GOOGLE ANALYTICS -RAJAPINNAN INTEGROINTI WEB-SOVELLUKSEEN

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Hämeenlinnan korkeakoulukeskus
Tietojenkäsittelyn koulutusohjelma

Kevät, 2020

Jesse Suvisuo

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn tradenomi
Hämeenlinnan korkeakoulukeskus

Tekijä Jesse Suvisuo **Vuosi** 2020

Työn nimi Google Analytics -rajapinnan integrointi web-sovellukseen

Työn ohjaaja/t Tommi Lahti

TIIVISTELMÄ

Tämän opinnäytetyön tarkoituksena oli selvittää, millaisia tekniikoita ja teknologioita käytetään modernissa web-kehityksessä, ja miten niitä voidaan käyttää rajapintojen integroimiseksi web-sovellukseen. Tämän opinnäytetyön tavoitteena oli myös tutkia data-analytiikkaa liiketoiminnan näkökulmasta.

Opinnäytetyön toimeksiantaja on Myyntimaatio Oy. Toimeksiantaja halusi luoda asiakkailleen yhteydenottosovelluksen, joka mahdollistaisi myös asiakkaiden verkkosivujen liikenteen seuraamisen. Sovelluksesta tullaan tekemään mobiilisovellus ja verkkosovellus. Tämä opinnäytetyö kattaa verkkosovelluksen Google Analytics -rajapinnan integraation luomisprosessin.

Opinnäytetyö koostuu teoreettisesta osasta ja käytännön osasta. Teoreettinen osa käsittelee data-analytiikkaa pienten ja keskisuurten yritysten näkökulmasta. Data-analytiikan tarkastelu keskittyy pääasiassa Google Analyticsiin ja sen käyttötapoihin pienissä ja keskisuurissa yrityksissä.

Opinnäytetyön käytännön osa käsittelee web-sovelluksen luomisprosessia. Luotu sovellus käyttää rajapintaa nimeltä Google Analytics API. Sovelluksen käyttöliittymä luodaan React- ja ECharts-nimisten JavaScript-kirjastojen avulla. Sovelluksen palvelinpuoli ohjelmoidaan käyttäen Node JS -suoritusympäristöä.

Avainsanat Data-analytiikka, React, Node JS

Sivut 27

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Bachelor of Business Administration
Visamäki

Author Jesse Suvisuo **Year** 2020

Subject Integrating Google Analytics API to a Web application

Supervisors Tommi Lahti

ABSTRACT

The purpose of this Bachelor's thesis was to find out what kind of techniques and technologies are being used in modern Web development and how to use them to create an integration of application programming interfaces to a Web application. This thesis also aims to examine data analytics from business' point of view.

This Bachelor's thesis was commissioned by Myyntimaatio Oy. The comissioner wanted to create an application for its clients. The application will contain features for interacting with clients and for monitoring the clients' website traffic. There will be a mobile application and a web application. This Bachelor's thesis covers the creation process of the API integration for the web application.

The thesis consists of a theoretical part and a practical part. The theoretical part discourses data analytics, mainly from a small and medium size business' point of view. The theoretical part focuses mainly on Google Analytics and ways that it's being used within small and medium size companies. In the theoretical part, the benefits of data analytics are being considered as a way of monitoring the efficiency of companies' marketing strategies.

The practical part of the thesis covers the process of creating a Web application. The application that is created utilizes an application programming interface called Google Analytics API. The front-end side of the application is created using React and a JavaScript library called ECharts.

Every technology and technique used in the practical part of this thesis was chosen after considering between different options. The technologies and techiques that were selected are typically used in modern web development. Every choice of library, framework or programming language is selected concerning the possibility for future development. Therefore the documentation and maintenance of the libraries and frameworks were some of the reasons for the choices.

Keywords Data analytics, React, Node JS

Pages 27

SISÄLLYS

1 JOHDANTO

Tämä opinnäytetyö käsittelee Google Analytics-rajapinnan integroinnin toteutusta mobiili- ja verkkosovellukseen, sekä data-analytiikan merkitystä liiketoiminnan kehittämisessä. Integraatio toteutetaan osana projektia, jossa luodaan mobiili- ja Web-sovellus toimeksiantajan asiakkaille. Projektin toimeksiantajana toimii digitaliseen markkinointiin erikoistunut yritys Myyntimaatio Oy. Sovelluksen on tarkoitus tarjota asiakkaille väylä, jonka kautta voi seurata verkkosivujen käyttäjädataa reaaliajassa. Aluksi tarkastellaan data-analytiikkaa liiketoiminnassa yleisesti, jonka jälkeen käydään läpi Google Analytics-rajapinnan integroinnin ohjelmointiprosessi vaiheineen.

Data-analytiikka on tärkeä mittausjärjestelmä liiketoiminnan kannattavuuden ja kehittämisen tueksi. Data-analytiikan avulla yritys voi tarkastella verkkosivustojensa kävijöiden määriä, ostokäyttäytymistä, sekä lukuisia muita kulutuskäyttäytymiseen piirteitä. Sen avulla yritykset voivat seurata markkinointikampanjoidensa tehokkuutta ja verkkokaupan myyntiä, kehittääkseen liiketoimintaansa.

Tämän opinnäytetyön tarkoitus on havainnollistaa Google Analytics -data-analytiikkarajapinnan integrointi Web-sovellukseen. Sovelluksen tarkoitus on tehdä markkinointikampanjoiden, brändin rakentamisen ja Web-sivujen kehittämisen vaikutusten seuraamisesta helpompaa – luoda väylä sivustojen kävijämäärien tarkkailuun, kirjautumatta Google Analytics -verkkosivuille. Toimeksiantaja halusi rakentaa sovellukseen kaavion, joka näyttää edellisen kahden viikon kävijät viivadiagrammina. Toimeksiantaja halusi myös sovellukseen listan, johon tulostetaan Google Analytics-rajapinnasta saatuja tietoja.

Opinnäytetyön käytännön osuus käsittelee Google Analytics-rajapinnan integroinnin toteutuksen eri vaiheita. Ensin käydään läpi sovelluksen palvelinpuolen ohjelmointi ja siihen käytettävät tekniikat. Käytännön osuuden toisella puoliskolla käydään läpi selainpuolen ohjelmointi, jossa Googlelta haetut tiedot asetetaan taulukkoon, käyttöliittymään. Opinnäytetyö vastaa seuraaviin kysymyksiin:

- Mitä teknologioita käytetään modernissa Web- sovelluksen ohjelmoinnissa ja miksi?
- Miten Googlen data-analytiikkarajapinta integroidaan käyttöliittymään?
- Miksi käyttäjädatan analyysi on liiketoiminnan kehittämisen näkökulmasta tärkeää?

2 DATA-ANALYTIIKKA LIIKETOIMINNAN KEHITTÄMISESSÄ

Tässä luvussa käsitellään data-analytiikan tarkoitusta liiketoiminnassa. Koska toimeksiantaja Myyntimaatio Oy on yritys, joka tarjoaa digitaalisia palveluita pääasiassa pienille ja keskisuurille yrityksille, painottuu tässä opinnäytetyössä tarkasteltava liiketoiminnan kehittäminen pienten ja keskisuurten yritysten liiketoiminnassa käytettävään data-analytiikkaan. Ensin tarkastellaan erilaisia hyötyjä ja työkaluja, joita data-analytiikka tarjoaa, sekä mittareita, joiden perusteella liiketoimintaa kehitetään.

Tässä luvussa tarkastellaan myös Google Analytics -palvelua ja sen tarjoamia työkaluja. Selvitetään mitkä ovat tärkeimpiä mittareita markkinointikampanjan onnistumisen seurannassa, sekä millaisia asioita mitataan, kun tutkitaan verkkosivujen käyttöä ja käyttäjien toimintaa. Tässä luvussa selvitetään myös, minkälainen verkkosivun käyttäjien suorittama toiminta verkkosivuilla on liiketoiminnan kehittämisen näkökulmasta olennaista.

2.1 Data-analytiikan tehtävä

Data on järjestäytymätöntä informaatiota, jolla ei itsessään ole usein informatiivista merkitystä. Informaatio on vastaavasti prosessoitua dataa, jota voidaan käyttää päätöksenteossa. Jotta datasta saadaan informaatiota, on sitä prosessoitava tai järjesteltävä. Kun data on muutettu informaatioksi, sitä voidaan jatkojalostaa tiedoksi. Tietoa voidaan pitää informaationa, jota on jatkojalostettu asiayhteydessään. Tätä prosessia, jossa dataa muutetaan tiedoksi, voidaan kutsua data-analytiikaksi. (Surbhi, 2012; Thierauf, 1999, s.6)

Jotta informaatiosta saadaan mahdollisimman hyödyllistä, on dataa käsiteltävä niin, että kerättävä dataotos on mahdollisimman hyödyllinen. On otettava huomioon seikkoja, jotka saattavat vääristää kerättävää aineistoa. Esimerkkinä tästä voidaan pitää yksinkertaisten kävijälaskureiden ja modernien verkkoanalytiikkapalveluiden välistä eroa. Yksinkertainen kävijälaskuri ei anna verkkosivuista niin merkittävää informaatiota, kuin verkkoanalytiikkapalvelu, joka käsittelee dataotosta ja huomioi sivustolla vierailleiden ihmisten tuottamaa dataa kokonaisvaltaisemmin. (AW Academy Group, 2019)

Verkkoanalytiikkapalveluiden, kuten Google Analyticsin, analysointiprosesseissa pyritään selvittämään mm. onko käyttäjä jo vieraillut sivustolla aiemmin, tarkentaen kuvaa todellisesta käyttäjämäärästä. Liiketoiminnan näkökulmasta tärkeää informaatiota sivuston kävijämäärän lisäksi on mm. sivuston käyttäjien sijainti, käytettävä päätelaite, verkkosivuilla käytetty aika ja sivustolla vain hetken vierailleiden käyttäjien osuus kaikista käyttäjistä. (Google LLC, 2020a)

Googlen ja Verto Analyticsin tekemän tutkimuksen mukaan kuluttajien käytös kuluttamisen suhteen on luultua monimuotoisempaa. Huolimatta tuotteesta tai palvelusta, kuluttajat ovat usein keskenään hyvin erilaisia. Digitaalisen teknologian kehityksen ansiosta kuluttajat ovat omien ostopäätöstensä suhteen tietoisia ja osaavia digitaalisten palveluiden käyttäjiä. Kuluttajat osaavat hyödyntää digitaalisia palveluita, joiden avulla tutkia tuhansien kategorioiden ja tuotteiden kirjoa. (Google LLC, 2020a)

Monet kuluttajat tekevät ostopäätöksensä harkiten ja käyttävät aikaa parhaimman mahdollisen tuotteen tai palvelun löytämiseksi. Samanaikaisesti moni tekee ostopäätöksen todella nopeasti. Tuotteiden ja palveluiden vertailuun ja etsimiseen käytetään pääsääntöisesti internetiä, ja yrityksille on elintärkeää päästä näkyville siellä, missä tuotteiden ja palveluiden vertailua tehdään. Havainto kuluttajien käyttäytymisen monimuotoisuudesta on olennainen havainto digitaalisten palveluiden kehittämisen näkökulmasta. (Google LLC, 2020a)

Digitaalisen teknologian kehitys ja mobiililaitteiden kuuluvuuden laajeneminen ovat muokanneet kuluttajien tietoisuutta tuotteiden etsintään ja vertailuun käytettävien digitaalisten palveluiden tarjoamista mahdollisuuksista. Koska mobiililaitteet ovat lähes aina käsien ulottuvilla, kuluttajat tietävät myös kaiken tarvittavan informaation olevan helposti saatavilla. Vaihtoehtojen vertailemisen helppouden ja nopeuden ansiosta, kuluttajat vertailevat laajasti ennen ostopäätöstä. Tämä havainto ei rajoitu pelkästään kalliisiin tuotteisiin: esimerkiksi Google-haku ”parhaat nappikuulokkeet”, on lisääntynyt 130% vuosina 2017-2018. Samaan aikaan, kun joku kuluttaja käyttää autoa ostaessaan eri vaihtoehtojen miettimiseen yhden päivän, saattaa toinen kuluttaja käyttää eri kahviloiden välillä puntaroimiseen kymmeniä päiviä. (Google LLC, 2020a)

Koska asiakkaat odottavat digitaalisilta palveluilta paljon enemmän kuin ennen, on markkinoinnissakin keksittävä tapoja, joilla vastata asiakkaan odotuksiin, sekä ennakoida asiakkaan käyttäytymistä. Samoin on löydettävä niitä mittareita, joita kannattaa data-analytiikassa huomioida ja tarkastella. Digitaalisen markkinoinnin kehityksen aallonharjalla olevat yritykset saavat syvemmän käsityksen nykyaikaisesta, epälineaarisesta ja nopeatempoisesta klikkausten ja hakujen prosessista, kuin vähemmän digitaaliseen markkinointiin keskittyvät yritykset. Tätä tietoa käytetään tarjoamaan kuluttajille henkilökohtaisempia, ja ostopäätöksen kannalta olennaisempia kokemuksia. (Google LLC, 2020a)

2.2 **Google Analytics**

Google Analytics -palvelussa seurattavalle sivulle asetetaan tähän tarkoitukseen luodun JavaScript-kirjaston avulla niin sanottu Google analytics tag. Tämä tag on JavaScript koodia, joka kirjoitetaan seurattavaksi asetettavan sivun HTML-tiedoston yläreunaan, ennen <script>- ja <css>-

merkintöjä. Kun Google Analytics tag on asetettu, alkaa Google mittaamaan sivun käyttäjädataa. (Google LLC, n.d.b)

JavaScriptiin perustuvan seurannan ongelmana voidaan pitää sitä, että JavaScript on mahdollista estää päätelaitteesta, ja näin ollen estää käyttäjätietojen saaminen. Nykyisin JavaScript on yleisimmissä selaimissa ja mobiililaitteissa oletuksena käytössä. Seuranta on mahdollista estää myös selaimeen asennettavilla lisäosilla. (Crandall, 2018)

Google Analyticsin yhtenä etuna on mahdollisuus luoda omia raportteja. Google Analytics -verkkosivulla niitä voi luoda ”raahaa ja pudota” -toiminnolla. Voidaan valita mitat ja tiedot, joiden perusteella raportti tehdään ja esitetään Google Analytics -verkkosivun käyttöliittymässä. (Thakur, 2017)

Omien raporttien luominen on mahdollista myös Google Analytics Core Reporting -rajapinnan avulla. Tämä rajapinta toimii ohjelmointimetodina kustomoitujen Analytics-kaavioiden rakentamisessa. (Google LLC, 2017). Tässä opinnäytetyössä luotiin palvelin, joka käyttää Google Analytics Core Reporting -rajapintaa luodakseen yhteyden Google Analyticsiin.

3 REACT

React on JavaScript ohjelmointikielen kirjasto, jonka avulla luodaan verkkosivustojen käyttöliittymiä. Alun perin Reactin kehitti Facebook ja nykyisin sitä ylläpitää Facebookin lisäksi joukko yksityishenkilöitä ja yhtiöitä. Sitä käytetään ensisijaisesti dokumenttioliomallin (DOM) manipuloimiseen dynaamisilla verkkosivuilla ja mobiiliapplikaatioissa. (React JS, 2020a)

Reactin rakennuspalikoita ovat komponentit. Komponentit ovat JavaScript-luokkia, tai JavaScript-funktioita. Ne ovat uudelleenkäytettäviä ja itsenäisiä koodin osasia, jotka palvelevat Web-kehityksessä samaa asiaa, kuin JavaScript-funktiot. Komponentit ottavat vastaan propseja, komponentin ominaisuuksia ja palauttavat React-elementtejä. Nämä elementit kuvaavat, mitä käyttöliittymässä näytetään. (React JS, 2020a)

3.1 Props-objekti

Reactissa komponentilla voi olla niin sanottu props-objekti, joka on lyhenne sanasta properties. Nämä objektit pitävät sisällään informaatiota, joka liittyy komponenttien renderöimiseen, dokumenttiobjektimallin piirtämiseen käyttöliittymään. Props-objektia voidaan pitää eräänlaisena komponentin konfiguraationa – asetuksina, jotka tulevat komponentille annettuina. (React JS, 2020b)

3.2 State-objekti

Komponenteilla voi myös olla niin sanottu state-objekti. State on JavaScript-objekti, jota hallitaan komponentissa. State- ja props-objektien ero on siinä, että props-objektin avulla informaatiota siirretään komponenttiin, kun taas state-objektin avulla informaatio siirtyy komponentin sisällä. State-objektilla on arvoja, jotka määrittävät miten komponentti näytetään käyttöliittymässä. Esimerkiksi setState-metodi ajoittaa päivityksen komponentin state-objektiin, jolloin komponentti vastaa uudelleenrenderöimällä. (React, 2020c)

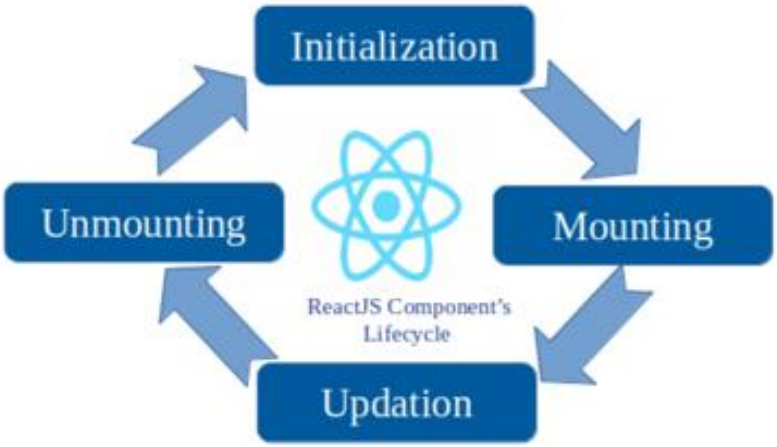
Komponentin state saa oletusarvon, kun komponentti asetetaan. Tästä asettamisesta käytetään termiä mounting. Tämän jälkeen komponentti kohtaa muutoksia elinkaarensa aikana. Nämä muutokset juontuvat usein käyttäjän tekemistä asioista, kuten hiiren klikkaus. State edustaa eräänlaista tilannekuvausta komponentin sen hetkisestä tilasta. Komponentti hallinnoi tilaansa (state) sisäisesti. (React JS, 2020b)

3.3 Komponenttien elinkaari

Komponenteilla on elinkaari, jota kutsutaan nimellä lifecycle. Se on prosessi, joka kuvaa komponentin vaiheita dokumenttiobjektimallissa. React tarjoaa metodeita, joilla kontrolloidaan komponentin elinkaarta (kuva 1).

- Initialization (alustaminen)
- Mounting (asettaminen)
- Updating (päivittäminen)
- Unmounting (irrottaminen)

(React JS, 2020d)



Kuva 1. Kuvassa komponentin elinkaari dokumenttioliomallissa. Kuva: Free Code Camp inc.

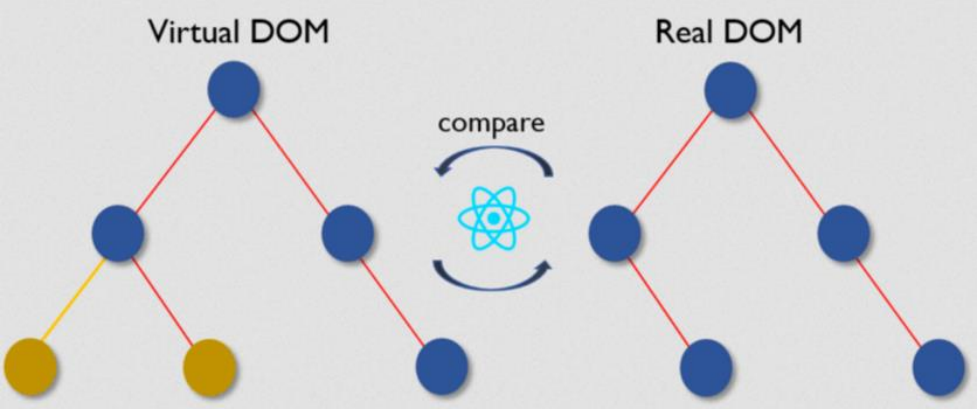
Komponentin pääasiallinen tehtävä on kääntää informaatiota HTML-muotoon. Props ja State-objektit muodostavat yhdessä raa'an datan, joka on pohjana HTML-muotoon kääntämiselle. Voidaan sanoa, että props ja state ovat eräänlainen datan lähde komponentin render-metodille. Jokaisella komponentilla on render-metodi, joka määrittää käyttäjälle näytettävän informaation.
(React JS, 2020b)

3.4 DOM

Dokumenttiobjektimallilla, lyhyesti DOM, tarkoitetaan objektien puuta, joka muodostaa ladatun verkkosivun rakenteen. Usein dokumenttiobjektimallilla tarkoitetaan HTML-dokumenttiobjektimallia. Dokumenttiobjektimalli on eräänlainen kirjaus siitä, mitä HTML-dokumentti pitää sisällään. HTML-dokumenttiobjektimalli määrittää HTML-elementit, HTML-elementtien asetukset, HTML-elementtien manipuloimiseen tarvittavat metodit ja tapahtumankäsittelyn HTML-elementeille. HTML-dokumenttiobjektimalli määrittää standardin sille, miten saada, muuttaa, lisätä tai poistaa HTML-elementtejä. (w3schools, n.d.)

3.5 Virtual DOM

Virtuaalinen dokumenttiobjektimalli (Virtual DOM) mahdollistaa tiettyjen komponenttien uudelleenlataamisen, sen sijaan, että ladattaisi jokainen komponentti uudelleen. Virtuaalinen dokumenttiobjektimalli perustuu siihen, että käyttöliittymän virtuaalinen esitys pidetään muistissa ja se synkronoidaan käyttäjälle näytettävän dokumenttiobjektimallin kanssa. Virtuaalinen dokumenttiobjektimalli voidaan päivittää ja sitä voidaan manipuloida. Kun virtuaalinen dokumenttiobjektimalli on päivitetty, voidaan päättää, mitä muutoksia halutaan tuoda käyttäjälle näytettävään varsinaiseen dokumenttiobjektimalliin. (React, 2020e)



Kuva 2. Havainnollistava kuva virtuaalisesta dokumenttioliomallista ja varsinaisesta dokumenttiolimallista. Kuva: Brain4ce Education Solutions Pvt.

React-ohjelmoinnissa luodaan JavaScript-kielellä React-komponentteja. React tekee itse dokumenttioliomallin päivittämisen. React käyttää yksisuuntaista tietojen sitomista, yksinkertaistaakseen prosesseja. Aina kun käyttäjä manipuloi dokumenttiobjektimallia käyttöliittymästä, komponentin tila ei suoraan muutu. Sen sijaan se päivittää tietomallin, jonka seurauksena käyttöliittymä päivittyy. Kuten varsinainen dokumenttioliomalli, myös virtuaalinen dokumenttioliomalli on tietomallirakenteeltaan hierarkinen puu. Reactin render-metodi luo komponenteista hierarkisen puun ja päivittää tätä puuta sitä mukaa, kun tietomallia muutetaan. Aina kun React-sovelluksen perustana oleva data muuttuu, usein käyttäjän

suorittamien asioiden seurauksena, luodaan uusi esitys käyttöliittymän dokumenttiobjektimallista. (Minnick, 2016)

3.6 JSX-syntaksi

React-ohjelmoinnissa suositellaan käytettävän JSX:ksi kutsuttua JavaScriptin syntaksilaajennusta. Sitä käytetään kuvaamaan, miltä käyttöliittymän tulisi näyttää. Se muistuttaa hieman HTML-kieltä, mutta se on JavaScript-lisäosa. JSX tuottaa React-elementtejä siirrettäväksi dokumenttiobjektimalliin. (React JS, 2020f)

Alla olevassa kuvassa 3 hyödynnetään JSX-syntaksia. Render-metodissa määritellään pylväsdiagrammin näytöllä esittämistä koskevia yksityiskohtia, kuten otsikoita ja fonttikokoja.

```
18    render(){
19      return (
20        <div className="chart">
21        <Bar
22          data={this.state.chartData}
23          options={{
24            title:{
25              display:this.props.displayTitle,
26              text:'',
27              fontSize: 25
28            },
29            legend:{
30              display:this.props.displayLegend,
31              position:this.props.legendPosition
32            }
33          }}
34          />
35        </div>
36      )
37    }
38  }
```

Kuva 3. Render-metodin sisällä määritellään JSX-syntaksia käyttäen, mitä käyttöliittymässä näytetään.

3.7 Reactin hyödyt

React-koodi suoritetaan selaimessa, toisin kuin esimerkiksi PHP-koodi, joka suoritetaan palvelimella. Tämän ansiosta sivuston käyttöliittymästä

saadaan helpommin interaktiivinen – kaikkia sivuston osia ei tarvitse ladata uudelleen, vaan ne voidaan suorittaa käyttäjän selaimessa. React-kirjaston hyödyntäminen tekee dynaamisten verkkosivujen käyttöliittymien rakentamisen helpommaksi, kuin pelkän JavaScriptin käyttäminen. (React JS, 2020a)

Dokumenttioliomallin manipulointi pelkällä JavaScriptillä on monimutkaisempi prosessi, kuin React-kirjastoa hyödyntämällä. Virtuaalinen dokumenttioliomalli yhdistettynä komponenttien elinkaarten hallintaan, yksinkertaistaa dokumenttioliomallin manipulointia ja vähentää kirjoitettavan koodin määrää. (Buna, 2017)

4 NODE JS

Node JS on JavaScriptin suoritusympäristö, jota käytetään koodin suorittamiseen palvelimella. JavaScript-koodia on aiemmin pääsääntöisesti käytetty niin, että JavaScript-koodi suoritetaan verkkoselaimen JavaScript-moottorilla, loppukäyttäjän päätelaitteessa. Node JS perustuu Googlen chrome V8 -JavaScriptmoottoriin. (fullstackopen, n.d.)

Node JS:n ero muihin yleisesti palvelinpuolen ohjelmoinnissa käytettyihin ohjelmointikieliin on yksisäikeinen teknologia, jonka ansiosta Node JS käyttää vain yhtä säiettä pyyntöjen lähettämiseen. Vastaavasti esimerkiksi Java -ohjelmointikielessä jokainen pyyntö käyttää yhden säikeen, käyttäen näin enemmän järjestelmän muistia. Monisäikeistä teknologiaa käyttävät ohjelmointikielet voivat tehdä jokaista säiettä kohtaan vain yhden prosessin kerrallaan, mikä joissain tapauksissa saattaa johtaa yksisäikeistä teknologiaa huonompaan suorituskykyyn. Node JS mahdollistaa niin sanotun non-blocking io calls -yhteyden, minkä ansiosta voidaan ylläpitää kymmeniä tuhansia yhteyksiä, ylikuormittamatta järjestelmän muistia. Vastaavasti Node JS saattaa kuormittaa tietokoneen prosessoria monisäikeistä teknologiaa enemmän. (Traversy, 2016)

Node JS:ää käytetään yleisesti REST-rajapintojen (Representional State Transfer) ja palvelinpuolen applikaatioiden ohjelmointiin. Node JS soveltuu hyvin reaaliajassa toimivien, ja tietokoneen prosessoria vähän kuormittavien tehtävien suorittamiseen. (Traversy, 2016)

4.1 Express-moduulin käyttö sovelluksessa

Express on Node.js-verkkosovelluskehys, joka tarjoaa joukon keskeisiä verkkosovelluksissa käytettyjä ominaisuuksia. Sillä voi esimerkiksi luoda komponentteja http-pyyntöjen käsittelyyn. Sitä käytetään Web-sovellusten ja palvelinten rakentamiseen. (expressjs, n.d.)

Express-kehystä käytetään tässä projektissa sovelluksen palvelinpuolen ohjelmoinnissa. Se mahdollistaa tiedon siirtämisen palvelimelta selainpuolelle. Sovellukseen luodaan yksinkertainen palvelin, jonka tehtävä on mahdollistaa tiedon siirto selainpuolelle http-protokollan avulla. Package.json -tiedostoon, React-tiedostojen yläkansioon, asetetaan välityspalvelin Tämä määrittää reitityksen, jota käytetään pyyntöjen ja vastausten lähettämiseen selaimen ja palvelimen välillä. Kun välityspalvelin määritellään, käytetään siihen määriteltyä URL-osoitetta pyyntöjen ja vastausten välitykseen palvelimelta käyttäjälle. Fetch-rajapinnan koodiin selainpuolelle asetetaan yksinkertaisempi polku, josta esimerkki kuvassa 4.

```
componentDidMount() {
  fetch('/api/views')
  .then(res => res.json())
  .then(views => this.setState({views}, () => console.log('Views fetched...', views)));
}
```

Kuva 4. fetch-metodin sisään asetettu polku on todellisuudessa https://localhost:5000/views. Tämä React-tiedosto käyttää package.json-tiedostoon määriteltyä välityspalvelinta, joten polku löytyy automaattisesti sijainnista: /api/views.

4.2 NPM, Node-moduulit ja package.json

NPM (Node Package Manager) on Node-moduulien hallintaan tarkoitettu työkalu. Se mahdollistaa moduulien julkaisun yleiseen käyttöön npmjs.com-sivuston kautta. NPM:ää käytetään komentorivillä moduulien asentamiseen. NPM asettaa riippuvuudet package.json -tiedostoon. Esimerkki package.json -tiedostosta on esitetty kuvassa 3. (Npm, n.d.a)

```
{} package.json > ...
{
  "name": "reactnode",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "server": "nodemon server.js",
    "client": "npm start --prefix client",
    "dev": "concurrently \"npm run server\" \"npm run client\""
  },
  "author": "Jesse Suvisuo",
  "license": "MIT",
  "dependencies": {
    "chart.js": "^2.9.3",
    "concurrently": "^5.1.0",
    "dotenv": "^8.2.0",
    "express": "^4.17.1",
    "react-chartjs-2": "^2.9.0"
  },
  "devDependencies": {
    "nodemon": "^2.0.2"
  }
}
```

Kuva 5. package.json-tiedoston sisältö. Riippuvuudet esitetään kohdassa ”devDependencies”.

Package.json -tiedostoon voidaan luoda scriptejä, komentorivillä suoritettavia komentoja. Yläpuolella, kuvassa 5, on package.json -tiedosto, johon on asetettu komento *dev*. NPM:n avulla on asennettu työkalu nimeltä concurrently. Tämä työkalu mahdollistaa useamman scriptin suorittamisen samanaikaisesti. Kun sovellukseen luodaan Back-end -palvelin ja React -palvelin, täytyy nämä molemmat käynnistää erikseen, ellei näiden käynnistämiselle luoda yhteistä scriptiä. Tällainen tilanne on tyypillinen, kun käytetään käyttöliittymän rakentamiseen Reactia ja Back-end -

palvelimen rakentamiseen Node JS:ää. Näiden molempien palvelimet täytyy käynnistää sovelluksen käynnistämiseksi.

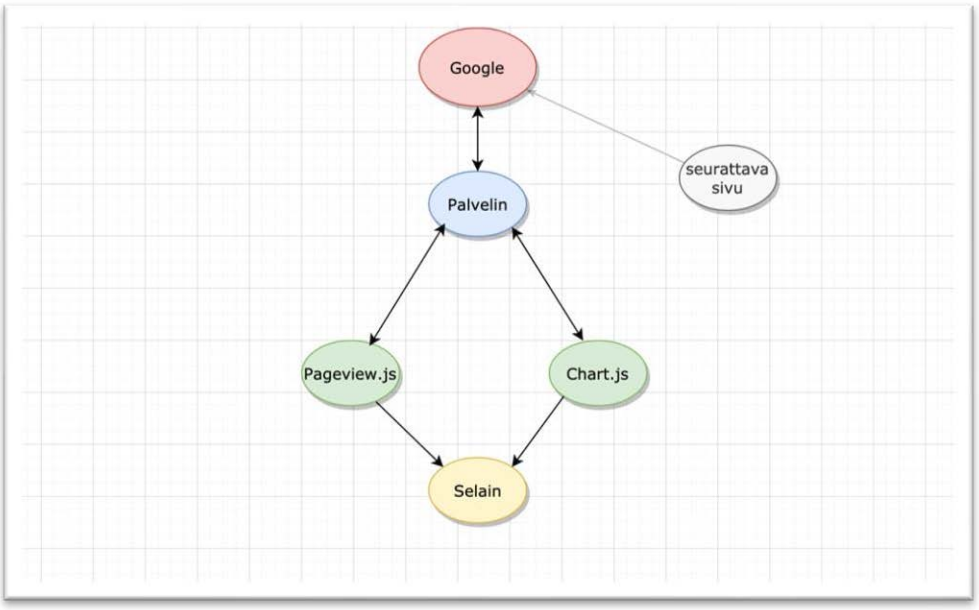
Kuvassa 5 on rivillä 10 asetettu React – ja Node JS -palvelinten käynnistämiselle yhteinen script nimeltä *dev*. Tämän yhteisen scriptin käyttämisen mahdollistaa concurrently-työkalu. Script suoritetaan navigoimalla sovelluksen kansion juureen ja kirjoittamalla komentoriville komento *npm run dev*. Tällöin sekä React -, että Node JS -palvelimet käynnistyvät, ja verkkosovellus toimii. (Npm, n.d.b)

Package.json-tiedosto pitää sisällään myös muuta metatietoa, koskien kyseistä projektia. Se voi sisältää muun muassa projektin kuvauksen, lisenssitietoja ja kokoonpanotietoja. (Node JS, 2011)

5 GOOGLE ANALYTICS INTEGRAATION SUUNNITTELU JA OHJELMOINTI

5.1 Sovelluksen rakenteen suunnittelu

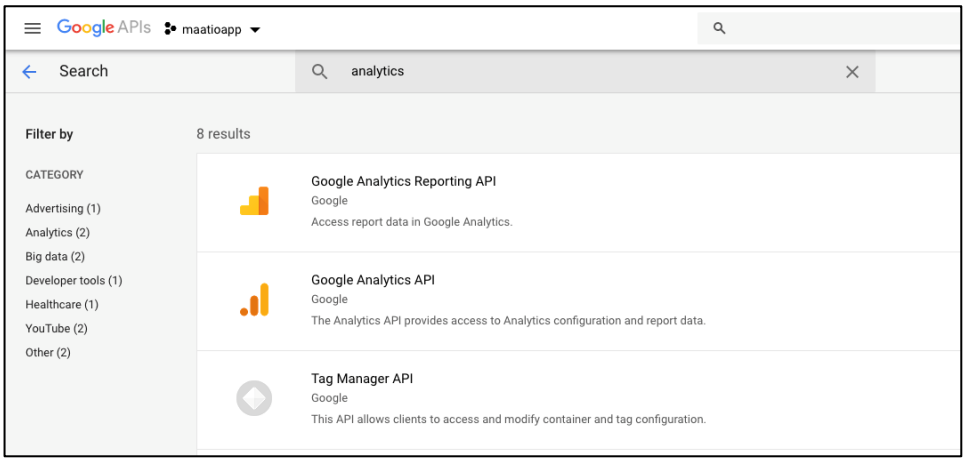
Integraatio aloitettiin suunnittelemalla applikaatiolle rakenne, joka kuvastaa tiedon kulkua, ja applikaation eri osien suhdetta toisiinsa. Luotiin mielekartta, jotta hahmotetaan applikaatioon tarvittavat tekniikat ja komponenttien määrä. Rakenne määräytyi käytettävien tekniikoiden ja toimeksiannossa esitettyjen käyttöliittymän elementtien perusteella. Kuvassa 6 esiintyvä palvelin kuvaa Node JS:n avulla ohjelmoitavaa palvelinta. *Pageview.js* ja *Chart.js* kuvaavat React-komponentteja, jotka luovat käyttöliittymässä näytettävät elementit.



Kuva 6. Miellekartta luotavansovelluksen rakenteesta.

5.2 Seurannan ja käyttöoikeuksien määrittäminen

Google Analytics -rajapinnan hyödyntäminen vaatii käyttäjän autentikaation. Käyttöoikeuksien määrittämistä varten käytetään Google Developers Console -sivustoa, sekä Google Analytics -sivustoa. Google Developers Console -sivustolta ladataan JWT (JSON Web Token) -autentikaatiota varten JSON -muodossa oleva tiedosto, joka pitää sisällään autentikaatiota varten tarvittavat tiedot. Näitä tietoja hyödynnetään, kun ohjelmoidaan palvelin, joka pyytää Google Analytics -rajapinnalta dataa.



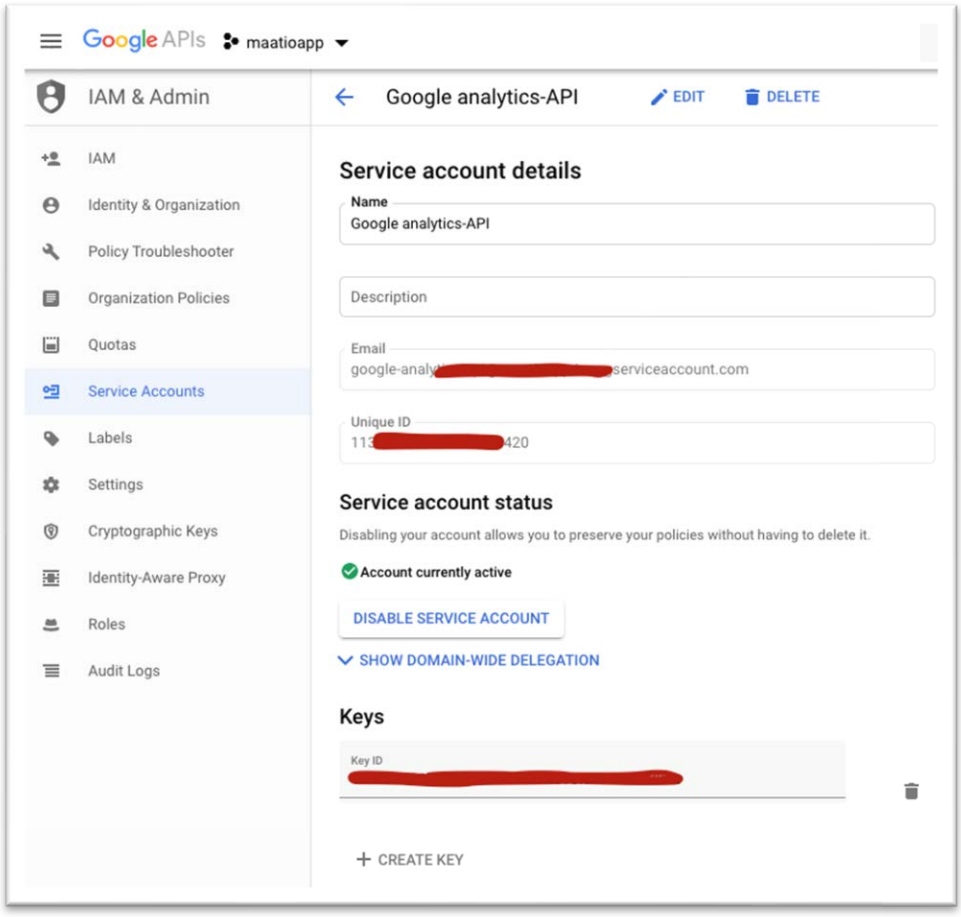
Kuva 7. Google Developers Console -sivustolla luodaan projekti, johon asetetaan Google Analytics Reporting API.

Kun projekti on luotu ja siihen on liitetty Google Analytics Reporting API, luodaan käyttäjätiedot, joita myöhemmin käytetään käyttöoikeuden varmentamiseen. Käyttäjätietojen avulla suoritettu autentikaatio tapahtuu, kun sovelluksen palvelimelta lähetetään pyyntö Googlen rajapintaan.

Google-rajapinnat mahdollistavat kolme erilaista tapaa suorittaa autentikaatio. Nämä ovat OAuth 2, API key ja Service to Service. OAuth 2 on näistä tietoturvan tasoltaan vahvin. Se on tarkoitettu ensisijaisesti sellaisiin applikaatioihin, joissa käyttäjä voi omalla toiminnallaan lähettää pyyntöjä Googlen rajapintaan. API key puolestaan on tietoturvaltaan heikompi, sekä sen käyttö on rajattu vain tietynlaisiin rajapintoihin, kuten Maps-rajapintaan. Service to Service -autentikaatiomalli käyttää niin sanottua service-käyttäjätiliä, tiliä, jolle annetaan lupa tarkastella Googlen rajapinnasta saatua tietoa.

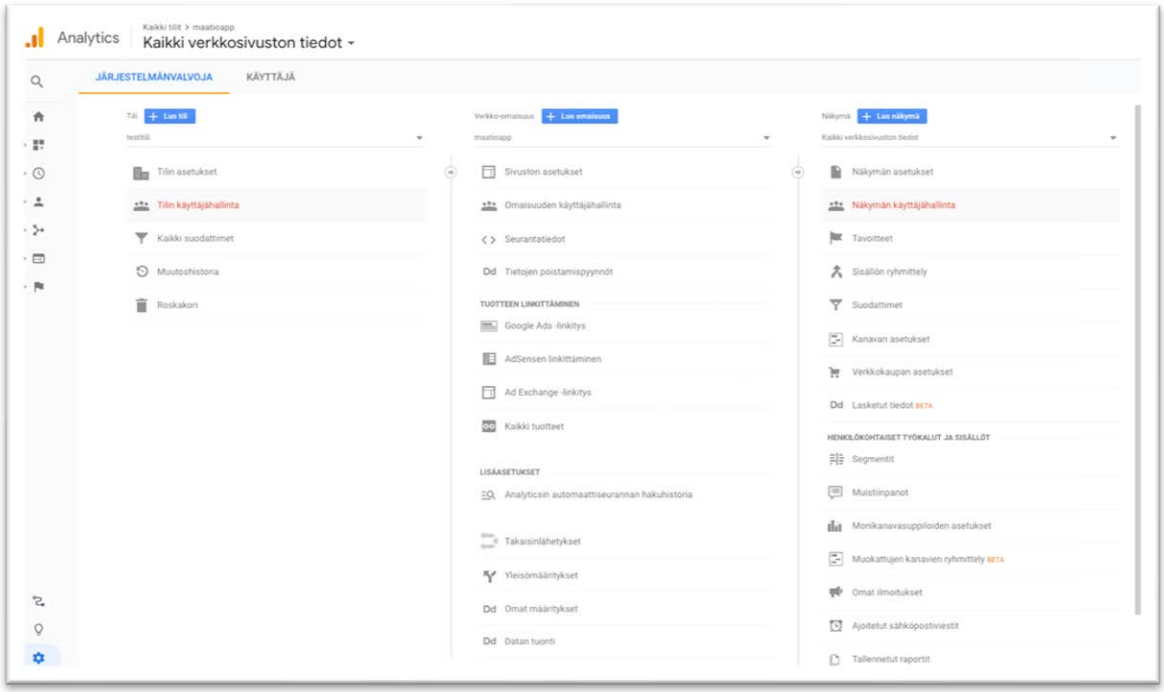
Koska tässä opinnäytetyössä tehtävässä projektissa luodaan palvelin, joka lähettää Googlen rajapintaan pyyntöjä, käytetään Service to Service -autentikaatiomallia. Mikäli applikaation myöhemmässä kehitysvaiheessa halutaan mahdollistaa käyttäjän aktiivisesti tekemät pyynnöt Googlen rajapintaan, voidaan autentikaatiomallia muuttaa tietoturvallisempaan vaihtoehtoon. Tällöin voidaan siirtyä käyttämään OAuth 2 -autentikaatiomallia.

Google Developers Consolen kautta luodaan service -käyttäjätili. Tämä suoritetaan navigoimalla Credientials-osioon, klikkaamalla sieltä Create credientials-painiketta ja edelleen kohtaa Service account key. Tämän jälkeen tiedosto, joka pitää sisällään luodun käyttäjän sähköpostiosoitteen, sekä henkilökohtaisen avaimen, on ladattavissa tietokoneelle. Tiedosto voidaan ladata joko JSON-, tai P12-muodossa. Tässä projektissa tiedosto ladataan JSON-muodossa. Kun JSON-muotoinen avaintiedosto on ladattu, asetetaan se tiedostopolkuun applikaatiolle saavutettavaan sijaintiin.



Kuva 8. Google Developers Dashboardin kautta luotu service-käyttäjätili kuvattuna Admin-sivulla.

Koska käytämme tämän opinnäytetyön projektissa Service to Service -autentikaatiomallia, pitää service-käyttäjätilin sähköpostiosoite asettaa myös Google Analytics -sivuston järjestelmänvalvojan paneelista oikeutetuksi seurattavan sivuston näkymän tarkasteluun. Tästä hallintapaneelista löytyy myös myöhemmin Googlen rajapintaan lähetettävässä kutsussa tarvittava näkymän tunniste.



Kuva 9. Google Analytics -sivuston järjestelmävalvoja hallintapaneeli. Kohdasta näkymän käyttäjähallinta asetetaan käyttöoikeudet Google Analytics API -sivustolla luodulle service-käyttäjätilin sähköpostiosoitteelle.

5.3 Tietojen hakeminen Google Analytics APIa hyödyntäen

Rajapintojen yhdistämiseen tarkoitettu Google Analytics Core Reporting -sovellusrajapinta mahdollistaa tietojen saamisen sovellukseen. Google Analytics Core Reporting -sovellusrajapinnan avulla pääsee käsiksi suurimpaan osaan Google Analyticsin raporttitiedoista. Core Reporting -sovellusrajapinnan avulla voi luoda mukautettuja hallintapaneeleja Google Analytics -tietojen näyttämiseksi. Tällä tavoin voi säästä aikaa automatisoimalla monimutkaisia raportointitehtäviä. Core Reporting -rajapinnalla voi integroida Google Analytics -tietoja muihin sovelluksiin. (Google LLC, 2017)

Google Analytics Core Reporting -rajapinta tarjoaa JavaScript-metodin Analytics-tietojen pyytämistä varten. Metodiin asetetaan parametreinä ne tiedot, jotka halutaan vastaanottaa Googlelta. Alla olevassa esimerkissä (kuva 10.) annetaan parametreinä aloitus- ja lopetuspäivä, joiden väliltä analytiikkatietoa halutaan vastaanottaa. (Google LLC, 2019)

Vastaanotettavaa dataa voidaan suodattaa niin, että data vastaanotetaan toivotulla tavalla järjestettynä, esimerkiksi käytetyimmän selaimen mukaan. Kyselyn tuloksia lajittelemalla, voidaan havainnollistaa esimerkiksi maat, jossa sivuston käyttäjiä on eniten, sekä selaimet, joita eniten käytetään sivustojen selaamiseen. Tästä esimerkkinä alla oleva kuva 11. (Google LLC, 2019)

```
&start-date=9daysAgo
&end-date=today
```

Kuva 10. Aloitus- ja lopetuspäivä vastaanotettavan datan suodattamiseksi annetaan parametreinä Google Analytics Core Reporting -rajapintaan. Kuva: Google LLC.

```
sort=ga:browser,ga:country
```

Kuva 11. Järjestetään vastaanotettava data selaimen ja maan mukaan, pienimmästä suurimpaan. Kuva: Google LLC.

Palvelimen ohjelmointi aloitetaan tarvittavien Node-moduulien lataamisella. Tässä projektissa käytettäviksi moduuleiksi valikoitui express ja googleapis. Express-moduuli mahdollistaa väylän palvelimelta selaimelle. Googleapis-moduuli pitää sisällään Googlen rajapintoihin yhdistämiseen ja tiedon siirtoon käytettävät metodit. Googleapis-moduuli tarjoaa myös autentikaatiomahdollisuuden, esimerkiksi JWT (JSON Web Token) – menetelmällä.

Tässä opinnäytetyössä luodaan yksinkertainen palvelin, joka ei toistaiseksi mahdollista käyttäjän aktiivisesti tekemiä pyyntöjä Googlen rajapintaan, joten Service to Service -autentikaatio riittää toistaiseksi. Autentikaatiossa käytettävät tiedot olisi mahdollista säilöä tietoturvallisemmin, mutta toistaiseksi ne säilötään auth.json- tiedostoon, luodun palvelimen kanssa samaan kansioon. Näin private key ja service-käyttäjätilin -sähköpostiosoite saadaan helposti käyttöön palvelimella. Kun tiedot haetaan erillisestä tiedostosta, ei niitä tarvitse kirjoittaa palvelimen koodiin.

Koska sovelluksen lopullinen kirjautumismenetelmä ei ole vielä selvillä, käytetään erillistä tiedostoa autentikaatiossa. Toimeksiantajan kanssa selvitetään, luodaanko tulevaisuudessa käyttöoikeuksia varten erillinen tietokanta. Tässä vaiheessa sovelluksen kehitystä pitäydytään yksinkertaisemmassa menetelmässä.

```
1  const express = require('express');
2
3  const app = express();
4
5  const { google } = require('googleapis')
6
7  const key = require('./auth.json')
8  const scopes = 'https://www.googleapis.com/auth/analytics.readonly'
9  const jwt = new google.auth.JWT(key.client_email, null, key.private_key, scopes)
```

Kuva 12. Palvelimen ohjelmoinnissa käytettävät Node-moduulit, express ja googleapis, otetaan käyttöön palvelimen koodissa. JWT-autentikaatiota varten luodaan key-muuttuja. Scopes-muuttuja määrittää käyttöoikeuksien tason. JWT-autentikaatio käyttää auth.json-tiedostossa olevia tietoja.

Express-moduulilla luodaan reitti palvelimelta selaimelle. Tätä reittiä varten luodaan app-muuttuja. App-funktio on JavaScript-funktio, joka käyttää http-metodeja, kuten get ja post. Nämä rajapintojen välistä reititystä koskevat metodit määrittävät takaisinkutsufunktiot, joita metodi käyttää tiedon lähettämiseen ja vastaanottamiseen. (expressjs.com). Esimerkkinä tästä voidaan pitää kuvassa 13 esiintyvää get-metodia ja sen argumentteja – polkua ja takaisinkutsufunktioita.

Kuvassa 14 palvelimelle asetetaan portti, jonka välityksellä tietoa siirretään selainpuolelle. Portti 5000 asetetaan app-objektille käytettäväksi listen-metodilla. Palvelimen konsoliin tulostetaan käytettävän portin numero, jolloin varmistetaan, että portti todella on käytössä ja toiminnassa.

```
app.get('/api/views', (req, res)
```

Kuva 13. Get-metodi ja argumentit: polku, pyyntö ja vastaus.

```
31  const port = 5000;
32
33  app.listen(port, () => console.log(`Server running on port ${port}`));
```

Kuva 14. Palvelimelle asetetaan portti jota app-funktio käyttää.

Palvelimelle luodaan lauseke, joka lähettää Google Analytics Core Reporting -rajapintaan pyyntöobjektin. Tähän objektiin asetetaan parametreinä ne tiedot, jotka Googlen rajapinnasta halutaan saada. Tässä tapauksessa tiedot ovat:

- Seurantatietojen aloituspäivä
- Seurantatietojen lopetuspäivä
- Kukin päivämäärä seurantajakson ajalta
- Käyttäjien määrä kunakin seurattavana päivänä

Objekti, joka Google rajapintaan pyyntönä lähetään, sisältää myös käytettävän autentikaatiomenetelmän, sekä seurattavan Google Analytics -näkymän numeron.

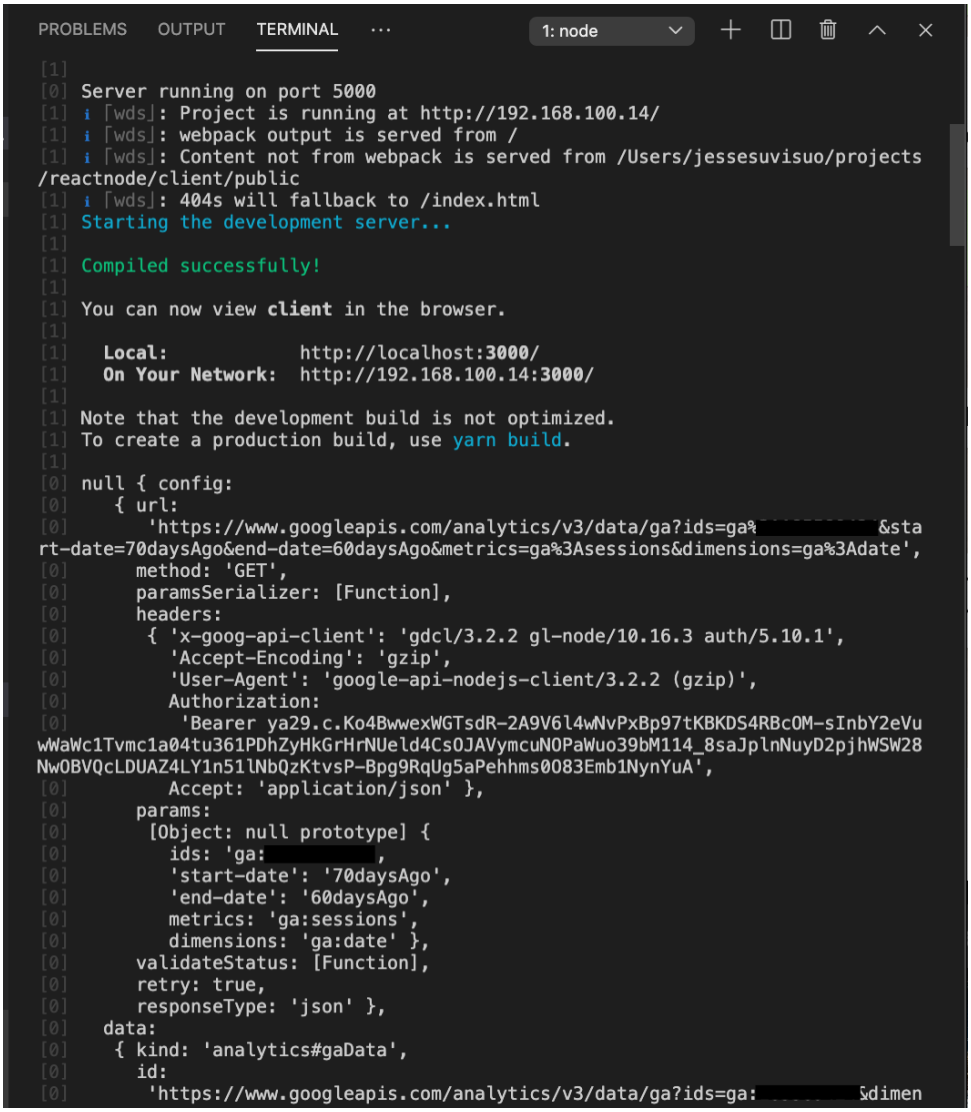
```
13      google.analytics('v3').data.ga.get(
14        {
15          auth: jwt,
16          ids: 'ga:' + key.view_id,
17          'start-date': '10daysAgo',
18          'end-date': 'today',
19          'metrics': 'ga:sessions',
20          'dimensions': 'ga:date',
21        },
```

Kuva 15. Google Analytics -rajapintaan lähetetään pyyntö – objekti, joka sisältää halutut tiedot parametreinä, autentikaatiomenetelmän, sekä seurantanäkymän numeron.

Google Analytics -rajapinnalta vastaanotettu objekti voidaan tulostaa palvelimen konsoliin. Tällöin varmistutaan, että halutut tiedot vastaanotetaan onnistuneesti palvelimelle. Objektista voidaan samalla havaita, millä nimellä ja missä muodossa kukin tieto objektissa on esitetty (Kuva 17).

```
11  app.get('/api/views', (req, res) => {
12    jwt.authorize((err, result) => {
13      google.analytics('v3').data.ga.get(
14        {
15          auth: jwt,
16          ids: 'ga:' + key.view_id,
17          'start-date': '10daysAgo',
18          'end-date': 'today',
19          'metrics': 'ga:sessions',
20          'dimensions': 'ga:date',
21        },
22        (err, result) => {
23          console.log(err, result);
24          res.json(result);
25        }
26      )
27    })
28
29  });
```

Kuva 16. Tämä lauseke lähettää pyynnön Googlen rajapintaan, asettaa "result"-muuttujan arvoiksi vastaanotetun objektin arvot ja asettaa objektin rajapintaan, josta se voidaan saada selaimessa käyttöön.



Kuva 17. Googlen rajapinnasta vastaanotettu objekti tulostettuna palvelimen konsoliin.

5.4 Tietojen esittäminen EChart-kirjastoa käyttäen

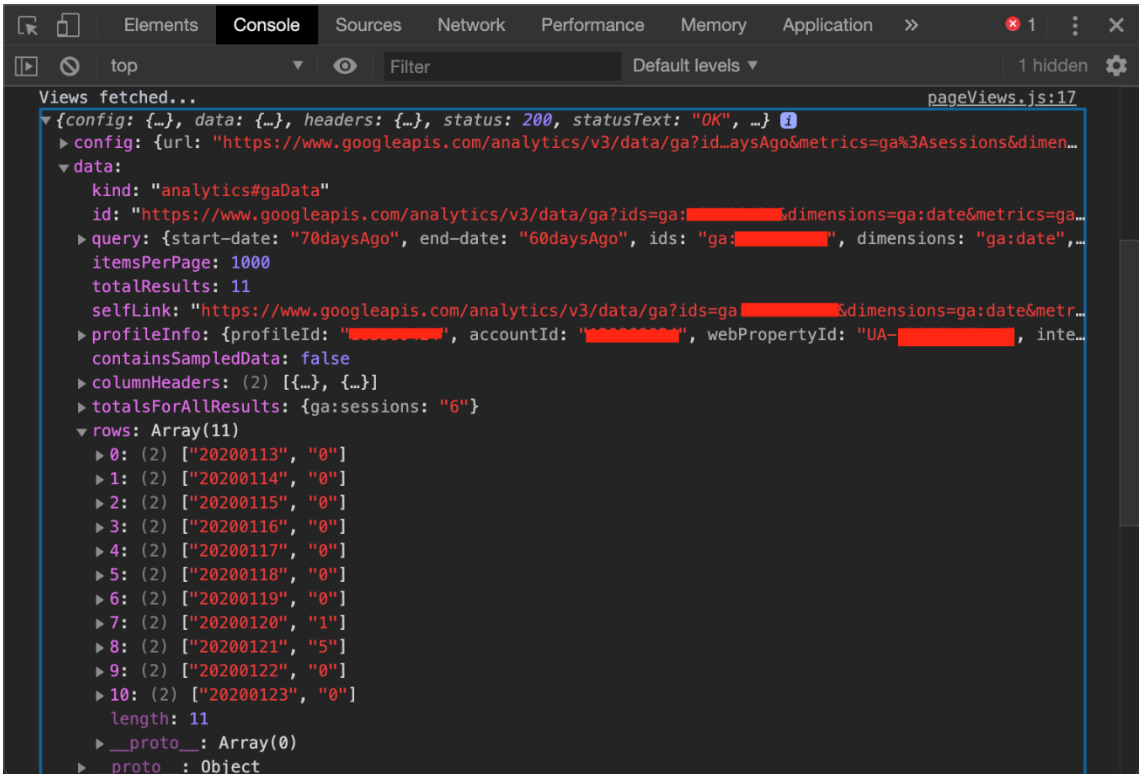
Kun palvelin on vastaanottanut onnistuneesti Google Analytics Core Reporting -rajapinnalta objektin, joka pitää sisällään halutut tiedot seurattavasta verkkosivusta, voidaan alkaa luomaan sovelluksen front-end -puolta. Ensin luodaan komponentti nimeltä ”Pageviews”. Googlen rajapinnasta saatujen tietojen siirtäminen selaimelle toteutetaan Express-rajapinnan avulla. Tiedot haetaan palvelimen portin kautta fetch-metodilla. Koska rajapinnan reititykselle on asetettu package.json-tiedostoon ”proxy” porttiin 5000, voidaan fetch-metodissa käyttää polkua ”/api/views”.

Kun Googlen rajapinnasta saatu objekti on saatu front-end -puolelle, voidaan sen arvot asettaa ”state”-objektiin. Tämä voidaan tehdä componentDidMount() -metodissa, koska se suoritetaan React-komponentissa ensimmäisenä. Tämän johdosta render() -metodi kutsutaan kahdesti, mutta se ei luultavasti aiheuta suuria ongelmia, sillä komponentissa tehtävät toimenpiteet eivät ole raskaita, jolloin viive jää häviävän pieneksi.

```
12   componentDidMount() {
13     fetch('/api/views')
14       .then(res => res.json())
15       .then(views => {
16         console.log(typeof views, views);
17         this.setState({ views }, () => console.log('Views fetched...', views))
18       });
19   }
```

Kuva 18. React-komponentin "componentDidMount"-metodin sisällä käytetään fetch-metodia objektin saamiseksi palvelimelta. Samassa metodissa asetetaan ”views”-muuttujalle ”state” ja tulostetaan ”views” selaimen konsoliin.

Ensin testataan, onnistuuko objektin tulostus selaimen konsoliin. Tällöin varmistutaan, että tiedon siirtäminen palvelimelta selaimelle toimii, ja Google Analytics Core Reporting -rajapinnan kautta vastaanotettu objekti on käytettävissä selaimessa. Google Chrome -selaimen kehittäjätyökalun avulla voidaan tarkastella tulostettua objektia.



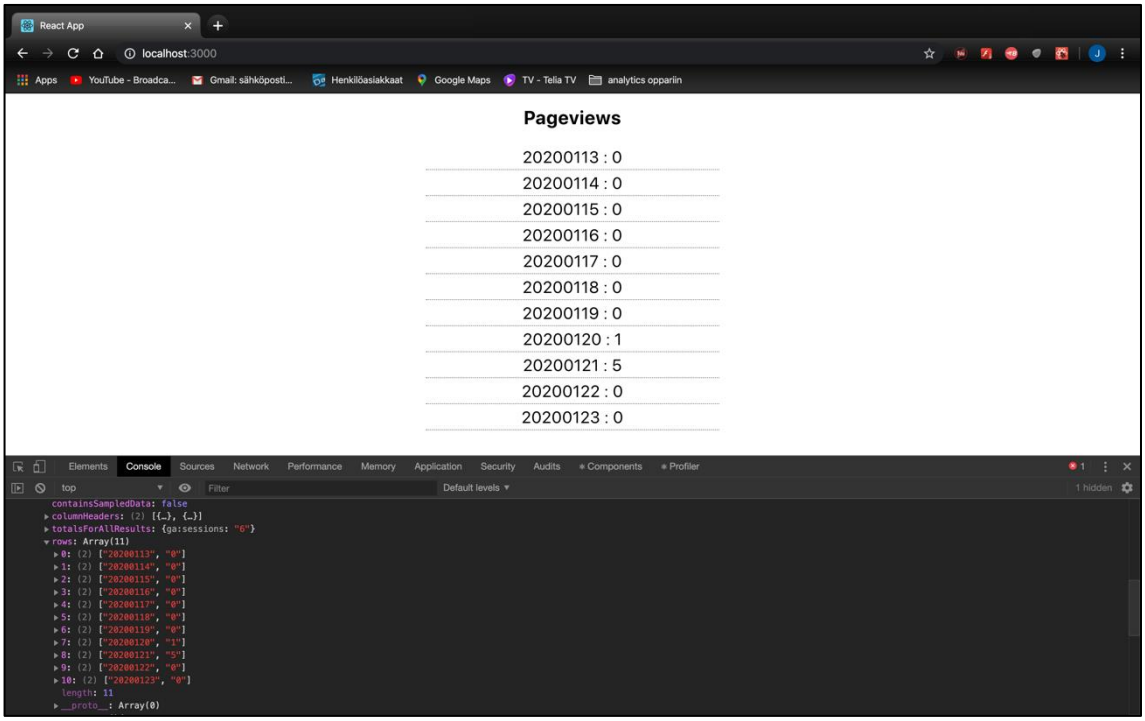
Kuva 19. Selaimen kehittäjätyökalun avulla voidaan tarkastella objektia, joka pyydettiin Google Analytics -rajapinnan kautta ja tulostettiin selaimen konsoliin. JSON-muodossa oleva objekti vastaanotetaan selaimelle fetch-metodin avulla.

Selaimen konsoliin tulostetusta objektista voidaan nähdä ne sarakkeet, jotka halutaan saada näkyviin selaimeen rakennettavassa kuvaajassa. Tämän jälkeen render-metodissa voidaan käyttää JavaScript-funktiota nimeltä map. Map-funktio mahdollistaa uuden arrayn luomisen käymällä läpi jokaisen elementin valitusta arraysta. Näin voidaan luoda lista tarkasteltavidta päivämääristä ja näiden päivämäärien kävijämääristä seurattavalla sivustolla. Tämä lista voidaan sitten tulostaa selaimeen. Koska

toimeksiannossa pyydettiin kuvaajan lisäksi listaa, voidaan se luoda sovellukseen ensin.

```
21   render() {
22     if (!this.state.views) {
23       return null;
24     }
25
26     console.log(this.state.views);
27
28     return (
29       <div className="Pageviews">
30         <h2>Pageviews</h2>
31         <ul>
32           {this.state.views.data.rows.map(row => {
33             return <li key={row.id}>{row[0] + ' : ' + row[1]}</li>
34           })}
35         </ul>
36
37       </div>
38     );
39   }
40 }
41
42 export default Pageviews;
```

Kuva 20. Render-metodissa käytetään views-objektiin map-funktiota, jonka avulla halutut sarakkeet saadaan näytettyä selaimessa.



Kuva 21. Selaimeen renderöity lista, jossa näkyy Googlen rajapinnasta pyydetyt päivät ja niiden päivien kävijämäärät. Konsoliin on tulostettu selaimeen tulostettuja tietoja edustava array, jonka sarakkeet ovat päivämäärät, sekä kunkin päivän kävijämäärä.

Kuvaajaa varten luodaan uusi komponentti nimeltä Chart. Kuvaaja luodaan käyttäen ECharts-kirjastoa. ECharts on JavaScrips-kirjasto, jolla luodaan graafisia elementtejä käyttöliittymään.

ECharts valikoitui tässä projektissa käytettäväksi kirjastoksi sen kattavan dokumentaation, sekä kuvaajien muokkausmahdollisuuksien ansiosta. EChrats-kirjasto asennetaan komentoriviltä komennolla *npm i echarts echarts-for-react*. Kun se on asennettu, voidaan se ottaa käyttöön komponentissa kirjoittamalla komponentin alkuun *import ReactEcharts from 'echarts-for-react';*.

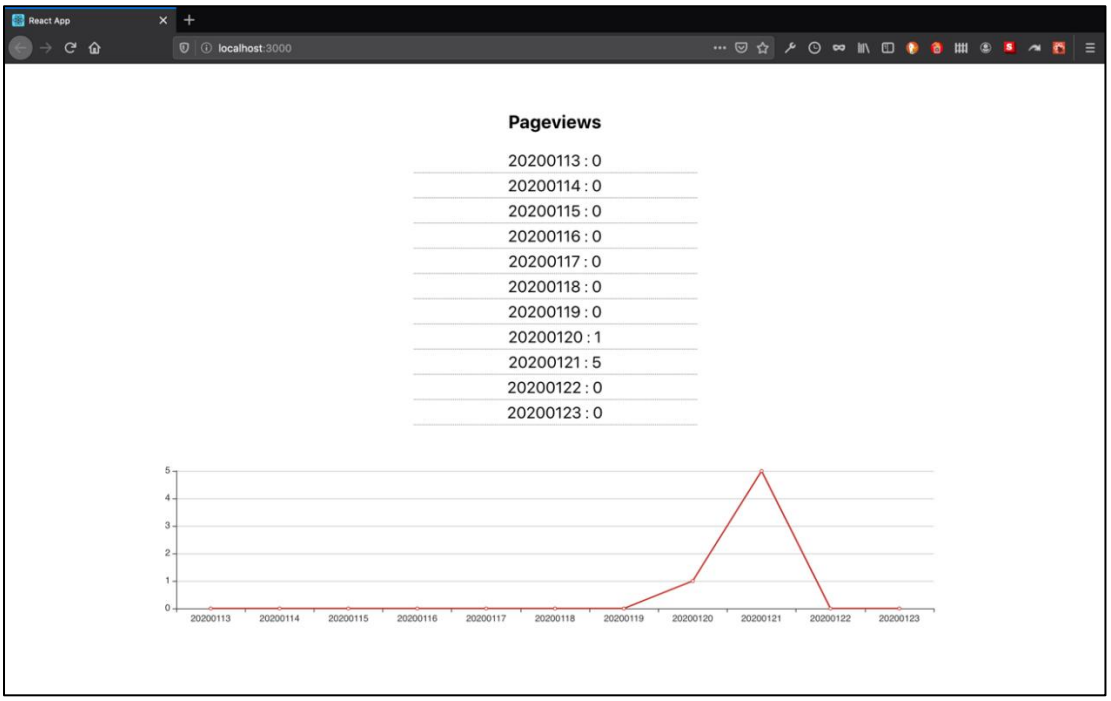
Kuten Pageview-komponentissa, myös Chart-komponentissa käytetään fetch-metodia Google Analytics -rajapinnasta pyydetyn objektin tuomiseksi Char-komponenttiin. Chart-komponentissa map-funktiota käytetään kahden muuttujan arvojen asettamiseksi. Nämä muuttujat edustavat kuvaajan x-akselille asetettavia sarakkeiden nimikkeitä, sekä kuvaajaan asetettavia arvoja. Labels- ja data-muuttujat saavat arvonsa, jonka jälkeen ne asetetaan state-objektiin.

Kuvaajan konfiguraation määrittämiseksi Chart-komponenttiin luodaan myös uusi metodi, joka asettaa labels- ja data -muuttujien arvot ECharts-komponentille. Tämän metodin nimi on getOption ja sitä kutsutaan React-komponentin render-metodissa.

```
24    getOption() {
25      return {
26        title: {
27          text: ''
28        },
29        xAxis: {
30          type: 'category',
31          data: this.state.labels,
32        },
33        yAxis: {
34          type: 'value',
35        },
36        series: [{
37          data: this.state.data,
38          type: 'line'
39        }]
40      }
41    }
42
43    render() {
44      console.log('render data', this.state.data);
45      return (
46        <div className="chart">
47          <ReactEcharts
48            option={this.getOption()}
49            notMerge={true}
50            lazyUpdate={true}
51            theme={"theme_name"}
52            onChartReady={this.onChartReadyCallback}
53            opts={{ renderer: 'svg' }} />
54        </div>
55      );
56    }
57  }
58
59  export default Chart;
```

Kuva 22. getOption-metodissa määritetään arvot, jotka render-metodissa asetetaan option-ominaisuudelle kutsumalla this.getOption-funktiota.

Edellä luotuja komponentteja kutsutaan app.js-tiedoston render-metodissa. App.js-komponentti suoritetaan ja asetetaan juurielementiksi index.js-tiedoston ReactDOM.render() -funktiossa. Tätä juurielementtiä vastaava tunniste löytyy index.html -tiedostosta, jonka ansiosta React-applikaatio saadaan tulostamaan selaimeen halutunlainen käyttöliittymä, joka sisältää kuvaajan, sekä taulukon.



Kuva 23. Sekä taulukko, että kuvaaja, joissa esitetään Google Analytics -rajapinnasta pyydetyt tiedot.

6 JATKOKEHITYS

Tässä opinnäytetyössä luotuun sovellukseen tulee vielä useita muita toimintoja. Luotu Google Analytics -integraatio on osa sovellusta, joka tulee pitämään sisällään muitakin komponentteja, kuten projektienseurantatyökalun. Toimeksiantajan kanssa sovittiin jatkotoimenpiteistä applikaatiota koskien.

Jatkokehityksessä tullaan tarkastelemaan mahdollisuuksia, joita Google Analytics Core Reporting -rajapinta antaa. Selvitetään muun muassa, minkälaisia tietoja Googlen rajapinnan avulla voidaan saada. Jatkokehityksessä selvitetään myös, mitä tietoja halutaan vastaanottaa, miten niitä halutaan järjestellä ja suodattaa.

Google Analytics -rajapintaan lähetettävässä pyynnössä, sovelluksen palvelinpuolella, voidaan pyyntöobjektiin tehdä muutoksia. Muutoksia voidaan tehdä objektiin asetettuihin parametreihin, kuten mittauksen aloituspäivään ja lopetuspäivään. Myös mahdollisesta autentikaatiomallin muuttamismahdollisuudesta keskusteltiin toimeksiantajan kanssa, eikä sovelluksen nykyisessä kehitysvaiheessa tarvetta vahvemmalle suojaukselle havaittu.

Luotujen komponenttien ja palvelimen hyödyntämistä sovelluksen React Nativella tehtävässä versiossa selvitetään. Tässä opinnäytetyössä luotu rajapintojen integraatio vaatii joitakin uudistuksia käyttäjän tunnistautumisen ja sisäänkirjauksen osalta. Tunnistetietoja ei voida säilöä sovelluksen tiedostoihin, vaan niille on luotava tietokanta. Jatkokehityksestä vastaa toimeksiantaja.

Toimeksiantaja on työn lopputulokseen erittäin tyytyväinen. Jokainen projektin toimeksiantoa koskeva tehtävä täytettiin, ja sovelluksen kaikki osat toimivat toivotulla tavalla. Opinnäytetyössä luotu sovellus testattiin toimeksiantajan edustajan kanssa, ja todettiin, että jatkokehittäminen asiakaskäyttöön voidaan aloittaa.

7 YHTEENVETO

Tässä opinnäytetyössä pyrittiin tarkastelemaan modernissa Web-kehityksessä käytettäviä työkaluja, kirjastoja ja tekniikoita. Tavoitteena oli myös selvittää, miten data-analytiikkaa hyödynnetään liiketoiminnassa ja miten Google Analytics -rajapinta integroidaan Web-sovellukseen. Opinnäytetyössä tarkastellaan, mitä työkaluja tällaisen sovelluksen luomiseen voidaan käyttää, ja miksi ne työkalut valittiin käytettäviksi tässä projektissa.

Google Analytics -palvelun käyttöä oli tarkoitus tarkastella Web-kehittäjän näkökulmasta, ja selvittää, miten integraatio kokonaisuudessaan toteutetaan. Data-analytiikan ja sovelluksen rakentamiseen valikoitujen tekniikoiden tarkastelun jälkeen keskityttiin itse sovelluksen luomiseen. Selvitettiin, mitä ohjelmoinnin eri vaiheessa tehdään, ja mitä saavutettiin.

Osa valituista tekniikoista ja työkaluista tuli toimeksiannossa, mutta näistä valinnoista keskusteltiin toimeksiantajan kanssa jo ennen sovelluksen rakentamisen aloittamista. Tämän opinnäytetyön tekijä pääsi vaikuttamaan tekniikoita ja työkaluja koskeneisiin valintoihin. Valinnoissa mietittiin käytettävien tekniikoiden toimivuutta, tulevaisuutta, ja niiden tukea ja kehitystä.

Valitut tekniikat ja teknologiat osoittautuivat käyttötarkoitukseensa toimiviksi. Lopputuloksen kannalta erinomainen asia on luodun sovelluksen ohjelmakoodin hyödyntämismahdollisuus vastaavanlaisen mobiilisovelluksen rakentamisessa. React ja React Native ovat molemmat komponentteihin ja JavaScriptiin perustuvia teknologioita, joiden samankaltaisuuden ansiosta tässä opinnäytetyössä luotua ohjelmakoodia voidaan ainakin osittain käyttää uudelleen mobiilisovellusta luotaessa. Molemmat edellä mainituista käyttävät React-kirjastoa.

React-kirjaston käyttäminen käyttöliittymän ohjelmoinnissa osoittautui nopeaksi ja tehokkaaksi tavaksi ohjelmoida yksinkertaisia elementtejä käyttöliittymään. Myös ohjelmoitavan koodin määrä pysyi vähäisenä, React-kirjaston avulla. Käyttöliittymän elementit latautuvat nopeasti, eikä ongelmia sovelluksen nykyisessä kehitysvaiheessa ilmene.

Myös Node JS, sekä valitut NPM-moduulit toimivat hyvin käyttötarkoituksessaan. Luodun palvelimen ja Google Analytics -rajapinnan välinen tietoliikenne on suhteellisen kevyttä, jolloin yksisäikeisen teknologian voidaan olettaa toimivan halutulla tavalla. Kaikkiin työkaluihin valitut kirjastot, moduulit ja ympäristöt olivat hyvin dokumentoituja, ja näin myös helppoja käyttää ja ymmärtää.

LÄHTEET

AW Academy group. (2019) Haettu 26.3.2020 osoitteesta: https://www.awacademy.fi/news/mita-data-analytiikka-on-ja-miten-se-pyorittaa-maailmaa

Buna, S. (2017). Yes, React is taking over front-end development. The question is why. Haettu 29.3.2020 osoitteesta: https://www.freecodecamp.org/news/yes-react-is-taking-over-front-end-development-the-question-is-why-40837af8ab76/

Crandall, T. (2018). Stop Google Analytics From Tracking My Visits (2018) - Hang Ten SEO. Haettu 16.2.2020 osoitteesta: https://hangtenseo.com/stop-google-analytics-from-tracking-my-visits-2018/

Express JS. (2020). Express - Node.js web application framework. Haettu 25.2.2020 osoitteesta: https://expressjs.com/

Fullstackopen. (n.d.). Fullstack osa3 | Node.js ja Express. Haettu 12.2.2020 osoitteesta: https://fullstackopen.com/osa3/node_js_ja_express

Google LLC. (2017). What Is The Core Reporting API – Overview. Haettu 26.2.2020 osoitteesta: https://developers.google.com/analytics/devguides/reporting/core/v3/

Google LLC. (2019). Core Reporting API - Reference Guide | Analytics Core Reporting API. Haettu 5.3.2020 osoitteesta: https://developers.google.com/analytics/devguides/reporting/core/v3/reference

Google LLC. (n.d.a). Search behavior has changed the path to purchase - Think with Google. Haettu 4.3.2020 osoitteesta: https://www.thinkwithgoogle.com/feature/path-to-purchase-search-behavior/

Google LLC. (n.d.b). Set up the Analytics tag - Analytics Help. Haettu 4.3.2020 osoitteesta: https://support.google.com/analytics/answer/1008080

Minnick, C. (2016). The Real Benefits of the Virtual DOM in React.js – Accelebrate. Haettu 30.3.2020 osoitteesta: https://www.accelebrate.com/blog/the-real-benefits-of-the-virtual-dom-in-react-js/

Node JS. (2011). What is the file `package.json`? | Node.js. Haettu 28.3.2020 osoitteesta: https://nodejs.org/en/knowledge/getting-started/npm/what-is-the-file-package-json/

Npm. (n.a.b). concurrently – npm. Haettu 15.3.2020 osoitteesta: https://www.npmjs.com/package/concurrently

Npm. (n.d.a). npm | build amazing things. Haettu 15.3.2020 osoitteesta: https://www.npmjs.com/

React JS, (2020). Add React to a Website – React. Haettu 30.3.2020 osoitteesta: https://reactjs.org/docs/add-react-to-a-website.html#optional-try-react-with-jsx

React JS, (2020c). Component State – React. Haettu 28.2.2020 osoitteesta: https://reactjs.org/docs/faq-state.html#what-is-the-difference-between-passing-an-object-or-a-function-in-setstate

React JS, (2020d). State and Lifecycle – React. Haettu 28.2.2020 osoitteesta: https://reactjs.org/docs/state-and-lifecycle.html

React JS, (2020e). Virtual DOM and Internals – React. Haettu 28.2.2020 osoitteesta: https://reactjs.org/docs/faq-internals.html#what-is-the-virtual-dom

React JS. (2020a). React – A JavaScript library for building user interfaces. Haettu 28.2.2020 osoitteesta: https://reactjs.org/

React JS. (2020b). Component State – React. Haettu 28.2.2020 osoitteesta: https://reactjs.org/docs/faq-state.html#what-is-the-difference-between-state-and-props

Surbhi, S. (2019). Difference Between Data and Information (with Comparison Chart) - Key Differences. Haettu 14.2.2020 osoitteesta: https://keydifferences.com/difference-between-data-and-information.html

Thakur, D. (2017). 10 Good Reasons Why You Should Use Google Analytics. Haettu 19.2.2020 osoitteesta: https://medium.com/@dineshsem/10-good-reasons-why-you-should-use-google-analytics-699f10194834

Thierauf, R. (1999). *Knowledge Management Systems for Business.* Westport, CT, USA: Quorum Books.

Traversy, B. (2016). Node.js Tutorial For Absolute Beginners. Haettu 12.2.2020 osoitteesta: https://www.youtube.com/watch?v=U8XF6AFGqlc

W3Schools, (n.d.). JavaScript HTML DOM. Haettu 30.3.2020 osoitteesta: https://www.w3schools.com/js/js_htmldom.asp