jamk.fi

ReactJS-kirjasto front-end-kehityksessä

Mikko Taipale

Opinnäytetyö
Toukokuu 2020
Tekniikan ala
Insinööri (AMK), tieto- ja viestintätekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Taipale, Mikko	Opinnäytetyö, AMK	Toukokuu 2020
	Sivumäärä 35	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
ReactJS-kirjasto front-end-kehityksessä

Tutkinto-ohjelma
Tieto- ja viestintätekniikka

Työn ohjaaja(t)
Ari Rantala

Toimeksiantaja(t)

Tiivistelmä

Opinnäytetyön tavoitteena oli tutkia ReactJS JavaScript-kirjaston komponenttipohjaista kehitystapaa ja sen etuja tiimiprojektin näkökulmasta. Tavoitteena oli myös ymmärtää mitä komponenttipohjainen web-kehitys mahdollistaa applikaation koodirakenteessa ja kuinka uudelleenkäytettävät komponentit suoraviivaistavat web-sivun rakentamista.

Opinnäytetyössä käytiin läpi ReactJS yleiset ominaisuudet ja sisältävät teknologiat. ReactJS-kirjastoa myös vertailtiin AngularJS ja Vue.js, jotka ovat toisia suosittuja JavaScript-ohjelmistokehyksiä. Opinnäytetyössä tutkittiin käyttötarkoitusta ohjelmistokehysten välillä ja miten eri ominaisuudet voivat vaikuttavat ohjelmistokehyksen valintaan.

Opinnäytetyössä myös tutkittiin ReactJS-applikaation koodirakennetta ja kuinka rakentaa web-applikaatio käyttäen ReactJS. Käytiin läpi geneerisen React-komponentin luonti ja sen käyttö. Opinnäytetyössä käytiin myös esimerkkien avulla läpi React-projektin potentiaalista rakennemallia ja web-sivun rakentamisen aloittamista.

Opinnäytetyön lopputuloksena todettiin ReactJS-applikaation komponenttimaisen rakenteen antavan etuja projektityölle tiimissä, jos projekti luodaan sen ympärille. Todettiin myös, että erilaiset ReactJS-kirjaston ominaisuudet vaikuttavat tiimiprojektin ominaisuuksien suunnitteluun, työnjakoon ja versionhallintaan.

Avainsanat (asiasanat)
JavaScript, front-end, web-kehitys, tiimikehitys

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Taipale, Mikko	Bachelor's thesis	May 2020
		Language of publication: Finnish
	Number of pages: 35	Permission for web publication: x

Title of publication
ReactJS library in front-end development

Degree programme
Information and Communication Technology, Software Engineering

Supervisor(s)
Rantala, Ari

Assigned by

Abstract

The goal of the thesis was to research ReactJS JavaScript library's component-based development methods and its benefits from a team project's perspective. The goal was to understand what component-based web design enables for the application's code structure and how recyclable components streamline web-site building.

The thesis goes through ReactJS's basic features and technologies. ReactJS library is also compared to AngularJS and Vue.js, which are other popular JavaScript frameworks. The thesis studies the use cases between the frameworks and how different features can affect the selection of a framework.

The thesis went through the ReactJS application's code structure in detail and how to build a web-application with ReactJS. It was also discussed how to create a generic React component and how to use it. The thesis showed examples of the potential structure and how to start building a web site with ReactJS.

The result of the thesis was that the component-based structure of a ReactJS application benefits team projects if the projects are created around it. It was also concluded that different ReactJS features affect team project's feature planning, work distribution and version control.

Keywords/tags (subjects)
JavaScript, front-end, web development, team development

Miscellaneous (Confidential information)

Sisältö

Kuviot

1 Johdanto

1.1 Työn tausta

Web-applikaatiokehityksessä sopivan sovelluskehyksen valinta voi olla haastavaa. Sovelluskehykset vaativat perehtymistä, ja niiden toiminnan ymmärtäminen ja oman kielirakenteen opetteleminen on vaativa prosessi. Sovelluskehysten välillä vaihtaminen on raskasta web-kehittäjälle, koska suosituimmat JavaScript-sovelluskehykset eroavat toisistaan paljon. Oikean sovelluskehyksen opettelun valinta on tärkeää web-kehittäjän tehokkuuden sekä työllistymisen kannalta. Mitä suositumpi sovelluskehys on, sitä suuremmalla todennäköisyydellä löytyy työmarkkinoilta projekteja, joissa kyseistä sovelluskehystä käytetään. Suosion lisäksi myös sovelluksen arvioitu elinkaari on otettava huomioon sovelluskehyksen valinnassa. Jos sovelluskehyksen tuen ja kehityksen tulevaisuus on epävarmaa, se vaikuttaa myös web-sovelluksen elinkaaren lyhenemiseen.

Facebookin kehittämä ReactJS on suosittu JavaScript-kirjasto, joka toimii sovelluskehyksenä web-applikaatioiden rakentamisessa. Sen komponenttikeskeinen rakenne on hyvin modulaarinen ja mahdollistaa koodin minimoinnin uudelleenkäytettävien komponenttien avulla.

1.2 Työn tavoite

Työn tavoitteena oli tutkia ReactJS-kirjastoa sekä verrata sitä muihin suosittuihin JavaScript-sovelluskehyksiin. Työssä tutkittiin, mikä tekee ReactJS-kirjastosta hyvän vaihtoehdon tiimiprojektiin, ja kuinka komponenttipohjainen rakenne vaikuttaa projektityön suunnitteluun ja versionhallintaan. Työssä käydään myös läpi, mitä etuja on komponenttipohjaisessa web-applikaation rakennuksessa, sekä mitä uudelleenkäytettävät komponentit mahdollistavat applikaation koodirakenteessa.

Työssä esitellään myös ReactJS-sovelluksen luonti, rakenne sekä kuinka ReactJS-projektia aletaan laajentamaan.

ReactJS-kirjastoa tutkittiin tiimiprojektinäkökulmasta, jossa on useita web-kehittäjiä luomassa samaa ReactJS-sovellusta. Tavoitteena oli myös käydä läpi, miten ReactJS-projektin tiedostorakennetta, sekä uudelleenkäytettäviä komponentteja voidaan käyttää hyväksi tiimityöskentelyyn kehittämisen ja versionhallinnan näkökulmasta.

2 ReactJS:n yleiset ominaisuudet

2.1 ReactJS

ReactJS on Facebookin ylläpitämä JavaScript-kirjasto, joka luotiin vuonna 2013 ja on siitä lähtien kehittynyt yhdeksi käytetyimmistä sovelluskehyksistä. ReactJS ei kuitenkaan sisällä kaikkea web-applikaation ajamiseen tarvittavia ominaisuuksia. Se on keskittynyt täysin web-sivun elementtien tulkintaan eikä sisällä tarvittavia backend-ominaisuuksia tai web-sivun reititystä. ReactJS web-applikaatio tarvitsee siis tuekseen lisäkirjastoja, kuten Redux ja React Router. (Morris 2019.)

ReactJS:ää käytetään rakentamaan interaktiivisia elementtejä web-sivuille. React DOM käsittelee HTML-, CSS- ja JavaScript-koodin ja tulostaa sen web-sivulle. ReactJS-applikaatio käyttää omaa syntaksia JSX. ReactJS:n vahvin ominaisuus on sen komponenttipohjainen rakenne. Se mahdollistaa yksittäisten komponenttien päivityksen web-sivulla ja keventää web-sivun rasitusta. Komponenttirakenne mahdollistaa myös samojen komponenttien uudellenkäyttämisen tekemällä geneerisiä komponentteja. (Morris 2019.)

Komponenttien välillä voidaan kuljettaa tietoa hakemalla sen ”import”-kutsulla tai valuttamalla dataa toiselta komponentilta. Datan kuljettaminen komponenttien

välillä sisäisesti välttää tarpeettomat API-kutsut. Komponenteilla on myös ”state”, komponentin tila, johon säilötään muuttujia, joiden arvoja voidaan dynaamisesti vaihtaa vaikuttamattaa muuhun web-applikaatioon. Komponentin state päivittyy vain sitä päivittämään sunnatulla metodilla, joten ristiriitaa muiden muuttujien välille ei tule. Staten muuttujan oletusarvon voi määrittää muuttujalla, mutta ei päivittää. Statessa voi säilöä ja manipuloida dataa ilman, että alkuperäistä dataa on pakko hakea uudestaan. Esimerkiksi jos listasta poistetaan jokin arvo, voidaan lähettää API-kutsu poistamaan arvo listasta, mutta listaa ei tarvitse hakea uudestaan API:lta, vaan voidaan suoraan komponentissa poistaa arvo myös staten listasta. Tällä tavalla listan datan hakemisen API-kutsu tarvitsee tehdä vain yhden kerran, ja silti komponentin data ja API:n data pysyvät täysin samanlaisina, ja oikea data näkyy reaaliaikaisena web-sivulla. Komponentin state-kentän päivityksellä voidaan luoda kevyitä web-toteutuksia, jotka vaativat vähimmäismäärän API-kutsuja. State-kentässä voidaan myös säilyttää arvoja, joita ei välttämättä haluta säilöä komponentin käytön jälkeen. Esimerkiksi jos komponentissa on hakukenttä, joka on oletusarvoltaan tyhjä, voidaan hakukentän arvo säilyttää statessa. Kun komponentti suljetaan ja avataan uudestaan, hakukentän arvo on taas tyhjä. (Morris 2019.)

2.2 HTML

Hypertext Markup Language eli HTML on koodikieli, joka toimii web-sivun rakenteena. Web-sivun visuaalinen toteutus luodaan HTML-elementeillä, jotka muodostavat lohkorakenteen. Lohkorakenteen järjestys vaikuttaa web-sivun elementtien sijoitteluun ja niiden vuorovaikutukseen. Teksti, linkit ja otsikot tehdään käyttäen HTML-elementtejä. HTML ei ole ohjelmointikieli, mikä tarkoittaa, ettei se voi luoda dynaamista toiminnallisuutta web-elementeille. (Domantas 2019).

2.3 DOM

Kun web-sivu ladataan, selain luo sivusta Document Object Modelin eli DOMin. DOM on pohjimmiltaan API web-sivulle. Se sallii ohjelmien lukea ja muuttaa sivun sisältöä, rakennetta ja tyylejä. DOM on objektipohjainen kuvaus alkuperäisestä HTML-dokumentista, eli sen tehtävä on muuttaa HTML-dokumentti objektimalliin, jonka avulla sitä voi käyttää ohjelmien kanssa. DOM ei kuitenkaan ole aina samanlainen kuin lähde HTML-dokumentti. Jos HTML-dokumentista puuttuu esimerkiksi <head> tai <body> elementti, DOM osaa automaattisesti luoda ne. DOM pystyy myös muuttumaan alkuperäisestä HTML-muodostaan JavaScript-funktioilla. (Aderinokun 2018.)

2.4 CSS

Cascading Style Sheets eli CSS on muotoilukieli, jonka tehtävä on muokata HTML-elementtien ominaisuuksia. CSS tyylitiedostoon asetetaan HTML-elementteihin viittaavia objekteja, joiden sisään kirjataan elementin ominaisuuksien arvoja, kuten koko ja väri. Näitä CSS objekteja voidaan käyttää yksittäisen tai useamman elementin muokkaamiseen. HTML-elementteihin viitataan niiden elementin nimen tai annetun luokan nimen perusteella. CSS objektien vaikutusta voidaan rajata myös sisentämällä objekteja toistensa sisään, jolloin ne vaikuttavat ainoastaan silloin, kun ne löytyvät ylemmän objektin elementistä. (What is CSS? n.d.)

2.5 JavaScript

JavaScript on koodikieli, joka suorittaa komentoja, jotka hallitsevat dynaamisesti web-sivun sisältöä. Web-sivun DOMin elementit lähettävät, tuovat ja käsittelevät

dataa JavaScript-funktioiden avulla. JavaScriptin tehtävä on lisätä animoituja ja vuorovaikutteisia elementtejä web-sivuille. (What is JavaScript? n.d.)

2.6 JSX

JSX on syntaksi, jota ReactJS käyttää perinteisen HTML-syntaksin sijaan. Toisin kuin HTML, JSX-syntaksia voi kirjoittaa suoraan JavaScript-koodiin. JSX:ää sisältävän JavaScriptin pitää kuitenkin prosessoida kääntäjällä kuten esimerkiksi Babelilla. Babelin päätehtävä on kääntää modernit versiot JavaScriptistä vanhaan versiomuotoon, jossa niitä voi käyttää esimerkiksi vanhemmat web-selaimet.

HTML-koodissa on pieniä eroja, jotka estävät sen HTML-tiedoston suoran toiminnan JSX-formaatissa. Attribuuttien nimissä on HTML- ja JSX-koodikielien välillä eroja, kuten esimerkiksi HTML-kielen ”class” on JSX-kielessä ”className”. (What is JSX? n.d.; What is Babel? n.d.)

2.7 ECMAScript (ES)

Koska JavaScriptilla ei ole versiojulkaisuja, ECMAScript on JavaScriptin kehityskieli, jolla laajennetaan sen toiminnallisuutta. ECMAScript muuttaa JavaScriptin kirjoitustapaa huomattavasti. Tällä hetkellä ES6 on uusin ECMAScript versio, jota kaikki modernit webselaimet tukevat. ES-versioita kehitetään jatkuvasti, mutta niiden käyttöönotossa menee aikaa. Versio pitää olla virallisesti tuettuna webselaimissa ennen kuin versioon päivittäminen on järkevää. Vanhan JavaScript-version päivittäminen uuteen ES-formaattiin vie huomattavan paljon aikaa. ES5- ja ES6-versioiden välillä on suuria eroja kirjoitusmuodossa. Vaikka ES-version päivittäminen ei ole välttämätöntä eikä anna minkäänlaista suoritusparannusta selaimelle, on viisasta pitää koodi mahdollisimman modernina. Uudet ES-versiot yksinkertaistavat

koodin kirjoitusmuotoa ja vähentävät mahdollisuutta koodivirheille. Toisin sanottuna ES-versiot eivät ole tehty käyttäjälle vaan kehittäjälle. (Li 2018.)

2.8 Node.js

Node.js on avoimen lähdekoodin JavaScript runtime-ympäristö. Node.js mahdollistaa JavaScript-koodin suorittamisen suoraan palvelimella, web-sivun HTML-koodiin upottamisen sijaan. (About Node.js® n.d.)

2.9 Node Package Manager

Node Package Manager eli npm on maailman suurin ohjelmistorekisteri ja JavaScript-kieleen tehty paketinhallintatyökalu. npm vaatii Node.js asennuksen ja yleensä npm asennetaan osana Node.js ympäristöä. npm hallinnoi kaikkia ReactJS-applikaatiossa olevia paketteja ja niitä käytetään komentorivin kautta. npm mahdollistaa helpon ulkoisten kirjastojen asennuksen, sekä projektin hallinnan terminaalikomentojen kautta. (About npm n.d.)

3 Vertailu muihin kehitysympäristöihin

3.1 AngularJS

AngularJS on Googlen ylläpitämä JavaScript-ohjelmistokehys, joka on suosittu vaihtoehto web-kehitykseen. Toisin kuin ReactJS, AngularJS sisältää kaiken web-applikaation rakentamiseen tarvittavat komponentit kuten lomakkeen validoinnin, HTTP-pyynnöt ja reitityksen. AngularJS kuitenkin vaatii web-kehittäjää käyttämään AngularJS:n omia ominaisuuksia ja tekee AngularJS:n ulkopuolisten kirjastojen käytön vaikeaksi. AngularJS pitää sisällää vakaat ja päiviteyt työkalut web-applikaatiokehitykseen, mutta ei anna web-kehittäjälle paljoa valinnanvaraa

työkalujen valinnassa. Se eroaa ReactJS UI-sentrisestä kehitysympäristöstä, joka kannustaa käyttämään muita avoimen lähdekoodin ohjelmistoja sen tukena. (Martin 2019; Starikov 2019.)

3.2 Vue.js

Vue.js on pienen tiimin kehittämä JavaScript-ohjelmistokehys. Vue.js on avointa lähdekoodia ja on hyvin riippuvainen yhteisön kehittämistä avoimen lähdekoodin ohjelmistoista. Vue.js kuitenkin sisältää jotain ominaisuuksia, joita ReactJS ei sisällä, kuten reititykseen tarvittavat työkalut, ja kooltaan Vue.js on AngularJS:n ja ReactJS:n väliltä. Irrallisuus isoista länsimaisista yhtiöistä toisin kuin ReactJS tai AngularJS tekee Vue.js ohjelmistokehyksestä suositun valinnan maissa kuten Kiina. (Martin 2019; Starikov 2019.)

4 Komponentti-pohjainen rakenne tiimikehityksessä

4.1 Kansiorakenne

Kun applikaatiota kehitetään tiimissä, ReactJS-projektin rakenteeseen kannattaa kiinnittää huomiota. Puumallinen tiedostorakenne pitää komponenttien kansiot organisoituina ja kaikille front-end-kehittäjille selkeinä. ReactJS tiedostorakenteessa ei kuitenkaan kannata sisentää liian syvälle kansioita, vaan säilyttää kaikki komponentit yhdessä kansiossa. Tällä tavalla komponentit eivät piiloudu kansioiden sisälle, joista muut kehittäjät eivät välttämättä niitä ymmärrä etsiä. Komponenttien omat uniikit JavaScript- ja CSS-tiedostot kannattaa pakata samaan kansioon. Jokaiselle JSX:n sisältävän tiedoston kanssa kannattaa tehdä oma CSS-tiedosto. CSS-koodin voi valuttaa yhdestä isosta tyylitiedostosta muille komponenteille, mutta se voi aiheuttaa muutoksia muissa komponenteissa. Omat CSS-tiedostot pitävät komponentin tyylit varmasti uniikkina, eikä ristiriitoja synny. Tästä syystä

säiliötiedostoille, joihin komponentit asetetaan, kannattaa luoda mahdollisimman tarkat CSS-luokat. Jos luokkia ei määritellä ja käytetään esimerkiksi HTML-elementtien määritelmiä, CSS-tyylit vaikuttavat kaikkiin komponentteihin, joita säiliötiedoston sisässä on.

4.2 Geneeriset komponentit

Yksi ReactJS-kirjaston vahvuuksista on luoda uudelleen käytettäviä komponentteja. Tällaisten geneeristen komponenttien tarkoitus on toimia uudelleenkäytettävinä rakennuspalikoina. Komponentin uudelleenkäyttö varmistaa sen, että web-sivun samantyyppiset elementit pysyvät yhtenäisinä. Elementit, kuten tekstikentät, napit tai modaalit voidaan luoda geneerisinä komponentteina, eikä niiden kokoa tai muita tyylimäärittelyitä tarvitse uudelleen määritellä. Komponentti pitää kuitenkin suunnitella niin, että sitä voi modifioida antamalla sille arvoja, joiden perusteella se palauttaa toivotun version komponentista. Komponentti kannattaa jo aikaisessa vaiheessa varustaa tulevaisuutta varten luomalla sen funktiot niin, että ne käyttävät muuttujia määriteltyjen arvojen sijaan.

Komponentin käyttö voi olla muille kuin komponentin kehittäjälle hankalaa, jos komponentti sisältää paljon muuttujia. Tästä syystä on hyvä luoda käyttöohjeet komponenteille, joissa voi selventää niiden käyttötarkoituksen ja mitä eri komponentille syötettäviä arvoja on ja mitä ne tekevät. Tälläiset käyttöohjeet voi säilyttää tekstitiedostossa, wiki-sivulla tai vaihtoehtoisesti jopa kommentoida ne suoraan komponentin sisään. Komponentin kehittäjän tulisi myös ilmoittaa muille tiimin jäsenille komponentista ja tarvittaessa opastaa sen käytössä.

Jos projektin myöhemmällä vaiheella huomataan tarve geneerisille komponenteille, on hyvä, että vain yksi kehittäjä on vastuussa vanhojen elementtien korvaamisesta

kyseisellä komponentilla. Komponenttia voidaan joutua muuttamaan korvausvaiheessa jos huomataan, ettei se ole täysin elementin tarpeiden mukainen. Tästä syystä vain yhden kehittäjän on kannattavaa olla vastuussa korvaamisprosessista, jotta hän löytää komponenttiin hyvän yhtenäisen tasapainon kaikkien korvattavien elementtien välillä. Näin myös vältytään tarpeettomilta ristiinkoodauksilta.

4.3 ReactJS-applikaation kehitys tiimissä

Komponenttipohjaisen rakenteen ympärille sprinttien suunnitteleminen ReactJS-projektissa on viisasta. Komponentteja voidaan kehittää rinnakkain ilman ristiriitoja ja komponentit tästä syystä kannattaa jakaa yksittäisille henkilöille kehitettäväksi. Komponentit kannattaa jättää mahdollisimman yksinkertaisiksi ReactJS projektissa, etteivät ne paisu kooltaan liian suuriksi. Jos JSX koodi paisuu liian suureksi, voi sen aina pilkkoa uusiin komponentteihin. Liian suuret tiedostot ovat vaikealukuisia ja niiden hallinta vai olla hankalaa. Tästä syystä myös JavaScript-funktiot voidaan sijoittaan toiseen JavaScript-tiedostoon, josta niitä kutsutaan. Aivan kuten geneeriset komponentit, kehittäjä voi luoda geneerisiä JavaScript-funktioita, jotka suorittavat jonkun yleisen funktion. Kuitenkin pelkkien funktioiden kierrättäminen on melko rajattua ja yleensä niitä käytetään vain asioihin, kuten esimerkiksi tekstin suodattamiseen.

Eri ulkoisten kirjastojen eli ”node moduulien” asennus on helppoa npm-työkalun avulla ReactJS-applikaatioon. Tiimiprojektissa on tärkeää määrittää, mitä kirjastoja ja mitä versiota niistä käytetään. Kaikki asennetut kirjastot listataan package.json -tiedostoon, jossa näkyy kaikki applikaation ulkoisten kirjastojen asennukset ja niiden versiot.

4.4 Git-käytännöt ReactJS-projektissa

ReactJS projektin kehittäminen tiimissä vaatii selvien versionhallintajärjestelmäkäytäntöjen eli git-käytäntöjen määrittelemisen. ReactJS komponenttimuotoista applikaatiorakennetta hyödyntäen pystytään välttämään konfliktit versionhallinnassa komponenttien jakamisella eri kehitysympäristöihin. Komponentit voidaan kehittää omassa versiohaarassan eli ”git-branchissä”. Git-branchit ovat kopioita alkuperäisestä haarastaan, jotka voidaan komponentin valmistuttua yhdistää päähaaraan. Päähaaraan ei pitäisi suoraan kehittää koodaamalla, vaan päivittää sitä muita haaroja yhdistämällä. Tällä tavoin voidaan helposti myös jäljittää ja palauttaa haitalliset muutokset. Kehityksen alla olevat haarat eivät tulisi sisältää muutoksia samoihin komponentteihin, ellei se ole välttämätöntä. Jos komponentissa on yhdistämiskonflikteja kuten päällekkäisiä muutoksia, ne pitää käydä manuaalisesti yhdistämässä haluttuun muotoon.

Jos applikaatiota halutaan demota tai pitää käynnissä jossain ympäristössä, tulisi päähaarasta tehdä vakaita versiohaaroja. Koska päähaara saatta elää uusien komponenttien ja ominaisuuksien mukana, on tärkeää tallentaa vakaita versioita ohjelmasta. Ohjelman versiohaara tehdään yleensä koodin jäädytyksen jälkeen. Jäädytysvaiheessa testataan ohjelma kriittisten bugien tai muiden puutteiden kannalta. Koodijäädytyksen aikaan ei lisätä haaraan uutta koodia edellämainittujen puutteiden korjaamisen ulkopuolelta.

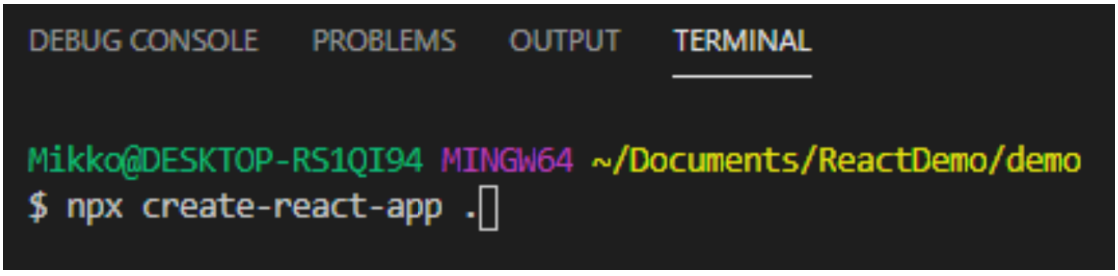
Jotkin tiedostotyypit ovat poikkeuksellisia versiohallinnan kannalta. Esimerkiksi generoidut tiedostot kuten käännöstiedostot, jotka rakennetaan UI:n viesteistä JSON tiedostoksi, on ongelmallisia käydä koodikonfliktin sattuessa muuttamassa manuaalisesti. Tästä syystä tällaiset suuret generoidut tiedostotyypit tulisi päivittää erikseen suoraan päähaaraan. Päähaaraan tulleiden muutosten jälkeen voidaan suoraan päähaarassa tehdä muutos generoimalla tiedostot uudelleen.

Ulkoisten kirjastojen koodi tallennetaan ”node_modules” -kansioon. ”node_modules” -kansio vie suuren osan applikaation koosta eikä sitä sen takia ole tapana tuoda versionhallintaan. Sen sijaan ulkoiset kirjastot asennetaan lokaalisti kehittäjän omaan ympäristöön. Uuden ympäristön pystyttäminen on ulkoiset kirjastot erikseen asentamalla nopeampaa. ReactJS-applikaatioon ulkoisten kirjastojen asennus ja päivitys voidaan tehdä terminaalikomennolla ”npm install” tai ”npm i”.

5 Koodirakenne ja käyttö

5.1 ReactJS-applikaation luonti

ReactJS-applikaation luonti vaatii Node.js:n asennuksen. Node.js:n asennuksen yhteydessä asennetaan myös sen oletus paketinhallintaohjelmisto npm. Kuviossa 1 ReactJS-applikaatio luodaan terminaalin kautta haluttuun sijaintiin komennolla ”npx create-react app .”. npx luo ReactJS-applikaation, mutta ei asenna sitä globaalisti. Globaaliin asennukseen voi käyttää komentoa ”npm” komennon ”npx” sijaan. Komennon loppuun lisättävä piste varmistaa että ReactJS-applikaatio luodaan valitun kansion sisään. Kuviossa 2 on ReactJS-applikaation asennuksen jälkeiset aloitusohjeet terminaalissa.

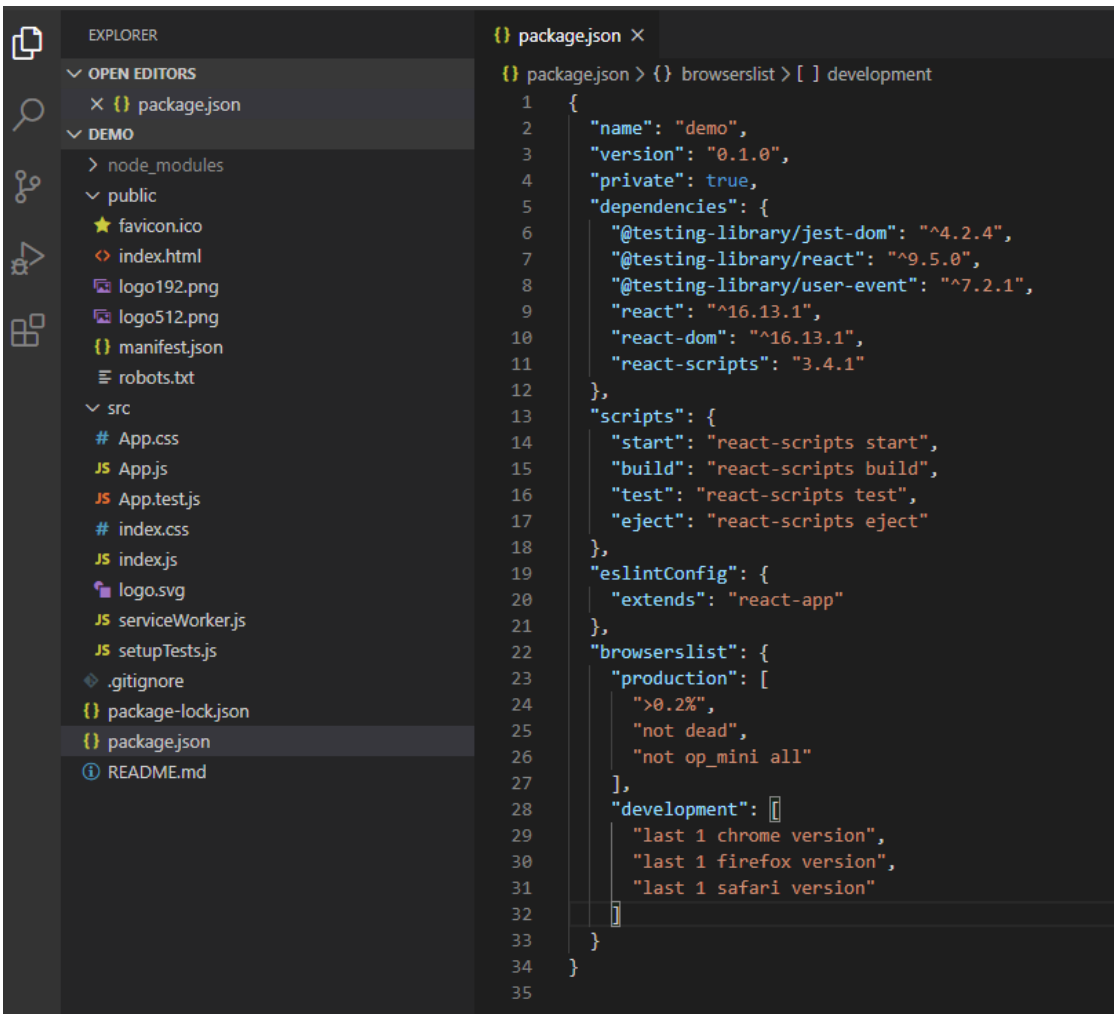


Kuvio 1. ReactJS-asennuksen terminaalikomento

```
Created git commit.

Success! Created demo at C:\Users\Mikko\Documents\ReactDemo\demo
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd C:\Users\Mikko\Documents\ReactDemo\demo
  npm start

Happy hacking!
```

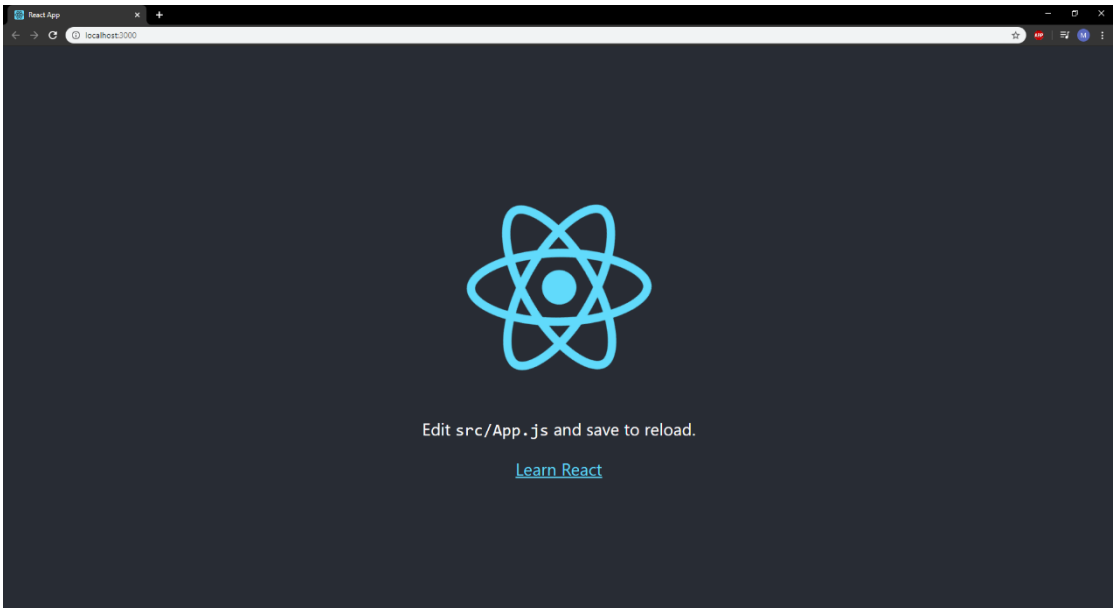
Kuvio 2. ReactJS:n terminaalikomento-ohjeet

Kun ReactJS-applikaatio on asennettu, npm luo valmiin tiedostopohjan applikaatiolle. Se asentaa automaattisesti uusimmat versiot riippuvuuksista ja luo komentoja applikaation käsittelyyn. React-applikaation asennus luo valmiin aloitussivun, joka toimii ”Hello world”-tyyppisenä esimerkkinä applikaation toiminnasta. Kuviossa 3 on asennetut npm-paketit package.json-tiedostossa, sekä niiden versionumerot.



Kuvio 3. Uuden ReactJS-projektin npm-paketit

ReactJS-applikaatio voidaan käynnistää ajamalla komento ”npm run start” terminaalissa. Applikaatio avaa automaattisesti oletusselaimen ja sivun ”http://localhost:3000”. Kuvio 4 näyttää esimerkkisivun käyttöliittymässä. Kun ReactJS-applikaatio on käynnissä, web-sivun sisältö päivittyy automaattisesti koodimuutoksien mukana, eikä applikaatiota tai selainta tarvitse erikseen päivittää.



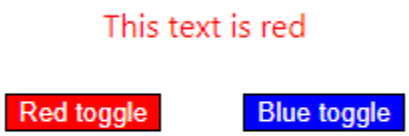
Kuvio 4. ReactJS-applikaation esimerkkisivu web-selaimessa

Tiedostoihin index.html (ks. liite 1) ja index.js (ks. liite 2) on kirjoitettu kommentteja avustamaan käyttäjää. Nämä tiedostot toimivat ReactJS-applikaation pohjana, jonka päälle react-komponentit rakennetaan. Tiedostoon index.js on tuotu komponentti App.js, joka toimii applikaation pääkomponenttina, jonka alle aletaan rakentamaan komponenttihierarkiaa. Kumpaankaan tiedostoon index.html tai index.js ei tehdä muutoksia tässä ReactJS-applikaation rakentamisen demonstraatiossa vaan toteutus aletaan rakentamaan App.js -tiedostosta lähtien.

App.js (ks. liite 3) tiedostoon on tehty esimerkkikoodia, jotka luovat ReactJS-applikaation aloitussivun. Kaikki ”App” luokan div-elementin sisältä voidaan poistaa. Myös logo.svg -tiedosto voidaan poistaa kokonaan applikaatiosta. App.css:ää voidaan myös tyhjentää, ettei oletustyylit vaikuta web-sivun ulkonäköön.

5.2 Geneerisen komponentin luonti

Nyt kun ReactJS-applikaatio on siistitty luon ensimmäisen react-komponentin. Komponentti on nappi, jota pystytään käyttämään muissa komponenteissa suorittamaan napille määritetyn funktion. Esimerkissä käytetään kahta nappi-komponenttia, jotka määrittävät niille yksilöllisesti liitetyn tekstimuuttujan näkyvyyden käyttöliittymässä. Kuviossa 5 on molemmat napit käyttöliittymässä, sekä yksi näkyväksi asetettu teksti.

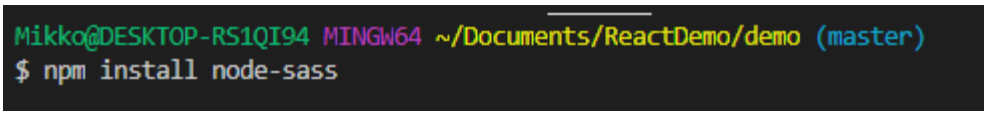


Kuvio 5. GenericButton-komponentti käyttöliittymässä

App tasolla on ”components”-kansion, johon tulee jokaiselle komponentille oma kansio. Komponentit kannattaa pakata omiin kansioihin, jotta rinnakkaistiedostot kuten funktiotiedostot tai tyylitiedostot voidaan pitää siististi yhdessä. Kansion nimi on ”GenericButton”, mutta itse JavaScript-tiedosto on ”index.js”. Tällä tavalla voi kutsua toisesta komponentista vain GenericButton-kansiota ja se automaattisesti palauttaa index.js -tiedoston.

Ulkoisia kirjastoja voidaan tuoda npm:n avulla ja Sass tyylikieli asennetaan ”npm install”-komennolla terminaalissa kuviossa 6. Sass tuo CSS-kieleen lisäominaisuuksia ja mahdollistaa Sass tyylitiedostojen, eli ”.scss”-päätteisen tiedostojen käytön (Sass Basics n.d.). Luon GenericButton-kansioon ”styles.scss” -tyylitiedoston.

Komponenttikohtaisen tyylitiedoston luominen ReactJS-projektissa välttää koodikonflikteja tiimikehityksessä. Myös jos tyyliluokitukset tehdään App.js -tiedoston tasolle, ne vaikuttavat kaikkiin App.js -tiedoston alle rakennettaviin komponentteihin React-applikaatiossa ja voivat tuottaa ei-toivottuja tyylimuutoksia komponentissa. App.js tasolle olisi suotavaa tehdä vain absoluuttisia CSS-sääntöjä, kuten kirjaisinlajimääritykset.



Kuvio 6. Sass-koodikirjaston asennus npm-komennolla terminaalissa

Jokaiseen JavaScript-tiedostotyypin react-komponenttiin tarvitsee tuoda erikseen React-kirjasto komponentin alussa. Myös kaikki muut komponenttiin tuotava data tuodaan komponentin alussa, kuten tyylitiedoston data. Komponenttiluokka sisältää ”render”-objektin, jonka sisällä oleva ”return” palauttaa web-sivuun halutun HTML-koodin JSX-muodossa. JSX-koodiin voidaan sijoittaa kaikki komponentin sisäiset muuttujat, komponenttiin valutetut ”prop”-omaisuudet tai ”import” -metodilla tuodut objektit. Jotta komponenttiluokkaa pystytään käyttämään toisessa komponentissa, se tarvitsee ”export”-metodin, joka tulee komponentin loppuun. Export-metodia voitaan käyttää myös yksittäisten objektien tai funktioiden vientiin react-komponenteille.

Geneerinen komponentti ”GenericButton” (liite 4) on nappi, jota pystytään kustomoida ”property”-muuttujilla laukaisemaan halutun funktion sekä muuttamaan napin väriä ja tekstiä. Komponentin ei tule sisältää mitään määriteltyjä arvoja, vaan toimia pelkkänä käytettävänä elementtinä, jotta siitä voidaan tehdä täysin geneerinen mihin vain tarkoitukseen. Geneerinen komponentti ei myöskään saisi sisältää JSX-koodissa muuttumattomia HTML-elementtien ”id”-kenttiä, jos

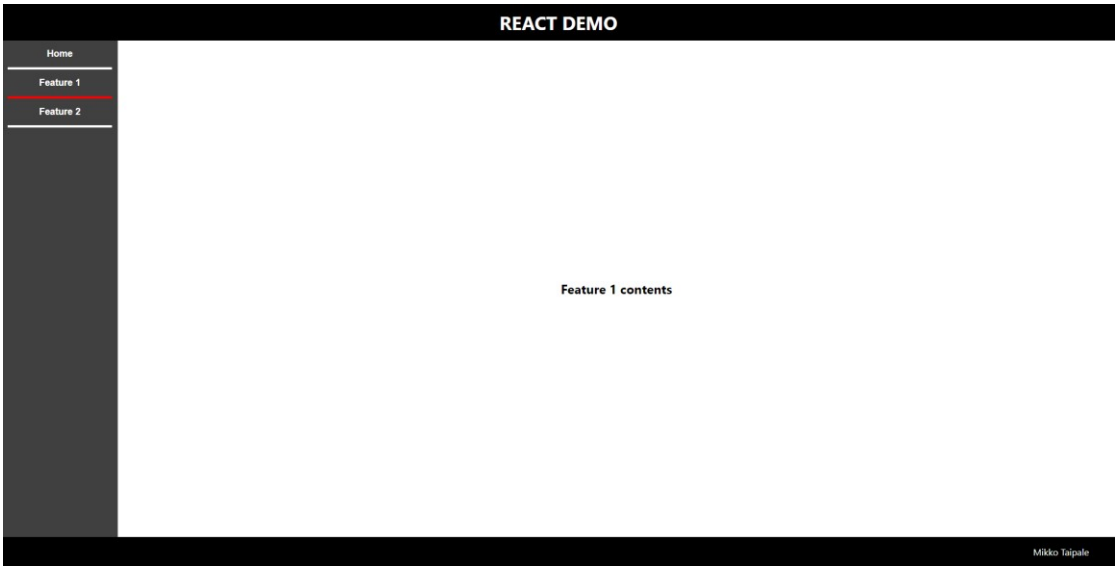
komponentteja voi olla käyttöliittymässä useampia samanaikaisesti. Tämä loisi ongelmallisia ”id”-duplikaatteja käyttöliittymään, joita ei saisi olla, sillä ”id”-kenttää käytetään yksittäisten HTML-elementtien identifioimiseen esimerkiksi automaatiotestauksessa.

Geneerinen komponentti voidaan nyt ottaa käyttöön toisessa komponentissa. Esimerkkinä toimii ”TextContainer”-komponentti (liite 5), joka toimii alustana geneerisen napin käyttöön. TextContainer sisältää GenericButton-komponentin ”return”-osiossa JSX koodin sisällä. GenericButton-komponentille annettavat ”property”-arvot, joiden avulla se määrittää itsensä, määritellään komponenttielementin sisään. GenericButton-komponentin ”OnClick”-funktio voidaan määrittää TextContainer-komponentissa liittämällä se funktioon, jonka nappi painettaessa laukaisee. Tässä esimerkissä nappi säätelee tekstimuttujan sisältöä, joka käyttöliittymässä heijastuu tekstin näkyvyyden asettamisella. Kummatkin GenericButton-komponentin onClick-funktiot ovat identtisiä, mutta aina lähettävät funktiolle ”toggleText” itselleen määritellyn arvon ”e”, jonka perusteella toggleText-funktio päättelee kumpaa nappia on painettu. GenericButton-nappien painamisella ei ole mitään vaikutusta toisiinsa, vaan toimivat täysin itsenäisesti.

Tekstiä sisältävät muuttujat ”blueText” ja ”redText” näkyvyyden tilaa hallitaan ReactJS-komponentin ”state”-muuttujilla. State-kenttään voidaan määritellä komponentille muuttujia, jotka eivät vaadi komponentin päivittymistä käyttöliittymässä, jolloin tekstin näkyvyyttä määrittävä state-muuttujan arvo pystyy muuttumaan dynaamisesti. Kaikki state-kenttään määritetyt muttujien arvot ovat väliaikaisia ja muuttuvat palaavat niiden oletusarvoihin, eli jos esimerkiksi sivu päivitetään, niin molemmat tekstin näkyvyyttä määrittelevät muuttujat palaavat arvoon ”false”.

5.3 Web-sivun luonti ReactJS-komponenteilla

Web-sivun komponentit rakennetaan App.js tasolle (liite 6). App.js -tiedostoa ja sen ylintä div-elementtiä on hyvä pitää ylimpänä tasona komponenttihierarkiassa ja käyttää sitä kehyksenä komponenteille, jotka sisältävät web-sivun rakentavat HTML-elementit. Aina web-sivulla näkyvät komponentit, kuten web-sivun header ja footer voidaan listata App.js tasolle. App.js -esimerkissä on vaik kolme komponenttia, jotka muodostavat sivun. Kaikki web-sivun muuttuva ja interaktiivinen sisältö tehdään ”PageContainer”-komponentin sisään. Web-sivun käyttöliittymä kuviossa 7.



Kuvio 7. ReactJS-esimerkki käyttöliittymässä

PageContainer-komponentti (liite 7) sisältää ”NavigationBar”-komponentin, joka on sivun vasemmassa laidassa sijaitseva navigointipalkki, sekä vaihtuvan sivun sisällön näyttävä ”PageContents”-komponentti. Vaikka näytettävä sivu valitaan NavigationBar-komponentissa, navigointipalkin muutokset kirjataan PageContainer-komponenttiin sen sijaan. Tällä tavalla näytettävän sivun tiedon siirtäminen PageContents-komponentille toimii sulavasti. State-muuttuja ”tab” vaihtuu NavigationBar-komponentin funktiolla ja muutos vaikuttaa Navigationbar- ja PageContents-komponenttiin samanaikaisesti. Tällä tavalla sivuvalinnan tieto pysyy

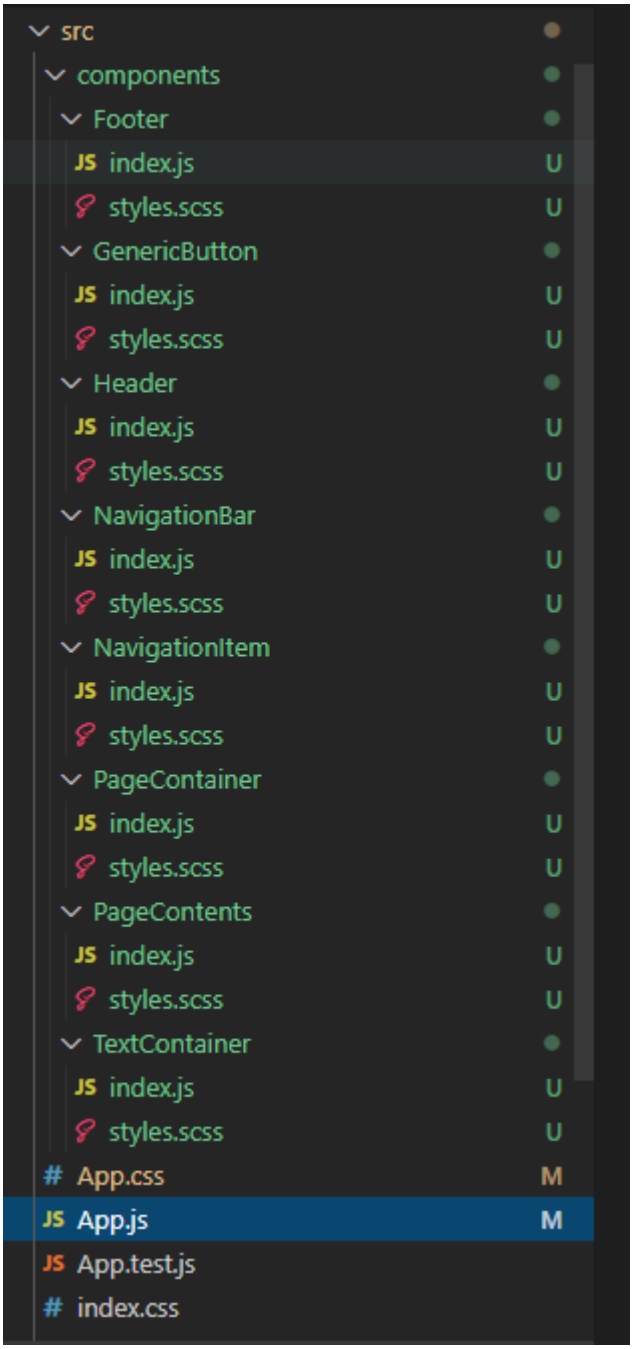
täysin synkronisoituna molemmilla komponenteilla, sillä tieto tulee samalta muuttujalta.

NavigationBar-komponenttiin (liite 8) tehdään ”NavigationItem”-komponentteja näytettävän sivun valintaan. NavigationBar pääasiassa toimii vain säiliönä NavigationItem-komponenteille. Tässä myös esimerkkinä miten ReactJS-property arvot voidaan valuttaa useamman komponentin läpi ja funktiot kuten PageContainer-komponentin ”onSelectTab” voidaan aktivoida useamman komponentin takaa. Tarpeeton propertyjen valuttaminen on kuitenkin ongelmallista isommissa applikaatio-toteutuksissa, sillä se tuottaa riippuvuuksia komponenttien välille. Komponettien riippuvuudet rajoittavat ReactJS-applikaation modulaarisuutta, joka on ReactJS-kirjastolla kehittämisen vahvuus. Jos tiettyä dataa tarvitsee kuljettaa usealle komponentille applikaatiossa, voidaan data säilyttää ”reducer”-tiedostoissa, joita voidaan hallita esimerkiksi Redux-kirjastolla. Vaihtoehtoisesti myös sessionStorage ja localStorage ovat paikkoja, joihon voidaan säilöä dataa ja hakea sitä komponenteille.

Propertyt päivittyvät välittömästi ja kun NavigationItem-komponenttia (liite 9) painetaan ja se lähettää valitun vaihtoehdon datan valitsemisfunktioon, se myös päivittää ”currentTab”-propertynsä heti sen muututtua. Tällä tavalla punaisella viivalla korostettu NavigationItem-vaihtoehto voidaan vaihtaa komponentin ulkopuolisen arvon perusteella ilman viivettä.

PageContents-komponenttiin (liite 10) listaataan kaikki tab vaihtoehdot. Komponentin ”render”-objektin sisällä on useampi ”return”-objekti. Render-objektissa voi olla vain yksi aktiivinen return-objekti. Sen sijaan, että käyttäisi sivuja komponentteina samassa JSX koodissa eri näkyvyysehtojen alla, return varmistaa että vain yksi sivu on voimassa. Jos ”if”-lausekkeen ehto toteutuu ja render palauttaa return-objektin, se myös jättää huomiotta kaikki sen jälkeen tulevat return-objektit, vaikka niiden ehdot toteutuisivat.

ReactJS-applikaation kansiorakenne on selkeä ja versionhallinnalle helppo seurata. Applikaation, sekä komponenttien kansiorakenne kuviossa 8.



Kuvio 8. Applikaation tiedostorakenne

6 Pohdinta

ReactJS-kirjastolla tehokkaasti kehittäminen vaatii web-kehittäjältä komponenttimallisen kokonaisuuden hahmottamisen. Komponenttien kierrättäminen ja rakennuspalikkamaisen käytön ymmärtäminen on tarpeellista, jotta kirjastosta saa maksimipotentiaalin irti. ReactJS vaatii kehittäjää rakentamaan web-sivun niin, että sen voi jakaa osiin.

Tiimityöskentelyssä ReactJS-kirjaston käyttö vaatii koordinaatiota ja suunnitelmallisuutta projektin rakenteen määrittelemisessä. ReactJS- projektin optimaalisen tehtävänjaon tulisi olla komponenttikeskeinen. Koska ReactJS ei pidä sisällään suoraan kaikki edellytyksiä nettisivun rakentamiseen, voi joillekin projekteille houkuttelevampi vaihtoehto sovelluskehykseksi olla esimerkiksi AngularJS.

Koska ReactJS on suosituin JavaScript-kirjasto, ei sillä kehittäminen ole iso riski. Myös Facebookin tuki tuo kirjaston tulevaisuudelle vakautta, sillä jatkokehittäminen ja ylläpitäminen on ison yhtiön takana. JavaScript on kielenä nopea ja skaalautuva, sekä on perustana usealle sovelluskehykselle (Kapoor 2019). JavaScriptin joustavuus tekee siitä jatkuvasti kehittyvän kielen ECMAScript-versioiden avulla, joten koodikielenä JavaScript ei ole helposti vanheneva kieli. JavaScriptin rooli web-kehityksessä ja web-kehityksen jatkuva nousu takaa työpaikkoja JavaScript-osaajille.

Lähteet

About npm. N.d. npm Documentation. Npmjs.com-verkkosivu. Viitattu 05.04.2020. https://docs.npmjs.com/about-npm/

Aderinokun I. 2018. What, exactly, is the DOM? Bitsofco.de-verkkosivu. Viitattu 10.04.2020. https://bitsofco.de/what-exactly-is-the-dom/

Domantas G. 2019. What is HTML? The Basics of Hypertext Markup Language Explained. Hostinger.com-verkkosivu. Viitattu 05.04.2020. https://www.hostinger.com/tutorials/what-is-html

Kapoor A. 2019. What is the future of JavaScript? Hackernoon.com-verkkosivu. Viitattu 03.05.2020. https://hackernoon.com/what-is-the-future-of-javascript-355b1f13b317

Li C. 2018. Why You Should Use ES6. Itnext.io-verkkosivu. Viitattu 13.04.2020. https://itnext.io/why-you-should-use-es6-56bd12f7ae09

Martin S. 2019. Angular vs React vs Vue: Which is the Best Choice for 2019? Hackernoon.com-verkkosivu. Viitattu 14.04.2020. https://hackernoon.com/angular-vs-react-vs-vue-which-is-the-best-choice-for-2019-16ce0deb3847

Morris S. 2019. TECH 101: WHAT IS REACT JS? Skillcrush.com-verkkosivu. Viitattu 05.04.2020. https://skillcrush.com/blog/what-is-react-js/

About Node.js®. N.d. OpenJS Foundation. Viitattu 13.04.2020. https://nodejs.org/en/about/

Sass Basics. N.d. Sass documentation. Sass-lang.com-verkkosivu. Viitattu: 27.4.2020. https://sass-lang.com/guide

Starikov M. 2019. Most Used JavaScript Frameworks for Quick Software Development: Which to Choose. Stfalcon.com-verkkosivu. Viitattu 14.04.2020. https://stfalcon.com/en/blog/post/javascript-frameworks-for-software-development

What is Babel? N.d. Babeljs.io-verkkosivu. Viitattu 10.04.2020. https://babeljs.io/docs/en/

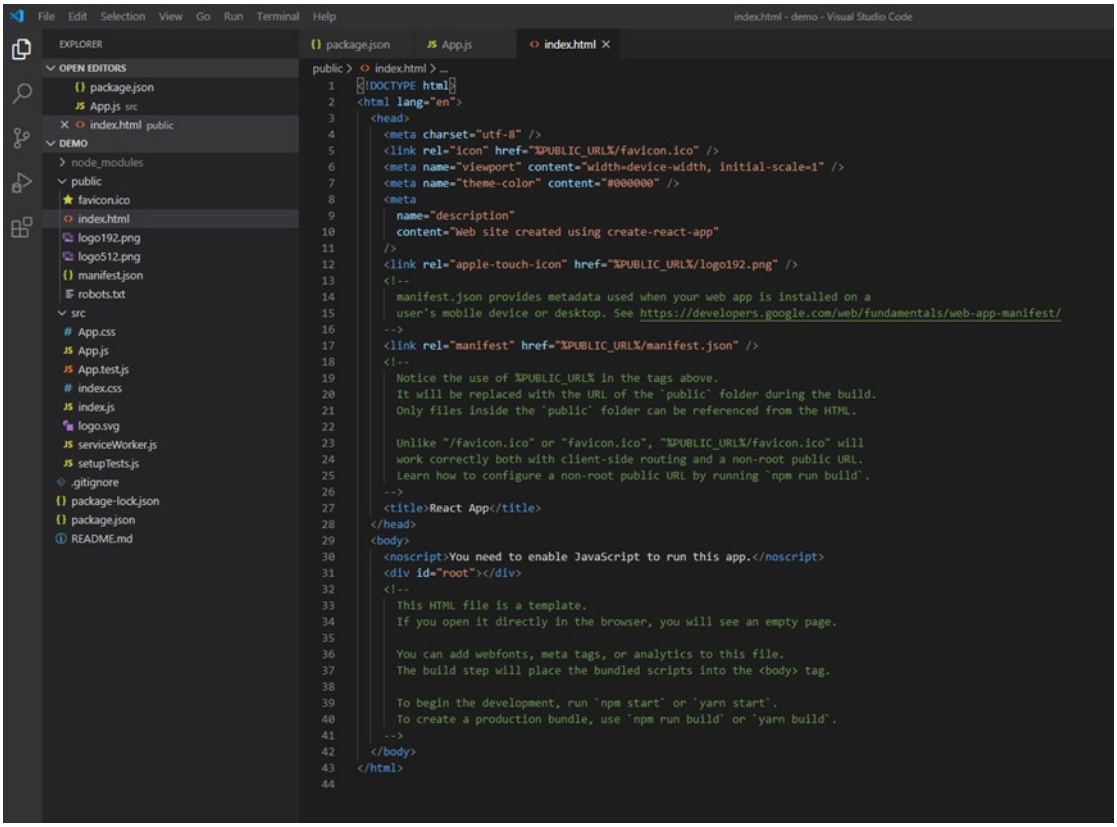
What is CSS? N.d. Tutorialspoint.com-verkkosivu. Viitattu 05.04.2020. https://www.tutorialspoint.com/css/what_is_css.htm

What is JavaScript? N.d. MDN web docs. Viitattu 05.04.2020. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

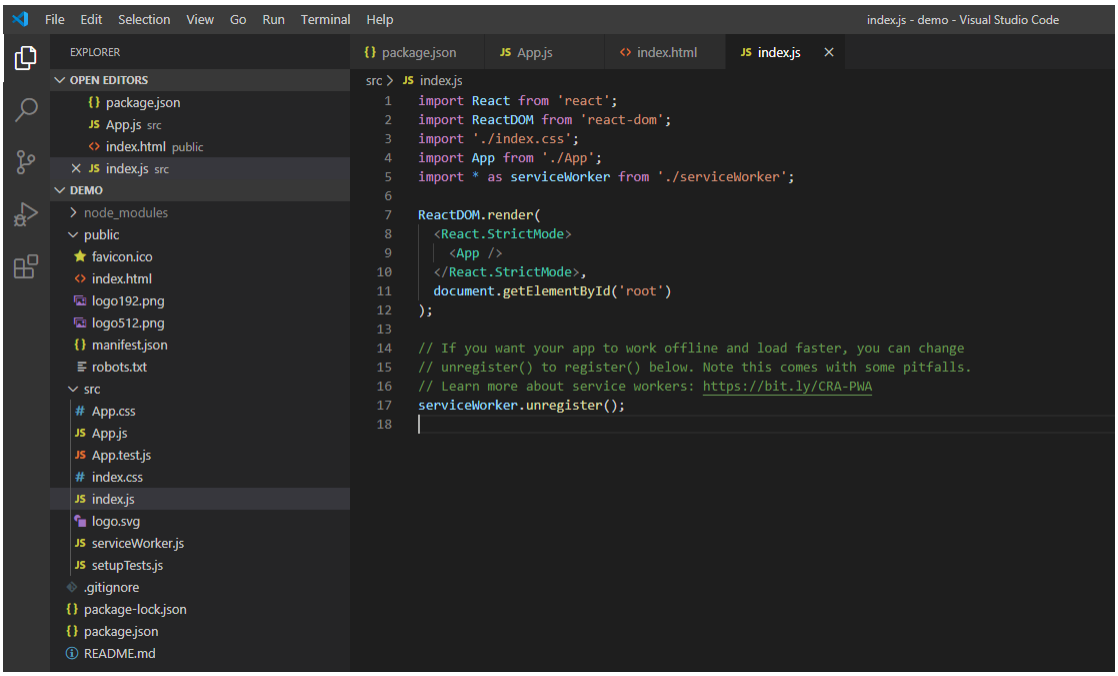
What Is JSX? N.d. Reactenlightenment.com-verkkosivu. Viitattu 10.04.2020. https://www.reactenlightenment.com/react-jsx/5.1.html

Liitteet

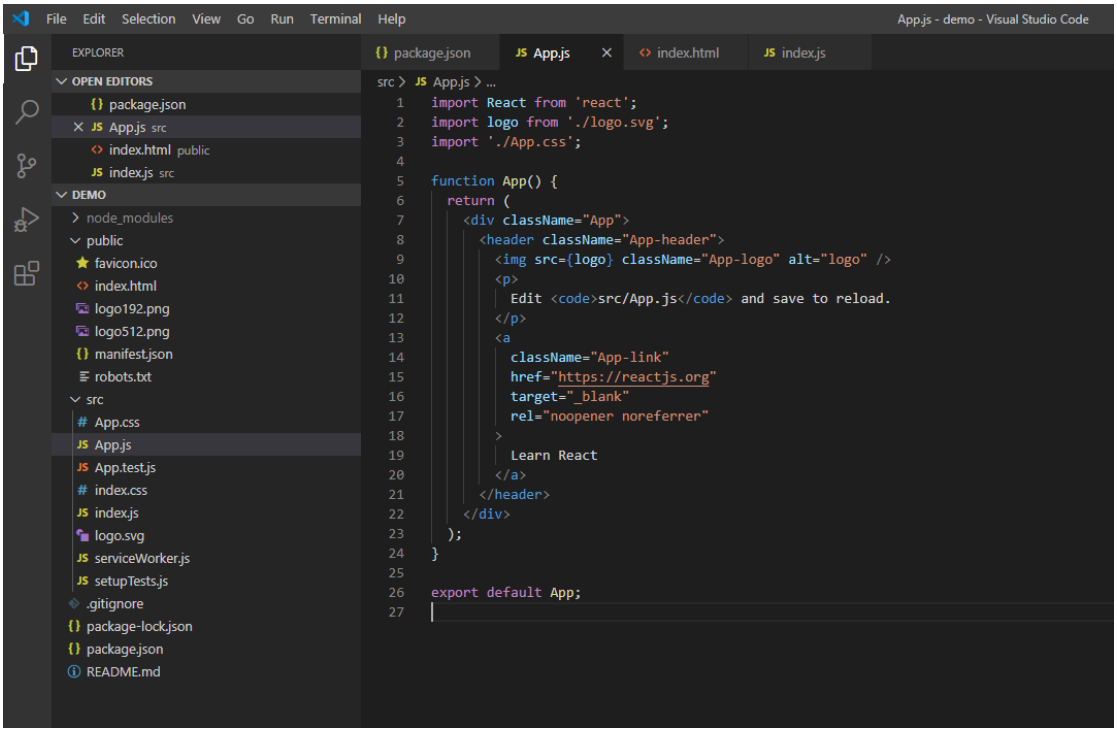
Liite 1. index.html-tiedosto



Liite 2. index.js-tiedosto



Liite 3. App.js-tiedosto



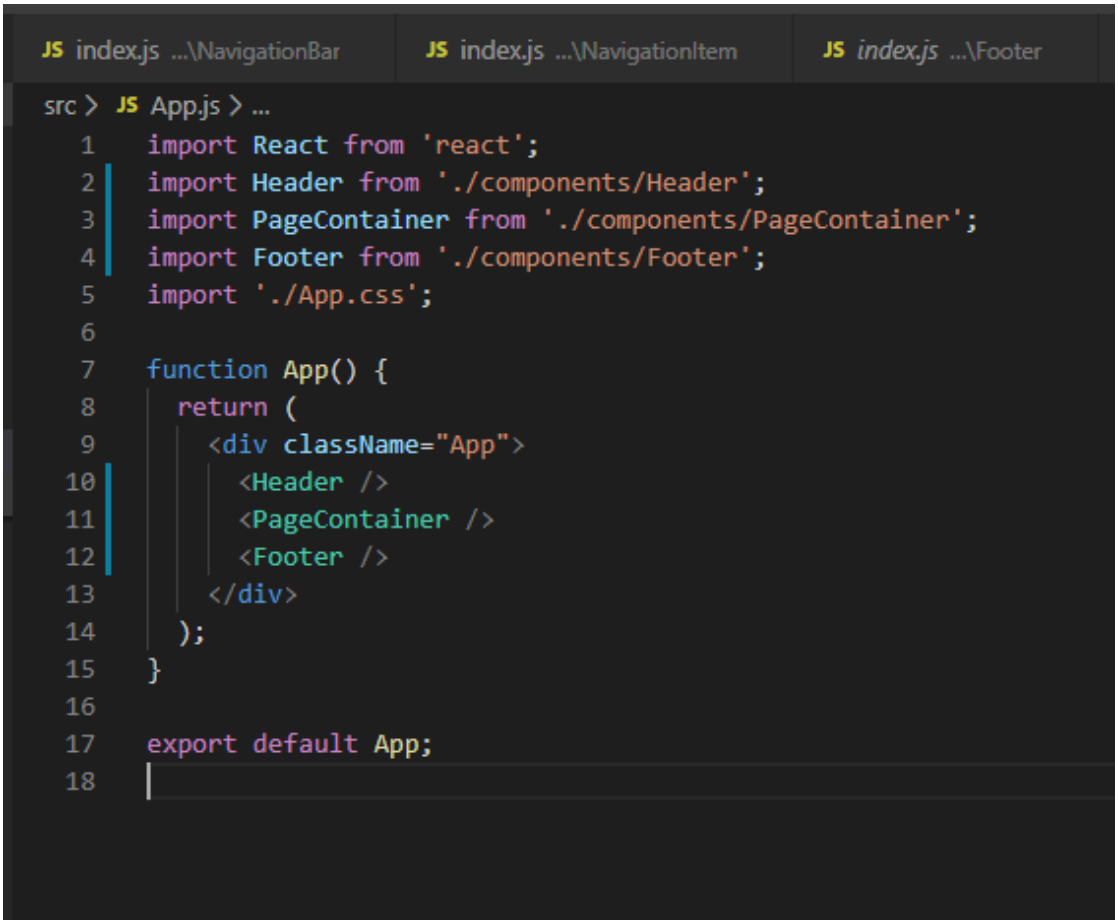
Liite 4. GenericButton-komponentti

```jsx
import React from 'react';
import PropTypes from 'prop-types';

require('./styles.scss');

class GenericButton extends React.Component {
    handleClick = event => {
        if (event) {
            this.props.onClick(this.props.color);
        }
    }

    render () {
        return (
            <div className="generic-button">
                <button className={`primary-button ${this.props.color}`} onClick={e => this.handleClick(e)}>
                    {this.props.text}
                </button>
            </div>
        );
    }
}

GenericButton.defaultProps = {
    text: '',
    color: '',
    onClick: undefined,
}

GenericButton.propTypes = {
    text: PropTypes.string,
    color: PropTypes.string,
    onClick: PropTypes.func,
}
export default GenericButton;
```

Liite 5. TextContainer-komponentti

```
class TextContainer extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            redVisibility: false,
            blueVisibility: false,
        }
    }

    toggleText = color => {
        if (color === 'red') {
            this.setState(prevState => ({ redVisibility: !prevState.redVisibility }));
        }
        if (color === 'blue') {
            this.setState(prevState => ({ blueVisibility: !prevState.blueVisibility }));
        }
    }

    render () {
        const redText = this.state.redVisibility ? 'This text is red' : '';
        const blueText = this.state.blueVisibility ? 'This text is blue' : '';
        return (
            <div className="text-container">
                <div className="text-red">
                    {redText}
                </div>
                <div className="text-blue">
                    {blueText}
                </div>
                <div className="left-button">
                    <GenericButton
                        text='Red toggle'
                        color='red'
                        onClick={e => this.toggleText(e)}
                    />
                </div>
                <div className="right-button">
                    <GenericButton
                        text='Blue toggle'
                        color='blue'
                        onClick={e => this.toggleText(e)}
                    />
                </div>
            </div>
        );
    }
```

Liite 6. App.js-tiedoston komponenttirakenne



Liite 7. PageContainer-komponentti

```
import React from 'react';
import NavigationBar from '../NavigationBar';
import PageContents from '../PageContents';

require('./styles.scss');

class PageContainer extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            tab: 'home',
        }
    }

    onSelectTab = tab => {
        if (tab) {
            this.setState({ tab });
        }
    }

    render () {
        return (
            <div className="page-container">
                <NavigationBar
                    currentTab={this.state.tab}
                    selectTab={e => this.onSelectTab(e)}
                />
                <PageContents
                    currentTab={this.state.tab}
                />
            </div>
        );
    }
}

export default PageContainer;
```

Liite 8. NavigationBar-komponentti

```
import React from 'react';
import PropTypes from 'prop-types';
import NavigationItem from '../NavigationItem';

require('./styles.scss');

class NavigationBar extends React.Component {
    render () {
        const { selectTab, currentTab } = this.props;
        return (
            <div className="navigation-bar">
                <NavigationItem
                    type='home'
                    onClick={selectTab}
                    currentTab={currentTab}
                />
                <NavigationItem
                    type='feature1'
                    onClick={selectTab}
                    currentTab={currentTab}
                />
                <NavigationItem
                    type='feature2'
                    onClick={selectTab}
                    currentTab={currentTab}
                />
            </div>
        );
    }
}

NavigationBar.defaultProps = {
    selectTab: undefined,
    currentTab: '',
}

NavigationBar.propTypes = {
    selectTab: PropTypes.func,
    currentTab: PropTypes.string,
}

export default NavigationBar;
```

Liite 9. NavigationItem-komponentti

```
class NavigationItem extends React.Component {
    handleClick = event => {
        if (event) {
            this.props.onClick(this.props.type);
        }
    }

    render () {
        const { currentTab, type } = this.props;

        let tabTitle = '';
        switch (this.props.type) {
            case 'home':
                tabTitle = 'Home';
                break;
            case 'feature1':
                tabTitle = 'Feature 1';
                break;
            case 'feature2':
                tabTitle = 'Feature 2';
                break;
            default:
                break;
        }

        const selected = currentTab === type ? 'highlighted' : '';

        return (
            <div className="navigation-item">
                <button className={`navigation-button ${selected}`} onClick={e => this.handleClick(e)}>
                    {tabTitle}
                </button>
            </div>
        );
    }
}

NavigationItem.defaultProps = {
    type: '',
    onClick: undefined,
    currentTab: '',
}

NavigationItem.propTypes = {
    type: PropTypes.string,
    onClick: PropTypes.func,
    currentTab: PropTypes.string,
}
```

Liite 10. PageContents-komponentti

```
class PageContents extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
        }
    }


    render () {
        const { currentTab } = this.props;

        if (currentTab === 'home') {
            return (
                <div className="page-contents">
                    <p>Home page contents</p>
                </div>
            );
        }

        if (currentTab === 'feature1') {
            return (
                <div className="page-contents">
                    <p>Feature 1 contents</p>
                </div>
            );
        }

        if (currentTab === 'feature2') {
            return (
                <div className="page-contents">
                    <p>Feature 2 contents</p>
                </div>
            );
        }
    }
}

PageContents.defaultProps = {
    currentTab: '',
}

PageContents.propTypes = {
    currentTab: PropTypes.string,
}

export default PageContents;
```