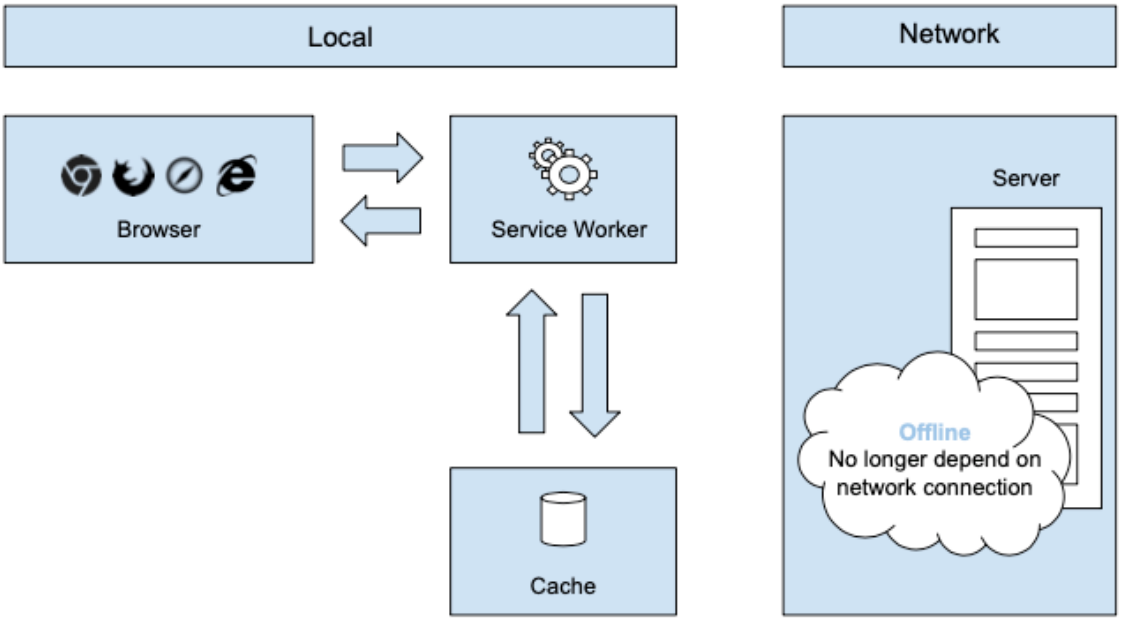
Kuva 4. Mikropalveluita arkkitehtuurinaan käyttävän web-sovelluksen toiminta-sykli (7).

Kolmas eniten käytössä olevista arkkitehtuureista on palvelimettomat sovellukset (kuva 5). Tämäntyyppisessä arkkitehtuurissa sovelluskehittäjät ulkoistavat palvelimen hallinnan ja sen ylläpidon kolmannelle, pilvipalveluita tarjoavalle osapuolelle. Usein palvelimeton arkkitehtuuri yhdistetään myös mikropalveluihin. Palvelimeton arkkitehtuuri on ihanteellinen vaihtoehto sellaiselle yritykselle, joka ei halua hallita palvelimia ja siihen tarvittavia laitteistoja, vaan keskittyä ainoastaan sovelluksen kehitystyöhön. (1; 4; 9.)



Kuva 5. Esimerkki palvelimettoman verkkokaupan toimintasyklistä (7).

Kolmen edellä mainitun tyypin lisäksi on olemassa myös muita, vähemmän käytössä olevia arkkitehtuurin tyyppejä, kuten usean sivun sovellukset ja progressiiviset sovellukset. Usean sivun sovellukset sisältävät nimensä mukaisesti useita sivuja ja sivu tulee ladata uudelleen joka kerta, kun käyttäjä lähettää pyynnön palvelimelle. (11.) Yleensä yritykset käyttävät usean sivun sovelluksia arkkitehtuurina, jos sivusto on erityisen laaja. Esimerkkinä usean sivun sovelluksista voidaan mainita eBay. Progressiiviset sovellukset puolestaan ovat web-pohjaisia sovelluksia, jotka on suunniteltu toimimaan kuten mobiilisovellukset. Tällaiset sovellukset mahdollistavat push-ilmoitukset, offline-käytön ja sovelluksen asentamisen aloitusnäytölle (kuva 6). (12.) Pinterest on hyvä esimerkki progressiivisesta web-sovelluksesta.



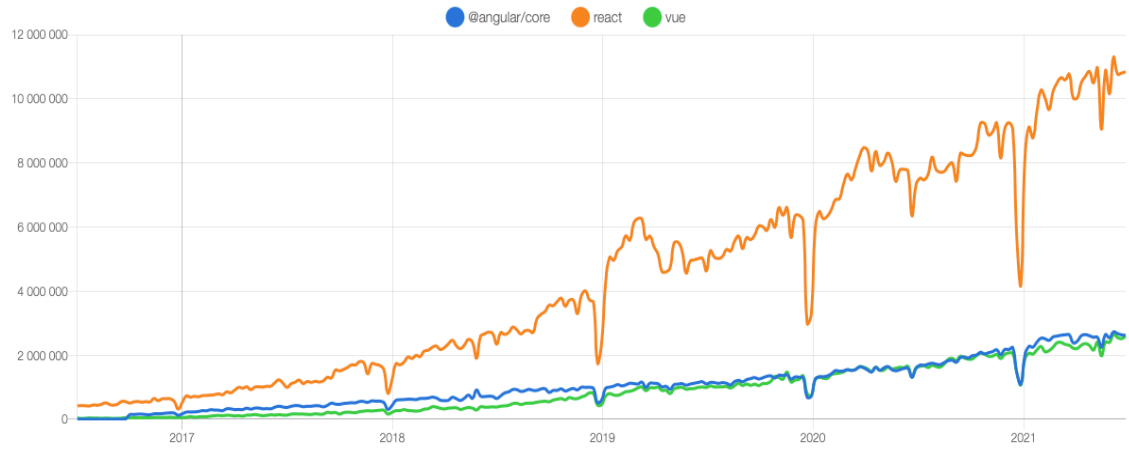
Kuva 6. Progressiivisen web-sovelluksen toimintasykli (7).

2.4 Ohjelmistokehykset web-sovelluskehityksessä

Kun Javascript tuli markkinoille, sen ominaisuudet eivät olleet samaa luokkaa kuin nykypäivänä. Alun perin ohjelmistokehykset kehitettiin paikkaaman puhtaan Javascriptin puutteita ja nopeuttamaan kehitystyötä. Ensimmäisenä laajemmin käytössä olevista ohjelmistokehyksistä voidaan mainita vuonna 2006 kehitetty jQuery, joka on edelleen yksi maailman ladatuimmista ja käytetyimmistä ohjelmistokehyksistä. jQuery mahdollisti ensimmäisten joukossa muun muassa selainriippumattoman kehitystyön. Lisäksi jQuery yksinkertaisti Javascriptin monimutkaisempia toimintoja, kuten standardisoidun AJAX:n käytön ja dokumenttioliomallien käsittelyn. Tämän ansiosta kehittäjät alkoivat ymmärtää paremmin Javascriptin todellisen potentiaalin. (13.)

Nykypäivänä erilaisia ohjelmistokehyksiä on kymmeniä ja jokainen kehittäjä voi valita niiden joukosta käyttötarkoitukseen ja omiin mieltymyksiinsä sopivimman vaihtoehdon. Kiivain keskustelu kuitenkin käydään kolmen suosituimman, uuden sukupolven ohjelmistokehyksen, Angularin, Reactin ja Vuen, välillä. Kolmesta vertailussa olevista ohjelmistokehyksistä React on latausmäärältään

selkeästi suosituin, Angular ja Vue tulevat perässä ja niiden latausmäärät ovat melko samalla tasolla (kuva 7).



Kuva 7. Angularin, Reactin ja Vuen latausmäärien vertailu vuosien 2016–2021 ajalta (14).

Kaikki kolmesta vertailussa olleista ohjelmistokehyksestä ovat toimintaperiaatteeltaan hyvin samankaltaisia, ja niiden väliset erot muodostuvat lähinnä teknisistä toteutustavoista. (15.)

2.4.1 Angular-ohjelmistokehys

Angular on Googlen vuonna 2009 kehittämä avoimen lähdekoodin komponenttipohjainen ohjelmistokehys, jota käytetään käyttöliittymien luomiseen, ja sen ensimmäinen versio kulkee nimellä AngularJS. Myöhemmin vuonna 2014 samat kehittäjät kehittivät uuden sukupolven Angularin, ja se tunnetaan nimellä Angular 2+, joka kattaa ohjelmistokehyksen kaikki versiot toisesta versiosta eteenpäin. Näitä kahta ei kuitenkaan tule sekoittaa keskenään, sillä kyseessä on käytännössä kaksi eri ohjelmistokehystä. (16.)

Uuden sukupolven Angular on suunniteltu siten, että paketti sisältää jo valmiiksi kaikki ominaisuudet, joita vaaditaan laajojen sovelluksien luomiseen. Tästä syystä Angular-paketti on kahteen muuhun ohjelmistokehykseen verrattuna huomattavasti suurempi ja vastaavasti sen käyttö on myös raskaampaa.

Suuresta ominaisuuksien määrästä vuoksi myös sen käyttäjältä vaaditaan useiden tekniikoiden hallitsemista, ja siksi se ei ole kaikista aloittelijaystävällisin ohjelmistokehys. Oman haasteensa Angularin käyttöön saattaa tuoda myös se, että sen ohjelmointikielenä käytetään Typescriptiä. Kaikesta huolimatta Angular sopii erityisen hyvin suurten sovellusten luomiseen, vaikka sen käyttäjältä vaaditaan hieman enemmän osaamista verrattuna muihin komponenttipohjaisiin ohjelmistokehyksiin. (15; 16.)

Googlen kehittämänä ohjelmistokehyksenä Angular on luonnollisesti käytössä suuressa osassa Googlen palveluita, kuten sen hakukoneessa ja Youtubessa. Muita yrityksiä, jotka käyttävät Angularia verkkosivuillaan, ovat muun muassa Apple, Nike ja PayPal. (16.)

2.4.2 React-kirjasto

Facebook julkaisi Reactin vuonna 2013, ja se keräsi runsaasti tukea myös ulkopuolisten kehittäjien joukosta. Se ei suoranaisesti ole ohjelmistokehys vaan Javascript-kirjasto, mutta koska sitä usein verrataan muihin ohjelmistokehyksiin, myös React mielletään siitä syystä ohjelmistokehykseksi. Kahden muun ohjelmistokehyksen tavoin Reactia käytetään web- ja mobiilisovellusten käyttöliittymien ja käyttöliittymäkomponenttien luomisessa. (16.)

Angularista poiketen Reactin peruspaketti on huomattavasti pienikokoisempi ja kevyempi: se sisältää ainoastaan minimaalisen määrän ominaisuuksia. Lisäominaisuuksien puolesta React luottaa kehittäjäyhteisön luomiin lisäosiin. Yleisesti Reactia käytetään yhden sivun sovelluksissa ja siihen liitetään usein muita Javascript-kirjastoja tueksi. Kahden muun ohjelmistokehyksen tavoin myös React on komponenttipohjainen ohjelmistokehys ja sen toimintaperiaate perustuu uudelleenkäytettävien komponenttien luomiseen, mikä nopeuttaa huomattavasti kehitystyötä. Reactin käyttö on esimerkiksi Angulariin verrattuna huomattavasti helpompi oppia, ja samasta syystä se sopii erittäin hyvin myös aloittelijoille. Reactilla on mahdollista luoda suuriakin sovelluskokonaisuuksia, mutta tähän tarkoitukseen esimerkiksi Angular sopii paremmin. (15; 16.)

Reactin lisäksi on saatavilla erityisesti mobiilisovelluskehitykseen tarkoitettu React Native, joka mahdollistaa täysin alustasta riippumattomien natiivisovellusten luomisen käyttäen Javascriptiä ja Reactia (17).

Myös Facebook käyttää Reactia omissa sovelluksissaan. React on yleensä käytössä kaikissa Facebookin hallinnoimissa palveluissa, kuten Facebookissa, Instagramissa ja WhatsAppissa. React on yksi maailman suosituimmista ohjelmistokehyksistä, ja myös useat suuret yritykset tukeutuvat siihen teknologiaratkaisunaan. Esimerkiksi Airbnb, Netflix ja New York Times käyttävät Reactia. (16.)

2.4.3 Vue-ohjelmistokehys

Tässä käsiteltävistä kolmesta ohjelmistokehyksestä nuorin on Vue, jonka kehitti Googlen entinen työntekijä vuonna 2014. Vue on jäänyt hieman kahden muun ohjelmistokehyksen varjoon, mutta viime vuosina se on kerännyt huomattavasti tukea kehittäjäyhteisössä. Huolimatta siitä, että sen taustalla ei toimi suuryritys, Vue on pysynyt hyvin kehityksen tahdissa ja näin ollen pystyy kilpailemaan muiden ohjelmistokehysten kanssa. (16.)

Verrattuna Angulariin ja Reactiin Vue asettuu käytettävyydeltään ja ominaisuuksiltaan suurin piirtein näiden kahden välimaastoon. Pakettien kokoja vertaillessa Vue on selkeästi kaikista pienikokoisin, joten sen käyttöönotto on nopeaa. Muiden ohjelmistokehysten tavoin Vue on komponenttipohjainen ohjelmistokehys, mutta muista poiketen Vuella luotuja komponentteja pystytään lisäämään joustavasti myös osaksi muilla ohjelmistokehyksillä tehtyjä projekteja. Lisäksi se on toimintatavaltaan erittäin yksinkertainen ja helppo oppia, minkä vuoksi se sopii hyvin myös aloitteleville kehittäjille. Vuen pienen markkinaosuuden vuoksi oman haasteensa sen opetteluun voi tuoda internetistä löytyvän oppimismateriaalin vähäisyys. (15; 16.)

Koska Vue ei itsessään sovellu erityisen hyvin suurien verkkosivujen luomiseen, se ei ainakaan toistaiseksi ole vielä käytössä kovinkaan monella yleisesti

tunnetulla verkkosivustolla. Tunnetuimpia Vuea käyttäviä yrityksiä ovat Adobe ja Alibaba. (16.)

3 Data ja sen visualisointi

Nykypäivänä datan kerääminen ja sen hyödyntäminen on kasvavassa määrin olennainen osa eri yritysten yritystoimintaa. Datan keräämisellä pyritään vastaamaan liiketoiminnan kannalta olennaisiin kysymyksiin, arvioimaan yrityksen tekemiä tuloksia ja ennustamaan tulevaisuuden todennäköisyyksiä ja suuntautumismahdollisuuksia. Tehokas datan kerääminen mahdollistaa tietoon perustuvan tutkimustyön ja laaduntarkkailun sekä liiketoimintapäätösten tekemisen. Erityisesti datan keräämistä hyödynnetään tekniikan ja luonnontieteiden aloilla. Datan hyödyntäminen ei kuitenkaan rajoitu näihin aloihin, vaan se on käytössä myös esimerkiksi valtion, rahoituksen, markkinoinnin, historian ja koulutuksen aloilla, jopa urheilussa. (18.)

Datan analysointi on prosessi, jossa tutkitaan kerättyä dataa esimerkiksi sellaisten trendien ja tapahtumaketjujen löytämiseksi, joiden tavoitteena on tukea päätöksentekoa yritystoiminnassa. Data-analyysillä pyritään luomaan malleja ja suhteita analysoitavien tietojen välille. Data-analytiikka puolestaan on laajempi termi, joka sisältää myös edellä mainitun datan analysoinnin. Data-analytiikka terminä kattaa datan koko elinkaaren, joka sisältää datan keräämisen, suodattamisen, järjestelyn, tallentamisen, analysoinnin ja hallinnan. (19.)

Pelkällä datalla ei itsessään ole kovinkaan suurta painoarvoa, jos dataa ei osata analysoida oikein ja näin saada tietoon olennaista informaatiota. Datan muuttaminen graafiseen muotoon saattaa joissain tapauksissa helpottaa sen analysointia ja auttaa ymmärtämään muutoksia kerättävässä datassa. Kaikissa tapauksissa datan graafinen visualisointi ei ole välttämätöntä, eikä edes suositeltavaa. Graafista visualisointia tulisi käyttää ainoastaan sellaisissa tilanteissa, joissa datan havainnointi ja analysointi on helpompaa graafisen kaavion avulla. Esimerkiksi mitattaessa verkkosivun kokonaiskävijämäärää pelkkä kävijäluku riittää kertomaan tarvittavan informaation, kun taas mitattaessa kävijämäärän

kasvua on havainnoinnin kannalta järkevämpää visualisoida kävijämäärä graafiseksi käyräksi.

3.1 Big data eli massadata

Big datalla tai vapaasti suomennettuna massadatalla tarkoitetaan suurien datamäärien analysointia, prosessointia ja tallentamista, kun data kerätään toisistaan erillään olevista lähteistä. Massadataratkaisuilla pyritään vastaamaan muun muassa useiden toisistaan riippumattomien datajoukkojen yhdistämisestä, suurien jäsentelemättömien datamäärien käsittelystä ja piilotetun informaation keräämisestä aikaherkällä tavalla. Yleisesti massadataratkaisut ovat välttämättömyys, kun perinteiset datan analysointi-, prosessointi- ja tallennustekniikat ovat lopputuloksen kannalta riittämättömiä, eikä niillä saavuteta haluttua lopputulosta. (19.)

Perinteisten tekniikoiden lisäksi massadataratkaisuihin on lisätty modernimpia tekniikoita, jotka hyödyntävät laskennallisia resursseja ja lähestymistapoja analyyttisten algoritmien suorittamiseen. Tällaiset dataratkaisut ovat tärkeitä, kun datajoukoista tulee jatkuvasti suurempia ja monimutkaisempia. Massadataympäristön tiedot kerääntyvät yleensä eri sovellusten, antureiden tai muiden ulkoisten lähteiden kautta, eikä kerätty data aina ole välttämättä kerätty tiettyä käyttötarkoitusta varten. Kerättyä dataa voidaan käyttää suoraan sellaisenaan rikastamaan jo olemassa olevaa dataa, tai se voidaan säilyttää myöhempää käyttöä varten. (19.)

Yhteenvetona massadatan ominaisuuksista voidaan mainita, että sen käsittelyssä käytetään dynaamista kaaviota, joka voi sisältää sekä jäsenneltyä että jäsentelemätöntä dataa. Perinteisen datan käsittelyssä puolestaan pystytään käsittelemään ainoastaan jäsenneltyä dataa, kuten relaatiotietokantoja ja erilaisia taulukoita. Perinteisen datan analysointi tapahtuu tavallisesti vasta analysoitavan tapahtuman tai ajanjakson jälkeen, kun taas massadataratkaisuiden avulla analyysi tapahtuu reaaliajassa. Tämä ominaisuus mahdollistaa nopeamman

reagoinnin tapahtumiin ja jatkuvan datan keräämisen. Massadatalle tyypillistä on myös se, että se kerätään useista eri lähteistä. (19; 20.)

3.2 Datatyypit

Datatyypit voidaan jakaa kolmeen eri tyyppiin, jäsenneltyyn, jäsentelemättömään ja puolijäsenneltyyn dataan. Näiden datatyyppien lisäksi on olemassa pienempiä edellä mainittujen sisäisiä datatyyppejä, jotka kuitenkin luetaan suurempiin datatyyppeihin. Kolmen päädatatyypin lisäksi on olemassa metadata, jota käytetään apuna datan käsittelyssä. Data voi olla ihmisten tai koneiden tuottamaa. Ihmislähtöinen data voi olla peräisin esimerkiksi vuorovaikutuksesta verkkopalveluiden tai laitteiden kanssa. Koneella tuotetusta datasta hyvä esimerkki on matkapuhelimen sijaintitiedot. (19.) Kaksi tärkeintä ja yleisimmin käytössä olevaa datatyyppiä ovat jäsennelty ja jäsentelemätön data.

Jäsennelty data on ainoa datatyyppi, jota pystytään käyttämään perinteisessä data-analytiikassa, minkä lisäksi sitä käytetään myös massadataratkaisuissa. Se on datamallin mukaista ja tallennetaan useimmiten taulukkomuodossa. Taulukkomuotoon tallennettu data on ulkonäöltään hyvin yksinkertaista (taulukko 1). Jäsenneltyä dataa käytetään erilaisten yksiköiden välisten suhteiden havainnoinnissa, minkä vuoksi se tallennetaan tavallisesti relaatiotietokantoihin. Jäsenneltyä dataa tuottavat yleisesti erilaiset yrityssovellukset ja tietojärjestelmät, kuten toiminnanohjaus- ja asiakkuudenhallintajärjestelmät. Eri tietokannat ja työkalut tukevat luonnostaan tätä datatyyppiä, joten se ei yleensä vaadi erityisiä toimenpiteitä käsittelyssä tai tallennuksessa. Pankkien tilitapahtumat ovat arkipäiväinen esimerkki jäsennellystä datasta. (19.)

Taulukko 1. Esimerkki jäsennellystä datasta, joka on tallennettu taulukkomuotoon.

OrderID	**ProductID**	**Quantity**
12345	123	2
67890	456	1
87654	789	3

Data, joka ei ole datamallin mukaista, on jäsentelemätöntä dataa. Tämä datatyyppi on joko tekstimuotoista tai binääristä, ja sitä välitetään usein itsenäisten ja toisistaan riippumattomien tiedostojen kautta. Tekstimuotoiset tiedostot voivat olla esimerkiksi sähköpostiviestejä tai blogikirjoituksia. Binääritiedostot puolestaan ovat yleensä mediatiedostoja, jotka sisältävät kuvia, ääntä tai videoita. Jäsentelemättömän datan käsittelyyn ja tallentamiseen vaaditaan useimmiten erityislogiikkaa, eikä sitä voida käsitellä suoraan relaatiotietokannan avulla. Esimerkiksi on välttämätöntä, että videotiedoston oikea koodekki on käytettävissä, jotta kyseinen tiedosto pystytään toistamaan. Jos mediatiedosto tallennetaan relaatiotietokantaan, se tallennetaan taulukkoon binäärisenä objektina. Jäsentelemätön data muodostaa arviolta 80 prosenttia kaikesta tuotetusta datasta. (19.)

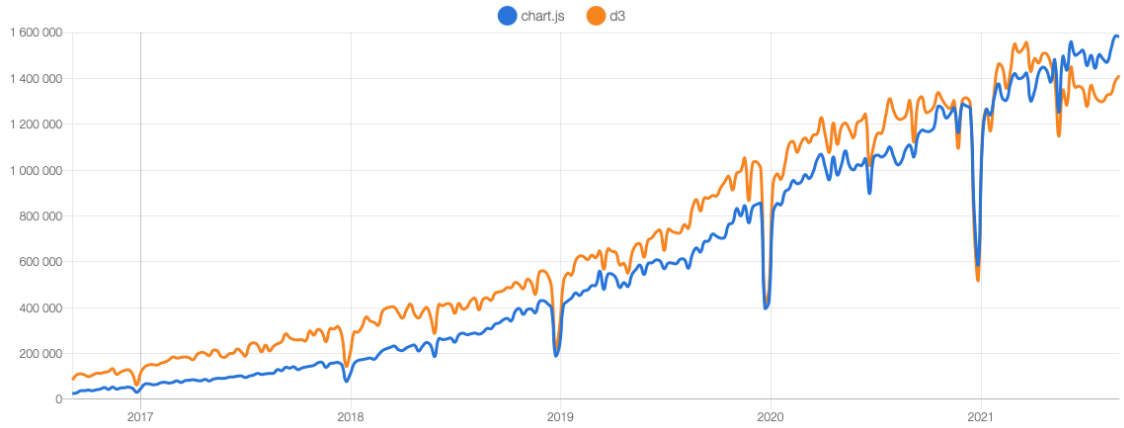
Puolijäsennellyssä datassa on määritelty jäsentelyn taso, mutta se ei itsessään sisällä relaatioita. Puolijäsennelty data tallennetaan yleensä tekstimuotoisiin tiedostoihin, ja se on hierarkkista tai kaaviopohjaista. Yleisiä puolijäsennellyn datan muotoja ovat esimerkiksi JSON- ja XML-tiedostot. Tiedostojen tekstimuotoisuuden vuoksi niiden käsittely on vaivattomampaa kuin jäsentelemättömän datan. Puolijäsennelty data ei kuitenkaan kaikissa tapauksissa ole välttämättä tallennettu tekstimuotoon, ja tässä tapauksessa sen käsittely ja tallennus vaatii myös erityislogiikkaa. Esimerkkinä puolijäsennellystä datasta voidaan mainita RSS-syötteet. (19.)

Metadata ei itsessään ole varsinaista dataa, mutta se sisältää tietoa kerätyn datan ominaisuuksista, rakenteesta ja sen alkuperästä. Metadata on yleisimmin tuotettu koneellisesti, ja se liitetään kerättyyn dataan rikastamaan informaatiota. Metadata on tärkeässä roolissa erityisesti jäsentelemättömän ja puolijäsennellyn massadatan käsittelyssä, säilytyksessä ja analyysissä. Metadataa ovat esimerkiksi XML-tunnisteet, joista selviää dokumentin tekijä ja luontipäivä, tai attribuutit, joista selviää digitaalisen kuvan tiedostokoko ja resoluutio. Metadatan avulla pystytään selvittämään yksityiskohtia, esimerkiksi mikä tiedosto on kyseessä. Se ei anna kuitenkaan ulkopuolisille pääsyä raakadataan, josta metadata on alun perin lähtöisin, ja näin ollen metadatan salaaminen ei ole suojauksen kannalta välttämätöntä. (19; 21.)

3.3 Visualisointiin käytettävät työkalut

Datan visualisointi helpottaa useimmissa tapauksissa kerätyn datan analysointia. Ympyrämallisella kaaviolla pystytään esittämään prosentuaalisia osia tietystä kohteesta, pylväsdiagrammilla vertailemaan tiettyjen ajanjaksojen muutoksia tai käyrällä esittämään kasvun kiihtyvyyttä. Visualisoitu data on myös esteettisesti miellyttävämpää, ja se auttaa ulkopuolisia henkilöitä havaitsemaan olennaisia pääkohtia tai tapahtumia kerätystä datasta.

Web-sovelluksissa datan visualisointiin käytetään useimmiten siihen tarkoitukseen luotuja Javascript-kirjastoja. Niiden avulla esteettisesti miellyttävien ja havainnointia helpottavien kaavioiden luominen onnistuu melko vaivattomasti. Ominaisuuksien ja käyttötapojen vertailuun on valittu kaksi tämän hetken suosituinta datan visualisointiin tarkoitettua Javascript-kirjastoa. Molempien vertailussa olevien kirjastojen latausmäärät ovat viimeisen viiden vuoden ajalla olleet hyvin samalla tasolla. (Kuva 8.)



Kuva 8. Chart.js- ja D3.js-kirjastojen latausmäärien vertailu vuosien 2016–2021 ajalta (22).

Datan visualisointiin tarkoitettujen Javascript-kirjastojen ominaisuudet ovat melko pitkälti samat, joskin joitain ominaisuuksiin tai käyttötapoihin liittyviä eroja kirjastojen välillä on.

3.3.1 Chart.js-kirjasto

Chart.js:n käyttöönotto on nopeaa, ja datan visualisointi vaatii käyttäjältä minimaalisen määrän varsinaista ohjelmointityötä. Kirjasto sisältää valmiiksi luotuja kaaviomalleja, joita käyttäjä voi muokata haluamallaan tavalla. Kaavion ulkoasuun voidaan tehdä muutoksia esimerkiksi vaihtamalla taustaväriä tai viivan paksuutta. Kaaviot ovat myös täysin responsiivisia itsessään. Chart.js:llä kaaviot luodaan HTML:n canvas-elementtiin, mikä parantaa suorituskykyä verrattuna SVG-kuvaan. Lisäksi canvas-renderöinnin etuna on, että kaavio pystytään lataamaan kuvatiedostona tietokoneelle vaivattomasti. Chart.js sopii peruskaavioiden luomiseen, mutta monimutkaisempien kaavioiden luominen kirjaston avulla voi osoittautua haasteelliseksi. Chart.js voidaan integroida ainoastaan Reactilla ja Vuella luotuihin projekteihin, mikä saattaa rajoittaa kirjaston käyttömahdollisuuksia. (23.)

Valittaessa projektiin sopivaa datan visualisointiin tarkoitettua kirjastoa useimmissa tapauksissa Chart.js on optimaalisempi vaihtoehto sen

helppokäyttöisyyden ja nopeuden ansiosta. Kirjaston verkkosivuilla on kattava dokumentaatio, joka sisältää muun muassa esimerkkikoodit erilaisten kaavioiden luomiseen. Dokumentaation avulla kirjaston käyttöönotto ja kaavioiden luominen onnistuu myös hieman kokemattomammalta käyttäjältä melko vaivattomasti.

3.3.2 D3.js-kirjasto

Kuten Chart.js-kirjastolla, myös D3.js-kirjaston avulla pystytään luomaan kaikki mahdolliset peruskaaviot, kuten käyrät, pylväsdiagrammit ja pistekaaviot. Poiketen kuitenkin edellisestä, D3.js-kirjastossa on huomattavasti enemmän sisäänrakennettuja kaaviomalleja, reilusti yli 100 erilaista. Kirjaston avulla pystytään luomaan esimerkiksi animoituja ja karttoihin perustuvia kaavioita. (23; 24.)

D3.js vaatii käyttäjältä huomattavasti enemmän aikaa ja vaivaa kuin Chart.js, eikä se ole kovinkaan aloittelijaystävällinen kirjasto. Se ei sisällä valmiiksi luotuja kaavioita, vaan kaaviot luodaan eräänlaisista rakennuspalikoista. Kaavioiden toteutustapa kuitenkin mahdollistaa hyvinkin monimutkaisten ja räätälöityjen kaavioiden luomisen. Kuten Chart.js:n kohdalla, myös D3.js:n kaaviot luodaan canvas-elementtiin, mutta kirjasto mahdollistaa myös SVG:n käytön. D3.js on mahdollista integroida kaikilla ohjelmistokehyksillä luotuihin projekteihin, mikä parantaa sen käyttömahdollisuuksia ja tekee kirjastosta alustasta riippumattoman. (23.)

D3.js-kirjasto sopii mainiosti projekteihin, joissa vaaditaan erityisesti monimutkaisten ja erikoisten kaavioiden luomista. Käytännössä tällä kirjastolla pystytään tekemään rajattomasti erilaisia kaavioita, joskin niiden luominen vaatii asiaan perehtymistä. Kirjaston verkkosivuilla on myös kattava dokumentaatio, joka helpottaa kirjaston käyttöönotossa ja kaavioiden luomisessa.

3.3.3 Muut kirjastot

Datan visualisointiin tarkoitettuja Javascript-kirjastoja on kymmeniä erilaisia. Perusperiaate jokaisella kirjastolla on sama, mutta käyttötavoissa ja -kohteissa voi olla eroja. Esimerkiksi Recharts- ja React-vis-kirjastot on luotu käytettäväksi ainoastaan Reactin kanssa, kun taas V Charts- ja Trading VueJS -kirjastot toimivat ainoastaan Vuella tehdyissä projekteissa. Osa visualisointiin tarkoitetuista kirjastoista on myös maksullisia. Lisäksi vähemmän käytössä olevien kirjastojen dokumentaatio on yleensä suppeampi, mikä luo omat haasteensa kirjaston käyttöönottoon ja käyttöön.

4 Koronadatan visualisointi

Alkuperäisen suunnitelman mukaan insinöörityöprojektin tavoitteena oli visualisoida Suomen koronatilastoja Terveyden ja hyvinvoinnin laitoksen avoimesta datasta. Kun asiaa tarkemmin tutkittiin, tultiin lopputulokseen, että Suomen koronatilastot ovat helposti saatavilla useissa lähteissä, joten projektissa päädyttiin visualisoimaan koko maailman koronatilastoja, jotka puolestaan eivät ole yhtä helposti saatavilla. Projektissa pyrittiin luomaan yksinkertainen ja helppokäyttöinen web-sovellus, josta selviää tartuntojen määrä kasvukäyränä sekä tartuntojen ja kuolemien kokonaismäärä. Lisäominaisuutena sovellukseen luotiin valikot, joiden avulla pystytään tarkastelemaan tietyn maan ja tietyn aikavälin tilastoja.

4.1 Projektin suunnittelu

Projektin vaiheistus rakentui melko perinteistä kaavaa noudattaen: työvaiheet olivat suunnittelu, työn toteutus, testaus ja lopuksi mahdolliset korjaukset ja parannukset. Suunnitteluvaihe sisälsi ulkoasun suunnittelun, työvaiheiden kartoituksen, teknologioiden valinnan ja vertailukohteiden tarkastelun. Suunnitteluvaiheessa luotiin suurpiirteinen hahmotelma ulkoasusta ja elementtien sijoittelusta Google Docsin Drawings-ominaisuudella. Kaksi viimeistä työvaihetta, testaus ja

mahdolliset korjaukset, suoritettiin tarpeen mukaan useampaan otteeseen, kunnes haluttu lopputulos oli saavutettu.

4.1.1 Projektissa käytetyt työkalut

Projektissa haluttiin luoda mahdollisimman yksinkertaistettu, selkeä ja helppokäyttöinen web-sovellus, jossa kaikki tieto on näkyvillä yhdellä sivulla, joten arkkitehtuuriksi valikoitui yhden sivun sovellus. Sovelluksen pohjana käytettiin React-ohjelmistokehystä, joka on tarkoitettu yksinomaan yhden sivun sovellusten luomiseen. Toteutuksen ja lopputuloksen kannalta React tarjoaa myös uudelleenkäytettävät komponentit ja erinomaisen suorituskyvyn.

Projektin tarkoituksena ei ollut luoda monimutkaista ja vaikeasti hahmotettavaa kaaviota, vaan yksinkertainen käyrämuotoinen kaavio. Pääpainona projektissa pyrittiin luomaan helppolukuinen kaavio asiaan syvemmin perehtymättömälle perustason käyttäjälle. Valmiin kaavion osalta kirjastolta ei vaadittu erityisiä ominaisuuksia, joten visualisointi toteutettiin Chart.js-kirjastolla, joka sisältää valmiit kaaviot ja on siksi nopea ottaa käyttöön. Tämä kirjasto valittiin toteutustavaksi sen helppokäyttöisyyden ja tarvittavien ominaisuuksien vuoksi.

Alun perin sovelluksessa oli tarkoituksena käyttää rajapintana Terveyden ja hyvinvoinnin laitoksen keräämää dataa. Suomen koronatilastoja on kuitenkin visualisoitu useissa eri lähteissä moneen otteeseen ja tilastot ovat helposti saatavilla, joten lopulta päädyttiin visualisoimaan koronadataa maailmanlaajuisesti. Lopulliseksi rajapinnaksi valikoitui COVID19API, joka on luotu Baltimoressa, Yhdysvalloissa sijaitsevan Johns Hopkinsin yliopiston kokoaman koronadatakirjaston pohjalta. Johns Hopkinsin yliopisto puolestaan on kerännyt datan yhteen kirjastoon muun muassa maailman terveysjärjestön ja eri maiden omista tilastoista.

Sovelluksen pääpainona ei ollut luoda esteettisesti näyttävää sovellusta, vaan ulkoasu pyrittiin pitämään selkeänä ja mahdollisimman yksinkertaisena. Toteutuksen tyylisivujen luomiseen käytettiin SCSS-esikääntäjää, joka kääntää

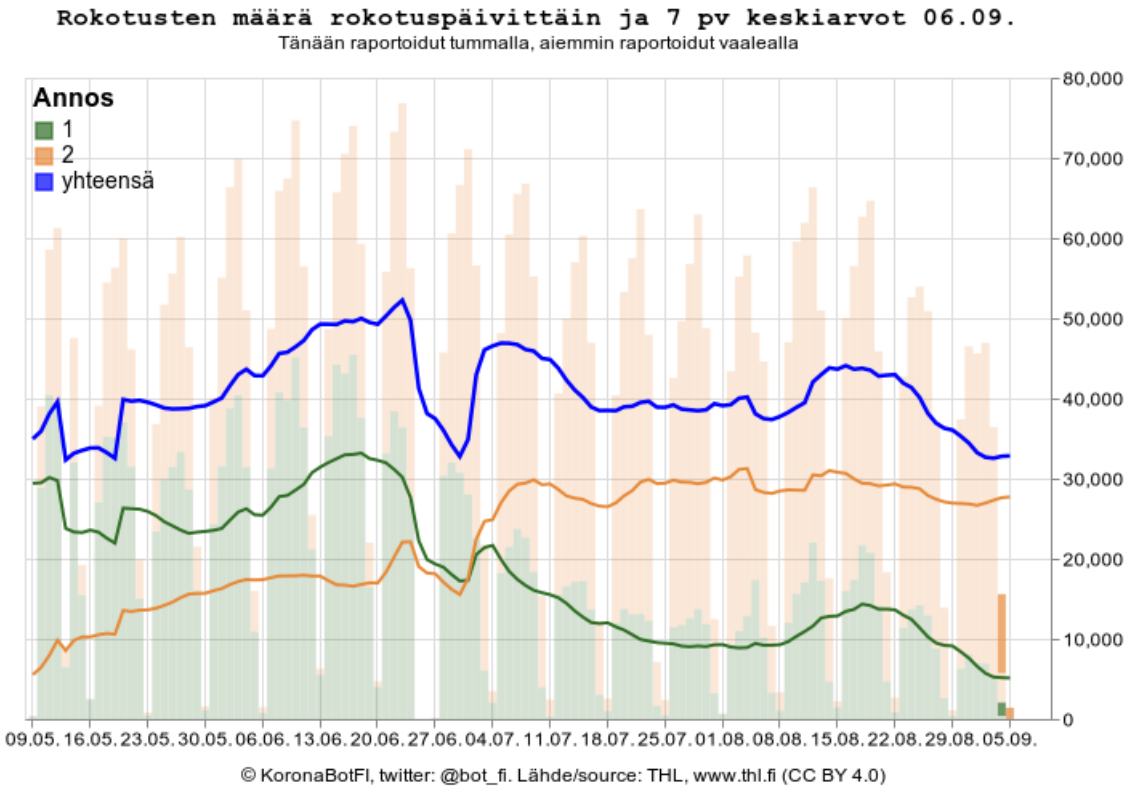
tyylisivun lopuksi CSS-muotoon. SCSS-tyylisivu luotiin Node-sass-kirjaston avulla, jolla tyylitiedostosta saadaan automaattisesti yhteensopiva kaikkien yleisimpien selainten välillä.

Projektiin lisättiin Axios-kirjasto, jolla pystytään parantamaan hieman sovelluksen suorituskykyä tehtäessä JSON-pyyntöjä. Kirjastolla on myös laajempi selaintuki kuin sisäänrakennetulla fetch-metodilla. (25.) Pelkkien lukujen visualisoinnin tueksi sovelluksessa käytettiin myös React-number-format-kirjastoa, jonka avulla suuret luvut pystytään muuttamaan helposti luettavaan muotoon.

4.1.2 Vertailukohteet

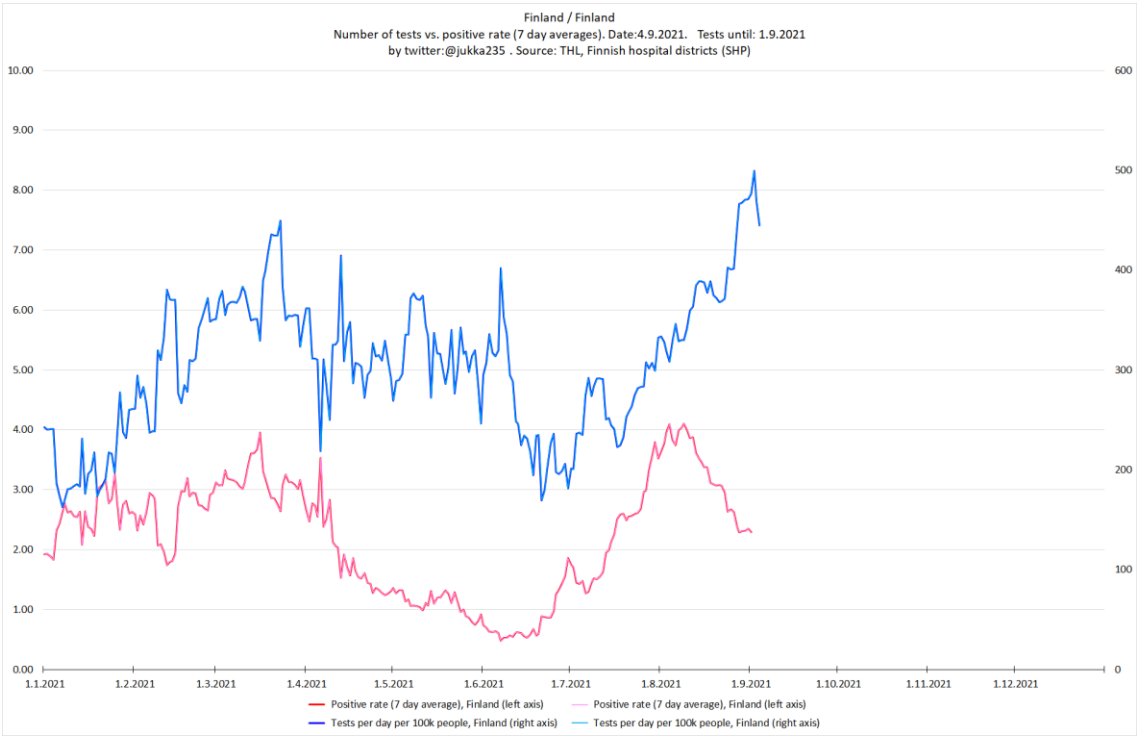
Projektin suunnitteluvaiheessa pyrittiin tutkimaan jo olemassa olevia koronadataan perustuvia visualisointeja, joiden pohjalta pystyttiin kartoittamaan visualisoitavia tilastoja. Suunnitteluvaiheessa pyrittiin myös tarkkailemaan, millaiset kaaviotyypit ja esitystavat sopivat lopputuloksen kannalta parhaiten toteutukseen.

Ensimmäisenä vertailukohteena käytettiin bot_fi-nimisen Twitter-käyttäjän visualisoimia kaavioita. Käyttäjä julkaisee päivittäin automaattisesti Terveyden ja hyvinvoinnin laitoksen avoimeen dataan perustuvia kaavioita useista eri tilastoista, esimerkiksi rokotettujen määrästä ja raportointiviiveestä. Lähteessä on käytetty monipuolisesti erilaisia kaaviotyyppejä, yleensä käyriä ja pylväsdiagrammeja. Yksinkertaisempia tilastoja, kuten uusien tartuntojen määriä, on julkaistu myös pelkkinä lukuina. Kaaviot ovat melko monimutkaisia, ja niiden ymmärtäminen vaatii aikaa ja asiaan perehtymistä, joten niiden osalta selkeää vertailukohtaa ei saatu. Lisäksi julkaistut kaaviot kuvaavat eri tapahtumia kuin projektissa kuvataan. (Kuva 9.)



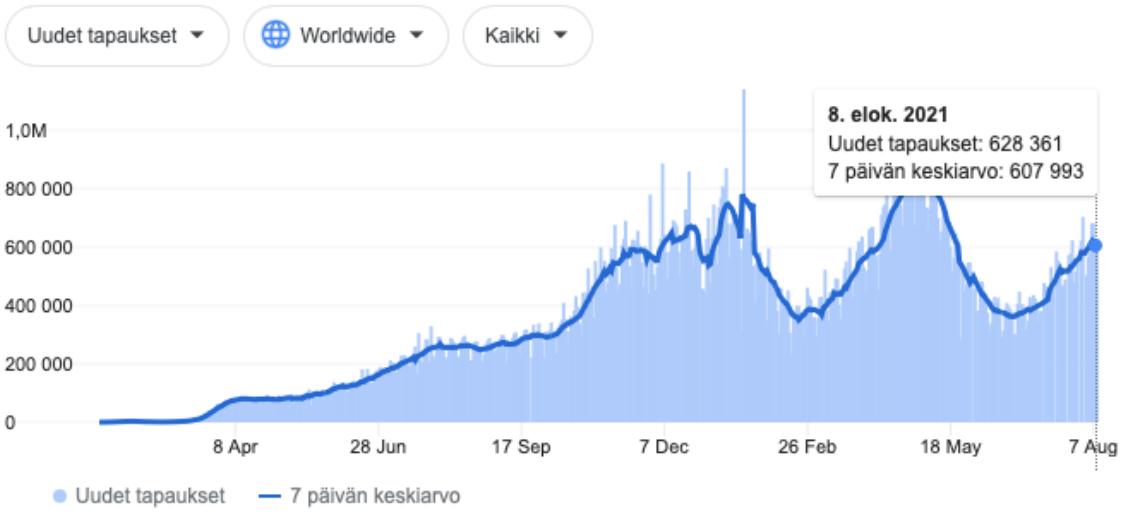
Kuva 9. Esimerkki bot_fi-nimisen Twitter-käyttäjän kaaviomuotoon visualisoidusta koronadatasta. Tilastoina kaaviossa on käytetty rokotteiden määrää ja rokotusten keskiarvoa. (26.)

Myös toisena vertailukohteena käytettiin Twitter-käyttäjän jukka235 visualisoimia kaavioita, jotka nekin perustuvat Terveyden ja hyvinvoinnin laitoksen avoimeen dataan. Käyttäjä visualisoi muun muassa ilmaantuvuutta tietyissä ikäryhmissä ja tietyillä alueilla ja otettujen testien määrää. Lähteessä käytetään yleensä käyrämuotoista visualisointitapaa, mutta jonkin verran myös pylväsdiagrammeja. Kaaviot ovat helposti ymmärrettäviä ja visualisoitavat tilastot samankaltaisia projektin kanssa. (Kuva 10.)



Kuva 10. Esimerkki jukka235-nimisen Twitter-käyttäjän kaaviomuotoon visualisoimasta koronadatasta. Tilastoina kaaviossa on käytetty tehtyjä koronatestejä ja positiivisia koronatestejä. (27.)

Kahdessa edellä esitellyssä vertailukohteessa on visualisoitu terveyden ja hyvinvoinnin laitoksen keräämää dataa, joten kolmanneksi vertailukohteeksi valittiin Google-haulla koko maailman koronatilastot. Tilastot ovat kokonaisuudessaan keränneet yhteen Wikipedia, The New York Times, Our world in data ja Johns Hopkinsin yliopisto. Edellä mainituista Johns Hopkinsin yliopisto on kerännyt yhteen koronadatakirjaston, joka oli pohjana myös insinöörityöprojektissa käytetyssä rajapinnassa. Lähteessä visualisoidaan uudet tartunnat ja kuolemat pylväsdiagrammeina ja valitun aikavälin keskiarvo käyrämuodossa (kuva 11). Lisäksi tilastoja voidaan tarkastella kunkin maan kohdalla erikseen ja eri aikaväleillä. Myös tässä lähteessä on visualisoitu tartuntojen kokonaismäärä ja uudet tartunnat sekä kuolemien kokonaismäärä ja uudet kuolemat lukuina.



Kuva 11. Esimerkki Googlen kaaviomuotoon visualisoimasta koronadatasta. Tilastoina kaaviossa on käytetty uusien tartuntojen määrää ja tartuntamäärien keskiarvoa viimeisen seitsemän päivän ajalta. (28.)

Tarkasteltaessa vertailukohteita havaittiin, että yleisimmät kaaviomuodot koronadataa visualisoitaessa ovat käyrät ja pylväsdiagrammit, ja tämän perusteella projektin kaaviomuodoksi valittiin käyrä. Yksinkertaisempien tilastojen visualisointi pelkkinä lukuina osoittautui myös yleiseksi esitystavaksi, joten myös se valittiin osaksi projektia. Projektiin lisättiin myös valikot, joista on mahdollista vaihtaa tarkasteltavaa maata ja aikaväliä.

4.2 Sovelluksen rakentaminen

4.2.1 Sovelluksen komponentit

Insinöörityössä tehdyn sovelluksen komponenttien määrä pyrittiin pitämään pienenä. Lopulliseen sovellukseen komponentteja kertyi yhteensä neljä, ja niitä käytetään App-pääkomponentin kautta (kuva 12).

Kuva 12. Insinöörityössä tehdyn sovelluksen sisäiset komponentit.

App-pääsovelluksessa ensimmäisenä on Heading-komponentti. Sen sisältö on hyvin yksinkertainen, ja nimensä mukaisesti se sisältää sovelluksen otsikon. Heading-komponenttiin lisättiin alaotsikko, joka näyttää tarkasteltavan maan kyseisellä hetkellä.

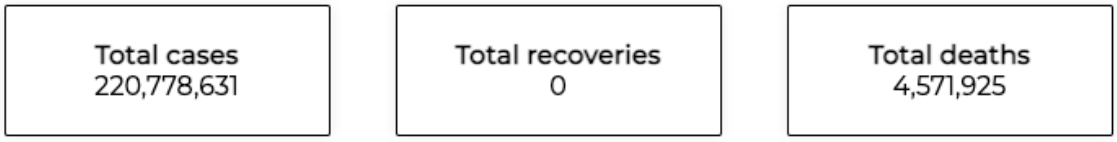
Toisena komponenttina App-pääkomponentissa on Totals-komponentti, joka sisältää tartuntojen, parantuneiden ja kuolleiden määrien tilastot lukuina. Visualisoidut luvut muutettiin helposti luettavaan muotoon lisäämällä erotteleva pilkku tuhansien väliin käyttäen react-number-format-kirjastoa. Totals-komponentin sisällä hyödynnettiin uudelleen käytettävää Card-komponenttia, jonka avulla visualisoidut luvut sidotaan kortin sisälle. Card-komponentti lisättiin projektiin ainoastaan esteettisistä syistä.

Projektin aiheen kannalta olennaisin komponentti on alimpana App-pääkomponentissa. Komponenttiin luotiin kaavio Chart.js-kirjastolla, johon tuodaan visualisoitava data rajapinnasta ja muutetaan käyrämuotoon.

4.2.2 Datan varsinainen visualisointi

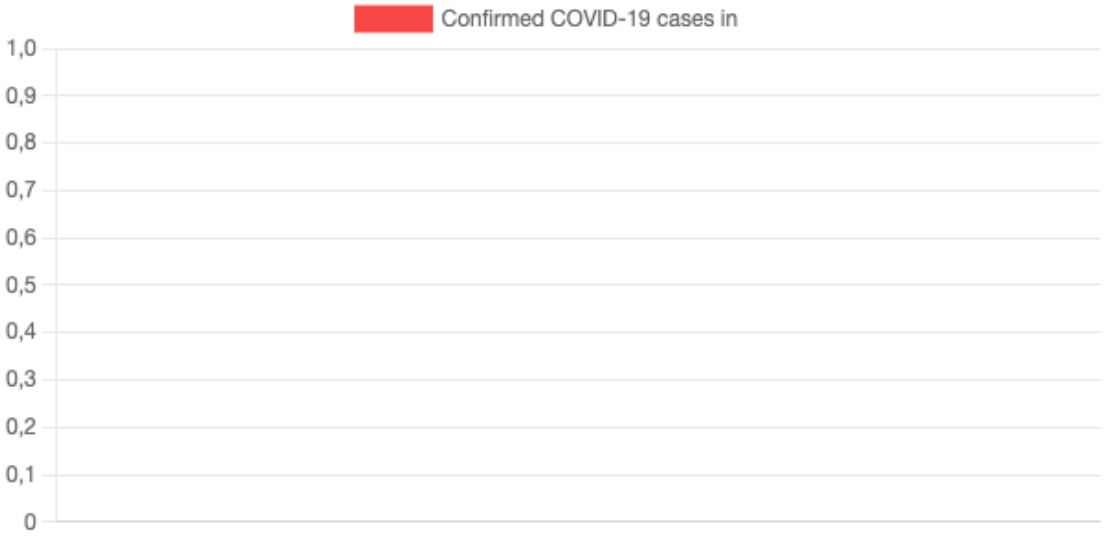
Visualisoinnin ensimmäisenä askeleena pyrittiin visualisoimaan tartuntojen, parantuneiden ja kuolemien kokonaismäärää pelkästään lukuina. Näin pystyttiin samalla testaamaan koodin ja rajapinnan toimivuutta. Tässä vaiheessa projektia ei vielä tehty vertailua tai tarkkailtu saatujen lukujen todenmukaisuutta, vaan se

suoritettiin vasta myöhemmin varsinaisessa testaus- ja vertailuvaiheessa. Saadut luvut sijoitettiin kutakin tilastoa visualisoivaan Card-komponenttiin (kuva 13).



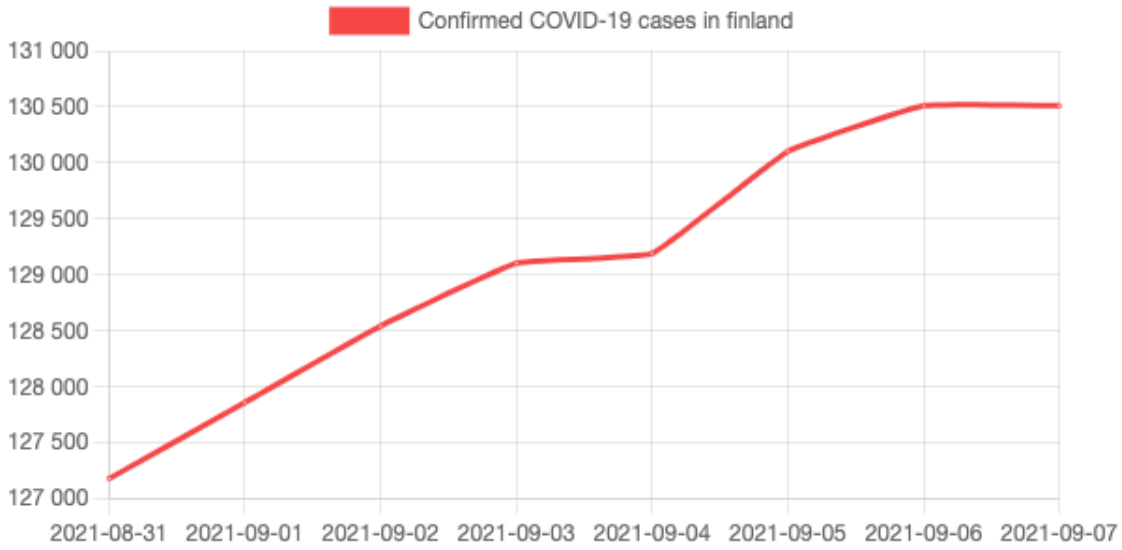
Kuva 13. Card-komponentit, joihin visualisoitiin tartuntojen, parantuneiden ja kuolleiden kokonaismäärä. Esimerkkitilastona käytettiin koko maailman lukuja.

Visualisoinnin toisessa vaiheessa toteutettiin käyrämuotoinen kaavio Chart.js-kirjastolla ChartLine-komponenttiin. ChartLine-komponentin sisällä käytettiin kirjastosta löytyvää Line-komponenttia, jolla pystytään luomaan käyrämuotoinen kaavio automaattisesti. Kaavion ulkonäköön pystytään tekemään muutoksia Line-komponentin datasets-jonon sisäisten attribuuttien avulla. Jono sisältää myös data-attribuutin, jonka arvoksi lisättiin myöhemmässä vaiheessa visualisoitava data rajapinnasta. Tämän työvaiheen lopputulokseksi saatiin tyhjä pohja käyrämuotoiselle kaaviolle (kuva 14).



Kuva 14. Chart.js-kirjastolla luotu tyhjä kaaviopohja.

Lopuksi edellä esiteltyyn kaavioon sijoitettiin kasvukäyrä tartuntojen kokonaismäärästä. Käyrän luomista testattaessa haettiin rajapinnasta By country -päätepiste, johon pystytään määrittämään yksityiskohtaisesti tarkasteltava maa, tilasto ja aikaväli. Testivaiheessa haettiin Suomen tartuntamäärät viikon aikavälillä ja lopputulokseksi saatiin onnistuneesti päällisin puolin oikealta näyttävä kasvukäyrä (kuva 15).

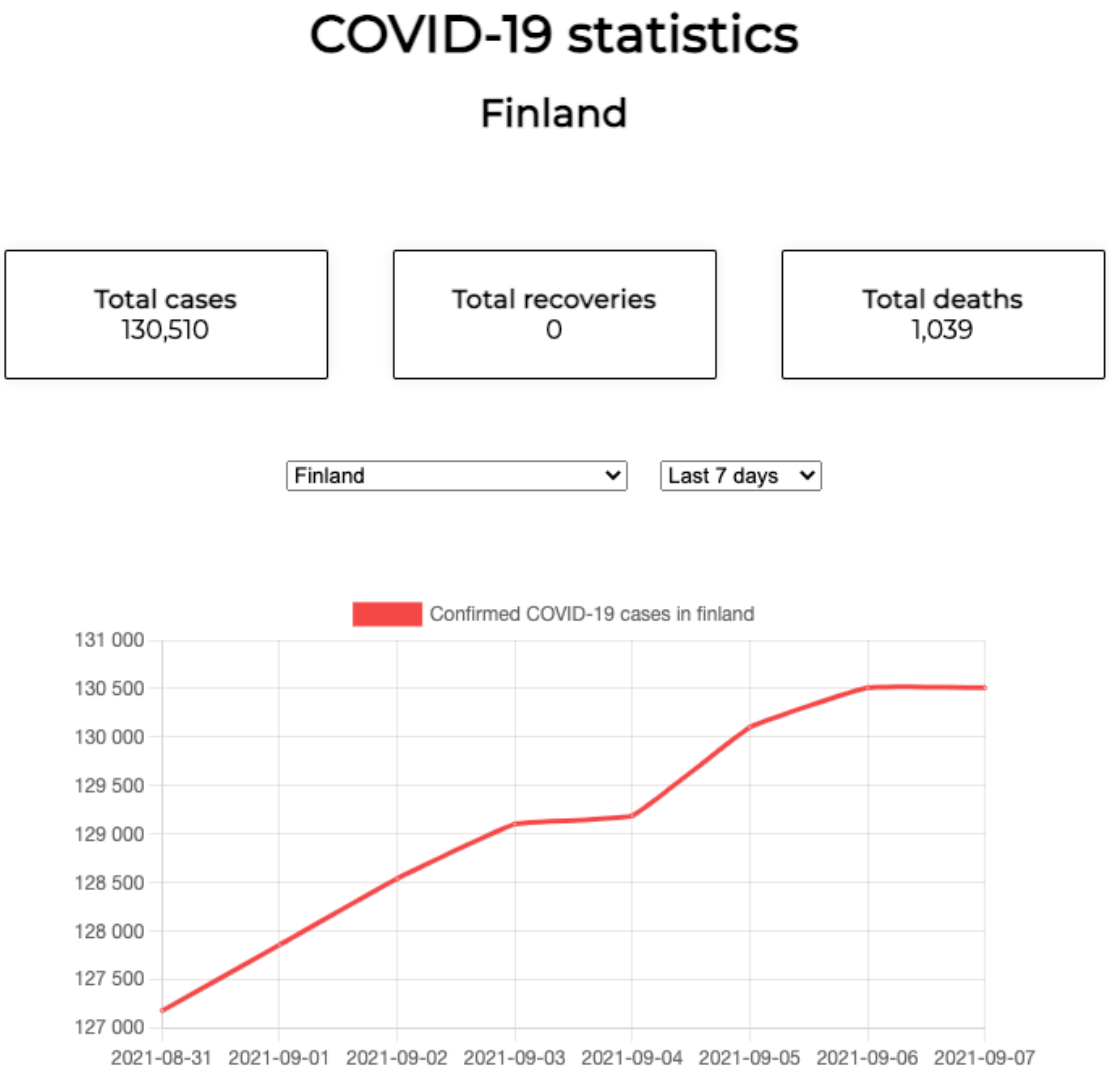


Kuva 15. Chart.js-kirjastolla luotu käyrämuotoinen kaavio, johon haettiin rajapinnasta Suomen tartuntamäärät viimeisen seitsemän päivän ajalta.

Lopuksi sovellukseen haluttiin luoda lisäominaisuutena mahdollisuus vaihtaa tarkasteltavaa maata ja aikaväliä pudotusvalikosta. Kaikki valtiot eivät raportoi omista koronatilastoistaan tai niissä ei ole havaittu yhtään tartuntatapausta, joten maavalikosta puuttuvat ainakin Pohjois-Korea, Turkmenistan sekä useat pienet Polynesian saarivaltiot (29). Tarkasteltavaksi aikaväliksi asetettiin oletuksena viimeiset seitsemän päivää, ja aikaväliä on mahdollista vaihtaa viimeisen 7, 14 tai 30 päivän välillä. Lisäksi Heading-komponenttiin lisättiin vielä alaotsikko, joka näyttää selkeästi tarkasteltavan maan kyseisellä hetkellä.

Projektin lopputulokseksi saatiin ulkonäöllisesti selkeä ja yksinkertainen sovellus, jossa luodut elementit ovat loogisessa järjestyksessä. Sovelluksesta

pystytään havaitsemaan kaikki olennaiset asiat nopealla vilkaisulla, eikä siihen ole lisätty tarpeetonta sisältöä esimerkiksi esteettisistä syistä. (Kuva 16.)



Kuva 16. Sovelluksen ulkoasu. Esimerkkitilastona kaaviossa on käytetty Suomen tartuntoja viimeisen seitsemän päivän ajalta.

4.3 Testaus ja tulosten vertailu

Testausvaiheessa tuloksia vertailtaessa apuvälineenä käytettiin Worldometerin visualisoimia lukemia. Ensimmäisenä haettiin Worldometeristä ja luodusta sovelluksesta Suomen tartuntojen kokonaismäärä. Worldometerin lukema näytti hieman suuremmalta kuin sovelluksessa, mutta ero selittyi rajapinnan

mahdollisella päivittymisviiveellä. Kun vähennettiin Worldometerin kokonaistartuntamäärästä uusien tartuntojen määrä, saatiin lopulliseksi lukemaksi sama kuin luodussa sovelluksessa. Päivittymisviive voidaan havaita myös luodusta kaaviosta, jossa tartuntamäärät näyttävät samaa lukemaa kahtena viimeisimpänä päivänä. Kuolleiden määrän osalta puolestaan Worldometerin lukema oli yhdeksän tapausta pienempi kuin luodussa sovelluksessa.

Toisena vertailukohteena käytettiin Italian tilastoja, jotka haettiin Worldometeristä ja luodusta sovelluksesta. Italian kohdalla tartuntojen kokonaismäärä ja kuolemien määrä olivat molemmissa lähteissä täsmälleen samat.

Kolmanneksi maaksi vertailuun valittiin Kiina. Kiinan osalta erot olivat huomattavan suuria. Worldometerin mukaan tartuntojen kokonaismäärä Kiinassa oli noin 95 000, kun taas luodussa sovelluksessa lukema oli noin 107 000. Myös kuolemien määrän osalta ero oli huomattava: Worldometerin lukema oli noin 200 tapausta vähemmän. Tarkemmin tutkittaessa havaittiin, että Kiinan lukemaan ei automaattisesti lisätä Hong Kongin lukemia Worldometerin toimesta. Kun lisättiin Hong Kongin lukemat Kiinan lukemiin, olivat Worldometerin ja luodun sovelluksen lukemat täysin samat.

Yhteenvetona voidaan todeta, että rajapinnasta saatavat lukemat ovat suurelta osin samassa linjassa todellisten lukemien kanssa. Erot tilastoissa ovat pieniä, ja ne selittyvät hyvin todennäköisesti rajapinnan ja Worldometerin päivittymisviiveellä ja eri maiden raportointiviiveellä.

Testausvaiheessa havaittiin myös, että rajapinnan avulla parantuneiden määrää ei saada visualisoitua, vaan niiden osalta lukema näyttää jokaisen maan kohdalla nollaa. Verrattaessa lukemaa Worldometerin lukemiin havaittiin, että maat raportoivat parantuneiden määrän, joten parantuneiden osalta rajapinnassa itsessään on mahdollisesti ongelma. Lopputuloksena sovelluksesta päädyttiin poistamaan Card-komponentti, joka näyttää parantuneiden määrän lukuna.

Lisäksi tarkasteltaessa eri maiden kaavioita voidaan havaita käyrän näyttävän hieman erilaiselta joidenkin maiden kohdalla. Erot käyrissä johtuvat siitä, että

esimerkiksi Australia, Kiina ja Yhdysvallat raportoivat tartuntamääriä useampaan otteeseen päivän aikana, kun taas esimerkiksi Italia ja Suomi raportoivat tartuntamäärät kerran päivässä.

4.4 Jatkokehitysmahdollisuudet

Insinöörityön sovelluksessa käytetystä rajapinnasta on saatavilla kolme eri kuukausimaksullista versiota. Maksulliset versiot tarjoavat esimerkiksi laajemman yhteenvedon, jossa on valmiiksi laskettu eri tilastoja miljoonaa ihmistä kohden. Maksullisilla versioilla on myös mahdollista visualisoida rokotustilastoja ja matkustusrajoituksia.

Rajapinnan maksuttomassa versiossa ominaisuudet ovat hyvin rajoitettuja, ja sen avulla pystytään visualisoimaan ainoastaan hyvin perustason tilastoja. Näin ollen maksuttomalla versiolla jatkokehitysmahdollisuuksia ei ole kovinkaan paljon. Sovellukseen pystytään lisäämään useita kaavioita, joissa visualisoidaan perustason tilastoja, mutta ilman rajapinnan maksullista versiota sovelluksesta on käytännössä mahdotonta tehdä kilpailukykyistä tai esimerkiksi kaupallisesti kannattavaa.

Lisäarvoa luodulle sovellukselle on mahdollista saada muuttamalla se mobiilisovellukseksi. Android-sovelluskaupasta vertailtavaksi sopivia mobiilisovelluksia löytyi yksi, ja iOS-sovelluskaupasta vastaavia sovelluksia ei löytynyt ollenkaan. Myös mobiilisovellukseen tulisi lisätä rajapinnan maksullinen versio, jotta sovellukseen voidaan lisätä uusia ominaisuuksia.

5 Yhteenveto

Insinöörityöprojektissa oli tavoitteena luoda yksinkertainen, selkeä ja helppokäyttöinen web-sovellus, jolla pystytään tarkastelemaan eri maiden koronatilastoja. Lisäksi pyrittiin kartoittamaan lopputuloksen kannalta järkeviä tekniikoita ja mahdollisia työkaluja, jotta opittuja asioita voidaan hyödyntää myös tulevaisuudessa.

Web-sovellukset rakentuvat yleensä selainpuolen ja palvelinpuolen komponenteista. Näistä ensimmäinen vastaa siitä, mitä käyttäjä näkee, kun taas palvelinpuolen komponentti vastaa siitä, mitä pinnan alla tapahtuu. Web-sovelluksissa on olemassa kolme eri komponenttimallia: yksi palvelin ja yksi tietokanta, useita palvelimia ja yksi tietokanta ja useita palvelimia ja useita tietokantoja. Näistä esimerkiksi ensimmäinen sopii parhaiten testaukseen ja oppimistarkoituksiin, kun taas viimeinen on komponenttimalleista tehokkain ja luotettavin, minkä vuoksi se sopii suuriin ja monimutkaisiin sovelluksiin. Arkkitehtuurin tyyppi määräytyy puhtaasti sen mukaan, millaista sovellusta ollaan luomassa. Arkkitehtuurin tyypeistä tällä hetkellä suosituin on yhden sivun sovellukset. Työssä vertailtiin myös kolmea suosituinta ohjelmistokehystä: Angularia, Reactia ja Vuea. Jokainen vertailluista ohjelmistokehyksistä on hyvin samankaltainen, ja käytettävän teknologian valinta on lähinnä tottumiskysymys. Vertailluista ohjelmistokehyksistä React ja Vue sopivat myös hyvin aloittelijoille.

Nykyisin dataa kerätään kasvavissa määrin eri lähteistä ja kerättyä dataa voidaan hyödyntää esimerkiksi vaikeiden liiketoimintapäätösten tekemisessä. Kerätty data jakautuu kolmeen eri datatyyppiin: jäsenneltyyn, jäsentelemättömään ja puolijäsenneltyyn dataan. Massadataratkaisuissa puolestaan dataa kerätään suuria määriä kerrallaan ja kerätty data voidaan tallentaa myöhempiä käyttötarkoituksia varten. Sen avulla dataa voidaan analysoida myös keräyksen aikana, kun taas perinteisessä datan keräyksessä data analysoidaan vasta keräykseen käytetyn ajanjakson jälkeen. Työssä vertailtiin kahta erityisesti visualisointiin tarkoitettua Javascript-kirjastoa, Chart.js:ää ja D3.js:ää. Vertailun tuloksena voitiin todeta, että Chart.js on sopivampi vaihtoehto, mikäli halutaan luoda yksinkertaisia kaavioita, kun taas D3.js sopii paremmin monimutkaisempien ja yksityiskohtaisempien kaavioiden luontiin.

Lopputuloksena projektissa saatiin aikaan tavoitelluilta ominaisuuksiltaan toimiva web-sovellus. Ensimmäisenä askeleena sovellukseen luotiin kortit, joihin tuotiin rajapinnasta tartuntojen, parantuneiden ja kuolemien kokonaismäärät lukuina. Testausvaiheessa rajapinnassa havaitun ongelman vuoksi sovelluksesta päädyttiin poistamaan kortti, joka näyttää parantuneiden määrän lukuna.

Sovellukseen luotiin Chart.js-kirjaston avulla käyrämuotoinen kaavio. Luotuun kaavioon tuotiin rajapinnasta valitun maan tartuntamäärä ja muutettiin tilastot kasvukäyräksi. Lisäominaisuudeksi sovellukseen tehtiin vielä lopuksi pudotusvalikot, joista pystytään valitsemaan tarkasteltava maa ja aikaväli.

Tavoitteena oli siis luoda tyhjästä web-sovellus, jonka avulla pystytään tarkastelemaan eri maiden koronatilastoja. Sovelluksesta pyrittiin tekemään selkeä ja helppokäyttöinen, ja kaaviot pyrittiin pitämään loppukäyttäjälle helppolukuisina. Haluttuun tavoitteeseen päästiin osittain. Ulkoasullisesti sovelluksesta saatiin suunnitelmien mukainen, mutta rajapinnan tuomien rajoituksien vuoksi sovelluksen sisältö jäi haluttua suppeammaksi. Projektin toisena tavoitteena oli saada selkeä kuva siitä, millaisia asioita ja tilastoja voidaan visualisoida ja millaisilla tekniikoilla ja työkaluilla visualisointi on järkevää tehdä. Tämän tavoitteen osalta päästiin haluttuun tavoitteeseen, ja opittuja asioita pystytään hyödyntämään myös tulevaisuudessa.

Sovelluksessa käytettiin COVID19API:n maksutonta versiota, jolla jatkokehitysmahdollisuudet ovat melko rajalliset. Maksuttoman version avulla ei ole mahdollista luoda uniikkia ja näin ollen kilpailukykyistä ja kaupallisesti kannattavaa sovellusta. Lisäarvoa sovellukseen on mahdollista saada muuttamalla se mobiilisovellukseksi ja käyttämällä rajapinnan kuukausimaksullista versiota, koska vastaavia ja vertailukelpoisia mobiilisovelluksia on saatavilla vain vähän.

Lähteet

1 Banga, Swapnil. 2021. What is web application architecture? Components, models, and types. Verkkoaineisto. Hackr.io. <https://hackr.io/blog/web-application-architecture-definition-models-types-and-more>. 28.5.2021. Luettu 14.6.2021.

2 MacPherson, Mary. 2019. Website vs web app: what's the difference? Verkkoaineisto. Medium. <https://medium.com/@essentialdesign/website-vs-web-app-whats-the-difference-e499b18b60b4>. 26.7.2019. Luettu 15.6.2021.

3 Nyakundi, Hillary. 2021. What is the difference between a website and a web application? Verkkoaineisto. FreeCodeCamp. <https://www.freecodecamp.org/news/difference-between-a-website-and-a-web-application/>. 24.3.2021. Luettu. 16.5.2021.

4 Buzan, Dan. 2019. Web application architecture: definitions, types, and components. Verkkoaineisto. DZone. <https://dzone.com/articles/web-application-architecture-definition-types-and>. 2.8.2019. Luettu 17.6.2021.

5 What is UI design? What is UX design? UI vs. UX: what's the difference. Verkkoaineisto. UX Planet. <https://uxplanet.org/what-is-ui-vs-ux-design-and-the-difference-d9113f6612de>. Luettu 21.6.2021.

6 Web application architecture: definition, components, models, and types. Verkkoaineisto. AHT TECH JSC. <https://www.arrowhitech.com/web-application-architecture-definition-components-models-and-types/>. Luettu 21.6.2021.

7 Dhaduk, Hiren. 2021. An ultimate guide of web application architecture. Simform. <https://www.simform.com/blog/web-application-architecture/>. 9.9.2021. Luettu 27.9.2021.

8 Introduction to the server side. Verkkoaineisto. Mozilla. <https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction>. Luettu 28.6.2021.

9 Web application architecture: definition, models, types, and more. 2021. Verkkoaineisto. Content snare. <https://contentsnare.com/web-application-architecture/>. Päivitetty 27.1.2021. Luettu 28.6.2021.

10 Normalization. Verkkoaineisto. Javatpoint. <https://www.javatpoint.com/dbms-normalization>. Luettu 28.6.2021.

11 Beyond SPAs: alternative architectures for your PWA. Verkkoaineisto. Google developers. <https://developers.google.com/web/updates/2018/05/beyond-spa>. Luettu 30.6.2021.

12 Definition for progressive web app (PWA). Verkkoaineisto. Software AG. <https://techradar.softwareag.com/technology/progressive-web-apps/>. Luettu 30.6.2021.

13 Katz, Yehuda; Bibeault, Bear & De Rosa, Aurelio. 2015. jQuery in action. Third edition. E-kirja. Manning publications.

14 @angular/core vs react vs vue. Verkkoaineisto. Npm trends. <https://www.npmtrends.com/react-vs-vue-vs-@angular/core>. Luettu 6.7.2021.

15 Tyson, Matthew. 2021. Angular, React, Vue: Javascript frameworks compared. Verkkoaineisto. InfoWorld. <https://www.infoworld.com/article/3606737/angular-react-vue-javascript-frameworks-compared.html>. 11.2.2021. Luettu 6.7.2021.

16 Das, Sneha. 2021. AngularJS vs. ReactJS vs. VueJS: a detailed comparison. Verkkoaineisto. DZone. <https://dzone.com/articles/angularjs-vs-react-js-vs-vue-js-a-detailed-compari>. 16.6.2021. Luettu 6.7.2021.

17 Introduction to React Native. Verkkoaineisto. Full stack open. <https://fullstackopen.com/en/part10/introduction_to_react_native>. Luettu 11.7.2021.

18 What is data visualization? Definition, examples, and learning resources. Verkkoaineisto. Tableau. <https://www.tableau.com/learn/articles/data-visualization>. Luettu 14.7.2021.

19 Buhler, Paul; Khattak, Wajid & Erl, Thomas. 2016. Big data fundamentals: concepts, drivers & techniques. E-kirja. Pearson.

20 Big data vs. traditional data: what's the difference? Verkkoaineisto. Treehouse technology group. <https://treehousetechgroup.com/big-data-vs-traditional-data-whats-the-difference/>. Luettu 27.7.2021.

21 Chapple, Mike. 2020. What is metadata? Verkkoaineisto. ThoughtCo. <https://www.thoughtco.com/metadata-definition-and-examples-1019177>. 4.1.2020. Luettu 30.7.2021.

22 Chart.js vs d3. Verkkoaineisto. Npm trends. <https://www.npmtrends.com/chart.js-vs-d3>. Luettu 8.9.2021.

23 Cook, Peter. 2019. D3 or Chart.js for data visualization? Verkkoaineisto. Create with data. <https://www.createwithdata.com/d3js-or-chartjs/>. 22.1.2019. Luettu 3.8.2021.

24 Bostock, Mike. 2020. Gallery. Verkkoaineisto. Observable. <https://observablehq.com/@d3/gallery>. 4.2.2020. Luettu 8.9.2021.

25 Difference between Fetch and Axios.js for making http requests. 2021. Verkkoaineisto. GeeksforGeeks. <https://www.geeksforgeeks.org/difference-between-fetch-and-axios-js-for-making-http-requests/>. Päivitetty 16.7.2021. Luettu 7.9.2021.

26 Rokotusten määrä rokotuspäivittäin ja 7 pv keskiarvot 06.09. Verkkoaineisto. Twitter. <https://twitter.com/bot_fi/>. Luettu 6.9.2021.

27 Rauvola, Ilkka. 2021. Number of tests vs. positive rate (7 day averages). Verkkoaineisto. Twitter. <https://twitter.com/jukka235>. 4.9.2021. Luettu 6.9.2021.

28 Uudet tapaukset ja kuolemat. Verkkoaineisto. Google. <https://www.google.com/search?q=covid+statistics&oq=covid+statistics&aqs=chrome..69i57j0i512l6j69i60.2715j0j7&sourceid=chrome&ie=UTF-8>. Luettu 6.9.2021.

29 Hubbard, Kaia. 2021. Places without reported COVID-19 cases. Verkkoaineisto. US News. <https://www.usnews.com/news/best-countries/slideshows/countries-without-reported-covid-19-cases>. 1.9.2021. Luettu 7.9.2021.