Haaga-Helia
ammattikorkeakoulu Oy

React Web-applikaation kehityksessä

Aleksi Pitkänen

Opinnäytetyö
Tietojenkäsittelyn
koulutusohjelma
2018

Haaga-Helia
ammattikorkeakoulu Oy

Tiivistelmä

Tekijä(t)
Aleksi Pitkänen

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
React Web-applikaation kehityksessä

Sivu- ja liitesivumäärä
33 + 2

Verkkoteknologiat ovat jatkuvan murroksen vallassa ja ripeä kehitys on synnyttänyt kovan tarpeen verkkokehitystöitä tehostaville työkaluille ja resursseille. Näiden joukossa yksi tämän hetken hehkuvimmista voimavaroista on resurssikokoelma nimeltä React. Teknologian pyrkimyksenä on tarjota verkkokehittäjälle tehokas luontiprosessi interaktiivisten käyttöliittymien rakentamiseen tehostaen työskentelyä perinteisiin kehitysmenetelmiin rinnastettuna.

Tässä opinnäytetyössä käsitellään React-kirjastoa vertailevasta tutkimusnäkökulmasta. Projektin tavoitteena on käytännöllinen perehtyminen työn aihealueeseen, tutustuminen sen tuoreimpiin työkäytäntöihin konkretiapohjalta sekä aiheeseen liittyvien kahden eri menettelytavan havainnollinen vertailu.

Vertailussa asetetaan konkreettisin esimerkein vastakkain React-kirjaston työmenetelmät ja perinteisemmät menettelytavat. Varsinaisen dokumentaation ohella luodaan alkeellinen verkkoapplikaatio, jonka perspektiivistä objektiivinen vertailu suoritetaan.

Tutkimuksen tuloksista rakennetaan yhteenveto, jossa luetellaan projektin aikana kertyneet konkreettiset havainnot ja tulokset. Näiden perusteella kootaan omia ajatuksia siitä, miten React poikkeaa perinteisitä kehitysmenetelmistä, mihin teknologia soveltuu sekä mitä hyötyjä ja haasteita kirjasto tuo verkkokehitystehtäviin. Lisäksi pohditaan vaihtoehtoisten ratkaisujen soveltamista.

Asiasanat
React, verkkokehitys, Web-applikaatio, käyttöliittymä

Sisällys

1 Johdanto

Nykypäivän verkko ja sen kehitysmenetelmät ovat muuttuneet. Mobiililaitteiden globaali räjähdysmäinen kasvu on luonut tarpeen saada verkkosivut toimimaan monenlaisilla eri laitteilla ja mukautumaan kaikenkokoisille ruuduille, jonka lisäksi vuorovaikutteiset verkkopalvelut kuten sosiaaliset mediat vaativat yhä kehittyneempiä järjestelmiä niiden sisällön hallinnointiin ja ripeämpiä kehitystyökaluja uusien ominaisuuksien rakentamiseen. Kokonaisuudessaan verkon kehitystyöt ovat siis monimutkaistuneet ja näiden haasteiden kohtaamana tehokkaammille kehitystyökaluille on syntynyt hanakka tarve.

Haasteeseen on vuosien mittaan haettu erilaisia ratkaisuja ja tuloksena on syntynyt lukuisa joukko erilaisia verkkokehitystehtävien aputyökaluja ja resursseja, joilla voidaan helpottaa sivustojen ja applikaatioiden rakennusprosessia. Jopa useat merkittävät tietotekniikan yritykset ovat tuoneet omat nappulansa mukaan peliin ja kehittäjäyhteisöjen tukemana rakentaneet omia avoimesti saatavilla olevia verkkokehityksen apuresursseja ja työkaluja. Eräs näistä voimavaroista on Facebookin kehittämä JavaScript -resurssikokoelma, eli kirjasto nimeltä React.

React -kirjasto tuo paljon helpottavia elementtejä JavaScript -pohjaisten käyttöliittymien & verkkoapplikaatioiden kehitykseen ja se on noussut tämän hetken suosituimmaksi JavaScript-kirjastoksi (Voss 2018). Kirjaston modulaaristen ohjelmointiaspektien ansiosta työskentely voi olla perinteisiin kehitysmenetelmiin verrattuna nopeampaa ja virtaviivaistetumpaa, mutta minkälaisiin käyttötapauksiin teknologia soveltuu ja mitä käytännön hyötyjä ja mahdollisia varjopuolia se tuo tavanomaisempaan JavaScript -pohjaiseen verkkokehitykseen verrattuna? Onko kyseessä vain nopean suosiovirtauksen aiheuttama buumi, joka on jo seuraavan muutaman vuoden sisällä siirtynyt historian kirjoihin ja korvautunut tuloksekkaammalla teknologialla?

Näitä asioita pohdittaessa on ensin syytä tutustua tarkemmin Reactin lähtökohtiin sekä sidosteknologioihin, joita käsitellään raportin tietoperustaosiossa eli luvussa kaksi. Kolmannessa luvussa käsitellään vertailevaa tutkimusta konkretiapohjalta ja lopuksi viimeisessä luvussa tehdään johtopäätökset tuloksista ja etsitään vastauksia edellä esitettyihin kysymyksiin.

Opinnäytetyön aihe on valittu omien kiinnostusten kohteiden lisäksi ensisijaisesti tukemaan omaa kehitystä. Tämän hetken trendinä verkkokehitysalan työmarkkinoilla tuntuu olevan JavaScript-kehyksien omaksuminen, ja React on sen nopean suosionkasvunsa vuoksi herättänyt oman mielenkiinnon teknologiaa kohtaan. Olen jo pitkään pohtinut

Reactiin tutustumista, mutta en tämänhetkisten verkkokehitystöideni puolesta ole saanut tähän tilaisuutta johtuen aihealueen ja työtehtävien poikkeavuuksista. Näin ollen opinnäytetyön valitseminen tältä saralta tuntui jouhevimmalta vaihtoehdolta tutustua teknologiaan sekä teorian, että konkretian nimissä.

1.1 Tehtävä

Tässä vertailevassa tutkimustyössä käsitellään React -kirjastoa sekä teorian että konkreettisen verkkoapplikaation rakentamisen pohjalta. Vertailtavina osapuolina ovat React -kirjasto sekä perinteinen JavaScript-ohjelmointikieli Web-applikaation kehittämisen perspektiivistä. Rakennettavan applikaation graafiset suunnitelmat löytyvät raportin liiteosiosta (liite 1).

Teoreettisessa osiossa (luku 2) esitellään React -kirjastoa yleisellä tasolla ja luodaan lukijalle kuva siitä, mitä teknologialla on tarkoitus saavuttaa. Havainnollisen applikaation rakennusprosessia (luku 3) tarkastellaan Reactin asennuksesta lähtien aina applikaation valmistumiseen saakka, jossa Reactin ja perinteisen JavaScript -kehityksen työmenetelmät asetetaan vastakkain konkreettisin esimerkein.

Tutkimuksella etsitään vastauksia siihen, mitä hyötyjä teknologia tuo Web-kehitys tehtäviin, miksi se on noussut nykyiseen suosioonsa ja miten teknologia eroaa perinteisestä JavaScript-kehityksestä työmäärän, koodisyntaksin ja suorituskyvyn puolesta.

Opinnäytetyön tuloksesta hyötyvät ensisijaisesti henkilöt, jotka ovat kiinnostuneita React -kehityksestä sekä yleisesti käyttöliittymien ja Web-applikaatioiden kehityksestä. Konkreettisen vertailututkimuksen tuloksista lukijat saavat merkittävää käytännöllistä tietoa Reactin ja perinteisen JavaScript-kehityksen eroista, josta myös mahdollisesti kokeneemmatkin alan kehittäjät voivat hyötyä.

Raportin empiirisen osion ymmärtämiseksi aiempi käyttökokemus Reactista ei ole välttämätöntä, mutta voi olla hyödyksi. Dokumentissa ei käydä läpi yleisen Web-kehityksen konsepteja, joten perustaitojen omaksuminen (html + CSS + JavaScript) on hyvin suositeltavaa, mikäli pyrkimyksenä on ymmärtää kunnolla dokumentaatiossa läpikäytävät aiheet. Mahdollinen aiempi kokemus muista JavaScript-kirjastoista tai -kehyksistä voi olla eduksi Reactin täyden potentiaalin hyödyntämiseksi, muttei myöskään tarpeellista.

1.2 Tavoitteet ja rajaukset

Raportin tavoitteena on luoda dokumentaatio, jolla kuvataan mahdollisimman kattavasti Reactia yleisellä tasolla, vertailla sen käyttöä Web-Applikaation kehityksessä suhteessa perinteiseen JavaScript-kehitykseen sekä laatia nykystandardipohjainen ohjeistava tietopaketti (empiirinen osio), jonka avulla lukija kykenee rakentamaan alkeellisen React-applikaation.

Projektissa syntyy dokumentaation lisäksi konkreettinen moderneihin standardeihin perustuva yksinkertainen Web-applikaatio, joka kirjoitetaan Reactin ja JavaScriptin vertailevasta toteutusperspektiivistä. Tuotetun applikaation agenda on valittu tukemaan raportin empiiristä osiota ja aihealueissa ei takerruta yksityiskohtiin, vaan keskitytään olennaisimpiin toimintoihin.

Raportin viimeisessä pohdiskelevassa luvussa kootaan ajatuksia työn herättämistä aiheista tutkimuksen kirjoittajan näkökulmasta ja esitellään kiteytetty yhteenveto tutkimuksen tuloksista. Lisäksi käydään läpi, mitä käytännön hyötyjä ja haasteita React tuo applikaatioiden kehitykseen, millaisissa tapauksissa teknologiaa kannattaa hyödyntää sekä pohditaan muiden vaihtoehtoisten ratkaisujen soveltamista.

Raportissa keskitytään ainoastaan Web-pohjaisiin applikaatioihin, joten Reactin natiiviapplikaatioihin rajoittuvaa React Native -kehystä ei raportissa käsitellä.

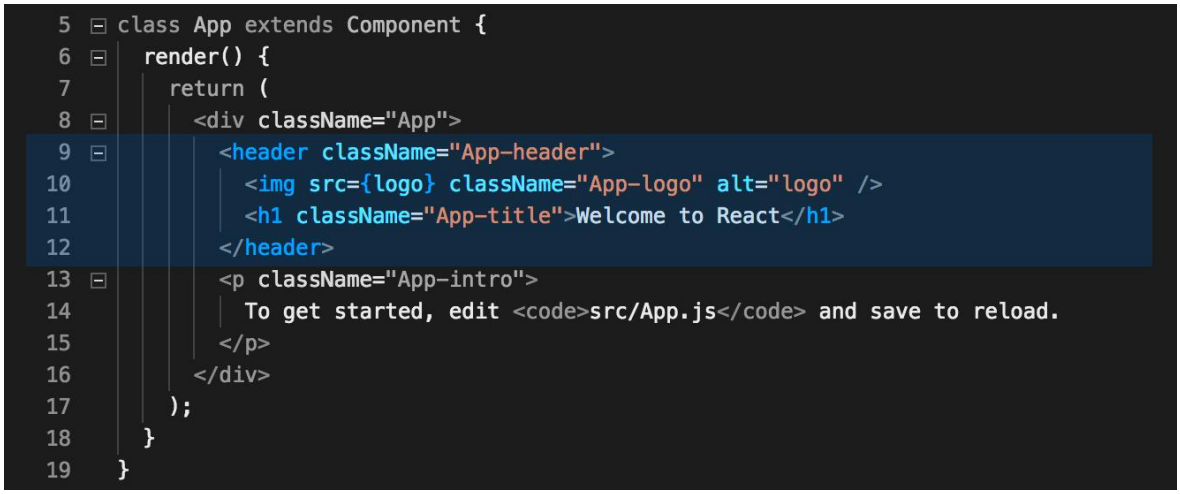
1.3 Tietoperusta

Tutkimuksen aihealueen kannalta keskeiset käsitteet ja teoreettinen yleistietoperusta pohjautuu aikaisempaan tietoon, joka koostuu ensisijaisesti verkkolähteistä, kirjallisuudesta, aikaisemmista tutkimuksista sekä aiheeseen liittyvistä artikkeleista. Kaikki käytetyt lähteet ovat lueteltuna raportin lopussa ja lähteisiin on viitattu Haaga-Helian asianmukaisilla viitemerkinnöillä.

Empiiristä vertailevaa tutkimustietoa käsitellään havaintopohjalta ja tulokset esitellään yhteenvetoina. Tuloksia analysoidaan empiirisessä luvussa konkreettisesti, joita sitten kiteytettynä abstraktina avataan pohdintaluvussa.

React -kirjastoon välittömästi liittyvän konkreettisen tiedon kuten koodikappaleiden käytössä hyödynnetään ensisijaisesti Reactin virallisia verkkosivuja ajantasaisimman ja luotettavimman tiedon takaamiseksi. Tätä raporttia kirjoittaessa Reactin uusin versio on 18. syyskuuta 2018 julkaistu 16.5.2.

Raportin esimerkkikoodikappaleet esitetään kuvakaappauksina Visual Studio Code -ohjelmasta tai komentorivistä ja mikäli johonkin tiettyyn koodipätkään halutaan kiinnittää lukijan huomio, niin kyseiset koodirivit ovat korostettuja sinisellä taustavärillä seuraavasti (kuva 1):



Kuva 1. Havainnollistava kuva koodiesimerkistä sekä korostetusta koodikappaleesta.

1.4 Käsitteet

Tässä kappaleessa avataan raportissa esiintyviä käsitteitä lyhyesti. Opinnäytetyön aihealueen termistö on valtaosaksi englanninkielisiä, joille ei välttämättä ole suoria suomenkielisiä ilmaisuja. Tästä johtuen raportissa on suomalaisesta raporttikielestä huolimatta käytetty paljon englanninkielisiä termejä.

HTML	HTML (lyhenne sanoista Hypertext Markup Language, suomennettuna hypertekstin merkintäkieli) on avoimesti standardoitu kuvauskieli, jolla voidaan kuvata hyperlinkkejä sisältävää tekstiä eli hypertekstiä.
CSS	CSS (Cascading Style Sheets) tarkoittaa tyylikielitekniikkaa, jolla määritellään ulkoasu HTML-kielellä kuvatulle rakenteelle.
JavaScript	JavaScript on ohjelmointikieli, jolla voidaan lisätä Web-sivuille dynaamista toiminnallisuutta, kuten sisällön muokkaus.
ECMAscript	Ohjelmointikielispesifikaatio, jolla JavaScript on standardisoitu. Pyrkimyksenä on edistää kielen toteutusta.
JSX	Syntaksilaajennus, joka mahdollistaa HTML-tyyppisen tekstirakenteen kirjoittamisen suoraan JavaScript-koodiin.

Kirjasto	Tietotekniikassa kirjastolla tarkoitetaan resurssikokoelmaa, joka voi sisältää esim. valmiiksi kirjoitettua koodia, luokkia, arvoja.
DOM	Dokumenttioliomalli, eli DOM (lyhenne sanoista Document Object Model) on tapa kuvata dokumentin, kuten HTML:n, rakenne puuna, jonka eri olioita voidaan hakea, tutkia ja manipuloida esim. JavaScriptin avulla.
MVC	Model-view-controler eli MVC on ohjelmistoarkkitehtuuri, jonka tarkoituksena on käyttöliittymän jakaminen kolmeen osaan: malli (model) kuvaa tiedon tallentamisen, näkymä (view) määrittää ulkoasun ja käsittelijä (controller) vastaanottaa käyttäjän käskyt ja muuttaa mallia sen mukaan.
Ohjelmistokehys	Ohjelmistotuote, joka muodostaa rungon sen päälle rakennettavalle tietokoneohjelmalle, jonka avulla nopeutetaan uusien ohjelmistotuotteiden valmistusta.
Käyttöliittymä	Tietotekniikassa käyttöliittymällä tarkoitetaan jonkin ohjelman osaa, jonka käyttäjä näkee tietokoneen näytöllä.
Responsiivisuus	Mukautuvuus ruutukoon mukaan
Git	Versionhallintajärjestelmä, jolla hallitaan ohjelmistoprojektin tuotoksia ja seurataan sen kehitystä.

2 React

Luvussa tarkastellaan Reactia yleisellä tasolla sekä sen teknistä rakennetta. Pyrkimyksenä on muodostaa yleiskuva Reactista sekä kirjastoon sidoksissa olevista teknologioista.

2.1 React lyhyesti

React on Facebookin kehittämä avoimen lähdekoodin JavaScript-kirjasto interaktiivisten käyttöliittymien kehitykseen (ReactJS 2018). Teknologian pyrkimyksenä on minimoida raskaan DOM-manipulaation operointia antamalla tekijälle käännettäväksi varsinaisen DOMin sijaan virtuaalisen DOM-kokonaisuuden, jolla vältetään toistavan koodin kirjoittamista (Sonpatki & A.M 2016).

Kirjasto perustuu modulaarisiin komponentteihin, jotka voivat vaihtaa niiden tietoa ilman peräkkäisiä sivun uudelleenlatauksia. React pyrkii tarjoamaan paremman käyttökokemuksen ja lujatekoisen kehitysalustan tuomalla reaktiivisen ohjelmoinnin aspekteja Web-applikaatioihin ja sivustoihin. MVC-ohjelmistoarkkitehtuurimallin mukaan sitä käytetään näkymäosan (view) määrittämiseen eli käyttöliittymän ulkoasun ja tietojen näytön esitykseen. (Aggarwal 2018, 133.)

React-kirjastoa ylläpitää Facebookin lisäksi laaja yksityiskehittäjä- ja organisaatioyhteisö, jonka ansiosta kirjasto on kehittynyt hurjaa tahtia. Kirjaston julkaistuttua vuonna 2013 sen suosio on kasvanut räjähdysmäisesti ja on noussut tämän hetken suosituimmaksi JavaScript-kirjastoksi (Voss 2018). Sadat sivustot ja toimistot hyödyntävät kirjastoa (github 2018), joista tunnetuimpia lienevät Facebook, Instagram, Pinterest, Skype ja Tesla.

React on tehokas työkalu monenlaisiin erilaisiin käyttötarkoituksiin, mutta sitä ei kuitenkaan tulisi käyttää kaikkeen. Argyle (2015) toteaa, että kirjasto soveltuu parhaiten applikaatioihin, joissa on runsaasti dynaamista, uudelleenlatautuvaa sisältöä ja kevyempien sovellusten kohdalla hyödyt jäävät melko vähäisiksi, eritoten suorituskyvyn kannalta.

2.2 Rakenne

Luvussa tarkastellaan Reactin teknistä rakennetta pintapuolisesti ja tutustutaan sen kolmeen tärkeimpään ominaispiirteeseen: komponentti, ominaisuudet sekä tilat. Koodiesimerkeissä hyödynnetään JavaScriptin syntaksilaajennusta nimeltä JSX, joka on Reactin virallisen dokumentaation (2018) suosittelema merkintätapa React-käyttöliittymien rakentamiseen.

2.2.1 Komponentit

React-komponentit mahdollistavat käyttöliittymän jakamisen omiin itsenäisiin, uudelleenkäytettäviin osiin. Jokainen komponentti voi sisältää muita alikomponentteja, joista voidaan muodostaa monimutkaisia, mutta organisoituja komponenttien puurakennelmia (ReactJS 2018).

React-komponentteja on kahta erilaista tyyppiä: funktionaaliset komponentit ja luokkakomponentit. Kenties yksinkertaisin tapa luoda komponentti on kirjoittaa funktiokomponentti. Seuraavassa esimerkissä (kuva 2) on esitettynä hyvin alkeellinen komponentti:

```
4
5   function Tervehdys(props) {
6     return <h1>Tervetuloa, {props.name}</h1>;
7   }
8
```

Kuva 2. Esimerkki funktionaalisesta React-komponentista

Ylläolevassa ”Tervehdys”-nimisessä komponentissa otetaan vastaan ”props” (eli ominaisuus) -olioargumentti datan kera ja palautetaan React-elementti. Komponenttia kutsuttaessa esim. argumentilla ”Matti” se siis palauttaisi h1-elementin merkkijonolla ”Tervetuloa, Matti”.

Tämän esimerkin kaltaisia komponentteja kutsutaan Reactissa funktionaalisiksi komponenteiksi, koska ne ovat kirjaimellisesti JavaScript-funktioita (ReactJS 2018). Tästä johtuen näiden kanssa ei voi käyttää luokkakomponenttien lisäominaisuuksia, kuten tilat (states) ja elinkaarikoukut (lifecycle hooks). Funktiokomponenteilla on kuitenkin etunsa huolimatta niiden puutteellisista ominaisuuksista. Kuten Jöch (2018) ilmaisee, niiden yksinkertaisuutensa vuoksi niitä on helpompi lukea ja testata, ne vaativat vähemmän koodia ja niillä saattaa ilmaantua suorituskykyparannuksia tulevissa React-versioissa. Funktiokomponentteja kannattaa siis lähes aina käyttää tilanteissa, joissa tiloille tai elinkaarillekoukuille ei ole tarvetta.

Kuten mainittu, komponentteja voidaan myös määritellä luokkakomponentteina, eli ECMAScript6 (ES6) -luokkina. Luokilla on funktiokomponentteihin verrattuna useita lisäominaisuuksia, joihin tutustutaan tarkemmin raportin empiirisessä osiossa. Ohessa esimerkki React-luokkakomponentista (kuva 3):

```
14
15  class Tervehdys extends React.Component {
16    render() {
17      return <h1>Tervetuloa, {this.props.name}</h1>;
18    }
19  }
20
```

Kuva 3. Esimerkki React-luokkakomponentista

Viimeiset kaksi komponenttiesimerkkiä tulostavat saman arvon. React tulkitsee nämä samanarvoisiksi ja tässä tapauksessa komponenttien erona on lähinnä syntaksi. Funktiokomponentti on tavallinen JavaScript -funktio, joka ottaa vastaan propsin argumentiksi ja palauttaa React-elementin. Luokkakomponentti taas vaatii erikseen sen laajentamista React-komponentiksi (extends React.Component) sekä luomaan ”render” -funktion, joka palauttaa React-elementin.

2.2.2 Ominaisuudet

Komponentin ominaisuudet huolehtivat komponentin rakenneasetuksista. Jokaisella React-komponentilla on käytännössä oma uniikki ominaisuus (tunnetaan Reactissa termillä ”props” eli properties), joka alkaa tyhjänä JavaScript-oliona. Nämä tyhjät oliot voidaan isäntäkomponentin puolesta täyttää jollain JavaScript-arvolla, jota komponentti voi käyttää tai siirtää alikomponentille (reactenlightenment 2018).

Oheisessa kuvassa (kuva 4) on havainnollistettuna aiemmin esitelty funktiokomponentti muutamalla lisäpropsilla; ”age” ja ”location”.

```
5
6   function Tervehdys(props) {
7     return <div>
8         <h1>Tervetuloa, {props.name}</h1>
9         <p>Olet {props.age} vuotta vanha.</p>
10        <p>Asut paikkakunnalla {props.location}</p>
11      </div>;
12  }
13
```

Kuva 4. Aiemmin esitelty funktiokomponentti kahdella lisäominaisuudella

2.2.3 Tilat

React-komponentin tilaa (state) käytetään tallentamaan tietoa, joka voi muuttua ajan myötä. Tyypillisesti muutos tulee käyttäjätapahtuman tai järjestelmätapahtuman yhteydessä, kuten esim. käyttäjän interaktion vastauksena, palvelinpyyntönä tai kellonajan päivityksenä (reactenlightenment 2018).

Sekä ominaisuudet että tilat ovat molemmat tavallisia JavaScript-olioita. Vaikka molemmat manipuloivat React-renderin ulostuloa, ne eroavat yhdellä tärkeällä tavalla: ominaisuudet **siirtyvät** komponentille (kuten JavaScriptissä funktioparametrit), kun taas tiloja hallinnoidaan itse komponentin **sisällä** (kuten funktion sisällä määritetyt muuttujat) (ReactJS 2018).

Seuraavassa esimerkissä (kuva 5) demonstroidaan tilojen käyttöä hyvin alkeellisella tasolla. Koodipätkässä käytetään ECMAScript6 (ES6) -mukaista koodisyntaksia.

```
35
36  class Tervehdys extends React.Component {
37    constructor() {
38      this.state = "Tervetuloa, Matti";
39    }
40    muutos() {
41      this.setState("Tervetuloa, Aleksi")
42    }
43  }
44
```

Kuva 5. Esimerkki yksinkertaisen tilan käytöstä React-komponentissa

Esimerkissä alustetaan Tervehdys -komponentti tilalla, joka sisältää merkkijonon ”Tervetuloa, Matti”. Komponentin tila voidaan muuttaa kutsumalla ”muutos” -metodia, joka ”setState” -metodilla asettaa komponentin uuteen tilaan, eli ”Tervetuloa, Aleksi”.

Tilojen hallintaan voidaan myös käyttää erillistä tilanhallintakirjastoa kuten Redux, Flux tai Mobx, mutta Reactia aluksi opeteltaessa on ensin hyvä tutustua Reactin perinteisiin tiloihin ennen lisäkirjastojen soveltamista.

2.3 Sidosteknologiat

Luvussa tarkastellaan Reactiin sidoksissa olevia teknologioita ja resursseja.

2.3.1 JavaScript

JavaScript (lyhyesti JS) on järjestelmäriippumaton, olioperustainen ohjelmointikieli, jonka tavoitteena on tehdä verkkosivut interaktiivisiksi, kuten animoinnilla, klikattavilla painikkeilla tai ponnahdusikkunoilla (MDN web docs 2018).

React on JavaScript-pohjainen kirjasto ja luonnollisesti sen myötä React-applikaatiot myös kirjoitetaan JS-ohjelmointikielellä. Reactin yhteydessä on myös mahdollista

hyödyntää JavaScriptin uusinta ES6-syntaksispesifikaatiota, joka tuo huomattavia etuja mm. tarvittavan koodimäärän suhteen.

Tämän ohella on myös hyvin yleistä ja jopa suositeltua (ReactJS 2018) hyödyntää JavaScriptin JSX-laajennusta React-applikaatioita rakennettaessa, jotta saavutetaan kirjaston täydet hyödyt.

2.3.2 Node.js ja npm

Node.js on avoimen lähdekoodin JavaScript-pohjainen palvelinympäristö. Kuten W3Schools (2018) verkkosivustollaan ilmaisee, Node.js:llä voidaan:

- tuottaa dynaamista sivustosisältöä
- luoda, avata, lukea, kirjoittaa, poistaa ja sulkea tiedostoja palvelimella
- kerätä lomaketietoja
- lisätä, poistaa ja muokata dataa tietokannassa

Node.js sisältää kaiken tarpeellisen JavaScript-pohjaisten applikaatioiden suorittamiseen, johon teknologia käyttää tapahtumapohjaista, estotonta I/O-siirräntämallia tehden siitä kevyen ja tehokkaan kokonaisuuden (Patel 2018).

Node.js:n mukana tulee pakettihallintatyökalu nimeltä **npm** (Node Package Manager). Paketit ovat koodin rakennuspalikoita, joilla JavaScript-kehittäjät voivat jakaa ja kierrättää koodipätkiä npm:n kautta. Npm-rekisteri sisältää laajan valikoiman kehittäjäyhteisön rakentamia paketteja monenlaisiin ohjelmointiympäristöihin, joka tekee Node.js-applikaatioiden rakennuksesta nopeaa ja helppoa (Nodejs 2018).

Node.js tai npm ei ole puhtaalta pöydältä Reactin toimivuuden kannalta pakollinen, mutta mikäli aikomuksena on hyödyntää kehitystöitä helpottavia paketteja – joita tässäkin opinnäytetyössä sovelletaan – on näiden asennus tarpeellista.

Node.js ja React ovat osa suosittua JavaScript Web-applikaation kehityspinoa nimeltä **MERN** (MongoDB, Express, React, Node.js), jolla on tarkoitus suorittaa sekä applikaation logiikka että pääsy tietokantaan. Nipussa MongoDB toimii tietokantana, Express huolehtii reitityksestä, React säätää käyttöliittymän ja Node.js operoi palvelimena. MERN:in kaltaisen standardi-applikaationipun käyttö helpottaa ja nopeuttaa uusien kehittäjien tuontia projekteihin ja saamaan heidät samaan vauhtiin, koska on hyvä mahdollisuus, että he ovat käyttäneet samaa teknologiaa aikaisemminkin (Morgan 2017).

2.3.3 Redux

Redux on ennustettava tilasäiliö JavaScript-applikaatioille, jota voidaan käyttää myös Reactin kanssa. Se auttaa kirjoittamaan applikaatioita, jotka käyttäytyvät johdonmukaisesti, ajautuvat eri ympäristöissä (asiakasohjelma, palvelin & natiivi) ja ovat helposti testattavia. Sen lisäksi Redux tarjoaa mainion kehityskokemuksen aputyökalujen myötä, kuten suora koodin muokkaus yhdistettynä aikamatkustus-debuggerilla (virheidenjäljittäjä). (github, Redux 2018)

Reduxilla on kolme perustavanlaatuista toimintaperiaatetta, jotka Reduxin virallisilla sivuilla (2018) luetellaan seuraaviksi:

- 1. Yksittäinen totuudenlähde – koko rakennettavan applikaation tila on varastoitu oliopuurakenteeseen yhden varaston sisälle
- 2. Tila on read-only (vain luku) – ainoa tapa vaihtaa tilaa on kutsua ”action” -olio (toiminta), joka kuvaa, mitä tapahtui
- 3. Muutokset tehdään puhtailla funktioilla – tilan puun muuttaminen toimintojen avulla määritellään kirjoittamalla puhtaita ”reducereita” (vähentäjiä).

Redux on melko yleisesti Reactin yhteydessä käytetty resurssi, mutta tämän opinnäytetyön puitteissa ei teknologiaa hyödynnetä applikaation pienen kokonsa vuoksi. Raportin puitteissa ei haluta poiketa liikaa olennaisesta, eikä Reduxia yleisestikään suositella (Abramov 2016) Reactin alkuopetteluvaiheessa. Teknologian käyttöä on syytä kuitenkin harkita suurempien applikaatioiden kohdalla.

2.3.4 Express

Express (tunnetaan myös nimellä Express.js) on minimaalinen ja joustava Node.js-pohjaisten Web-applikaatioiden kehys, joka tuo vantteran joukon eri ominaisuuksia Web- ja mobiiliapplikaatioille (Express 2018). Express tehostaa applikaation reititystä sekä koodin organisointia ja Serby (2012) kuvaa teknologiaa ”De facto” -standardi-verkkokehykseksi Node.js:lle.

Expressin GitHub-sivuilla (2018) luetellaan kehyksen ominaisuudet seuraavasti:

- lujatekoinen reititys
- fokus korkeassa suorituskyvyssä
- superkorkea testikattavuus
- http-avustajat (uudelleenohjaus, välimuistitallennus)
- näkymäjärjestelmätuki 14+ mallimoottorille

- sisältöneuvottelu
- käyttökelpoinen nopeasti tuotetuille applikaatioille

2.3.5 MongoDB

MongoDB on avoimen lähdekoodin NoSQL-tietokanta, eli ”ei-relaatiotietokanta”. Tietokannan tiedot tallennetaan ns. dokumentteihin, joka tarjoaa sitkeyttä applikaation datalle sekä yhteyttää avainarvo-varaston (nopea & skaalautuva) ja relaatiotietokannan (toimintorikas) välisen aukon (Byers 2018).

MongoDB on tarkoitettu ensisijaisesti isoihin kokonaisuuksiin, joissa saatavuus, skaalautuvuus ja joustavuus ovat applikaation tärkeimmät aspektit. Perinteisempään relaatiotietokantaan poiketen MongoDB tallettaa datan tietokantaan BSON-dokumentteina, joka on JSON-dokumenttien (JavaScript Object Notation) binäärinen edustaja (MongoDB Documentation 2018).

3 React -applikaation rakennus

Luvussa tutustutaan empiirisesti React-applikaation kehittämiseen. Rakennusprosessia tutkaillaan vertailevasta toteutusperspektiivistä, jossa asetetaan vastakkain React-työmenetelmät ja perinteiset JavaScript -menettelytavat. Tavoitteena on saada kuvattua konkreettisesti, miten teknologiat eroavat toisistaan työskentelytapojen, koodisyntaksin ja koodimäärän suhteen. Vertailut kuvataan konkreettisina koodiesimerkkeinä, jossa erot esitetään vierekkäisinä kuvakaappauksina.

Agendassa edetään johdonmukaisessa järjestyksessä käyden läpi Reactin perustoiminnallisuudet. Tämä alkaa aihealueen lähtökodista, joka koostuu tarvittavien resurssien asennuksesta ynnä muista vaatimuksista. Tämän jälkeen siirrytään käytännöllisen ohjelmoinnin pariin, jossa aloitetaan hyvin yksinkertaisista aihealueen konsepteista ja vähitellen hivuttautuen kehittyneempien ominaisuuksien pariin. Peruskäsitteiden jälkeen tutustutaan applikaation testausprosessiin sekä mahdollisiin jatkotoimenpiteisiin. Tutkimuksen tuloksia käsitellään lopuksi yhteenveto-luvussa, jossa kiteytetään applikaation rakennusprosessin aikana kertyneet konkreettiset havainnot

Luvussa rakennetaan yksinkertainen applikaatio nimeltä ”Muistiinpano-applikaatio”, jolla nimensä mukaisesti voidaan luoda omia muistiinpanoja. Muistiinpanot talletetaan selaimen lokaaliin säiliöön (local sorage), jolloin kirjoitetut muistiinpanot säilyvät myös sivun uudelleenlatauksen yhteydessä. Lokaalisäiliön etuna on sen yksinkertaisuus, jolloin erillisille palvelimille, ohjelmille tai kirjastoille ei ole tarvetta. Tämä tukee raportin agendaa, jossa tarkoitus on demonstroida ja vertailla Reactin yksinkertaisia konsepteja.

Rakennettavan applikaation ulkoasusuunnitelmat löytyvät opinnäytetyön liiteosiosta (liite 1).

3.1 Lähtökohdat

Tämän raportin puitteissa hyödynnetään **create-react-app** -nimistä työkalua, joka on kenties yksinkertaisin tapa aloittaa React-applikaatioiden luonti. Create-react-app vaatii toimiakseen Node.js -palvelinympäristön asennuksen (+ sen myötä asentuvan NPM-pakettihallinnoijan), joka on ladattavissa osoitteesta https://nodejs.org/.

Virallisen dokumentaation mukaan (2018) React on versiosta 16.0 lähtien hyödyntänyt kokoelmatyyppejä, joita vanhemmissa verkkoselaimissa (kuten Internet Explorer 10 ja vanhemmat versiot) ei tueta. Tässä työssä käytetään Reactin uusinta 23. toukokuuta 2018

julkaistua versiota 16.6.0 ja verkkoselaimena käytetään Google Chromea, jonka uusin versio tätä raporttia kirjoittaessa on 70.0.3538.

3.2 Asennus

Kuten mainittu, tutkimuksessa hyödynnetään Create-react-app -työkalua. Välinettä ei tarvitse asentaa erikseen, mikäli käytössä on npm:n versio 5.3 tai uudempi, jolloin npm:n mukana asentuu komento npx. Komennon avulla voidaan asentaa väliaikainen create-react-app ja kutsua se ilman globaalia asennusta tai ylimääräisiä komentoja.

Asennusprosessi suoritetaan komentorivin kautta (Windows-käyttöjärjestelmällä cmd tai Mac OS:llä terminal). Komentorivillä siirrytään applikaation halutun asennuskansio alle ja suoritetaan järjestyksessä seuraavat kolme komentoa:

- npx create-react-app muistiinpano-applikaatio
- cd muistiinpano-applikaatio
- npm start

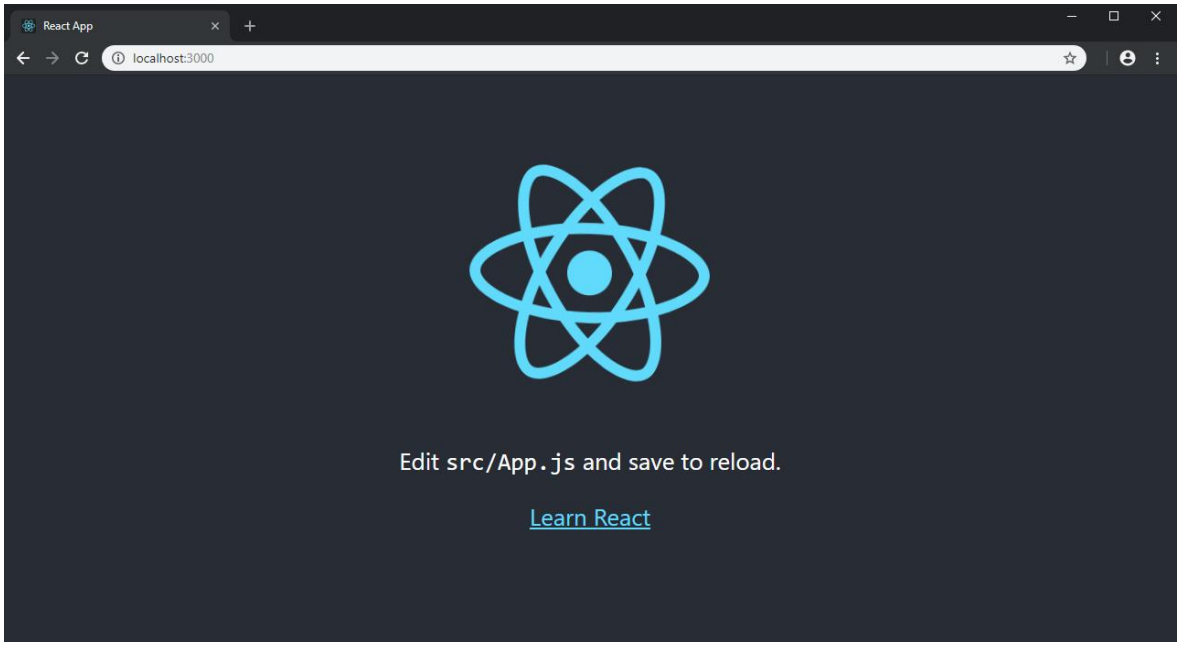
Ensimmäisellä npm-komennolla asennetaan create-react-app, jolle annetaan nimi ”muistiinpano-applikaatio” ja komento luo applikaatiolle oman projektihakemiston. Npm aloittaa applikaation asennusprosessin, jonka aikana komentorivillä esitetään asennuksen edistyminen. Create-react-app:in asennus lisäosineen päivineen saattaa kestää useita minuutteja riippuen verkon nopeudesta. Jos prosessi suoriutuu loppuun ilman virheitä, komentorivillä näytetään seuraavat ilmoitukset ja ohjeistukset (kuva 6):

```
Success! Created muistiinpano-applikaatio at C:\Users\Allu\Desktop\muistiinpano-applikaatio
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 muistiinpano-applikaatio
  npm start

Happy hacking!
```

Kuva 6. Valmis create-react-app -asennusprosessi

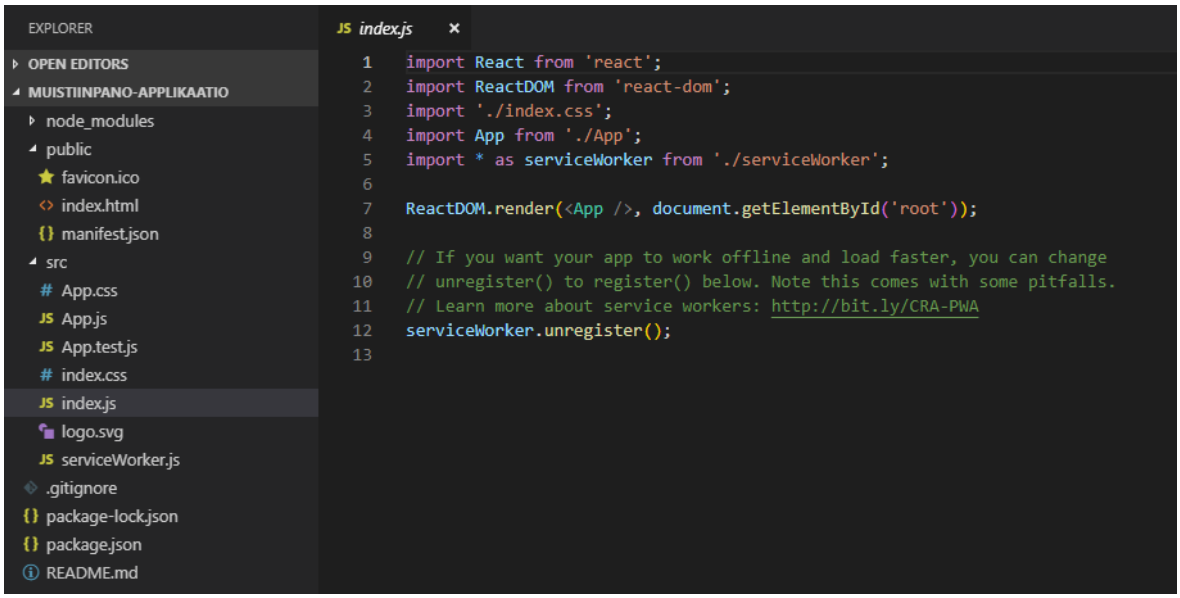
Kyseisessä ohjeistuksessa selostetaan, mitä komentoja voidaan seuraavaksi suorittaa vastaluodun hakemiston alla. Tässä vaiheessa alkuun pääsemiseksi keskitytään ainoastaan olennaisiin komentoihin, eli kahteen alimmaiseen.

Edellisen kuvan ohjeistusten mukaisesti siis aloitetaan siirtymällä vastaluodun hakemiston alle ”**cd muistiinpano-applikaatio**”-komennolla ja lopuksi käynnistetään applikaatio ”**npm start**” komennolla. Sovellus käynnistyy localhostin (paikallinen isäntä) porttiin 3000, eli osoitteeseen http://localhost:3000 ja avaa applikaation samalla suoraan verkkoselaimessa, joka oletusarvoisesti näyttää seuraavalta (kuva 7):



Kuva 7. Create-react-appin luoma oletusarvoinen applikaatio avattuna verkkoselaimessa

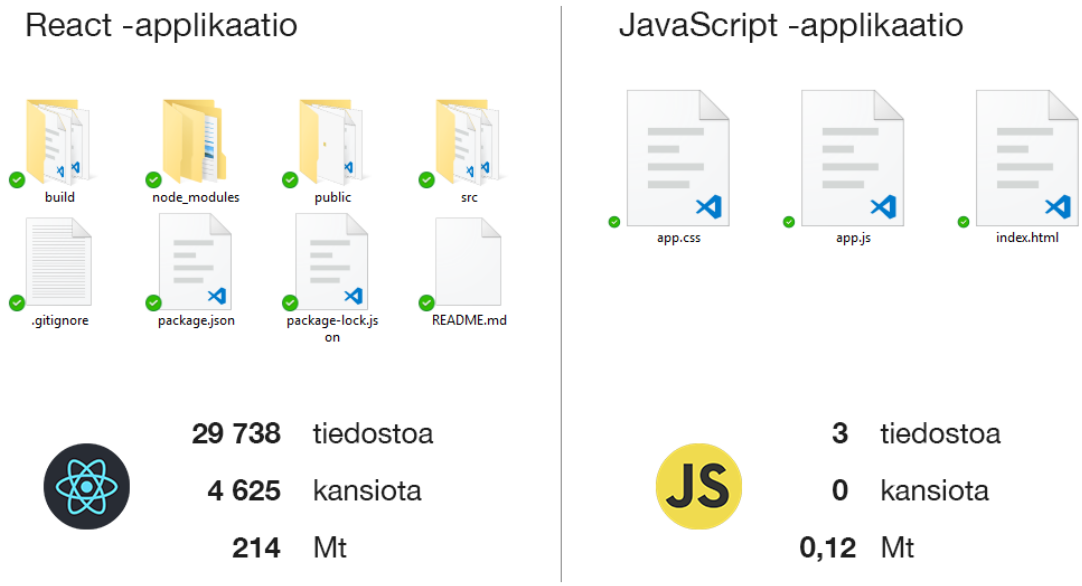
Create-react-app on luonut projektin oman kansiorakenteen, johon sisältyy kaikki tarvittavat applikaation koodi- ja konfiguraatiotiedostot sekä toiminnan kannalta välttämättömät Node-moduulit. Itse sovelluksen koodi löytyy hakemistosta src ja applikaation koko hakemistorakenne näyttää seuraavalta (kuva 8):



Kuva 8. Applikaation kansiorakenne. Vasemmalla esitettynä kansiot (src, public, node_modules) ja oikealla puolella avattuna src-hakemiston index.js -tiedosto.

Tätä asennusprosessia vertailtaessa perinteisempään JavaScript-kehitykseen, JS erottuu edukseen merkittävästi. JavaScript itse ei vaadi mitään erillisien resurssien – kuten Node.js, npm, create-react-app - asennuksia tai alustuksia, joten nämä kaikki aiemmat vaiheet voidaan applikaation JS-version puolesta ohittaa.

Lisäksi kansiorakenne ja tiedostojen määrä ovat Reactin ja JS:n välillä aivan omaa luokkaansa. Create-react-app on ladannut verkosta jopa tuhansia toiminnan kannalta välttämättömiä tiedostoja kansioineen ja moduuleineen, kun taas JavaScriptissä alkuvaiheessa pärjää yleensä pelkästään kolmella tiedostolla (HTML-, CSS- ja JS-tiedosto) tai jopa pelkästään yhdellä HTML-tiedostolla, johon on upotettu kaikki näiden kolmen koodi ja merkinnät. Ohessa demonstroituna konkreettisesti Muistiinpano-applikaation Reactin ja JavaScript-version väliset erot näin alkuvaiheessa (kuva 9).



Kuva 9. Create-react-app ja perinteisen JavaScript -applikaatioversioiden väliset erot alustusvaiheen jälkeen. Vasemmalla React-versio ja oikealla JS-versio

Sovelluksen alkuvaiheessa JavaScript siis erottuu Reactista huomattavasti ajankäytön, yksinkertaisuuden ja tiedostomäärän puolesta. Lopputulos kuitenkin ratkaisee eniten ja on ehdottomasti otettava huomioon, että nämä ovat vain Reactin kehitysvaiheessa käytettyjä tiedostoja (esim. Node.js-moduulit) ja lopullinen applikaatio pakkautuu vain muutamiin tiedostoihin. Näin ollen muutamien minuuttien ajansäästö tai tiedostojen määrä ei vielä näin aikaisessa vaiheessa kerro käytännössä mitään applikaation lopputuloksesta.

Tässä kohtaa React-applikaation asennusvaihe on valmis ja seuraavaksi voidaan siirtyä itse applikaation kehityksen pariin.

3.3 Elementin luonti

Reactissa on yleisesti ottaen kolme merkittävää osaa, jotka tekevät kirjastosta tehokkaan voimavaran kehitystöihin: JSX syntaksi, komponentit sekä virtuaalinen DOM. Nämä tuovat helpottavia ominaisuuksia, jotka vähentävät tarvittavan koodin määrää ja tekevät käyttöliittymien kehityksestä huomattavasti tehokkaampaa perinteiseen JavaScriptiin katsottuna.

Näiden ominaisuuksien puute perinteisessä JavaScript-koodauksessa on huomattava haitaste (alkuvalmistelujen tarpeettomuutta lukuun ottamatta), mutta nämä vajeet voivat tuoda myös muutamia pieniä etuja. React-applikaatiota suorittaessa verkkoselaimen täytyy jäsentää (parse) kaikki toimivuuden kannalta välttämätön kirjaston sisältämä JavaScipt-koodi, jota React sisältää hyvin runsaasti. Tämä voi suorituskyvyn ja sivun

latausnopeuden kantilta vaikuttaa merkittävästi, mutta yleensä erot kaventuvat applikaation kasvaessa.

JSX mahdollistaa kirjoittamaan puhdasta HTML-ulkoasuista kieltä huolimatta siitä, että ollaan kirjoittamassa JavaScriptiä. Elementtien ja komponenttien jäljittäminen tavalla, joka visuaalisesti vastaa JSX:llä kirjoitettua puhdasta HTML:ää, on tavallisella JavaScriptillä kirjoitettuna huomattavasti vaivalloisempaa ja koodirunsasta. Replikoinnin kompleksisuutta on demonstroitu seuraavassa kuvassa (kuva 10), jossa on yritetty jäljentää dynaamista React-elementtiä, jolla voidaan tulostaa h1 HTML-elementti.

```jsx

const luoElementti = <h1>Heipähei!</h1>;

```

```js
function luoElementti(tyyppi, teksti) {
    var elementti = document.createElement(tyyppi);
    var tekstiNode = document.createTextNode(teksti);

    elementti.appendChild(tekstiNode);

    return elementti;
}

var h1 = function(teksti) {
    return luoElementti('h1', teksti)
};

// H1-elementin kutsu
document.body.appendChild(h1('Heipähei'));
```

Kuva 10. JSX-syntaksijäljitteisen React-elementin luonti puhtaalla JavaScriptillä. Vasemmalla JSX-versio ja oikealla JS-versio

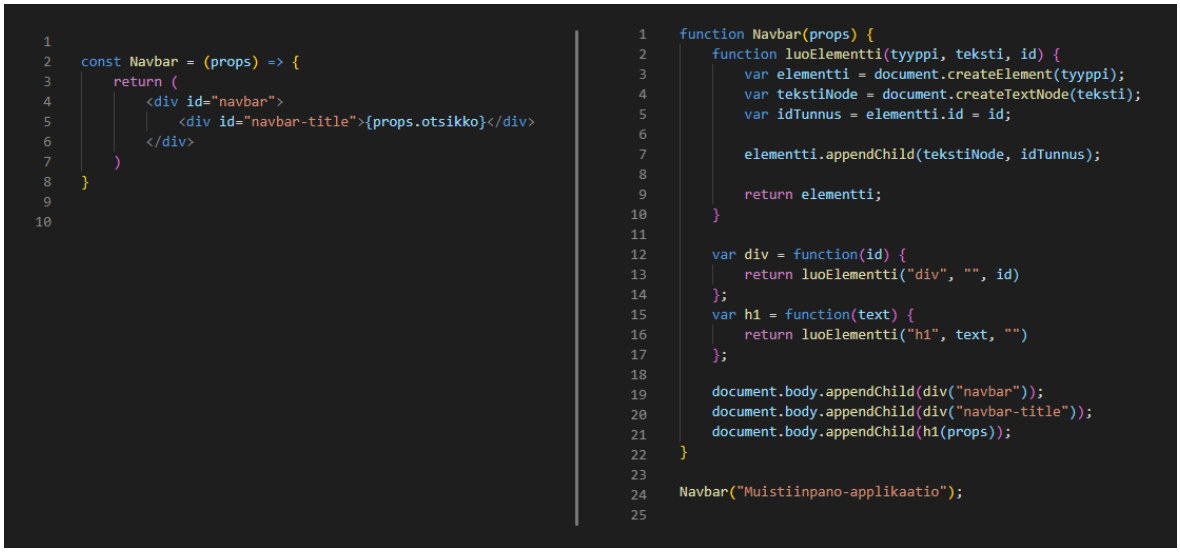
Kuvan JavaScript koodissa hyödynnetään tekaistua virtuaalista DOM-teknologiaa, jolla voidaan luoda elementti ilman sen kirjoittamista itse HTML-dokumenttiin. Kyseisellä ”dynaamisella” koodilla voidaan myös poissulkea toistavan ***document.createElement()***- ja ***makeElement(”h1”)*** -dokumenttiolioiden kutsuminen ja tekstiä voidaan toiminnallisesti lisätä suoraan elementin kutsukomennossa.

Tämänkaltaisessa JS-koodaamisessa ei kuitenkaan ole käytännössä mitään mieltä, kun kyseisen h1-elementin voisi aivan yhtä hyvin kirjoittaa suoraan HTML-koodiin sellaisenaan, kuin se perinteisestikin kirjoitettaisiin h1-tageilla. Lopputulos olisi käytännössä täysin sama.

3.4 Komponentin luonti

JavaScriptissä ei puhtaalta pöydältä ole ominaisuutta luoda suoraa React-komponentin vastinetta, ellei sitten ole kyseessä React funktiokomponentti (josta aiemmin luvussa 2.2.1 mainittiin). Hyvin kärjistettynä React-komponentti onkin vain funktio, joka palauttaa elementin - eli käytännössä komponenttia olisi mahdollista jäljittää ryhmittämällä funktioita

funktion sisälle ja Reactin konventioiden mukaisesti antaa tälle iso alkukirjain, jolloin se periaatteessa vastaisi React-komponenttia. Tälle ”komponentille” pystyy jopa myös määrittämään ominaisuuksia (props) Reactin tapaisesti, jota on demonstroituna seuraavassa kuvassa (kuva 11).



```jsx
const Navbar = (props) => {
    return (
        <div id="navbar">
            <div id="navbar-title">{props.otsikko}</div>
        </div>
    )
}
```

```js
function Navbar(props) {
    function luoElementti(tyyppi, teksti, id) {
        var elementti = document.createElement(tyyppi);
        var tekstiNode = document.createTextNode(teksti);
        var idTunnus = elementti.id = id;

        elementti.appendChild(tekstiNode, idTunnus);

        return elementti;
    }

    var div = function(id) {
        return luoElementti("div", "", id)
    };
    var h1 = function(text) {
        return luoElementti("h1", text, "")
    };

    document.body.appendChild(div("navbar"));
    document.body.appendChild(div("navbar-title"));
    document.body.appendChild(h1(props));
}

Navbar("Muistiinpano-applikaatio");
```

Kuva 11. Applikaation navigaatiopalkki-komponentin luonnin vertailua. Aiemman kuvan mukaisesti vasemmalla jälleen React-versio ja oikealla JS-versio.

Tässä esimerkissä siis luodaan ”Navbar” niminen komponentti, joka ottaa propsina navigaatiopalkin otsikon ja palauttaa sen div-rakenteineen kokonaisuutena, joka luo koko navigaatiopalkki-elementin. Vasemmanpuoleinen React-esimerkki on hyvin tyypillinen tapa luoda yksinkertainen komponentti, jonka toiminnallisuutta & ominaisuuksia on kuvan oikealla puolella lähdetty jäljittämään JavaScript-syntaksin mukaisesti.

Luokka-pohjaisen komponentin jäljittäminen saattaa koitua hankalahkoksi etenkin, kun käytössä ei ole React-koodin kirjoittamista helpottavia resursseja, kuten virtuaalinen DOM tai JSX. Lisäksi, jos JavaScriptiä ajatellaan itsenäisenä kokonaisuutena, niin luokkia yleisestikään ei voida hyödyntää, koska tämä on ECMAscript 2015:ssa esitelty ominaisuus, vaikkakin tätä voitaisiin teoriassa jäljitellä hyödyntäen olion prototyyppejä.

3.5 Komponentin tila

Komponentit ja niiden tilat ovat yksi olennaisimmista osista React-kirjastosta. Tiloilla varmistetaan, että React päivittää ainoastaan tarvittavat osat applikaatiosta, joka tapahtuu React DOM:issa vertaamalla elementtiä ja sen lapsiobjektia edelliseen, ja asettaa vain tarvittavat DOM-päivitykset haluttavan tilan saavuttamiseksi.

Jos ajatellaan esimerkiksi verkkosivulla olevaa tikittävää kelloa, joka reaaliajassa päivittää itsensä joka sekunti, parhaimman suorituskyvyn saavuttamiseksi aikapäivityksen tulisi koskea ainoastaan kello -komponenttia häiritsemättä tai päivittämättä muita elementtejä.

Komponentin tiloja voidaan myös konkreettisesti hyödyntää rakennettavassa muistiinpano-applikaatiossa esimerkiksi muistiinpanoja lisätessä ja tyhjennettäessä. Oheisessa esimerkkikuvassa (kuva 12) esitellään applikaation komponentti ”NoteApp”, jossa itse muistiinpanoille alustetaan oma oletustila, joka hakee talletetut muistiinpanot selaimen lokaalisäiliöstä (local store). Komponentin sisään on lisäksi luotu elementti ”poistaMuistiinpanot”, joka poistaa säiliöön tallennetut muistiinpanot hyödyntämällä uuden tilan asettamista, jolloin komponentti päivittää itsensä päivittämättä itse sivua uudelleen

```
class NoteApp extends React.Component {
    constructor() {
        super();
        this.state = {
            notes: [localStorage.getItem("muistiinpanot")]
        }
    }

    render() {
        const poistaMuistiinpanot = (e) => {
            e.preventDefault();
            if(window.confirm(
                "Haluatko poistaa kaikki muistiinpanot?")) {
                this.setState({
                    notes: [localStorage.clear()]
                });
            }
        }
    }
}
```

```
function poistaMuistiinpanot(e) {
    e.preventDefault();
    if (window.confirm(
        "Haluatko poistaa kaikki muistiinpanot?")) {
        localStorage.clear();
    }
}
```

Kuva 12. Tilallisen React-komponentin vertailua

Kuvan 12 mukainen samalla logiikalla rakennettu JavaScript-pohjainen verrokki ei näytä tehtyjä muutoksia – eli muistiinpanojen poistoa – ilman, että sivu päivitetään erikseen. Vaikka päivitys hoituisi automaattisesti, kyseessä olisi silti varteenotettava ero React-pohjaiseen versioon käyttömukavuuden ja suorituskyvyn kannalta.

Tilojen toiminnollisuuksia on mahdollista viedä askeleen eteenpäin hyödyntämällä erillistä tilanhallintakirjastoa (kuten Redux tai MobX), jolla näiden hallintaan saadaan vielä täydempi kontrolli sekä sukelletaan syvemmälle applikaation datan hallintaan. Tähän ei kuitenkaan tämän tutkimusprojektin puitteissa perehdytä applikaation pienen kokonsa vuoksi.

3.6 Applikaation testaus

Reactille on rakennettu useita erilaisia vaihtoehtoja testausprosessiin, jotka vaihtelevat selaimen kehitystyökalulaajennuksista aina kokonaisvaltaisiin testausalustoihin.

Testausmahdollisuuksia löytyy kaikenkokoisille applikaatioille, joista suurikokoisten sovellusten kohdalla voidaan hyödyntää esimerkiksi Jest-nimistä testausalustaa, jota on mahdollista käyttää myös pelkän JavaScript-pohjaisen applikaation kanssa.

Tämän projektin puitteissa ei ole hyödynnetty erillistä testaukseen omistautunutta alustaa, vaan testaukseen on käytetty lähinnä Chrome kehitystyökaluja ja varmistettu, että applikaatio ei sisällä virheitä. Perinteisen kehitystyökalun lisäksi on hyödynnetty ”**React Developer Tools for Chrome**” -nimistä aputyökalua, joka tuo erillisiä React-keskeisiä virheenetsintäominaisuuksia. Työkalulla voidaan esim. valita jokin tietty komponentti puurakenteesta, ja tarkastamalla (inspect) muokata sen nykyisiä propseja ja tiloja.

Testaukseen löytyy myös muita tunnettuja testaustyökaluja, kuten Reactin oma ReactTestUtils ja Airbnb:n kehittämä Enzyme, joka jQueryn API:ta jäljittämällä manipuloi DOM-rakennetta ja tämän kautta suorittaa määritetyt testaustoimenpiteet.

3.7 Mahdolliset jatkotoimenpiteet

JavaScript-ohjelmoinnilla tuotetut applikaatiot eivät sinällään vaadi testausten lisäksi mitään jatkotoimenpiteitä, paitsi jos koodi halutaan minimoida ja ellei applikaatio ole sidoksissa mahdollisiin ohjelmistokehyksiin tai lisäresursseihin, jotka vaativat erillisiä operaatioita.

React-applikaation kohdalla on tehtävä vielä viimeinen toimenpide – tuotantorakenteen luonti (production build). Applikaatio on tähän mennessä ollut kehitysrakenteen muodossa (development build), jonka tarkoituksena nimensä mukaisesti on toimia kehityksen aikaisena kokonaisuutena applikaatiolle. Kehitysrakenteessa kaikki tiedostot ja tiedostosidokset ovat jäsennettynä lähdekarttatiedostoihin (source map), joilla varmistetaan, että kaikki tarvittavat koodikokoojat ja -jäsentimet, debuggerit, pakettikokonaisuudet ynnä muut tarvittavat resurssit ovat yhteydessä applikaation kokonaisrakenteeseen ja kommunikoivat keskenään.

Vastavuoroisesti applikaation tuotantorakenteella tarkoitetaan applikaation tuotantotilassa pyörivää koodia, joka ajautuu asiakasohjelman koneella. Tuotantorakenteessa kaikki applikaation koodi kääritään minimoituun JavaScript-muotoon sekä lopuksi kootaan kaikki lähdetiedostot yhteen tai useampaan minimoituun tiedostoon. Tuotantoversioon myös poimitaan kaikki kuvat, CSS-tyylitiedot ja muut vastaavat lähdetiedostot, jotka sitten ladataan paalainohjelmaan (esim. create-react-appin kohdalla Webpack-niminen työkalu). Tässä

vaiheessa applikaatio ei enää hyödynnä lähdekarttatiedostoja, vaan tiedostorelaatiot luodaan paalainohjelmalla suoraan lähdetiedostoihin.

Se, miten kehitysrakenne eroaa tuotantorakenteesta konkreettisesti, riippuu applikaation asetuksista ja vaatimuksista – eli käytännössä siitä, miten määritykset on kirjattu paalainohjelmaan. Create-react-appia hyödynnettäessä näihin määrityksiin ei yleensä ole tarvetta koskea, koska asetukset on jo valmiiksi määritelty hyvin pitkälle sen mukana tulevassa Webpack-paalaintyökalussa.

Applikaation nykyinen rakenne on mahdollista tarkistaa käyttäen React Developer Tools for Chrome -lisäosaa verkkoselaimessa, joka osaa ilmoittaa, ajautuuko avattu applikaatio kehitys- vai tuotantorakenteessa.

Siinä vaiheessa, kun applikaatio on kehitystöiden puolesta valmis, voidaan sovelluksesta tehdä tuotantoversio. Jälleen kerran, tämä on create-react-app -pohjaisissa React-applikaatioissa tehty hyvin yksinkertaiseksi ja työ hoituu yhdellä komentorivin npm-komennolla; **npm run build**. Komentorivillä siirtyessä applikaation juurikansio alle ja kutsumalla tämän kyseisen komennon, create-react-app hoitaa koko rakennusprosessin itsestään, joka seuraavassa kuvassa (kuva 13) on esiteltynä.

```
PS F:\ONEDRIVE\HAAGA-HELIA\OPINNAYTETYO\Notes\notes> npm run build


Creating an optimized production build...
Compiled successfully.

File sizes after gzip:

  32.39 KB  build\static\js\1.4fd91c74.chunk.js
  1.34 KB   build\static\css\main.9de8e48e.chunk.css
  1.04 KB   build\static\js\main.e944d898.chunk.js
  763 B     build\static\js\runtime~main.229c360f.js

The project was built assuming it is hosted at the server root.
You can control this with the homepage field in your package.json.
For example, add this to build it for GitHub Pages:

  "homepage" : "http://myname.github.io/myapp",

The build folder is ready to be deployed.
You may serve it with a static server:

  npm install -g serve
  serve -s build

Find out more about deployment here:

  http://bit.ly/CRA-deploy
```

Kuva 13. Applikaation tuotantorakenteen luominen npm run build -komennolla.

Kun kyseinen komento on hoitanut tehtävän loppuun, niin komentorivi antaa siitä onnistumisilmoituksen. Tämän lisäksi komentorivi kertoo applikaation lopullisten kooditiedostojen koot, jotka tämän Muistiinpano-applikaation kohdalla näyttävät olevan siis yhteensä vain 35.47 KB. Tämä on applikaation alkuperäiseen create-react-appin kehitysrakenteeseen katsottuna niin huima ero, että React-versio ja JavaScript-versio ovat käytännössä nyt samankokoisia.

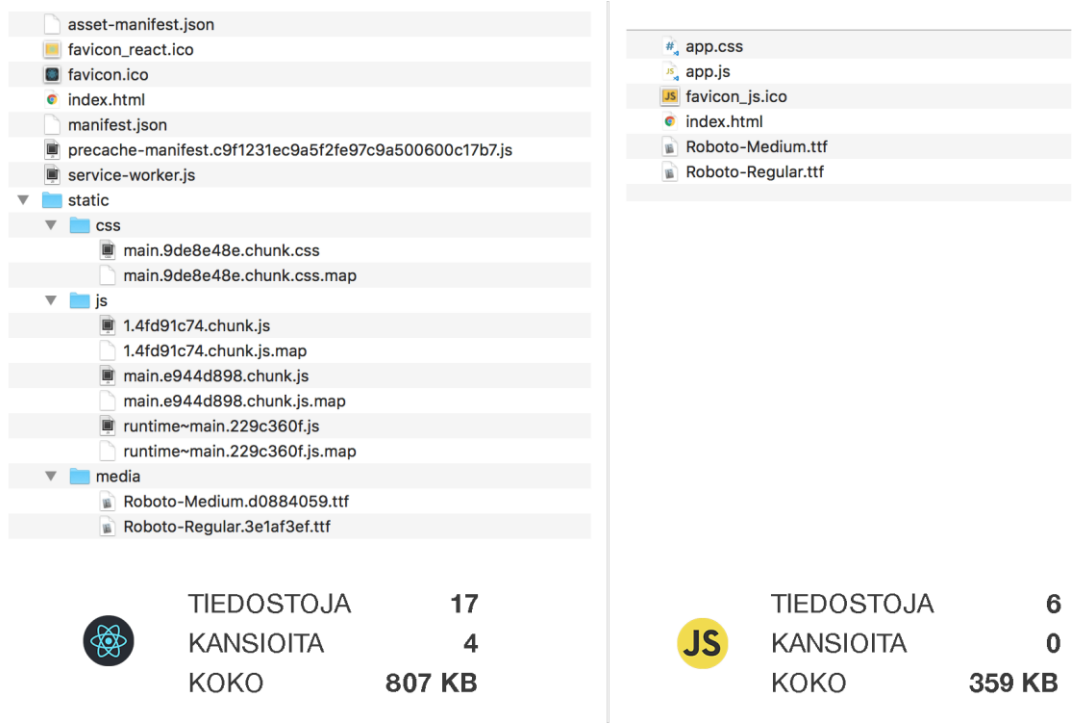
Kun tuotantoversion rakennus on suoritettu loppuun, voidaan applikaatio sijoittaa lopulliseen sijaintiinsa, eli web-palvelimelle. Vaihtoehtoisesti tämä voidaan isännöidä vaikka GitHubissa.

Npm run build on luonut valmiin hakemistorakenteen applikaatiolle, mutta rakennetta on tarvittaessa mahdollista muokata. Tällöin kaikki muutokset täytyy kirjata myös pääkansiosta löytyvään ”asset-manifest.json” -manifestitiedostoon (kuva 14), joka kerää kaikkien tarvittavien resurssien hakemistopolut.

Kuva 14. Applikaatiorakenteen asset-manifest.json tiedostoon kerätyt hakemistopolkutiedot.

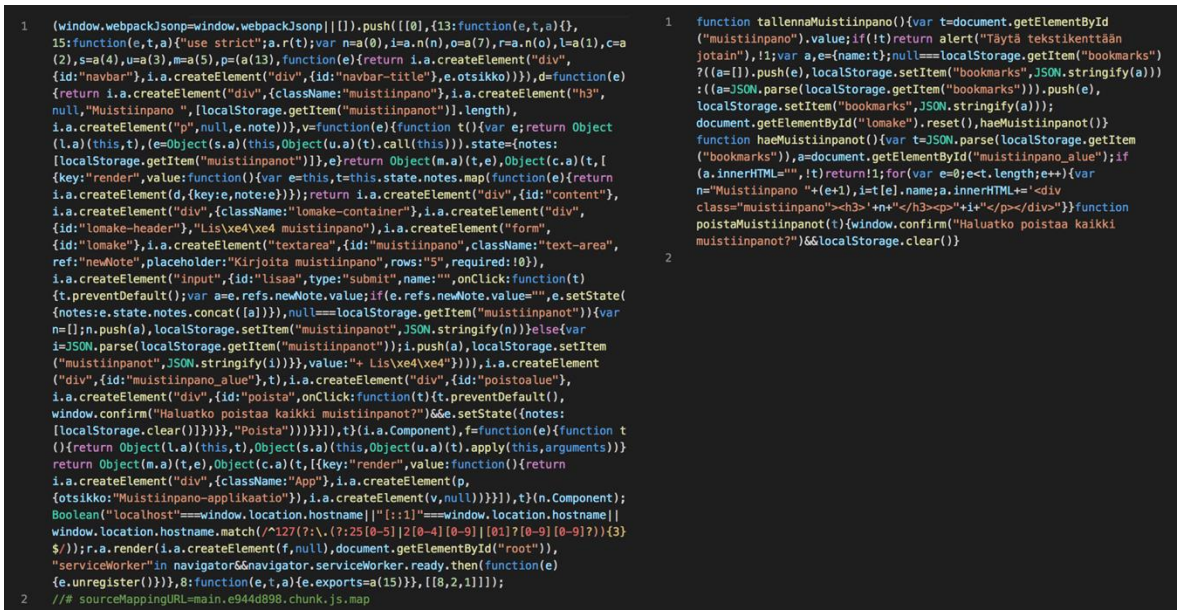
3.8 Yhteenveto

Applikaation rakennusprosessin alkupään radikaaleista eroista poiketen sovelluksen lopputulos Reactin ja JavaScriptin välillä on melko samanlainen. Molemmat applikaatiot ovat koodisyntaksi- ja tiedostomäärältään samaa luokkaa, sovellus avautuu & pyörii selaimessa nopeasti pienen kokonsa vuoksi ja tiedostokokojen ero on muutamien satojen kilobittien luokkaa. Seuraavassa kuvassa (kuva 15) on esiteltynä tuotantoversioiden väliset erot tiedostomäärän, kansiomäärän ja tiedostokokojen suhteen.



Kuva 15. Lopullisten applikaatioiden kansiorakenteet tiedosto- ja kansiomäärineen sekä kokoineen. Vasemmalla React-versio ja oikealla JavaScript-versio.

Koodimäärän vertailua on hieman hankalahkoa lähteä vertailemaan käytännöllisesti, koska Reactin toiminnot on siroteltu useaan eri tiedostoon ja lisäksi koodi on JavaScriptistä poiketen automaattisesti minimoitu. Näistä jälkimmäinen on tosin mahdollista suorittaa manuaalisesti hyödyntäen verkosta löytyviä kodin minimointityökaluja, jolla JavaScript-koodi saadaan pienennettyä yhdelle riville. Näin koodi voidaan saada mahdollisimman lähelle Reactin tuotantorakennetta, ja jos vertailtavaksi otetaan applikaation olennaisin koodi tiedosto, niin näiden kahden koodimäärää voidaan kuvan 16 mukaisesti vertailla melko todenmukaisesti.



```
(window.webpackJsonp=window.webpackJsonp||[]).push([[0],{13:function(e,t,a){},
15:function(e,t,a){"use strict";a.r(t);var n=a(0),i=a.n(n),o=a(7),r=a.n(o),l=a(1),c=a
(2),s=a(4),u=a(3),m=a(5),p=(a(13),function(e){return i.a.createElement("div",
{id:"navbar"},i.a.createElement("div",{id:"navbar-title"},e.otsikko))}),d=function(e)
{return i.a.createElement("div",{className:"muistiinpano"},i.a.createElement("h3",
null,"Muistiinpano ",[localStorage.getItem("muistiinpanot")].length),
i.a.createElement("p",null,e.note))},v=function(e){function t(){var e;return Object
(l.a)(this,t),(e=Object(s.a)(this,Object(u.a)(t).call(this))).state={notes:
[localStorage.getItem("muistiinpanot")]},e}return Object(m.a)(t,e),Object(c.a)(t,[
{key:"render",value:function(){var e=this,t=this.state.notes.map(function(e){return
i.a.createElement(d,{key:e,note:e})});return i.a.createElement("div",{id:"content"},
i.a.createElement("div",{className:"lomake-container"},i.a.createElement("div",
{id:"lomake-header"},"Lis\xe4\xe4 muistiinpano"),i.a.createElement("form",
{id:"lomake"},i.a.createElement("textarea",{id:"muistiinpano",className:"text-area",
ref:"newNote",placeholder:"Kirjoita muistiinpano",rows:"5",required:!0}),
i.a.createElement("input",{id:"lisaa",type:"submit",name:"",onClick:function(t)
{t.preventDefault();var a=e.refs.newNote.value;if(e.refs.newNote.value="",e.setState(
{notes:e.state.notes.concat([a])}),null===localStorage.getItem("muistiinpanot")){var
n=[];n.push(a),localStorage.setItem("muistiinpanot",JSON.stringify(n))}else{var
i=JSON.parse(localStorage.getItem("muistiinpanot"));i.push(a),localStorage.setItem
("muistiinpanot",JSON.stringify(i))}},value:"+ Lis\xe4\xe4"}))),i.a.createElement
("div",{id:"muistiinpano_alue"},t),i.a.createElement("div",{id:"poistoalue"},
i.a.createElement("div",{id:"poista",onClick:function(t){t.preventDefault(),
window.confirm("Haluatko poistaa kaikki muistiinpanot?")&&e.setState({notes:
[localStorage.clear()]})}},"Poista")))}}]),t}(i.a.Component),f=function(e){function t
(){return Object(l.a)(this,t),Object(s.a)(this,Object(u.a)(t).apply(this,arguments))}
return Object(m.a)(t,e),Object(c.a)(t,[{key:"render",value:function(){return
i.a.createElement("div",{className:"App"},i.a.createElement(p,
{otsikko:"Muistiinpano-applikaatio"}),i.a.createElement(v,null))}}]),t}(n.Component);
Boolean("localhost"===window.location.hostname||"[::1]"===window.location.hostname||
window.location.hostname.match(/^127(?:\.(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)){3}
$/));r.a.render(i.a.createElement(f,null),document.getElementById("root")),
"serviceWorker"in navigator&&navigator.serviceWorker.ready.then(function(e)
{e.unregister()})},8:function(e,t,a){e.exports=a(15)}},[[8,2,1]]]);
//# sourceMappingURL=main.e944d898.chunk.js.map
```

```
function tallennaMuistiinpano(){var t=document.getElementById
("muistiinpano").value;if(!t)return alert("Täytä tekstikenttään
jotain"),!1;var a,e={name:t};null===localStorage.getItem("bookmarks")
?((a=[]).push(e),localStorage.setItem("bookmarks",JSON.stringify(a)))
:((a=JSON.parse(localStorage.getItem("bookmarks"))).push(e),
localStorage.setItem("bookmarks",JSON.stringify(a)));
document.getElementById("lomake").reset(),haeMuistiinpanot()}
function haeMuistiinpanot(){var t=JSON.parse(localStorage.getItem
("bookmarks")),a=document.getElementById("muistiinpano_alue");if
(a.innerHTML="",!t)return!1;for(var e=0;e<t.length;e++){var
n="Muistiinpano "+(e+1),i=t[e].name;a.innerHTML+='<div
class="muistiinpano"><h3>'+n+"</h3><p>"+i+"</p></div>"}}function
poistaMuistiinpanot(t){window.confirm("Haluatko poistaa kaikki
muistiinpanot?")&&localStorage.clear()}
```

Kuva 16. Minimoidun applikaatiokoodin vertailua, jossa tuttuun tapaan vasemmalla puolella Reactin luoma koodi ja oikealla puhtaan JS-version koodi.

Applikaation lopputuloksen puolesta näiden kahden erot tosiaan ovat melko vähäiset. Tutkimus olisi vaatinut laajempikokoisemman applikaation testattavaksi, jotta konkreettiset erot näiden välillä olisivat olleet huomattavia.

Kokonaisuudessaan JavaScript-pohjaisen version rakennus oli React-versiota nopeampi toteuttaa, joka suurimmaksi osaksi johtuu applikaation pienestä koosta sekä omasta aiemmasta JS-kokemuksesta. Itse kehitystyö (eli ohjelmointi) oli tasaisempaa ajankäytön suhteen ja applikaation koon kasvaessa erot olisivat pelkästään kaventuneet ja jossain vaiheessa React olisi varmasti kiihdyttänyt edelle.

4 Pohdinta

Raportin viimeisessä luvussa avataan projektin herättämiä ajatuksia, tutkimuksen aikana kertyneitä havaintoja sekä arvioidaan omaa tekemistä ja käsitellään opinnäytetyön aikana puhjenneita haasteita.

4.1 React – Hyödyt ja varjopuolet

Rakentamani Muistiinpano-applikaatio oli kokonaisuudessaan hyvin yksinkertainen, mutta sovellusta rakennettaessa pystyin silti havaitsemaan monin eri tavoin, mitä käytännön etuja ja haittapuolia React tuo verkkokehitystöihin sekä mihin käyttötapauksiin kirjasto soveltuu.

Applikaation koon puolesta päällimmäisenä aspektina huomasin, kuinka paljon teknologian konkreettiset hyödyt voidaan alkaa havaitsemaan vasta applikaation koon kasvaessa. React vaati suhteellisen paljon alkuvalmisteluja vaadittavine asennuksineen, jolloin näin pienen applikaation kokoluokassa valmistelujen ajankäyttö alkoi näkyä jo koko applikaatiokokonaisuuden mittakaavassa.

Reactin modulaariset ohjelmointiaspektit ovat mainioita rakennettaessa sovelluksia, joissa hyödynnetään paljon toistettavia käyttöliittymäelementtejä, joita luomassani applikaatiossa tosin ilmaantui kaiken kaikkiaan melko vähän. Näin ollen pääsin soveltamaan modulaarista näkökulmaa jokseenkin vähän ja näiden todelliset hyödyt jäivät hieman hämärän peittoon.

Reactin ominaisuus, josta pidin paljon ja koin hyvin hyödylliseksi oli virtuaalinen DOM (Document Object Model) sekä JSX-syntaksi. Applikaation koodin ja merkinnän järjestäminen HTML-muotoisessa puurakennelmassa oli hyvin selkeää ja entuudestaan tuttua, joka helpotti kokonaiskuvan ymmärtämistä. JSX-syntaksissa tosin oli muutamia totuttelua vaativia kummallisuuksia – kuten HTML-elementtien luokkien merkintäsyntaksi, joka normeista poiketen täytyy kirjoittaa muodossa ”className”, koska ”class” on jo JavaScriptissä käytössä oleva merkintä. Ajan myötä nämä erikoisuudet toisaalta alkoivat tuntua luontevammalta syntaksin tullessa läheisemmäksi, eivätkä enää sekoittaneet kokonaiskuvaa.

Reactiin oli alkuvaiheessa suhteellisen helppo päästä käsiksi - erityisesti Create-react-app -aputyökalun avulla, jonka koin aloitusvaiheessa lähes korvaamattomaksi. En olisi ilman tätä resurssia päässyt koodaamisessa alkuun läheskään yhtä nopeasti ja alkuvalmistelut olisivat ilman create-react-appia luultavasti venyneet vielä pidemmäksi. Itse

ohjelmoinnissa virallinen dokumentaatio toimi mainiona tietolähteenä, jonka konkreettiset koodiesimerkit auttoivat myös suunnattomasti alkuunpääsyä.

4.2 Vaihtoehtoiset ratkaisut

Kilpaileviin ratkaisuihin verrattuna Reactin hyötynä on sen yksinkertaisuus sekä tavanomaisempi lähestymistapa JavaScriptiä hyödyntäen. Tunnetuimmista vaihtoehtoisista ratkaisuista esim. Angular hyödyntää TypeScript -nimistä JavaScriptin tehostavaa lisäosaa, joka voi alkuvaiheessa jyrkistää oppimiskäyrää Reactin tavanomaisempaan JavaScriptin JSX + ECMAscript -yhdistelmään verrattuna. Aiempaa verkkokehityskokemusta omaaville henkilöille Reactin syntaksi voi tuntua tutummalta, jolloin ajankäytön tarve uuden opettelulle on vähäisempää.

Angularilla tuotetut applikaatiot ovat kooltaan React-applikaatioita suurempia ja saattavat vaatia enemmän perehtymistä pelkästään jo TypeScript-pohjaisen ohjelmointikielen myötä. Angular on Reactista poiketen kokonaisvaltainen ohjelmistokehys - eikä pelkkä JavaScript-kirjasto – ja teknologia perustuu komponenttipohjaiseen kehykseen, joka tehokkaine komentorivi-työkaluineen mahdollistaa applikaation helpon skaalautuvuuden. Näin ollen Angular voi soveltua paremmin isompien applikaatiokokonaisuuksien rakentamiseen, mikäli jatkossa tuleville laajennuksille ja jatkokehityshankkeille on tarvetta.

Loppujen lopuksi se, mikä JS-kehyksistä soveltuu omaan käyttötapaukseen, riippuu täysin siitä mitä ollaan tekemässä ja miten se halutaan tehdä. Mikäli yksinkertaisuus ja applikaation koko ovat tärkeimmät aspektit, niin React on luultavasti paras ratkaisu. Vastavuoroisesti, mikäli jyrkempi oppimiskäyrä ei ole este ja skaalautuvuus on applikaation tärkeä näkökulma, niin Angular on varteenotettava vaihtoehto.

Reactille on monia vaihtoehtoisia kehys- ja kirjastoratkaisuja, ja näiden kilpailu valta-asemasta on ollut melko värikästä – ja on edelleen. Google Trends -palvelua hyödyntäessä voidaan helposti tarkastella kolmen suosituimman JavaScript-kehyksen suosiokäyrää (kuva 17) ja havainnoida konkreettisesti esim. viimeisen viiden vuoden aikana tapahtuneet verkkohaun muutokset koko maailman mittakaavassa.

Kuva 17. Google Trends palvelusta kuvakaapattu Reactin, Angularin ja Vue.js:n välinen hakusuosiokäyrä viimeisen viiden vuoden aikajanalta. Keskiarvon numerot esittävät haun suosiota valitulla ajanjaksolla ja alueella suhteutettuna kaavion suurimpaan arvoon asteikolla 0 - 100.

Kuten aikajanasta selviää, React on Googlen hakutermeissä noussut ykkössijalle – ohittaen Angularin vuoden 2018 alkupuoliskolla – ja on pysynyt tällä sijalla siitä lähtien. Tosin Angular on keskiarvon suhteen vielä toistaiseksi etusijalla johtuen sen alkuaikojen suosiosta.

Vastaavanlaista suuntausta on havaittavissa myös alan rekrytointihakemusten parissa, kun lähdetään vertailemaan kyseisillä hakusanoilla ilmaantuvia työpaikkailmoituksia. Käyttäessä esim. suosittua Stack Overflow -sivuston rekrytointiportaalia, voidaan huomata samankaltainen suosiomieltymys käyttämällä samoja hakusanoja: ”React”, ”Angular” ja ”Vue.js”, jolloin työpaikkailmoituksia ilmaantuu seuraavat tulosmäärät:

- React – 743 tulosta
- Angular – 387 tulosta
- Vue.js – 178 tulosta

Suosiojärjestys on siis edelleen sama, mutta Googlen hakusanatuloksiin verrattuna Reactin ja Angularin välinen suosio näyttää olevan maailmanlaajuisesti vielä laajempi ja React-hakusanalla työpaikkailmoituksia löytyy lähes tuplamäärä. Tämän lisäksi Vue.js:n kohdalla suhdeluku on Googlen hakutuloksiin verrattuna myös huomattavasti suurempi ja tämän tuloksen perusteella voisi päätellä, että Vue.js on vähitellen saavuttamassa Reactin ja Angularin suosion tasoa

4.3 Ajatukset ja ammatillinen kasvu

Opinnäytetyön aihe tuli alun perin valittua puhtaasti mielenkiinnosta aihealuetta kohtaan. React oli teknologia, josta olin kuullut paljon, mutta en ollut lainkaan tietoinen sen käyttötarkoituksista ja hyödyistä. Projektin alkuperäinen agenda olikin puhtaasti käytännönläheinen tutustuminen aiheen teoriaan ja applikaation rakennukseen, mutta opinnäytetyön ohjaajan kommenttien ja omien havaintojen pohjalta tämä kuitenkin osoittautui turhan suppeaksi aihealueeksi opinnäytetyölle. Näiden perusteella otin sitten yhtälöön mukaan nykyisen vertailevan tutkimusnäkökulman.

Vaikka opinnäytetyö oli kokonaisuudessaan varsin opettavainen ja oli mielenkiintoista nähdä konkreettisesti Reactin ja JavaScriptin eroja, koin silti tämän vertailevan aspektin osittain hidasteena työn edistymisen ja oman kehityksen kannalta. Vaikka React on käytännössä pelkkää JavaScript-ohjelmointikieltä, niin tästä huolimatta näiden kahden ominaisuudet poikkeavat osittain merkittävästi. Lähes kaikissa esimerkkitapauksissa JavaScriptillä ei ollut React-elementin tai -ominaisuuden suoraa vastinetta, jolloin toiminnoiltaan vastaavan JS-pohjaisen elementin rakennus olisi vaatinut huomattavasti enemmän syvempää osaamista ja kirjoitettavaa koodia. Ymmärrän, että tämä nimenomaan on kirjasto- ja kehysteknologioiden idea, ja tältä saralta vertailtavaksi olisikin luultavasti ollut viisaampaa valita puhtaan JavaScriptin tilalle jokin toinen JS-kehysvaihtoehto, kuten esim. Angular tai Vue.js.

Projektissa pääsin loppujen lopuksi kuitenkin hyvin käsiksi Reactin peruskonsepteihin ja jokseenkin yllättäen myös JavaScriptin puolelta tuli paljon uutta tietoa. Muutaman teknologian käyttö (esim. lokaalisäiliö) ei ollut projektia aloittaessa ollenkaan tuttu konsepti ja näiden kokemuksien perusteella saatan käyttää näitä ominaisuuksia jatkossakin myös perinteisellä JavaScriptillä.

4.4 Haasteet opinnäytetyössä

Opinnäytetyö osoittautui suhteellisen karikkoiseksi ja projektin aikana ilmaantui lukuisia haasteita, joista päällimmäisenä painaa tarvittavan ammattitaidon puute aihealueen saralta. Opinnäytetyötä aloittaessa en omannut yhtään aiempaa kokemusta React-kirjastosta, eivätkä ohjelmointitaitoni yleisestikään ulotu juuri pintatasoa syvemmälle. Myös niin sanotun ”backend”-puolen osaaminen on jäänyt hyvin vähäiselle, joten palvelimien ja verkkoapplikaatioiden välinen interaktio on kokemuksen puolesta jäänyt pintaraapaisutasolle. Edellä mainittujen ammattitaitopuutteiden näkökulmasta opinnäytetyön aiheen valinta voikin kuulostaa lähes järjenvastaiselta, jonka pystyn jälkiviisaana myös itse

allekirjoittamaan. Näin projektin kuitenkin loistavana tilaisuutena tutustua teknologiaan sekä teorian että konkretia kautta.

Tätä raporttia kirjoittaessa työskentelen opintojen ohella verkkokehitystöiden parissa, mutta aihealueen läheisyydestä huolimatta työtehtävät poikkeavat melko paljon opinnäytetyön teemasta ja lisäksi vastuualueeni työssä on ollut melko pintapuolinen. Näin ollen omasta alan työkokemuksestakaan ei ole ollut varteenotettavaa hyötyä opinnäytetyön edistämiseksi. Lisäksi näiden kyseisten työtehtävieni herkeämättömät työkiireet hankaloittivat myös projektin työstöä, kun opinnäytetyön ahertaminen oli usein jätettävä viikonlopuille sekä arki-illoille. Tämä opintojen ja töiden yhdistäminen loi toisinaan stressaavan ilmapiirin, joka ei lainkaan edesauttanut projektin valmistumista.

Olosuhteet huomioon ottaen onnistuin mielestäni ylittämään nämä haasteet varsin hyvin loppujen lopuksi. Olisin vallan hyvin pystynyt valitsemaan opinnäytetyön aiheen helpommalta ja itselleni tutummalta aihealueelta, mutta halusin tutkimusprojektin ohessa oppia jotain sellaista, jota tämän päivän työmarkkinoilla oikeasti arvostetaan ja josta tulen varmasti jatkossa hyötymään.

Lähteet

Abramov, D. 2016. Medium – You might not need Redux. Luettavissa: https://medium.com/@dan_abramov/you-might-not-need-redux-be46360cf367. Luettu: 27.10.2018.

Aggarwal, S. 2018. Modern Web-Development using ReactJS. International Journal of Recent Research Aspects, 5, 1, s. 133.

Argyle, Z. 2015. Medium. Stop Using React for Everything. Luettavissa: https://medium.com/@zackargyle/stop-using-react-for-everything-c8297ac1a644. Luettu. 20.10.2018.

Byers, D. 2018. Alligator.io - An Easy Way to Get Started with the MERN Stack. Luettavissa: https://alligator.io/react/mern-stack-intro/. Luettu: 27.10.2018

Express 2018. Express – Fast, unopinionated, minimalist web framework for Node.js. Luettavissa: https://expressjs.com/. Luettu: 28.10.2018

GitHub 2018. Express. Luettavissa: https://github.com/expressjs/express. Luettu: 28.10.2018

Github 2018. Sites using React. Luettavissa: https://github.com/facebook/react/wiki/Sites-Using-React. Luettu: 3.10.2018.

GitHub 2018. Redux. Luettavissa: https://github.com/reduxjs/redux. Luettu: 27.10.2018.

Joch, D. 2018. Medium. Functional vs Class-Components in React. Luettavissa: https://medium.com/@Zwenza/functional-vs-class-components-in-react-231e3fbd7108. Luettu: 21.10.2018.

MDN web docs 2018. JavaScript Introduction. Luettavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction. Luettu: 22.10.2018.

MongoDB Documentation. 2018. MongoDB – Documents. Luettavissa: https://docs.mongodb.com/manual/core/document/. Luettu: 28.10.2018.

Morgan, A. 2017. Introducing the MEAN and MERN stacks. Luettavissa: https://www.mongodb.com/blog/post/the-modern-application-stack-part-1-introducing-the-mean-stack. Luettu: 6.11.2018.

Node.js. 2018. Node.js Docs - Dependencies. Luettavissa: https://nodejs.org/en/docs/meta/topics/dependencies/. Luettu: 27.10.2018

Patel, P. 2018. freeCodeCamp – What exactly is Node.js? Luettavissa: https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5. Luettu: 27.10.2018.

Prathamesh Sonpatki, Vipul. A. M. 2016. ReactJS by Example - Building Modern Web Applications with React. Packt Publishing Ltd. Birmingham, Iso-Britannia.

ReactJS 2018. A JavaScript library for building user interfaces. Luettavissa: https://reactjs.org/. Luettu: 1.10.2018.

ReactJS 2018. Introducing JSX. Luettavissa: https://reactjs.org/docs/introducing-jsx.html. Luettu: 5.10.2018.

ReactJS 2018. Components and Props. Luettavissa: https://reactjs.org/docs/components-and-props.html. Luettu: 5.10.2018.

ReactJS 2018. Component state. Luettavissa: https://reactjs.org/docs/faq-state.html. Luettu: 15.10.2018.

ReactJS 2018. JavaScript Environment Requirements. Luettavissa: https://reactjs.org/docs/javascript-environment-requirements.html. Luettu: 5.10.2018.

React Enlightenment 2018. What are Component Props? Luettavissa: https://www.reactenlightenment.com/react-props/7.1.html Luettu 10.10.2018

React Enlightenment 2018. What Is Component State? Luettavissa: https://www.reactenlightenment.com/react-state/8.1.html. Luettu 11.10.2018

Redux. 2018. Redux – Three principles. Luettavissa: https://redux.js.org/introduction/threeprinciples. Luettu: 27.10.2018.

Serby, P. 2012. Venturebeat - Case study: How & why to build a consumer app with Node.js. Luettavissa: https://venturebeat.com/2012/01/07/building-consumer-apps-with-node/. Luettu: 28.10.2018

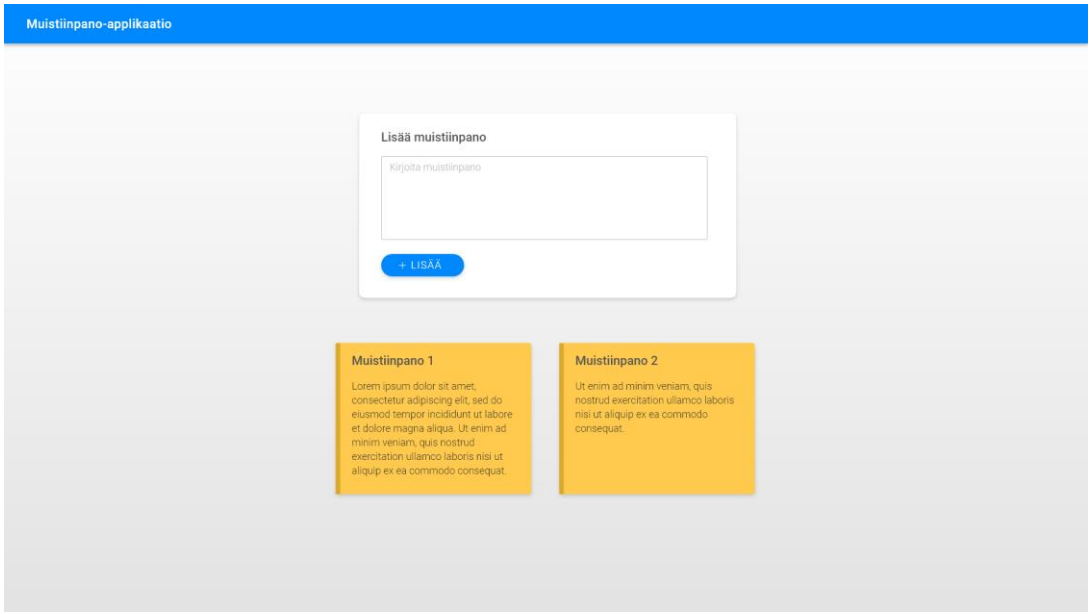
Voss, L. 2018. NPM Inc. The state of JavaScript frameworks. Luettavissa: https://www.npmjs.com/npm/state-of-javascript-frameworks-2017-part-1. Luettu: 1.10.2018.

W3Schools 2018. Node.js Introduction. Luettavissa: https://www.w3schools.com/nodejs/nodejs_intro.asp. Luettu: 26.10.2018

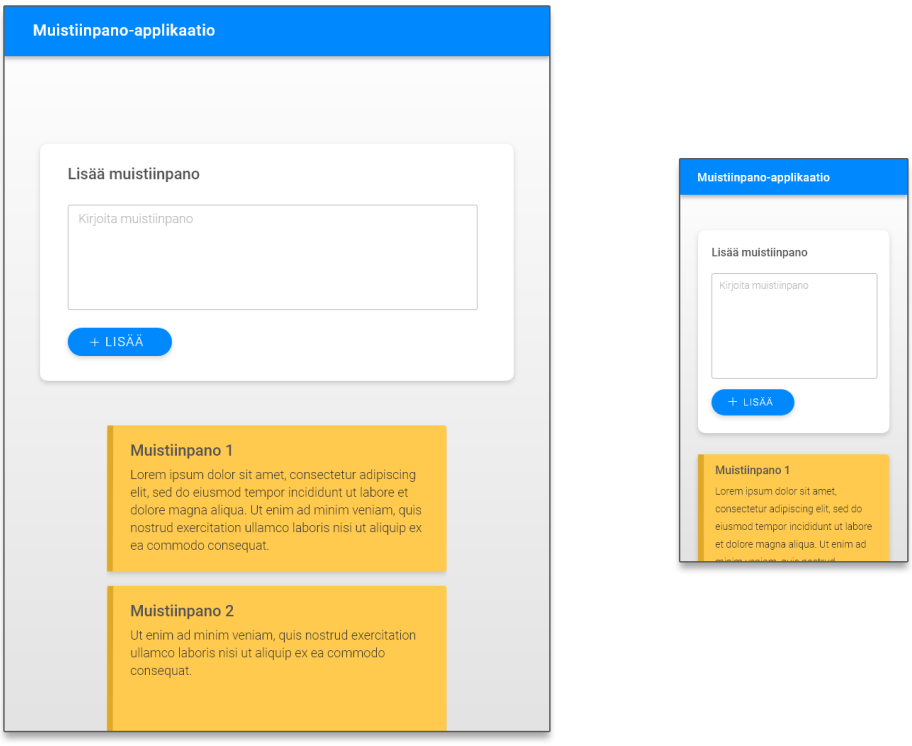
Liitteet

Liite 1. Muistiinpano-Applikaatio – Graafiset suunnitelmat

Muistiinpano-Applikaatio – Graafiset suunnitelmat

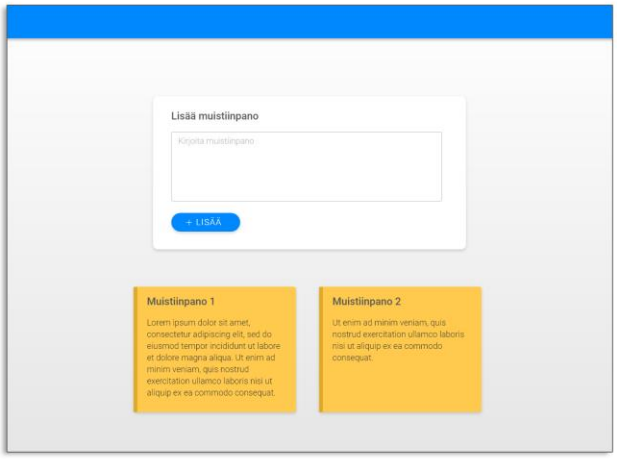


Kuva 1. Työpöytä-koon malli

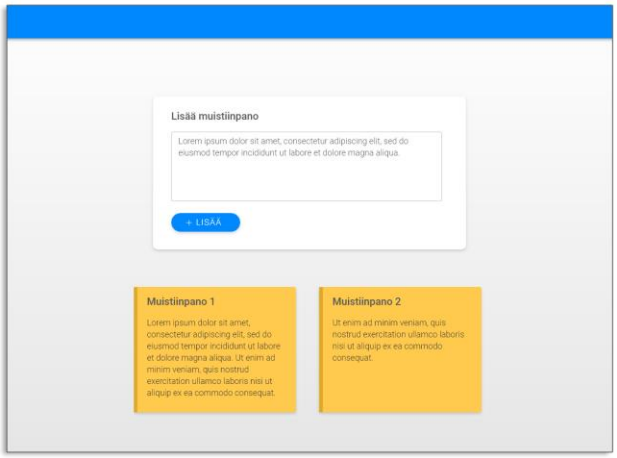


Kuva 2. Mobiiliruutujen mallit – tablettikone & kännykkä

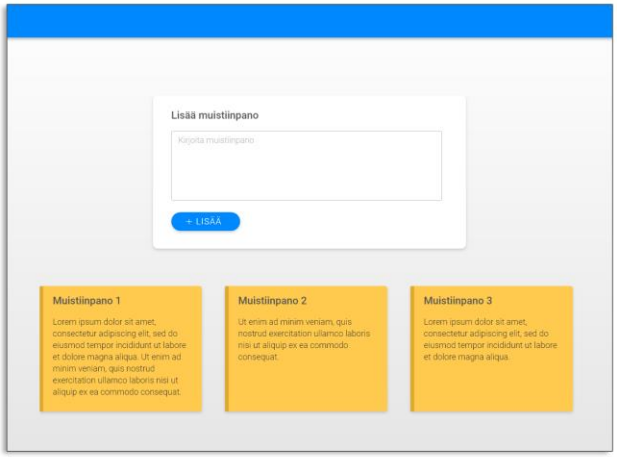
Applikaation virtaussuunnitelma



1. Applikaatio on avattuna. Muistiinpanoalueella näkyy mahdollisia aiemmin syötettyjä muistiinpanoja.



2. Käyttäjä syöttää lomakekenttään haluamansa muistiinpanotekstin ja painaa ”+ Lisää” – nappulaa.



3. Käyttöliittymä tuo uuden muistiinpanon käyttäjän syöttämällä tekstillä. Lomakekenttä tyhjentyy lisäyksen myötä uutta muistiinpanoa varten.