LAB University of Applied Sciences

YLLÄPITOKÄYTTÖLIITTYMÄ

Case: Digiloop

LAB-AMMATTIKORKEAKOULU
Insinööri (AMK)
Ohjelmistotekniikka
Kevät 2020
Erno Viitanen

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Viitanen, Erno	Opinnäytetyö, AMK	Kevät 2020
	Sivumäärä	
	33	

Työn nimi

Ylläpitokäyttöliittymä
Case: Digiloop

Tutkinto

Insinööri (AMK)

Tiivistelmä

Suomessa ympäristöministeriö on aloittanut jätelainsäädännön uudistuksen, joka pohjautuu EU:ssa hyväksytyn jätesäädöspaketin toimeenpanoon. Sen tavoitteena on vähentää jätteen määrää ja lisätä uudelleenkäyttöä ja kierrätystä.

Opinnäytetyön tavoitteena oli luoda Digiloop-kierrätyspalvelun taustajärjestelmä. Digiloop-kierrätyspalvelu on tarkoitettu tehostamaan SER-jätteiden ja akkujen kierrättämistä. Ylläpitokäyttöliittymä on tarkoitettu operaattoreille, ylläpitokäyttöliittymässä operaattorit voivat seurata ilmoituksia ja varata itselleen sopivat jäte-erät, joko listanäkymästä tai karttanäkymästä.

Työssä esitellään React ja Redux JavaScript-kirjastot ja lisäksi React Redux -kirjasto ja niiden toiminnat. Sovellus on toteutettu verkossa toimivana, koska loppukäyttäjien, eli jätteiden ilmoittajien oletetaan käyttävän sovellusta mobiililaitteella.

Työn lopputuloksena saatiin toimiva sovellus testikäyttöä varten. Siinä toimivat kaikki olennaiset toiminnot, kuten jätteiden ilmoittaminen ja varaaminen. Ylläpitokäyttöliittymää on tarkoitus käyttää pääasiallisesti tietokoneella, mutta sovelluksesta luotiin myös mobiilisovellus, joka on tarkoitettu lähinnä loppukäyttäjille.

Digiloop-kierrätyspalvelun tekeminen sai AIKO-rahoitusta kehittämistä varten. Jatkokehityksenä palvelun pohjalta on kehitetty jo RESELL-kierrätyspalvelu. RESELL-palvelu on LUT:n ja LAMK:n yhteisprojektina kehitetty kierrätyspalvelu.

Asiasanat

React, Redux, React Redux

Abstract

Author(s)	Type of publication	Published
Viitanen, Erno	Bachelor's thesis	Spring 2020
	Number of pages	
	33	

Title of publication

Administrative interface
Case: Digiloop

Name of Degree

Bachelor of Engineering

Abstract

In Finland, the Ministry of the Environment has started a reform of waste legislation which is based on the implementation of the waste legislative package approved in the EU. It aims to reduce the amount of waste and increase reuse and recycling.

The aim of the thesis was to create a background system for the Digiloop recycling service. The Digiloop recycling service is designed to increase the efficiency of SER waste and battery recycling. The administrative interface is designed for operators, where they can follow the notifications and reserve appropriate waste items for themselves, either from the list view or the map view.

This thesis presents the React and Redux JavaScript libraries and the React Redux-library and their features. The application was implemented to be a web application because end users, i.e. waste notifiers, are assumed to use the application with a mobile device.

As result of the thesis, a working application was produced for test use. It has all the relevant functions, such as waste notifying and reservation. The administrative interface is intended to be used mainly with a computer, but a mobile application was also created, which is intended mainly for end users.

The development of the Digiloop recycling service received AIKO funding. As further development, the RESELL recycling service has already been developed based on this service. The RESELL recycling service has been developed as a joint project between LUT and LAMK.

Keywords

React, Redux, React Redux

SISÄLLYS

LYHENNELUETTELO

CLI	Command line interface, komentokehote
HIGHER-ORDER FUNCTION	funktio, joka voi ottaa toisen funktion parametrina tai palauttaa funktion tuloksena
NPM	Node package manager, pakettien hallinta työkalu
NPX	Npm package runner, auttaa suorittamaan paketteja asentamatta niitä suoraan
OBJECT LITERALS	pilkulla erotettu lista nimi-arvo-pareista aaltosulkujen sisällä
PROMISE	objekti, joka edustaa asynkronisen toiminnan loppuun saattamista tai epäonnistumista
PROPS	lyhenne sanasta properties, käytetään siirtämään dataa komponentista toiseen
WEBPACK	kartoittaa kaikki projektin tarvitsemat moduulit ja luo yhden tai useamman paketin

1 JOHDANTO

Ihmiskunta hukkuu jätteisiin (Vesa 2019), koska osa jätteistä ei päädy kierrätykseen. Jätteitä kertyy ihmisten koteihin sekä yritysten varastoihin, osittain sen takia että kierrätyskeskukset voivat olla kaukana eikä ihmisillä ole mahdollisuutta kuljettaa niitä sinne. Yle Tieteen Taloustutkimuksella teettämän kyselyn mukaan ihmiset eivät välttämättä edes tiedä, mihin mikäkin jäte kuuluu. Kyselyssä yleisimpiä syitä jätteiden lajittelematta jättämiseen olivat jäteastian puuttuminen taloyhtiöstä ja lähin kierrätyspiste liian kaukana. (Hallamaa & Kanerva 2019.)

Suomessa ympäristöministeriö on aloittanut jätelainsäädännön uudistuksen, joka pohjautuu EU:ssa kesällä 2018 hyväksytyn jätesäädöspaketin toimeenpanoon. Sen tavoitteena on vähentää jätteen määrää ja lisätä uudelleenkäyttöä ja kierrätystä. (Ympäristöministeriö 2019.) Direktiivin mukaan EU:n jäsenvaltioiden on taattava SER-jätteiden vuosittainen vähimmäiskeräysaste. Vuodesta 2016 lähtien se on ollut 45 prosenttia, ja vuodesta 2019 lähtien 65 prosenttia. Laskenta tehdään kolmen viime vuoden aikana myytyjen sähkö- ja elektroniikkalaitteiden kokonaispainosta. (EUR-Lex 2019.)

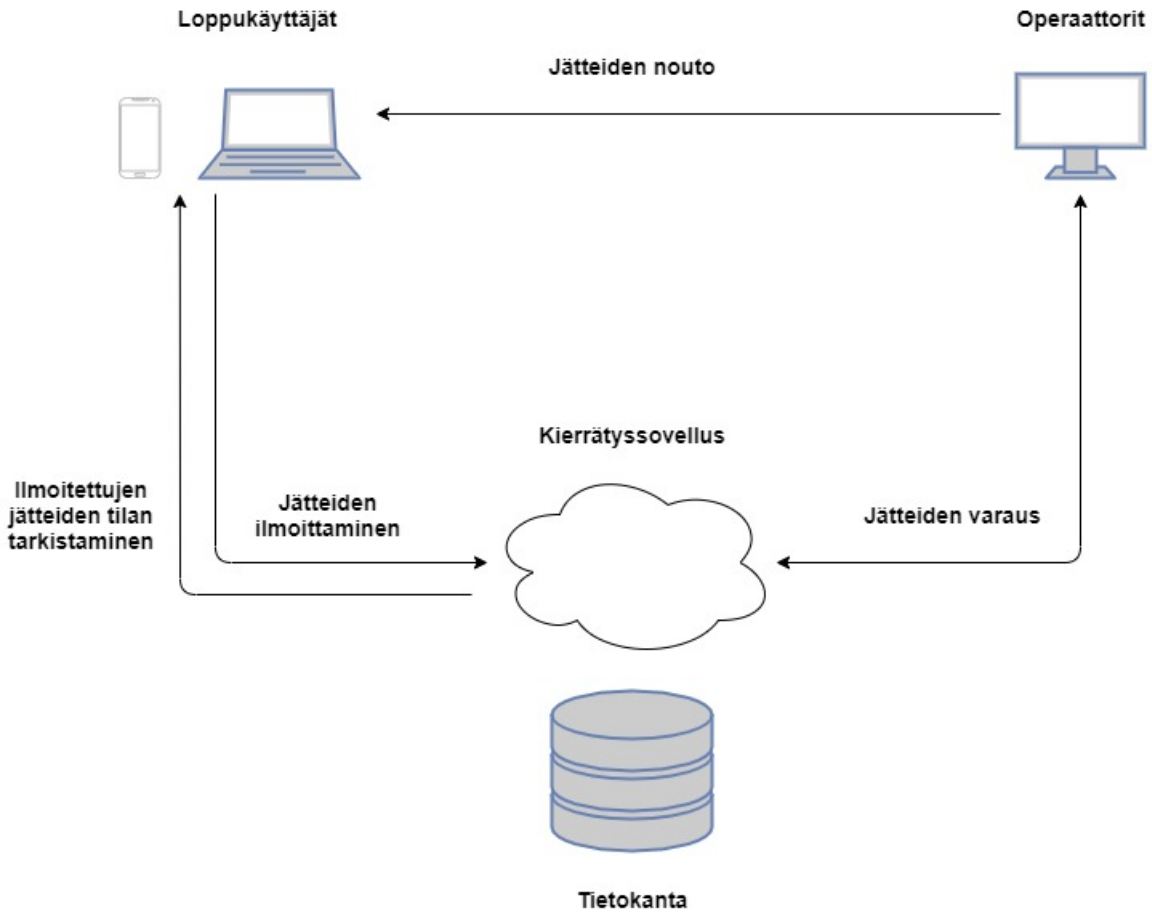
Digiloop on lähinnä SER-jätteiden ja akkujen keräämiseen tarkoitettu palvelu. Palvelussa loppukäyttäjät voivat ilmoittaa omista noudettavista jätteistä mobiililaitteessa toimivan sovelluksen avulla. Palveluun voi ilmoittaa useampia jätteitä kerralla ja lähettää tilauksen vasta kun kaikki jätteet on syötetty. Operaattorit taas voivat ylläpitokäyttöliittymän avulla valita itselle sopivat jäte-erät ja noutaa ne asiakkaalta.

Opinnäytetyön tavoitteena on toteuttaa ylläpitokäyttöliittymä Digiloop-palvelulle. Projektia on tehty syksystä 2017 asti ja tarkoitus on saada toimiva ylläpitokäyttöliittymä testivaiheeseen. Opinnäytetyön toimeksiantaja on LAMK. Lisäksi projektiin osallistui myös Tramel Oy:n edustaja sekä BLTK Banaanilaatikko Oy:n edustaja.

Tässä työssä esitellään React- ja Redux Javascript-kirjastot, tutustutaan niiden arkkitehtuuriin ja toimintaan. Lisäksi esitellään kirjastot React Redux ja Axios ja tutustutaan niiden toimintaan.

2 TOIMINTAYMPÄRISTÖN KUVAUS

Digiloop-kierrätyspalvelun toimintaympäristö koostuu useammasta käyttäjäryhmästä ja toiminnosta, joista keskeisimmät on esitelty kuvassa 1. Tärkeimpiä toimintoja ovat jätteiden ilmoittaminen, varaaminen ja noutaminen.



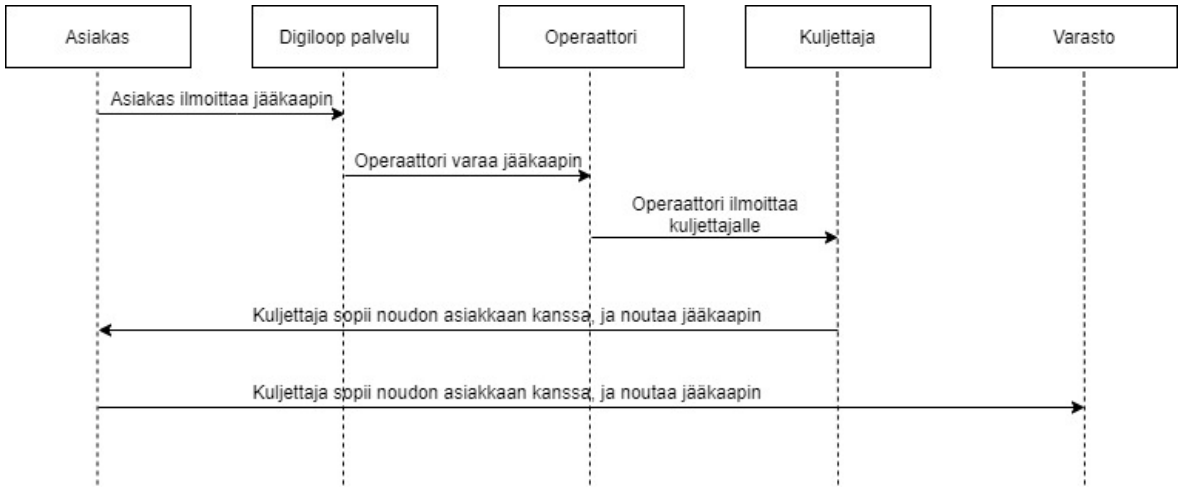
Kuva 1. Kierrätyspalvelun toimintaympäristö

Loppukäyttäjät ovat jätteiden ilmoittajia, joiden oletetaan käyttävän palvelua pääasiallisesti mobiililaitteelle ladattavan sovelluksen avulla, mutta palvelun tekovaiheessa on myös varauduttu verkkoselaimen kautta tehtäviin ilmoituksiin. Loppukäyttäjien keskeisimmät toiminnot ovat jätteiden ilmoittaminen ja jo ilmoitettujen jätteiden tilan seuranta. Kaikki loppukäyttäjien ilmoittamat jätteet tallentuvat sovelluksen tietokantaan.

Operaattoreiden oletetaan käyttävän sovellusta lähinnä tietokoneella, mutta sovellus mahdollistaa myös mobiililaitteella käyttämisen. Operaattoreiden käyttöliittymä lataa kaikkien varattavien jätteiden tiedot tietokannasta ja lisäksi päivittää jätelistaa tarvittaessa. Mahdollistaakseen optimaalisen noutoreitin, operaattorit näkevät jätteiden sijainnin myös

karttanäkymällä, jolloin heidän on helpompi suunnitella mahdollisimman tehokas reitti jätteiden noutoa varten. Operaattoreiden keskeisimmät toiminnot ovat jätteiden varaaminen ja noutaminen. Operaattorit kirjautuvat samaan sovellukseen kuin loppukäyttäjätkin, mutta heidän näkymänsä on erilainen, ja sillä ei voi ilmoittaa jätteitä.

Jätteen elinkaari on esitelty kuvassa 2. Toiminta alkaa siitä, kun asiakas ilmoittaa jätteen, minkä jälkeen jäte siirtyy palveluun. Operaattori varaa jätteen/jätteet karttanäkymää hyväksi käyttäen. Karttanäkymässä pystyy myös näkemään muut lähiympäristössä sijaitsevat jätteet, jolloin operaattori voi varata kerralla useampia jäte-eriä. Kun operaattori on varannut jätteet, hän ilmoittaa siitä kuljettajalle. Kuljettaja sopii noudot ja toimittaa noudetut jätteet varastolle odottamaan jatkokäsittelyä.



Kuva 2. Jätteen elinkaari

Yhteys palveluun tulee toteuttaa REST (Representional State Transfer) -rajapintana, joka hyödyntää http(s)-protokollaa kommunikointiin. Palvelin on yhteydessä tietokantaan ja huolehtii kutsujen vaatimien tietojen välittämisestä. Http-protokolla tarjoaa muun muassa get- ja post-metodeja käytettäväksi.

3 REACT

3.1 Yleistä Reactista

React on Javascript-kirjasto, jota käytetään luomaan sovelluksen käyttöliittymiä. Sen kehitti Jordan Walke, ja se julkaistiin Facebookin toimesta ratkaisemaan joitain suuria, datapohjaisia verkkosivustoja koskevia ongelmia. (Banks & Porcello 2017, 1.)

Se otettiin ensimmäisenä käyttöön Facebookin uutisvirran kommenttiosiossa alkuvuodesta 2012. Kesällä 2012 se otettiin käyttöön Instagram.com:ssa ja loppuvuodesta 2012 Facebookin mainoksissa (Hunt 2015). Tämän jälkeen se julkaistiin vapaasti käytettäväksi (open-sourced) vuonna 2013. Yleisesti ottaen Reactia pidetään V:nä (view) MVC-arkkitehtuurissa (model-view-controller).

Yksi syistä, mikä tekee Reactin käytöstä tehokasta, on se, että React käyttää deklaratiivista ohjelmointia eli selittävän ohjelmoinnin mallia. Ero imperatiivisella ohjelmoinnilla ja deklaratiivisella ohjelmoinnilla on se, että imperatiivinen ohjelmointi (kuva 3) kuvailee, kuinka asiat toimivat vaihe vaiheelta. Deklaratiivinen ohjelmointi taas (kuva 4) kuvailee, mitä halutaan saavuttaa. (Bertoli 2017, 8.)

```
const toLowerCase = input => {
  const output = []
  for (let i = 0; i < input.length; i++) {
    output.push(input[i].toLowerCase())
  }
  return output
}
```

Kuva 3. Imperatiivinen ohjelmointi (Bertoli 2017, 8)

```
const toLowerCase = input => input.map(
  value => value.toLowerCase()
)
```

Kuva 4. Deklaratiivinen ohjelmointi (Bertoli 2017, 9)

Kuvan 3 mukaisella esimerkillä eli imperatiivisella ohjelmoinnilla kerrotaan siis ohjelmalle seuraavasti:

- Luo tyhjä taulukko.
- Ota ensimmäinen alkio ja muunna se pieneksi kirjaimeksi.
- Siirrä se taulukkoon.

- Toista niin kauan alkioiden muuntamista pieniksi kirjaimiksi ja siirtämistä taulukkoon kuin alkioita riittää.
- Tulosta valmis tuotos.

Kuvan 4 mukaisella eli deklaratiivisella ohjelmoinnilla kerrotaan ohjelmalle vain, että muunna kirjaimet pieniksi kirjaimiksi. Tämä vähentää kirjoitettavan koodin määrää ja koodi on helpommin tulkittavissa.

Reactin mukaan deklaratiivisella ohjelmoinnilla tarkoitetaan siis mallia, jossa on helppo luoda interaktiivisia käyttöliittymiä. Sovelluksessa suunnitellaan näkymät jokaiselle tilalle (state), ja React vastaa siitä, että vain oikeat komponentit päivitetään. Näin koodi on ennustettavampaa ja helpompaa testata ja korjata virheitä. (React 2018.)

3.2 Rakenne

Jotta voi luoda React-sovelluksen, täytyy koneella olla asennettuna Node.js. Node.js on JavaScript-koodin suorittamiseen tarkoitettu ajoympäristö (runtime environment). React projekti my-app luodaan create-react-app skriptillä kuvan 5 mukaisesti (Kahler 2019).

```
npx create-react-app my-app
```

Kuva 5. React-sovelluksen luonti

Kun projekti on luotu, projektin kansiorakenne näyttää kuvan 6 mukaiselta. Jotta projektista on mahdollista luoda suoritettava sovellus, ei index.html- ja index.js-tiedostojen kirjoitusasuun saa koskea. Kaikkia muita tiedostoja voi nimetä uudelleen tai tuhota.

```
my-app/
  README.md
  node_modules/
  package.json
  public/
    index.html
    favicon.ico
  src/
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
```

Kuva 6. Reactin kansiorakenne (Abramov 2018)

React käyttää WebPack-moduulia JavaScript-moduulien kokoamiseen. Kun WebPack käsittelee sovellusta, se rakentaa sisäisen riippuvuusgraafin. Se kartoittaa kaikki projektin tarvitsemat moduulit ja luo yhden tai useamman paketin sen perusteella. (WebPack 2020.)

Omia alihakemistoja voi luoda src-kansion sisään. Tämä sen takia, koska Webpack käsittelee nopeampien uudelleenkoontien (rebuilds) takia vain src-kansion sisällä olevia tiedostoja. Niinpä kaikki .js- ja .css-tiedostot täytyy olla sen sisällä, jotta Webpack löytää ne. (Abramov 2018.)

3.3 Komponentit

Komponentit ovat pieniä, uudelleenkäytettäviä osia, jotka palauttavat React-elementin renderöitäväksi sivulle. Yksinkertaisin versio React-komponentista on tavallinen Javascript-funktio, joka palauttaa React-elementin (kuva 7). Komponentit voidaan jakaa erilaisiin toiminnallisuuksiin ja käyttää toisissa komponenteissa. Komponentit voivat palauttaa toisia komponentteja, taulukoita, merkkijonoja ja numeroita. (React 2019a.)

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Kuva 7. Yksinkertainen React-komponentti (React 2019a)

Komponentit ovat siis Reactin rakennuspalikoita ja voivat olla joko funktionaalisia komponentteja (functional components) tai luokallisia komponentteja (class components). Kuvan 7 funktiota kutsutaan funktionaaliseksi komponentiksi, koska se on kirjaimellisesti JavaScript-funktio (React 2019a).

Funktionaaliset komponentit ovat siis funktioita, jotka saavat syötteen ja palauttavat tulosteen. Syöte on props-objekti ja tuloste on komponentin ilmentymä. Funktionaalisilla komponenteilla ei ole sisäistä tilaa, eikä niihin näin ollen voi viitata. Lisäksi niillä ei ole elinkaarimetodeita (lifecycle methods). (Wieruch 2017, 56.)

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

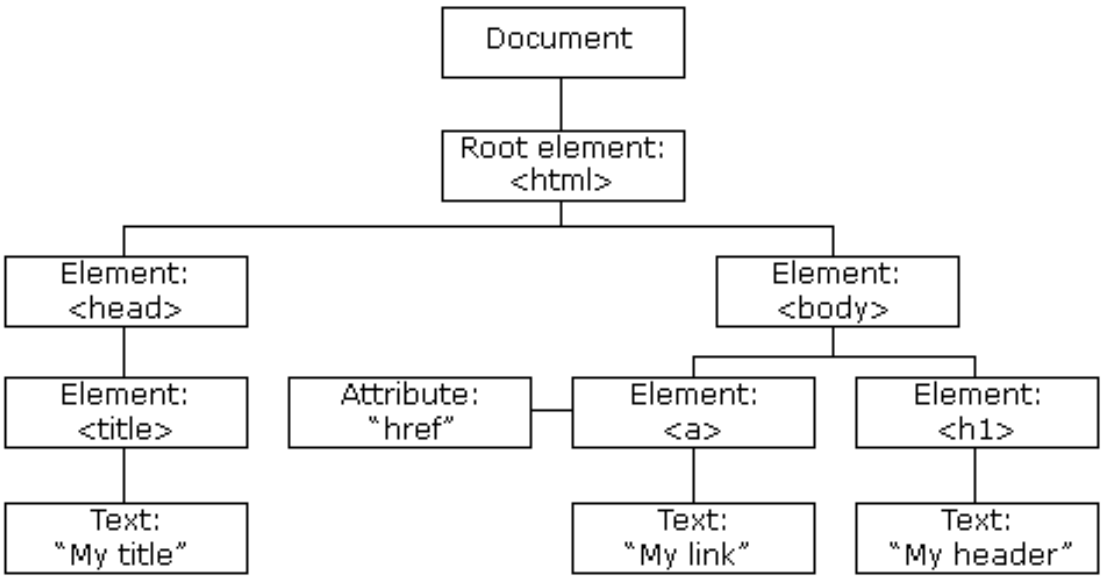
Kuva 8. Luokallinen komponentti (React 2019a)

Luokallisilla komponenteilla (kuva 8) taas voidaan käyttää elinkaarimetodeita ja niissä voidaan tallentaa ja käsitellä tiloja (Wieruch 2017, 56). Tila on samanlainen kuin ominaisuus, mutta se on yksityinen ja komponentti hallitsee sitä täysin (React 2019b).

Pääsääntönä pidetään, että funktionaalisia komponentteja pitäisi käyttää aina kun ei tarvita komponentin sisäistä tilaa eikä elinkaarimetodeita (Wieruch 2017, 56). Luokallisilla komponenteilla on huomattavasti suurempi merkkimäärä, joten niiden käyttäminen liiallisesti ja tarpeettomasti voi vaikuttaa haitallisesti suorituskykyyn, koodin luettavuuteen, ylläpidettävyyteen ja testattavuuteen (Kagga 2018).

3.4 Virtual DOM

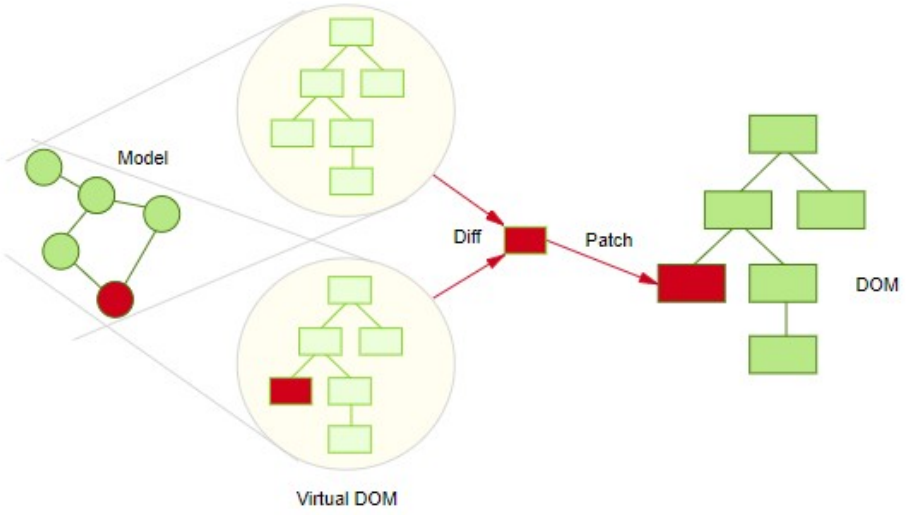
Document Object Model (DOM) on ohjelmointirajapinta HTML- ja XML-tiedostoille. Se määrittelee tiedostojen loogisen rakenteen ja tavan, jolla tiedostoihin pääsee käsiksi, ja sen, kuinka niitä voidaan muokata. DOM:n avulla ohjelmoijat voivat rakentaa tiedostoja, navigoida niiden rakenteessa ja lisätä, muokata tai poistaa sisältöä. DOM:n tietorakennetta kuvataan puun kaltaisena rakenteena (kuva 9). (Robie 1998.)



Kuva 9. DOM puu (w3schools.com 2019)

Ongelma todellisen DOM-rakenteen kanssa on se, että DOM-puut ovat nykyään valtavia. Koska ollaan menossa enemmän ja enemmän kohti dynaamisia web-sovelluksia, täytyy DOM-puuta muokata tauotta ja paljon. Tämä on suorituskyvyllinen ja kehityksellinen ongelma. (Krajka 2015.)

Virtuaalinen DOM (VDOM) on ohjelmointikonsepti, missä virtuaalinen UI:n esitys pidetään muistissa ja synkronoidaan todellisen DOM:n kanssa kirjastolla kuten ReactDOM (kuva 10). Tätä prosessia kutsutaan sovitteluksi (reconciliation). (React 2019c.)

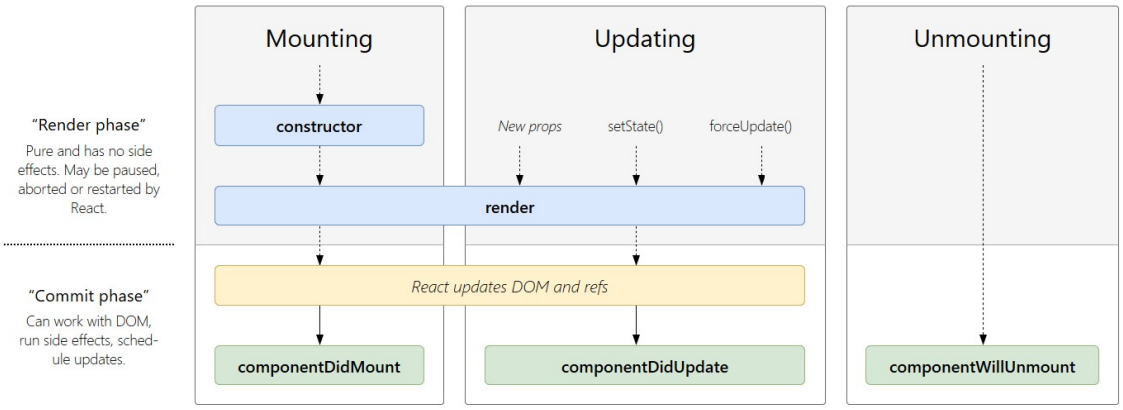


Kuva 10. Virtual DOM (Parviainen 2015)

Tämä lähestymistapa mahdollistaa Reactin deklaratiivisen ohjelmointirajapinnan. Reactille kerrotaan, missä tilassa UI:n halutaan olevan. Ja React varmistaa, että DOM vastaa tätä. Tämä poistaa ominaisuuksien manipuloinnin, tapahtumien käsittelyn ja DOM.n manuaalisen päivittämisen, joita muuten olisi käytettävä sovelluksen rakentamiseen. (React 2019c.)

3.5 Elinkaarimetodit

Jokaisella luokallisella komponentilla on useita elinkaarimetodeita, joita voi käyttää koodin suorittamiseksi prosessin tiettyinä aikoina. Kuvasta 11 voidaan nähdä yleisimmin käytetyt elinkaarimetodit sekä metodien jako kolmeen eri vaiheeseen: kiinnitys (mounting), päivitys (update) ja irrotus (unmounting). (React 2019d.)



Kuva 11. Reactin elinkaarimetodit (Maj 2019)

Kiinnitysvaiheessa voidaan kutsua seuraavia metodeja:

- constructor() -metodia kutsutaan ennen kiinnitysvaihetta. Tyypillisesti konstruktoria käytetään vain kahteen tarkoitukseen, alustamaan paikallinen tila tai sitomaan tapahtumien käsittelijöiden metodeita instansseihin.
- render() -metodi on ainoa vaadittu metodi luokalliselle komponentille. Se tutkii this.props- ja this.state -objektit, ja palauttaa esimerkiksi React elementin, joka on yleensä tehty JSX:ä käyttäen.
- componentDidMount() -metodia kutsutaan välittömästi kun komponentti on kiinnitetty. DOM-solmuja (DOM nodes) edellyttävä alustus tulisi suorittaa tässä metodissa. Myös jos komponenttille tarvitsee ladata tietoja etäpisteestä, tämä on paikka luoda verkkokutsu.

Päivityksen voi aiheuttaa ominaisuuksien tai tilojen muutos. Näitä metodeja kutsutaan, kun komponenttia renderöidään uudestaan:

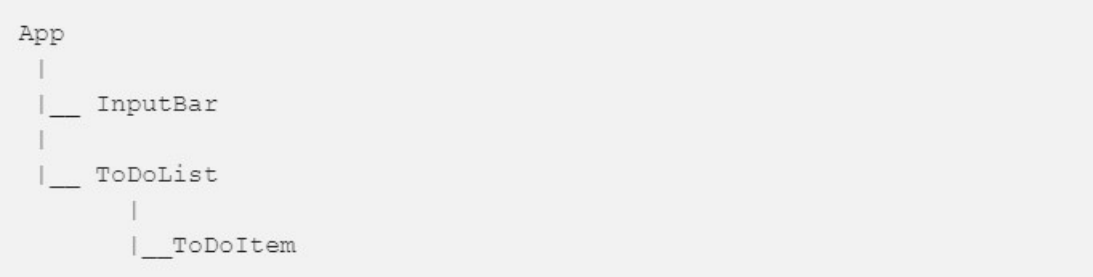
- render() -metodia päivittää komponentin tilan.
- componentDidUpdate() -metodia kutsutaan heti päivityksen jälkeen. Tätä metodia ei kutsuta alustavassa renderöinnissä. Tämä on hyvä paikka tehdä verkkokutsuja, kun verrataan nykyisiä ominaisuuksia edellisiin ominaisuuksiin.

Irrotuksessa kutsutaan vain componentWillUnmount() –metodia, jolla irrotetaan komponentti DOM:sta. Tätä kutsutaan välittömästi ennen kuin komponentti irrotetaan ja hävitetään. Tässä voi suorittaa tarvittavat siivoukset, kuten verkkokutsujen peruuttamiset tai componentDidMount()-metodissa luotujen tilausten peruuttaminen. (React 2019d.)

3.6 Komponenttien välinen tiedonsiirto

Komponenttien välistä tiedonsiirtoa voidaan tehdä kahdella tavalla, äitikomponentilta lapsikomponentille (parent => child) tai lapsikomponentilta takaisin äitikomponentille (child => parent). Esimerkkinä käytetään kuvan 12 mukaista komponenttien välistä suhdetta, jossa äitikomponentin (App) alla on kaksi lapsikomponenttia (InputBar ja ToDoList) ja ToDoList-komponentin alla on yksi lapsikomponentti, jonka suhteen myös ToDoList-komponentti on äitikomponentti. (Pardee 2016.)

Rinnakkaisten (child => child) komponenttien väliseen tiedonsiirtoon on aina käytettävä äitikomponenttia välittäjänä. Joten siirrettäessä ToDoList-komponentilta dataa InputBar-komponentille, on App-komponenttia käytettävä välittäjänä. (Pardee 2016.)



Kuva 12. Komponenttien välinen suhde (Pardee 2016)

3.6.1 Parent to child

Äitikomponentilta lapsikomponentille on helpoin suunta siirtää dataa Reactissa. Kuvan 12 mukaisessa komponenttien välisessä suhteessa äitikomponentilla on dataa, jota lapsikomponentti tarvitsee, joten se voidaan siirtää yksinkertaisesti props-ominaisuuksien avulla (kuva 13). Sen jälkeen ToDoList-komponentissa voidaan käyttää this.props.listNameFromParent muuttujaa päästäkseen käsiksi dataan. (Pardee 2016.)

```
class App extends React.Component {

    render() {

    [... somewhere in here I define a variable listName which I
think will be useful as data in my ToDoList component...]

        return (
            <div>
                 <InputBar/>
                 <ToDoList listNameFromParent={listName}/>
            </div>
        );

    }
}
```

Kuva 13. Kommunikointi äitikomponentilta lapsikomponentille (Pardee 2016)

3.6.2 Child to parent

Datan siirtäminen lapsikomponentilta äitikomponentille vaatii hieman enemmän työtä. Jotta datan saaminen onnistuu, täytyy ensin määritellä takaisinkutsu-funktio (callback) äitikomponenttiin (ToDoList), joka vastaanottaa datan lapsikomponentilta. Sen jälkeen välitetään takaisinkutsufunktio lapsikomponentille props-ominaisuutena. Tässä tapauksessa funktio nimeltä myCallback (kuva 14).

```
class ToDoList extends React.Component {

    myCallback = (dataFromChild) => {
        [...we will use the dataFromChild here...]
    },

    render() {
        return (
            <div>
                 <ToDoItem callbackFromParent={this.myCallback}/>
            </div>
        );

    }
}
```

Kuva 14. Takaisinkutsu-funktio (Pardee 2016)

Tämän jälkeen voidaan lapsikomponentissa kutsua takaisinkutsu-funktiota this.props. callbackFromParent ja siirtää sinne tarvittava data (kuva 15).

```
class ToDoItem extends React.Component{

    someFn = () => {
        [...somewhere in here I define a variable listInfo which
I think will be useful as data in my ToDoList component...]

        this.props.callbackFromParent(listInfo);
    },

    render() {
        [...]
    }
};
```

Kuva 15. Lapsikomponentin takaisinkutsu-funktio (Pardee 2016)

3.7 JSX

JSX on JavaScript-laajennus, jonka avulla voidaan määrittää React-elementtejä syntaksilla, joka näyttää samanlaiselta kuin HTML. Facebookin React-tiimi julkaisi sen samaan aikaan, kun he julkaisivat Reactin. Heidän tarkoituksenaan oli tehdä Reactista luettavampaa. (Banks & Porcello 2017, 81.)

React suosittelee käyttämään JSX:ää kuvailemaan, miltä UI:n tulisi näyttää, mutta se ei ole pakollista. Kuvassa 16 nähdään molemmat tavat luoda React-elementtejä, ylemmässä käytetään JSX:ää, alempi on kirjoitettu ilman sitä. JXS:ssä elementin tyyppi määritellään <tageilla> ja elementin lapset voidaan lisätä aloitus- (<>) ja lopetustagin (< />) sisään. (React 2019e.)

```
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);
```

```
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

Kuva 16. JSX (React 2019e)

JSX:n kanssa voidaan käyttää myös mitä tahansa JavaScript-lausekkeita aaltosulkujen sisällä. Kääntämisen (compilation) jälkeen, JSX-lausekkeista tulee tavallisia JavaScript-objekteja. Tämä tarkoittaa, että JSX:ää voidaan käyttää if-lauseiden ja for-silmukoiden sisällä, määrittää muuttujille, hyväksyä parametreina ja palauttaa funktiosta. JSX:ää suositellaan jakamaan useammalle riville luettavuuden takia. (React 2019e.)

3.8 Tiedonsiirto taustajärjestelmien kanssa

Tiedonsiirtoon taustajärjestelmien on tarkoitus käyttää promise-pohjaista Axios http-client -kirjastoa, joka toimii sekä selaimessa että Node.js-ympäristössä. Axioksen tärkeimpiin ominaisuuksiin kuuluu mahdollisuus tehdä XMLHttpRequest-pyyntöjä selaimesta ja http-kutsuja Node.js-ympäristöstä. Se tukee Promise API:a ja osaa muuntaa pyyntö- ja vastaustiedot, peruuttaa pyynnöt ja tehdä automaattiset muunnokset JSON-tiedoille. (GitHub 2019.)

Axios asennetetaan komentokehotetta käyttäen ja Axios tukee muun muassa get- ja post-metodien (kuva 17) lisäksi put-, patch- ja delete-metodeja. Vastauksessa Axios palauttaa seuraavat asiat:

- data, sisältää vastauksen minkä palvelin lähettää
- status, sisältää HTTP-tilakoodin (200, onnistuessaan)
- statusText, sisältää HTTP-tilatekstin (OK, jos tilakoodi on 200)
- headers, sisältää otsikkotiedot palvelimelta
- config, sisältää konfigurointitiedot, jotka Axiokselle on annettu.

Ja käytettäessä catch-metodia, virheen sattuessa palvelin palauttaa vastauksen, joka sisältää virheen tilakoodin. (GitHub 2019.)

```
// GET request
axios.get('/user?ID=12345')
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });



// POST request
axios.post('/user', {
    firstName: 'Fred',
    lastName: 'Flintstone'
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

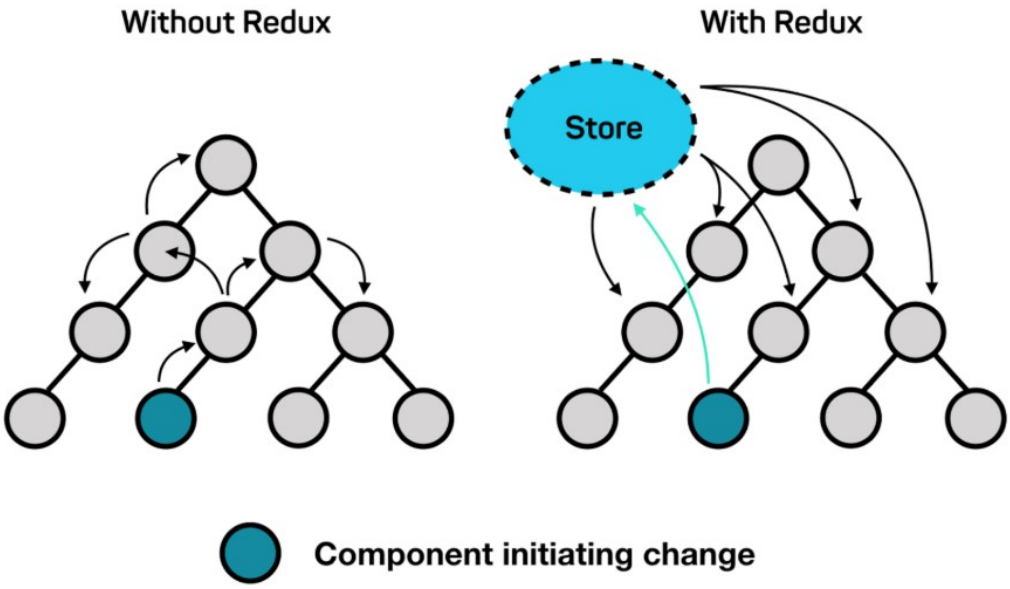
Kuva 17. GET- ja POST-metodit (GitHub 2019)

4 REDUX

4.1 Yleistä Reduxista

Redux on ennustettavissa oleva tilasäiliö JavaScript-sovelluksille. Sen tarkoitus on helpottaa kirjoittamaan sovelluksia, jotka toimivat eri ympäristöissä. Lisäksi sitä voi käyttää minkä tahansa näkymäkirjaston (view library) kanssa. (Redux 2018a.)

Reduxin kehitti Dan Abramov ja Andrew Clark, ja Dan Abramov esitteli sen ensi kertaa React Europe -tapahtumassa 2015 ja se kehitettiin ratkaisemaan ohjelman tilan hallinnan ongelmia (Wieruch 2018). Sen tärkein ominaisuus on, että kaikki tilat on tallennettu yhteen paikkaan. Näin sovelluksen jokaisella komponentilla on suora pääsy sovelluksen tilaan tarvittaessa, eikä tarvitse lähettää tiloja erikseen lapsikomponenteille tai käyttää takaisinkutsuja tietojen saamiseksi takaisin äitikomponentille (kuva 18). (Erikson 2017.)



Kuva 18. Redux (Weck 2017)

4.2 Perusperiaatteet

Reduxia voidaan kuvata kolmella perusperiaatteella:

- Datalla on vain yksi lähde.
- Tila on vain luettava.
- Muutoksia tehdään puhtailla funktioilla.

Datalla on vain yksi lähde (single source of truth) tarkoittaa, että koko sovelluksen tila on tallennettu tilapuuhun yhteen paikkaan, yhteen tietovarastoon (store). Tämä helpottaa universaalien sovellusten luomista. Ja koska koko sovelluksen tila on tallennettu yhteen tilapuuhun, helpottaa se myös sovelluksen debuggausta ja tutkimista. (Redux 2018b.)

Tila on vain luettava (state is read-only) tarkoittaa, että ainoa tapa muuttaa tilaa on lähettää toiminto (action), eli objekti, joka kuvaa mitä tapahtuu. Näin varmistetaan, etteivät näkymät tai verkon takaisinkutsut koskaan pääse kirjoittamaan suoraan tilaan, vaan sen sijaan ne ilmaisevat aikomuksensa muuttaa tilaa. (Redux 2018b.)

Muutoksia tehdään puhtailla funktioilla (changes are made with pure functions) tarkoittaa, että tilan muutokset toteutetaan reducer-funktioilla. Reducerit ovat vain funktioita, jotka ottavat edellisen tilan sekä toiminnon ja palauttavat seuraavan tilan. Koska reducerit ovat vain funktioita, niiden kutsumisjärjestystä voidaan ohjata, siirtää ylimääräistä dataa niiden avulla tai tehdä uudelleenkäytettäviä reducereita tavallisiin tehtäviin, kuten sivuttamiseen. (Redux 2018b.)

4.3 Rakenne

Redux on saatavilla NPM-pakettina ja se asennetaan komentokehotetta käyttäen (Kuva 19). Redux on pieni tiedosto kooltaan (2kB, mukaan lukien riippuvuudet). (Redux 2018a).

```
npm install --save redux
```

Kuva 19. Redux asentaminen (Redux 2018a)

Koska Redux on vain tietovarasto-kirjasto, sillä ei ole tiettyä rakennesuositusta. On olemassa kuitenkin joitain yleisiä malleja, joita Redux-kehittäjät yleensä käyttävät:

- Rails-tyyli: erilliset kansiot toiminnoille, vakioille, reducereille, storelle ja komponenteille
- Domain-tyyli: erilliset kansiot toimintoa tai domainia varten, mahdollisesti alikansioita tiedostotyyppiä kohden
- ”Ducks”: vastaava kuin domain-tyyli, mutta toiminnot ja reducerit usein samassa tiedostossa.

4.4 Actions

Toiminnot (actions) ovat tietoja, jotka lähettävät dataa sovelluksesta tietovarastoon. Ne ovat ainoa tietolähde tietovarastolle, ja ne lähetetään store.dispatch()-metodilla. Toiminnot (kuva 20) ovat tavallisia JavaScript-objekteja, joilla täytyy olla tyyppiarvo (type), joka

ilmaisee, minkä tyyppinen toiminto suoritetaan. Tyyppiarvon lisäksi toiminnon rakenne on käyttäjän päätettävissä, mutta on suositeltavaa lähettää niin vähän dataa kuin mahdollista. (Redux 2018c.) Aina kun toiminto lähetetään, se käy läpi kaikki Reduxin reducerit (Wieruch 2018). Toiminnon tyyppiarvo on yleensä isoilla kirjaimilla kirjoitettu merkkijono, joka kuvaa toimintaa (Banks & Porcello 2017, 177). Reduxin mukaan toiminnot kuvaavat vain mitä on tapahtunut, mutta eivät kuvaa miten sovelluksen tila muuttuu (Redux 2018d).

```
{ type: 'ADD_TODO', text: 'Use Redux' }
{ type: 'REMOVE_TODO', id: 42 }
{ type: 'LOAD_ARTICLE', response: { ... } }
```

Kuva 20. Redux actions (Redux 2018c)

Action creators (kuva 21) on tapa toteuttaa kyseiset toiminnot. Ne ovat funktioita, jotka luovat toimintoja. Reduxissa, action creatorsit yksinkertaisesti palauttavat toiminnon, minkä takia ne ovat siirrettäviä ja helppoja testata. (Redux 2018c.)

Action creatorsit eivät ole pakollisia, mutta niillä on helpompi pitää arkkitehtuuri selkeämpänä. Vaihtoehtoisesti voidaan määrittää toiminto joka kerta uudelleen, mutta action creatorseilla voidaan luoda funktio, jota voidaan uudelleenkäyttää useammassa tilanteessa. Funktiolle siirretään hyötykuorma, jolloin se palauttaa objektin (Wieruch 2018).

```
export function addTodo(text) {
  return {
    type: 'ADD_TODO',
    text
  }
}
```

Kuva 21. Action creators (Redux 2018c)

4.5 Reducer

Reducerit määrittävät kuinka sovelluksen tila muuttuu tietovarastoon lähetettyjen toimintojen vastauksena (Redux 2018d). Reducerit (kuva 22) ovat funktioita, jotka ottavat argumentteina nykyisen tilan ja toiminnon ja käyttävät niitä uuden tilan luomiseen ja palauttamiseen (Banks & Porcello 2017, 190).

Reduxin dokumentaation (Redux 2018d) mukaan reducerin sisällä ei tulisi ikinä tehdä seuraavia asioita:

- Muuttaa sen argumentteja.

- Suorittaa ohjelman osia, joilla on sivuvaikutuksia, kuten API-kutsuja ja reitityksen muutoksia.
- Kutsua ei-puhtaita funktioita, esim. Math.random().

```
(prevState, action) => newState
```

Kuva 22. Reducer (Wieruch 2018)

Reducer on puhdas funktio, joka tuottaa aina saman tuloksen, jos syöte pysyy samana. Sillä ei ole sivuvaikutuksia, koska se on vain syöte- / tulostoiminto. Reducerilla on kaksi syötettä, tila ja toiminto, joista tila on aina globaali tilaobjekti tietovarastosta ja toiminto on lähetetty toiminto, jolla on tyyppi ja mahdollisesti hyötykuorma. (Wieruch 2018.)

4.6 Store

Reduxissa on vain yksi tietovarasto (store), joka pitää sisällään yhden globaalin tilaobjektin (Wieruch 2018). Reduxin (Redux 2018e) mukaan tietovarastolla on seuraavat tehtävät:

- pitää yllä sovelluksen tilan
- mahdollistaa pääsyn tilaan
- mahdollistaa tilan päivittämisen
- rekisteröi kuuntelijat (listeners)
- huolehtii kuuntelijoiden poistamisesta.

Redux tietovarasto on ensimmäinen kirjastoriippuvuus, jonka Redux tarvitsee. Store-objektin toiminnot tuodaan Redux-kirjastosta import-komennolla (kuva 23). Tämän jälkeen sitä voidaan käyttää luomaan tietovaraston ainoa instanssi. Instanssi luodaan createStore-funktiolla. CreateStore-funktio tarvitsee vain yhden pakollisen argumentin, reducerin, tässä esimerkissä todoApp, joka on tuotu reducer-kansiosta. Lisäksi on mahdollista syöttää toinen valinnainen argumentti, joka on alkutila (initial state). (Wieruch 2018.)

```
import { createStore } from 'redux'
import todoApp from './reducers'

const store = createStore(todoApp)
```

Kuva 23. Redux store (Redux 2018e)

5 REACT REDUX

5.1 Yleistä React Redux -kirjastosta

React Redux on Redux-tiimin ylläpitämä kirjasto Reduxin käyttämiseen React-sovelluksessa. Se sallii React-komponentin lukea tietoa Reduxin tietovarastosta ja välittää toimintoja tietovarastoon tilan päivittämiseksi. React Redux tarjoaa Provider-komponentin, millä Reduxin tietovarasto saadaan koko sovelluksen käyttöön ja connect-funktion, jolla komponentit saadaan yhdistettyä tietovarastoon. (React Redux 2019a.)

React-komponentit voidaan jakaa kahteen eri kategoriaan: esityksellisiin komponentteihin (presentational components) ja säiliökomponentteihin (container components). Esityksselliset komponentit ovat komponentteja, jotka renderöivät vain UI-elementtejä. Ne eivät liity tiiviisti mihinkään data-arkkitehtuuriin, vaan niiden tarkoitus on vain vastaanottaa dataa props-ominaisuuksina ja lähettää dataa äitikomponentille takaisinkutsu-funktion avulla. Ne huolehtivat puhtaasti UI:sta ja niitä voi käyttää uudestaan muualla sovelluksessa. (Banks & Porcello 2017, 221.)

Container-komponentit taas ovat komponentteja, jotka yhdistävät esitykselliset komponentit dataan. Ne tuottavat esitykselliset komponentit kartoittamalla ominaisuudet, tilan ja takaisinkutsu-funktion ominaisuuksiin, ja edelleen tietovaraston dispatch-metodiin. Tällä arkkitehtuurilla on useita etuja: esitykselliset komponentit ovat uudelleenkäytettäviä, niitä on helppo vaihtaa ja testata ja niitä voidaan yhdistää UI:n luomiseksi. Container-komponentit eivät vaikuta UI:hin ollenkaan, niiden päätarkoitus on yhdistää esitykselliset komponentit data-arkkitehtuuriin. (Banks & Porcello 2017, 221.)

React Redux-kirjasto on myös Reduxin luojan Dan Abramovin tekemä ja se asennetaan komentokehotteella npm:n avulla. Sitä ei ole pakko käyttää Reduxin kanssa, mutta se vähentää koodin monimutkaisuutta ja voi auttaa rakentamaan sovelluksen nopeammin. (Banks & Porcello 2017, 223.)

5.2 Provider ja connect

Provider-komponentti (kuva 24) voidaan ottaa käyttöön missä päin komponenttipuuta tahansa. Mutta eniten hyötyä siitä on, kun sen sisään kääritään root-komponentti, tässä tapauksessa App-komponentti. Tämä siitä syystä, että Provider-komponentti antaa tietovaraston käyttöön kaikille tiedostorakenteessa alapuolella oleville ns. lapsikomponenteille. (React Redux 2019b.)

```
ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
)
```

Kuva 24. Provider (React Redux 2019b)

React Reduxin mukana tulevalla connect-funktiolla voidaan ui-komponentit pitää puhtaasti esityksellisinä. Tämä onnistuu luomalla container-komponentteja. Container-komponentti kartoittaa tietovaraston nykytilan ja välittää sen esitykselliselle komponentille. Lisäksi se kartoittaa myös tietovaraston dispatch-funktion takaisinkutsun ominaisuudet. Connect-funktio on Higher Order -funktio, millä tarkoitetaan funktiota, joka voi ottaa toisen funktion parametrina tai palauttaa funktion tuloksena. Connect-funktio palauttaa funktion, joka palauttaa komponentin. (Banks & Porcello 2017, 224-225.)

Connect-funktio ei muokkaa komponenttiluokkaa, joka sille on siirretty, vaan palauttaa uuden kytketyn komponenttiluokan käytettäväksi. Connect-funktio voi käyttää neljää eri muuttujaa, jotka ovat vapaavalintaisia. Taulukossa 1 on esitetty käytettävissä olevat muuttujat ja niiden tyypit.

Taulukko 1. Connect-funktio (React Redux 2019c)

Muuttuja:	Tyyppi:
mapStateToProps	Funktio
mapDispatchToProps	Funktio / Objekti
mergeProps	Funktio
options	Objekti

5.2.1 MapStateToProps

MapStateToProps-funktio (kuva 25) voi ottaa maksimissaan kaksi muuttujaa, tila ja omat props-ominaisuudet (ownProps). Jos tämä funktio on määritelty, uusi komponentti tilaa Redux tietovaraston päivityksiä, eli joka kerta kun tietovarasto päivittyy, mapStateToProps-funktiota kutsutaan. Funktion tuloksen täytyy olla tavallinen objekti, joka yhdistetään komponentin props-ominaisuuksiin. Jos funktion ei haluta vastaanottavan päivityksiä tietovarastosta, tulee sen tilalle asettaa null tai undefined. (React Redux 2019c.)

```
function mapStateToProps(state, ownProps) {
  return { todos: state.todos[ownProps.userId] }
}
```

Kuva 25. MapStateToProps-funktio (React Redux 2019c)

Jos mapStateToProps-funktio on määritelty ottamaan kaksi muuttujaa, ensimmäinen muuttuja sisältää tietovaraston tilan ja toinen kytketyn komponentin omat props-ominaisuudet, ja sitä uudelleen kutsutaan aina, kun kytketty komponentti saa uusia props-ominaisuuksia. (React Redux 2019c.)

5.2.2 MapDispatchToProps

Jos mapDispatchToPropsia käytetään objektina, kaikkien sen sisällä olevien funktioiden oletetaan olevan Reduxin action creatorseja. Objekti, jolla on samat funktion nimet, mutta jokainen niistä kääritään dispatch-kutsuun, jotta niitä voidaan kutsua suoraan, yhdistetään komponentin props-ominaisuuksiin. (React Redux 2019c.)

```
function mapDispatchToProps(dispatch) {
  return {
    todoActions: bindActionCreators(todoActionCreators, dispatch),
    counterActions: bindActionCreators(counterActionCreators, dispatch)
  }
}
```

Kuva 26. MapDispatchToProps-funktio (React Redux 2019c)

Funktiota (kuva 26) käytettäessä sille annetaan dispatch-metodi ensimmäisenä muuttujana. Miten objekti palautetaan, on käyttäjän päätettävissä, siihen voi käyttää esimerkiksi bindActionCreators():a apuna. Jos mapDispatchToProps-funktiolle syötetään kaksi muuttujaa, dispatch-metodi on ensimmäinen ja omat props-ominaisuudet toinen muuttuja, ja sitä uudelleen kutsutaan aina kun kytketty komponentti saa uusia props-ominaisuuksia. (React Redux 2019c.)

5.2.3 MergeProps

MergeProps-funktiolle välitetään mapStateToProps-funktion ja mapDispatchToProps-objektin/funktion tulokset ja omat props-ominaisuudet. Objekti, joka palautetaan tästä funktiosta, välitetään kytketylle komponentille. Tätä toimintoa käytetään, kun halutaan vaan osa props-ominaisuuksista välittää kytketylle komponentille. Jos tätä ei käytetä, kytketty komponentti saa käyttöönsä omat props-ominaisuudet, mapStateToProps-funktion props-

ominaisuudet ja mapDispatchToProps-objektin/funktion props-ominaisuudet. (React Redux 2019c.)

```
function mergeProps(stateProps, dispatchProps, ownProps) {
  return Object.assign({}, ownProps, {
    todos: stateProps.todos[ownProps.userId],
    addTodo: (text) => dispatchProps.addTodo(ownProps.userId, text)
  })
}
```

Kuva 27. MergeProps-funktio (React Redux 2019c)

5.2.4 Options

React Reduxin (2019c) mukaan options-objektin avulla voidaan mukauttaa vielä lisää connect-funktion toimintaa, vaihtoehtoina on:

- pure: välttää uudelleen renderöintiä ja kutsuja mapStateToPropsiin, mapDispatchToPropsiin ja mergePropsiin tekemällä yhtäsuuruustarkastuksia
- areStateEqual: vertailee tulevan tietovaraston tilaa edelliseen arvoon
- areOwnPropsEqual: vertailee tulevia props-ominaisuuksia edelliseen arvoon
- areStatePropsEqual: vertailee mapStateToPropsin tulosta edelliseen arvoon
- areMergedPropsEqual: vertailee mergePropsin tulosta edelliseen arvoon.

6 DIGILOOP-KIERRÄTYSPALVELU

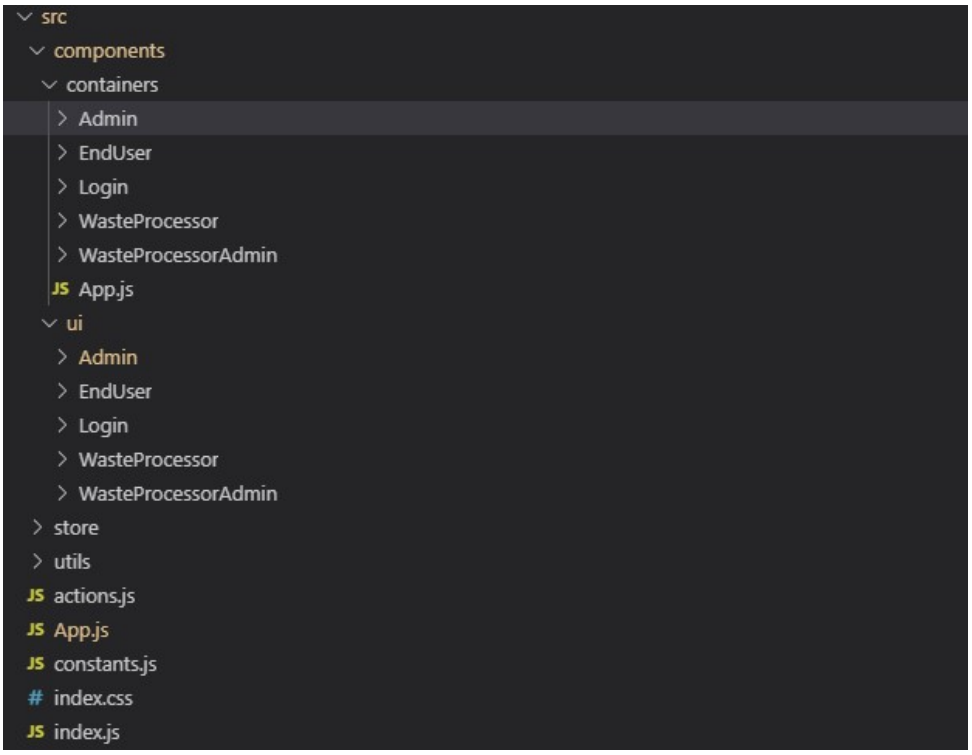
6.1 Toteutus

Digiloop toteutettiin käyttäen Microsoftin ilmaista Visual Studio Code-tekstieditoria. Palvelu on suunniteltu tehostamaan akku-, SER- ja tietoturvajätteiden kierrätystä. Sitä käyttämällä yksityiset henkilöt voivat ilmoittaa jätteistään, joista haluaisivat eroon.

Ylläpitokäyttöliittymän avulla kierrätysyritykset voivat valita mitä tavaraa he haluavat noutaa ja näkevät missä kyseiset tavarat ovat. Näin yritykset voivat suunnitella optimoidun ja kustannustehokkaan noutoreitin. Ylläpitokäyttöliittymä sisältää kolme eri toimialuetta, admin, operaattori ja operaattorin työntekijät, joilla kaikilla on eri oikeudet ja näkymät sovellukseen.

6.2 Rakenne

Sovellukseen luotiin kuvan 28 mukainen tiedostorakenne. Tiedostorakenne muokattiin siten, että jokaisella toimialueella on oma kansiorakenteensa. Utils-kansio sisältää palvelimelle tehtävät kutsut. Toiminnot ovat omassa tiedostossaan (actions.js) ja reducer-funktiot omassa tiedostossaan, joka löytyy store-kansion alta.

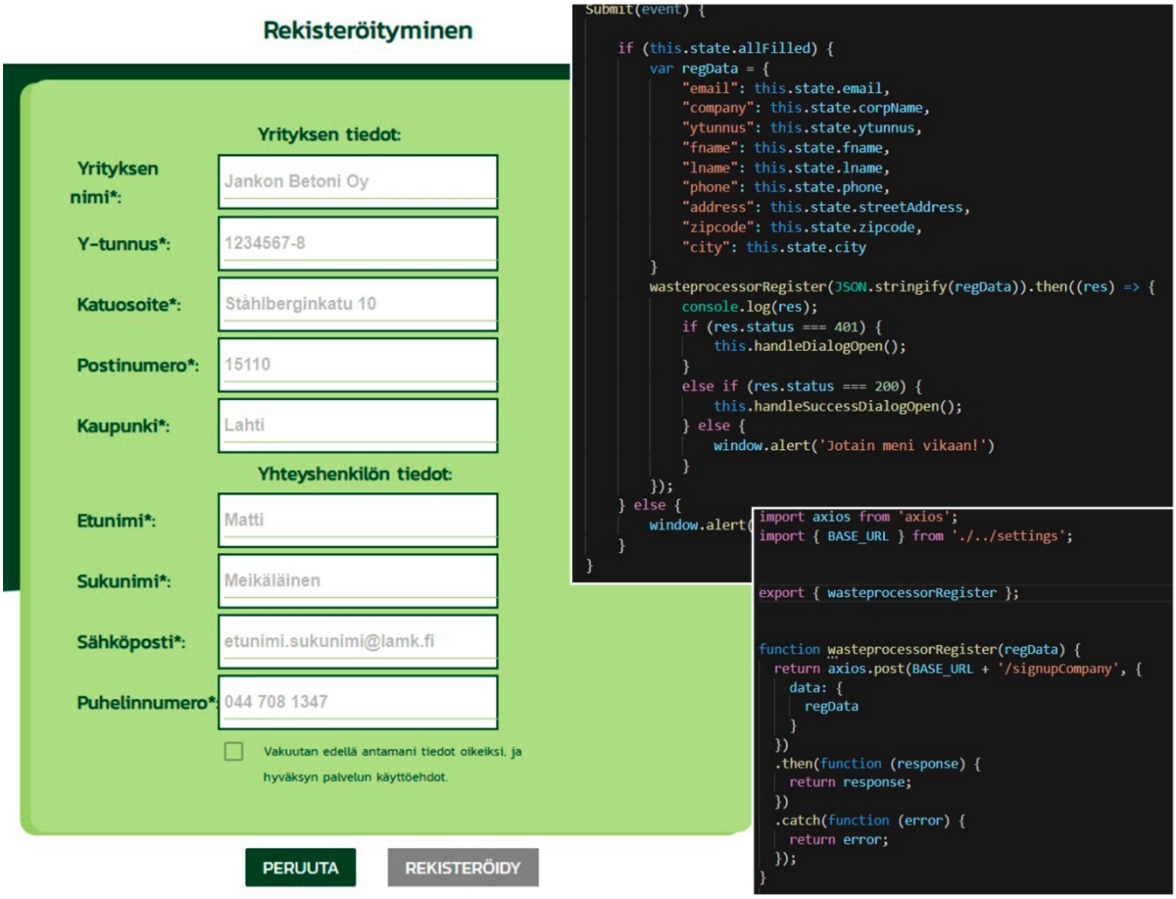


Kuva 28. Sovelluksen tiedostorakenne

Toimialueiden tiedostot on lisäksi jaettu erikseen container- ja ui-hakemistoihin. Container-hakemiston komponentit sisältävät kytkennät Reduxin tietovarastoon, ja ui-hakemiston komponentit sisältävät käyttöliittymässä näytettävän sivun. Container-komponentit huolehtivat props-ominaisuuksien välittämisestä tietovaraston ja käyttöliittymän välillä, ja lisäksi container-komponentti huolehtii tehtävistä muutoksista sovelluksen tilaan.

6.3 Rekisteröityminen

Sovelluksessa on kaksi eri rekisteröitymismahdollisuutta, loppukäyttäjille ja yrityskäyttäjille omansa. Ero näillä kahdella on se, että loppukäyttäjät pääsevät rekisteröitymisen jälkeen suoraan käyttämään sovellusta, kun taas yrityskäyttäjät pitää adminin varmistaa ja hyväksyä, ennen kuin tiliä voi käyttää. Tällä tavalla voidaan varmistaa, että kyseessä on oikea yritys.



Kuva 29. Rekisteröityminen

Yrityspuolen tilille rekisteröidytään kuvan 29 mukaisella sivulla. Kun kaikki tiedot on syötetty, tarkistamiseen käytetään elinkaarimetodia componentDidUpdate(), jota kutsutaan joka kerta kun jotain päivitetään. Kun käyttöehdot on hyväksytty, aktivoituu rekisteröidy nappi, jota painettaessa käynnistyy kuvassa oikealla ylhäällä oleva submit-funktio.

Submit-funktio ottaa kaikki syötetyt tiedot talteen ja yhdistää ne yhdeksi objektiksi, tässä tapauksessa regData. RegData sisältää kaikki tarvittavat nimi-arvo-parit. Submit-funktio välittää regDatan utils-kansiosta löytyvälle wasteprocessorRegister-funktiolle (kuvassa 29 oikealla alhaalla), joka lähettää ne edelleen palvelimelle.

Rekisteröinti-funktio lähettää tiedot post-kutsuna palvelimelle. Post-kutsuun on mahdollista lisätä dataa mukaan. Vastaavasti taas get-kutsuun kaikki data, mitä lähetetään, pitää lisätä kutsuriville, eli URL:iin. Lisäksi post-kutsua pidetään turvallisempana kuin get-kutsua. Saatuaan tiedot palvelin tarkistaa ensin tietokannasta onko sähköpostiosoitteella jo rekisteröidytty. Jos on, palauttaa palvelin virhekoodin 401, joka tarkoittaa, että kyseisellä sähköpostiosoitteella on jo rekisteröidytty. Kaiken ollessa kunnossa, vastaa palvelin koodilla 200 ja tili jää odottamaan adminin hyväksyntää.

6.4 Kirjautuminen

Sovelluksen ensimmäisenä sivuna toimii index-komponentti, jolla luodaan store-objekti. Koska kaikilla komponenteilla on oltava pääsy Reduxin tietovarastoon, otetaan tällä sivulla käyttöön myös React Redux -komponentti Provider, joka antaa kaikille kyseisen komponentin alapuolisille komponenteille tietovaraston käytettäväksi.

Index-komponentissa kutsutaan myös ReactDOM-kirjastoa, joka huolehtii DOM:n päivittämisestä, ja renderöidään sovelluksen ensimmäinen komponentti App. App-komponentti käyttää switch-lauseketta ohjaamaan käyttäjä oikealle sivulle. Lisäksi se lataa näytettävät kategoriat ja niiden alakategoriat, kun sivu aukeaa. Kategoriat ladataan käyttäen elinkaarimetodia componentDidMount(), jota kutsutaan välittömästi kun komponentti ladataan ensimmäisen kerran. Ja lopuksi App-komponentti välittää kategoriat Reduxin tietovarastoon this.props.setCategories-funktiolla (kuva 30), missä ne tallennetaan. Oletusarvona komponentti ohjaa käyttäjän Login.js-sivulle, jolla käyttäjä voi kirjautua palveluun. Sisäänkirjautuminen todennetaan lähettämällä post-kutsun mukana sähköpostiosoite ja salasana palvelimelle. Palvelin tarkistaa käyttöoikeuden, ja tunnusten ollessa oikeat, palauttaa käyttäjädatan, joka myös tallennetaan Reduxin tietovarastoon.

```
componentDidMount() {

  // src/App.js
  getCats().then((cats) => {
    this.props.setCategories(cats);
  })
```

```
// utils/fetchGategories.js
function getCats() {
  return axios.get(BASE_URL+'/categories')
  .then(response => response.data)
  .catch(function (error) {
    return error;
  });
}
```

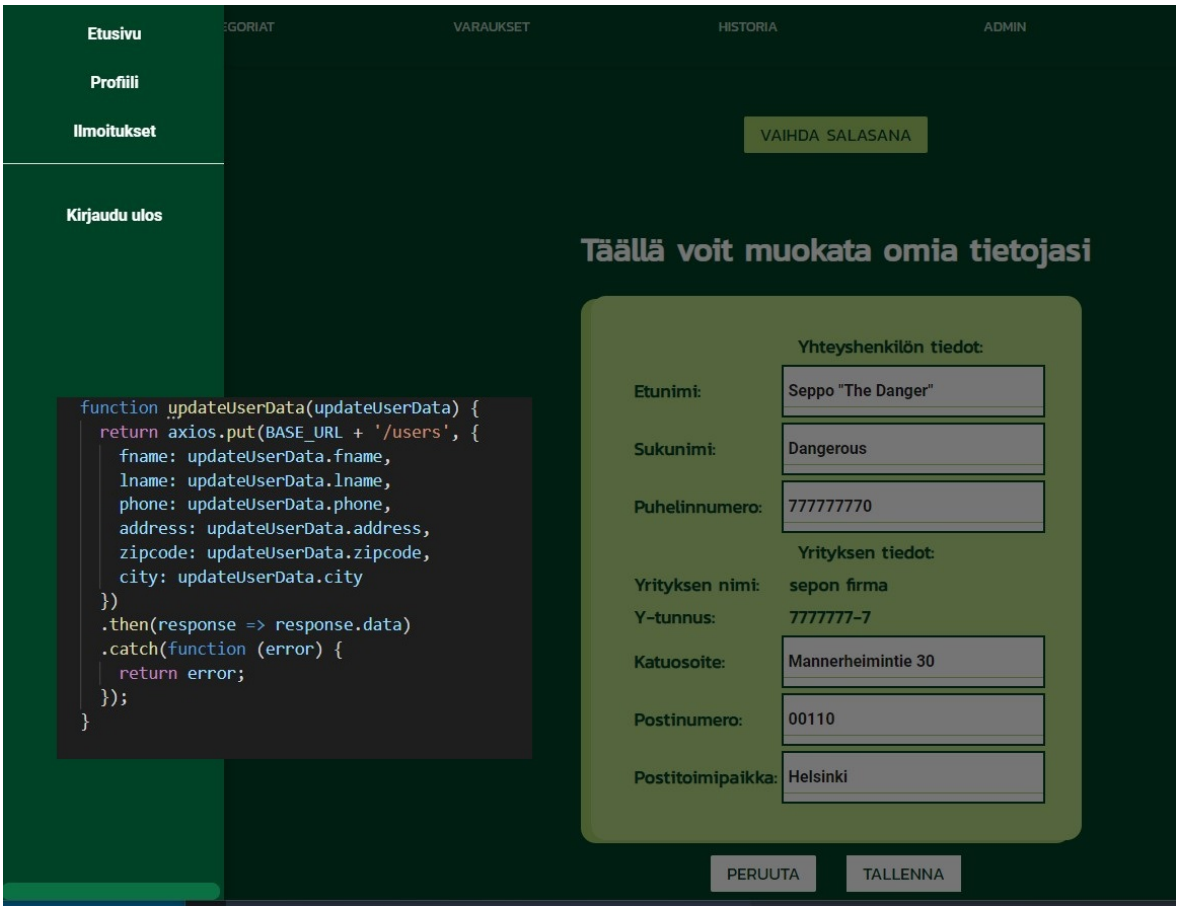
```
// containers/App.js
const mapDispatchToProps = dispatch =>
  ({
    setCategories(cats){
      dispatch(
        setCategories(cats)
      )
```

Kuva 30. Kategorioiden lataus ja tallennus Reduxin tietovarastoon

Sisäänkirjautumisen jälkeen sivusto lataa kyseisen käyttäjäprofiilin aloituskomponentin, mikä sisältää sivun perusrungon (ylävalikko, sivuvalikko ja niiden toiminnot sekä ulkoasuun liittyvät CSS-tiedot). Kyseinen komponentti lataa aina ylä- tai sivuvalikosta valitun sivun sisäänsä, jolloin ylä- ja sivuvalikko pysyy koko ajan käytettävissä.

6.5 Profiili

Profiilinäkymä aukeaa oikeassa yläkulmassa olevan hampurilaispainikkeen kautta ja profiilinäkymä on samanlainen kaikilla käyttäjäprofiileilla. Profiilinäkymän kautta voi muokata omia käyttäjätietoja ja vaihtaa salasanan (kuva 31). Profiilissa muokatut tiedot lähetetään palvelimelle axioksen put-metodin avulla. Put-metodia käytetään, koska sen avulla voidaan korvata edelliset tiedot eikä sillä ole sivuvaikutuksia, vaikka sitä kutsuttaisiin useampiakin kertoja (MDN Web Docs 2019a).



Kuva 31. Profiilin muokkaus

Sama profiilisivu ladataan yhdestä paikasta useammallekin käyttäjäprofiilille (admin, operaattori ja operaattorin työntekijät). Tämä voidaan tehdä näin, koska sen sivun rakenne pysyy samana kaikille eikä näin ollen ole järkevää tehdä useampaa samanlaista komponenttia tiedostorakenteeseen. Käyttäjän omat tiedot ladataan kuitenkin aina erikseen palvelimelta kirjautumisen yhteydessä ja tallennetaan Reduxin tietovarastoon, jolloin sivulla on aina oikean käyttäjän tiedot.

Näytettävä profiilisivu koostuu kolmesta komponentista: Profilemain.js, UserInfo.js ja ChangePassword.js. ProfileMain-komponentti (kuva 32) sisältää kuvassa 31 näkyvän napin, jossa on vaihda salasana / muokkaa tietoja -teksti, riippuen siitä kumpi alasivuista komponentin sisään on ladattu. Sivun vaihto pääkomponentin sisään on toteutettu ternary-operaatiolla, joka on ainoa JavaScript operaattori, joka ottaa sisään kolme operandia (MDN Web Docs 2019b). Ternary-operaatio vertaa arvoa, joka on true / false, ja näyttää arvoon perustuvan sivun komponentin sisällä.

```
import React, { Component } from 'react';
import { FlatButton } from 'material-ui'
import UserInfo from '../../../containers/Admin/Profile/UserInfo'
import ChangePassword from './ChangePassword'

class ProfileMain extends Component {
    constructor(props) {
        super(props);
        this.state = {
            value: true
        }
        this.handleChange = this.handleChange.bind(this);
    }

    handleSnackbar = (snackbarvalue) => {
        this.props.onUpdate(snackbarvalue)
    }

    handleChange = (value) => {
        this.setState({
            value: !this.state.value
        })
    }

    render() {
        return (
            <div>
                <FlatButton key={'i'}
                    label={this.state.value ? "Vaihda salasana" : 'Muokkaa tietoja'} onClick={this.handleChange}
                    hoverColor="#8CE30B"
                    style={{ margin: '3%' }}
                    backgroundColor="#A6CE6B"
                    labelStyle={{
                        fontFamily: 'kanit',
                        float: 'left',
                        borderRadius: '0',
                        fontSize: '17px',
                        color: '#004225'
                    }} />
                {this.state.value ? <UserInfo handleSnackbar={this.handleSnackbar} /> : <ChangePassword handleSnackbar={this.handleSnackbar} />}
            </div>
        );
    }
}
export default ProfileMain
```

Kuva 32. ProfileMain-komponentti

Komponentissa toteutetaan myös komponenttien välistä tiedonsiirtoa, tässä tapauksessa siirretään tietoa lapsikomponentilta äitikomponentille. Siirrettävä arvo on aina joko true tai false. Arvo määräytyy kyseisen komponentin tapauksessa siitä, onko muutoksia tehty ala-sivuilla. Jos muutoksia on tehty, tieto välitetään aina pääkomponentille asti, joka näyttää Material-Ui:n snackbarin avulla tietojen päivittämisestä. Tämä on tehty sen takia, koska sekä tallenna- että peruuta-napit vievät aina takaisin pääsivulle ja näin voidaan visuaali-sesti näyttää, että tietojen päivitys on onnistunut.

6.6 Admin

Admin-puolella voidaan muokata tai luoda kategorioita ja hallita käyttäjätilejä. Lisäksi siellä voidaan tarkkailla sovelluksessa tehtyjä ilmoituksia jätteistä ja niiden varauksia, sekä tar-vittaessa muuttaa varauksia. Yhtenäistä operaattoripuolen kanssa on kartta-komponentti sekä sovelluksen sisäisten ilmoituksien tekoon tarkoitettu komponentti.

6.6.1 Kategorioiden luonti ja muokkaus

Kategorioiden lisäys alkaa aina luomalla ensin pääkategoria ja sille alakategoria, jonka jälkeen voidaan luoda sovelluksessa näkyvä proxykategoria. Pääkategoria voi olla esim. SER, alakategoria Iso-SER ja proxykategoria jääkaappi. Tämä sen takia, että sovelluksen käyttäjän olisi helppoa valita oikea tuote, minkä haluaa ilmoittaa. Loppukäyttäjä valitsee sovelluksessa ensin pääkategorian ja sen jälkeen hänelle tulee suoraan proxykategorian tuotteet valittavaksi. Proxykategorian tuotteet ovat siis suoraan lopputuotteita, kuten jääkaappi, joista valita oikea.

Operaattorit taas tarvitsevat pääkategorian ja alakategorian tiedot, jonka takia kategoriatasoja tarvitsee olla kolme. Tämä sen takia, että tilastoinnin takia lopputuotteella ei ole niinkään merkitystä, vaan enemmän merkitystä on sillä mihin alakategoriaan kyseinen tuote kuuluu.

Kategorioiden muokkaus, kuten uuden lisääminen, nimen vaihtaminen, aktivointi/deaktivointi tai kuvan lisääminen tehdään kaikki post-kutsuilla. Pääkategorioiden lisäyksessä (kuva 33) ei tarvitse palvelimelle lähettää kuin uuden kategorian nimi, ja palvelin lisää sille tarvittavat lisätiedot, kuten id:n, automaattisesti luodessaan uuden kategorian. Tämän jälkeen käynnistetään kutsu this.getCategories(), jolla kategoriat haetaan uudestaan ja tallennetaan sovelluksen tilaan, ja näin voidaan varmistua, että kategoria on luotu ja näkyy sovelluksessa. Jos Admin-puolella kategoriat tallennettaisiin Reduxin tietovarastoon, näkyisivät mahdolliset muutokset vasta, kun sovellukseen kirjauduttaisiin uudestaan. Tämän takia kategorialuetteloa päivitetään sovelluksen tilaan.

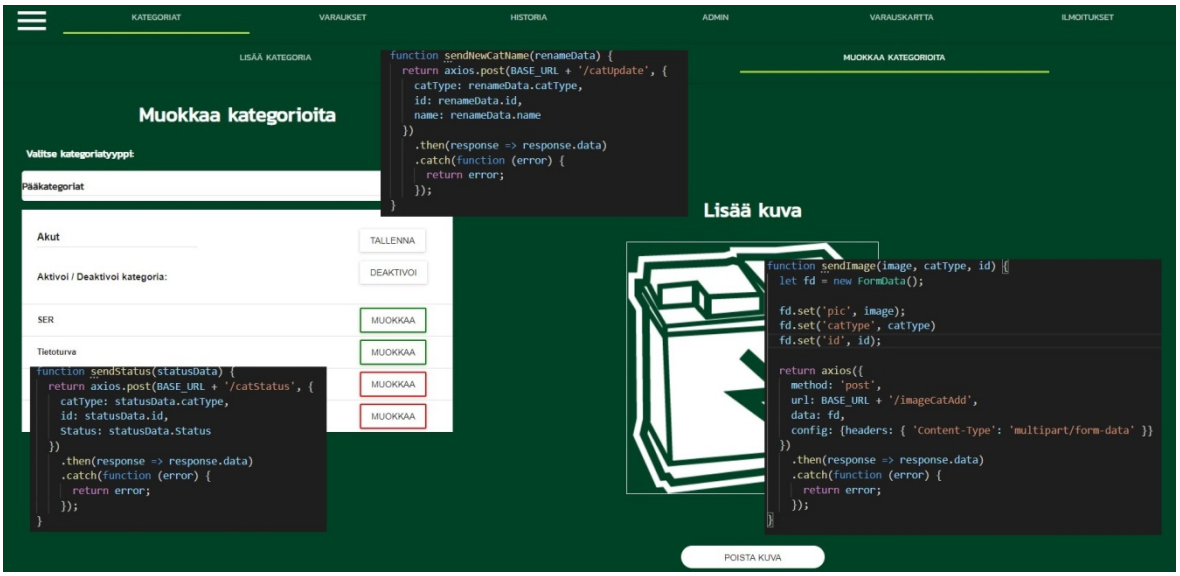
```
// Add Category name
addCategory() {
    var addCatName = {
        'catname': this.state.addCat
    }
    addNewCat(JSON.stringify(addCatName)).then(() => { this.getCategories() });
}

// Add SubCategory name
addSubCategory() {
    var addSubCatName = {
        'catid': this.state.value,
        'subcatname': this.state.addCat,
    }
    addNewSubCat(JSON.stringify(addSubCatName)).then(() => { this.getSubCategories() });
}
```

Kuva 33. Kategorian lisäys

Alakategoriaa luodessa (kuva 33) pitää lisäksi palvelimelle kertoa pääkategorian id, jotta palvelin tietää mihin kategoriaan kyseinen alakategoria kuuluu. Loput tarvittavat tiedot palvelin lisää automaattisesti. Tietojen lähetyksen jälkeen käynnistetään kutsu this.getSubGategories(), jotta muutokset näkyisivät heti. Proxykategoriaa luodessa palvelimelle lähetetään nimen lisäksi alakategorian id, jonka perusteella palvelin lisää tarvittavat tiedot automaattisesti. Alakategoriaa tai proxykategoriaa luodessa sovellus tarkistaa aina ensin, että uusi kategoria on linkitetty johonkin olemassa olevaan pää- ja/tai alakategoriaan ennen kuin lähettää kutsun palvelimelle.

Kuvien lisääminen on mahdollista pää- ja proxykategoriaan. Tämä sen takia, koska kuvat on tarkoitettu vain loppukäyttäjän käytön helpottamiseksi, ja loppukäyttäjä ei koskaan näe todellista alakategoriaa, joten sille ei myöskään tarvitse lisätä kuvia. Kuvien lisääminen (kuva 34) tehdään muokkaa kategorioita -valikon alla, missä voidaan lisäksi nimetä kategoria uudelleen ja aktivoida/deaktivoida kyseinen kategoria.



Kuva 34. Muokkaa kategorioita

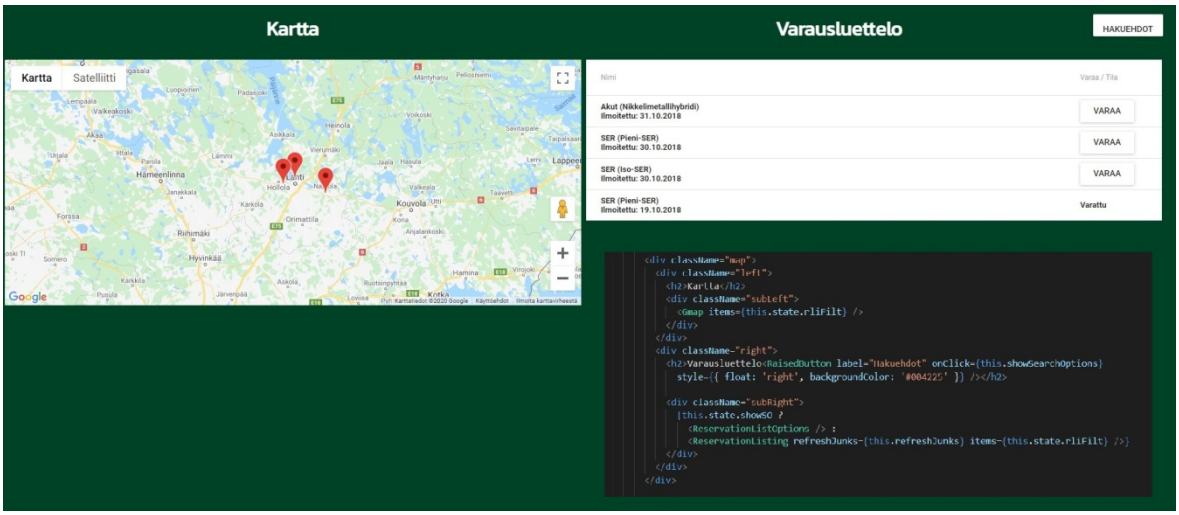
Kuvien lisääminen tehdään formData-objektin avulla, koska formDataan voi syöttää avain- / arvoparien joukon. Kuviin tarvitaan kuvan lisäksi myös tieto mihin kategoriaan se kuuluu ja kategorian id, jotta se voidaan yhdistää oikeaan kategoriaan. FormData-objektissa käytetään tietojen syöttämisen yhteydessä set-metodia, koska set-metodi korvaa edelliset arvot, jos sellaiset olivat olemassa. Tämä on tärkeää silloin, kun kuva vaihdetaan, jottei kategorioihin jää tietoja vanhasta kuvasta, joka voisi aiheuttaa jonkinlaisia ongelmia jatkossa.

Kategorian uudelleen nimeäminen tehdään lähettämällä palvelimelle kategorian id, uusi nimi ja kategoriatyyppi, jotta tiedetään, mistä kategoriasta nimeä ollaan muuttamassa.

Kategorian aktivointi/deaktivointi toimii samalla lailla, nimen tilalla lähetään vain muuttunut statuksen arvo (true tai false). Aktiiviset kategoriat on korostettu vihreällä värillä muokkaa-painikkeen ympärillä, kun taas ei-aktiiviset on ympäröity punaisella värillä. Jos kategoria on deaktivoitu, näkyy se ainoastaan admin-käyttäjälle. Kategoriaa ei poisteta kokonaan, koska kategoriassa saattaa olla ennestään syötettyjä tietoja, jotka näkyvät historia tiedoissa. Kategorian poistaminen voisi myös aiheuttaa ongelmia tietokannan eheydessä.

6.6.2 Kartta-komponentti

Näytettävä karttasivu (kuva 35) on luotu useammasta komponentista ja se on sama sivu adminilla, operaattorilla ja operaattorin työntekijällä. Sivu koostuu Varauskartta.js-pääkomponentista, jonka sisään muut komponentit ladataan. Varauskartta-komponentti on jaettu kahteen sarakkeeseen. Kartta-komponentti näkyy aina vasemmalla, ja oikealla joko varausluettelo-komponentti tai hakuehdot-komponentti. Valintaan kumpi komponentti oikealla puolella näkyy, käytetään ternary-operaatiota. Varausluettelo-komponenttiin viedään myös lajiteltu tavaraluettelo (rliFilt). Tämä siksi koska palvelin ei huolehdi lajittelusta, vaan koko tavaraluettelo ladataan aina kerralla palvelimelta. Varauskartta-komponentti huolehtii, ettei varausluettelossa näy jo noudettuja tuotteita. Tätä samanlaista lajittelua käytetään myös varaukset- ja historia-komponentin kanssa.



Kuva 35. Varauskartta-komponentti

Hakuehdoissa voidaan muuttaa minkä kategorian tuotteita halutaan hakea sekä rajata tuotteita painon, koon ja etäisyyden perusteella. Hakuehdoissa näytetään aina vain aktiiviset kategoriat ja niiden aktiiviset alakategoriat. Kaikki tehdyt valinnat tallennetaan Reduxin

tietovarastoon ja ladataan aina kirjautumisen yhteydessä, jolloin tehtyjä muutoksia ei tarvitse tehdä joka kerta uudestaan.

Karttana sovelluksessa käytetään react-google-maps -komponenttia, joka asennetaan npm:n avulla. Kyseinen komponentti mahdollistaa kartassa näkyvien markkereiden käytön ja niihin saa tarvittaessa myös tietoikkunat, joista näkyy kyseisessä paikassa olevan tavaran tiedot. Markkerit ladataan karttaan lajitellun tavaraluettelon perusteella.

6.7 Operaattori ja operaattorin työntekijät

Operaattoreilla ja operaattorin työntekijöillä on samat historia-, varaukset-, varauskartta- ja ilmoitukset-näkymät kuin adminillakin. Operaattorilla on lisänä oma käyttäjien hallinta -näkymä, josta voi lisätä omia työntekijöitä. Luodut työntekijät kirjautuvat kyseisen operaattorin työntekijöiksi ja heille kirjataan yrityksen tiedot automaattisesti.

Käyttäjien hallintasivu on luotu kolmesta komponentista: pääkomponentti on UserManagementMain.js, joka sisältää sivun rungon ja ternary-operaation. Ternary-operaation perusteella sivulle ladataan jompikumpi kahdesta vaihtoehdosta, joko NewUser.js- tai UserManagement.js-komponentti. Sivun toteutus on hyvin samanlainen kuin kuvassa 35 näkyvä varauskartta-sivun toteutus.

NewUser-komponentissa operaattori täyttää työntekijän tiedot ja lähettää ne palvelimelle Axios-kirjaston post-kutsulla. Palvelin tarkistaa ensin onko sähköpostiosoite jo käytössä ja palauttaa tarvittaessa virheviestin. Jos kaikki on kunnossa, rekisteröidään uusi käyttäjätili tietokantaan, jonka jälkeen se on heti käytettävissä. Rekisteröinnin jälkeen käyttäjälista ladataan uudestaan palvelimelta Axioksen get-kutsulla ja päivitetään sovelluksen tilaan, jotta uusi käyttäjä näkyy operaattorin käyttäjälistalla.

UserManagement-komponentissa operaattori voi tarkastella omien käyttäjien tietoja ja tarvittaessa aktivoida/deaktivoida käyttäjän. Deaktivoinnin yhteydessä käyttäjä ei voi enää kirjautua palveluun, mutta tiliä ei poisteta kokonaan. Tämä siitä syystä, koska on mahdollista, että kyseinen käyttäjä on joskus tehnyt varauksia, ja tilin poistaminen kokonaan voisi aiheuttaa ongelmia tietokannan eheydessä.

7 YHTEENVETO

Työn tavoitteena oli luoda Digiloop-kierrätyspalvelun ylläpitokäyttöliittymän prototyyppi. Digiloop-kierrätyspalvelun avulla jätteiden kierrätystä on mahdollista tehostaa. Kierrätyspalvelusta saatiin ensimmäinen testiversio valmiiksi, missä toimivat keskeisimmät toiminnot. Sovelluksesta jätettiin pois sellaisia ominaisuuksia, joilla ei ollut merkitystä sovelluksen toiminnan kannalta. Lisäksi osa vähemmän tärkeistä toiminnoista jäi puutteellisiksi, mutta nekään eivät vaikuta sovelluksen toimintaan. Digiloop-kierrätyspalvelua testattiin useamman kerran kokousten yhteydessä. Näissä saatiin määriteltyä mahdolliset vakavat toimintojen puutteet, kuten myös vähemmän tärkeät puutteet. Näiden perusteella tehtiin rajaukset tarvittavista toiminnoista, jotta palvelua voitaisiin koekäyttää.

Alussa haasteena oli kokemuksen puute, mikä myös vaikutti osittain koodin toimivuuteen ja siisteyteen. Lisähaasteena olivat myös kokouksissa sovitut uudet ominaisuudet, jotka saattoivat aiheuttaa isojakin koodin uudelleenkirjoituksia. Lisäksi myös kokemuksen lisääntyminen aiheutti osittain jo toimivien komponenttien uudelleenkirjoituksia, lähinnä sen takia, että kaikki komponentit olisivat yhdenmukaisia.

Kokemuksesta sovellussuunnittelussa erityisesti projektin alussa on hyötyä, koska nyt aloitettaessa sovelluksen suunnitteluun tulisi kiinnitettyä erityisesti huomioita. Nyt on tiedossa, miten asiat tulisi toteuttaa. Lisäksi tiedostorakenteessa olisi enemmän uudelleenkäytettävien komponenttien hyötykäyttöä. Nyt sovelluksessa on joitakin komponentteja turhaan useampia, koska alussa ei ollut tiedossa, miten samaa komponenttia pienillä muutoksilla voisi käyttää useammassa paikassa.

Kierrätyspalvelusta luotiin myös mobiilisovellus, mutta se oli lähinnä loppukäyttäjiä varten. Sovellusta ei jaettu missään yleisesti ja se oli ainoastaan Androidille tarkoitettu testiversio. Sovellukseen pystyi myös kirjautumaan operaattoritkin, mutta se ei ollut kovin hyvin toimiva, koska ylläpitokäyttöliittymä oli alun perin ajateltu käytettäväksi tietokoneella eikä mobiililaitteella.

Kierrätyspalvelu sai Päijät-Hämeen liitolta AIKO-rahoitusta mobiilipalvelualustan kehittämistä varten. Kierrätyspalvelu jatkaa uutena versiona RESELL-kierrätyspalvelun nimellä. RESELL on LUT:n, LAMK:n ja Saimaan AMK:n yhteinen hanke ja se on jo saanut TUTLI-rahoituksen. RESELL-hankkeessa on tarkoitus valmistella palvelu Euroopan markkinoille. (LAMK 2018).

LÄHTEET

Abramov, D. 2018. Create React App [viitattu 9.4.2019]. Saatavissa: https://facebook.github.io/create-react-app/docs/folder-structure

Banks, A. & Porcello, E. 2017. Learning React: Functional Web Development with React and Redux. Sebastopol: O'Reilly Media.

Bertoli, M. 2017. React Design Patterns and Best Practices. Birmingham: Packt Publishing Ltd.

Erikson, M. 2017. Redux and Why it's Good For You [viitattu 1.5.2019]. Saatavissa: https://www.fullstackreact.com/articles/redux-with-mark-erikson/

EUR-Lex 2019. Sähkö- ja elektroniikkaromun maksimaalinen hyödyntäminen [viitattu 3.2.2020]. Saatavissa: https://eur-lex.europa.eu/legal-content/FI/TXT/?uri=legissum:200403_1

GitHub 2019. Axios [viitattu 10.4.2019]. Saatavissa: https://github.com/axios/axios

Hallamaa, T. & Kanerva, J. 2019. Muovi mietityttää, mutta sen kierrättäminen takkuaa: Yksi syy on liian kaukana olevat kierrätyspisteet [viitattu 26.4.2020]. Saatavissa: https://yle.fi/uutiset/3-10606963

Hunt, P. 2015. TXJS 2015. Video [viitattu 28.12.2018]. Saatavissa: https://www.youtube.com/watch?v=A0Kj49z6WdM

Kagga, J. 2018. Understanding React Components [viitattu 6.1.2019]. Saatavissa: https://medium.com/the-andela-way/understanding-react-components-37f841c1f3bb

Kahler, R. 2019. Create React App [viitattu 9.4.2019]. Saatavissa: https://facebook.github.io/create-react-app/docs/getting-started

Krajka, B. 2015. The difference between Virtual DOM and DOM [viitattu 15.1.2019]. Saatavissa: https://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/

LAMK 2018. LUT-korkeakoulujen toimintamalli avaa uusia mahdollisuuksia projekti- ja hanketoimintaan [viitattu 19.4.2020]. Saatavissa: https://www.epressi.com/tiedotteet/hanketiedotteet/lut-korkeakoulujen-toimintamalli-avaa-uusia-mahdollisuuksia-projekti-ja-hanketoimintaan.html

Maj, W. 2019. React lifecycle methods diagram [viitattu 28.3.2019]. Saatavissa: http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

MDN Web Docs 2019a. PUT [viitattu 8.10.2019]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT

MDN Web Docs 2019b. Conditional (ternary) operator [viitattu 8.10.2019]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_Operator

Pardee, M. 2016. Passing Data Between react Components [viitattu 22.4.2019]. Saatavissa: https://medium.com/@ruthmpardee/passing-data-between-react-components-103ad82ebd17

Parviainen, T. 2015. Change And Its Detection In JavaScript Frameworks [viitattu 28.3.2019]. Saatavissa: https://teropa.info/blog/2015/03/02/change-and-its-detection-in-javascript-frameworks.html

React 2018. A Javascript library for building user interfaces [viitattu 21.12.2018]. Saatavissa: https://reactjs.org/

React 2019a. Glossary of React Terms [viitattu 6.1.2019]. Saatavissa: https://reactjs.org/docs/glossary.html#components

React 2019b. Glossary of React Terms [viitattu 6.1.2019]. Saatavissa: https://reactjs.org/docs/state-and-lifecycle.html

React 2019c. Virtual DOM and Internals [viitattu 7.1.2019] Saatavissa: https://reactjs.org/docs/faq-internals.html#what-is-the-virtual-dom

React 2019d. React.Component [viitattu 28.3.2019]. Saatavissa: https://reactjs.org/docs/react-component.html

React 2019e. Introducing JSX [viitattu 4.9.2019]. Saatavissa: https://reactjs.org/docs/introducing-jsx.html

React Redux 2019a. Quick Start [viitattu 3.9.2019]. Saatavissa: https://react-redux.js.org/5.x/introduction/quick-start

React Redux 2019b. Provider [viitattu 3.9.2019]. Saatavissa: https://react-redux.js.org/api/provider

React Redux 2019c. Connect [viitattu 6.9.2019]. Saatavissa: https://react-redux.js.org/5.x/api/connect

Redux 2018a. Getting Started with Redux [viitattu 24.4.2019]. Saatavissa: https://redux.js.org/introduction/getting-started

Redux 2018b. Three Principles [viitattu 2.5.2019]. Saatavissa: https://redux.js.org/introduction/three-principles

Redux 2018c. Actions [viitattu 6.5.2019]. Saatavissa: https://redux.js.org/basics/actions

Redux 2018d. Reducers [viitattu 8.8.2019]. Saatavissa: https://redux.js.org/basics/reducers

Redux 2018e. Store [viitattu 8.8.2019]. Saatavissa: https://redux.js.org/basics/store

Robie, J. 1998. What is the Document Object Model? [viitattu 11.1.2019]. Saatavissa: https://www.w3.org/TR/REC-DOM-Level-1/introduction.html

w3schools.com. 2019. What is the HTML DOM? [viitattu 11.1.2019]. Saatavissa: https://www.w3schools.com/whatis/whatis_htmldom.asp

WebPack 2020. Concepts [viitattu 26.4.2020]. Saatavissa: https://webpack.js.org/concepts/

Weck, S. 2017. Developing modern offline apps with ReactJS, Redux and Electron [viitattu 1.5.2019]. Saatavissa: https://blog.codecentric.de/en/2017/12/developing-modern-offline-apps-reactjs-redux-electron-part-3-reactjs-redux-basics/

Vesa, M. 2019. Näkymät jätteiden Mount Everestiltä pysäyttävät [viitattu 19.4.2020]. Saatavissa: https://www.is.fi/ulkomaat/art-2000006140754.html

Wieruch, R. 2017. The Road to learn React. Leanpub.

Wieruch, R. 2018. React Redux Tutorial for Beginners [viitattu 24.4.2019]. Saatavissa: https://www.robinwieruch.de/react-redux-tutorial/

Ympäristöministeriö 2019. Jätesäädöspaketti [viitattu 3.2.2020]. Saatavissa: https://www.ym.fi/fi-FI/Ymparisto/Lainsaadanto_ja_ohjeet/Ymparistonsuojelun_valmisteilla_oleva_lainsaadanto/Jatesaadospaketti