Ilari Matikka

ReactJS-kirjaston käyttö ohjelmointirajapintojen yhteydessä

Opinnäytetyö
Kevät 2018
SeAMK Tekniikka
Tietotekniikan tutkinto-ohjelma

SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Tekniikan yksikkö

Tutkinto-ohjelma: Tietotekniikka

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Tekijä: Ilari Matikka

Työn nimi: ReactJS-kirjaston käyttö ohjelmointirajapintojen yhteydessä

Ohjaaja: Hilkka Niemelä

Vuosi: 2018 Sivumäärä: 42 Liitteiden lukumäärä: 0

Tämän opinnäytetyön ideana oli hyödyntää React-kirjastoa. Työssä tutustuttiin React-kirjaston rakenteeseen ja sen ominaisuuksiin, joita voidaan hyödyntää koulun avoimiin PEPPI-ohjelmointirajapintoihin. React-kirjaston ominaisuuksia voidaan käyttää tehostamaan ja nopeuttamaan datan saantia avoimista ohjelmistorajapinnoista. Työssä toteutettiin tietojen hakuja Seinäjoen Ammattikorkeakoulun PEPPI-järjestelmän rajapintoihin.

Työssä perehdyttiin myös HTTP-pyyntömetodien käyttöön sekä niiden erilaisiin toteutuksiin käyttäen kolmea erilaista menetelmää, jotka olivat jQuery AJAX, Axios ja Fetch. Työn lopputuloksena saatiin tulostettua dataa PEPPI-ohjelmointirajapinnoista React-pohjaiselle sivulle GET- ja POST-pyynnöillä käyttäen kolmea eri menetelmää.

Avainsanat: React-kirjasto, JavaScript, Ohjelmointirajapinta, HTTP

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Technology

Degree programme: Information Technology

Specialisation: Software Engineering

Author: Ilari Matikka

Title of thesis: Using ReactJS Library with an Application Programming Interface

Supervisor: Hilkka Niemelä

Year: 2018 Number of pages: 42

The idea behind this thesis was to utilize the React library. The aim was to get acquainted with the ReactJS structure and its features, which can be used in conjuction with the PEPPI open API. The library's features can be used to optimize and speed up the data access from the open API. The data search was implemented inside the PEPPI system's API at Seinäjoki University of Applied Sciences.

The thesis also introduced the use of HTTP requests and their implementations using three different methods to acquire data from the API, which were called jQuery AJAX, Axios and Fetch. As the result, the data pulled from the PEPPI open API was successfully printed on the React-based site using the GET and POST requests with three different methods.

Keywords: ReactJS, JavaScript, API, HTTP

SISÄLTÖ

Kuvaluettelo

Käytetyt termit ja lyhenteet

AJAX	Asynchronous JavaScript and XML on joukko web-sovelluskehityksen tekniikoita, joiden avulla web-sovelluksista voi tehdä vuorovaikutteisempia.
API	Application programming interface eli ohjelmointirajapinta.
Asynkroninen	Tarkoitetaan ohjelmoinnissa sitä, että useita toimintoja suoritetaan koodissa samaan aikaan.
CSS	Cascade Style Sheet on kieli dokumenttien tyylien kuvaamiseksi.
DOM	Document object model eli dokumenttioliomalli, joka kuvaa rakenteisen dokumentin.
Elinkaarimetodi	React-komponentin eri vaiheissa suoritettavat toiminnot.
ES6	EcmaScript 6 on Ecma Internationalin määrittelemä tavaramerkki, johon JavaScript perustuu.
HTML	Hypertext Markup Language eli hypertekstin merkintäkieli, joka on standardoitu kuvauskieli.
HTTP	Hypertext Transfer Protocol. Webissä sijaitsevien komponenttien välistä kommunikaatiota ohjaava protokolla.
HTTP-metodi	HTTP-protokollalle pyynnön aikeita kuvaavia verbejä, kuten GET ja POST.
JavaScript	Komentosarjakieli, jota käytetään dynaamisen verkkosivun toteutuksessa.
JavaScript runtime	Ajoympäristö, joka tarjoaa suoritettavalle JavaScript-ohjelmakoodille rajapinnan selaimen tarjoamiin toimintoihin.

JSON	JavaScript Object Notation on JavaScriptissä riippumaton avoimen standardin tiedostomuoto, joka koostuu avain-arvopareista.
Non-blocking I/O	Syötteiden ja tulosteiden käsittelymalli. Niiden käsittely ei keskeytä ohjelmakoodin suorittamista.
NPM	Node Package Manager. Työkalu pakettien hallintaan.
Polyfill	Ohjelmakomponetti, joka implementoi toimintoja selainympäristöön.
Promise	Lupaukset (promise) ovat olioita, jotka kuvaavat tapahtumaa tai tulevaisuudessa saatavilla olevaa arvoa.
Props	Props on data, joka siirtyy isäntäkomponentista lapsikomponentille.
SOA	Service Oriented Architecture eli palvelukeskeinen arkkitehtuuri.
SPA	Single-page application. Selaimessa käytettävä sovellus, jonka tarkoituksena on käyttää vain yhtä sivun latausta.
State	Tila. Komponentin sisäinen yhteinen data.
URI	Uniform Resource Identifier. Resurssin yksilöivä tunniste.
Virtual DOM	Kopio DOM-mallista. Verrataan muutoksia alkuperäiseen DOM-malliin.

1 JOHDANTO

1.1 Työn tausta

Web-ohjelmointi sisältää paljon erilaisia JavaScript-sovelluskehyksiä. Tässä opinnäytetyössä keskitytään yhteen suosituimmista kirjastoista nimeltään ReactJS.

React-kirjaston avulla kehittäjät voivat luoda web-sovelluksia, joiden data sekä näkymä voivat muuttua tietyin aikavälein lataamatta sivua uudelleen. Tämän kirjaston tarkoituksena on tarjota nopeutta, yksinkertaisuutta ja skaalautuvuutta web-ohjelmointiin. Tätä työtä voi käyttää esimerkkinä niille, jotka haluavat käyttää React-kirjastoa web-ohjelmoinnissa rajapintojen yhteydessä.

1.2 Työn tavoite

Työn tavoitteena on käyttää React-kirjastoa ja sen toimintoja Seinäjoen Ammattikorkeakoulun avoimien PEPPI-ohjelmointirajapintojen yhteydessä. Opinnäytetyössä käydään läpi React-kirjaston arkkitehtuuria, termejä sekä datan hakua rajapinnoilta erilaisilla HTTP-metodeilla.

Lukijan pitäisi pystyä tämän työn avulla ymmärtämään React-kirjaston perusteita. Perusteita ovat sivuston luominen käyttäen Node.js-ohjelmaa, React-arkkitehtuuri ja sen toiminta sekä erilaisien HTTP-pyyntömetodien käyttö datan hakuun ohjelmointirajapinnoista.

1.3 Työn rakenne

Luvussa 2 esitellään React-kirjaston perusteita ja sen rakenteita, sekä käydään läpi React-kirjaston asennusta ja sivun luontia käyttäen node.js-työkalua. Luvussa 3 tutustutaan SeAMK:n PEPPI-rajapintoihin ja niiden käyttöön React-kirjaston yhteydessä erilaisilla HTTP-pyyntöesimerkeillä. Luku 4 sisältää työn yhteenvedon.

2 REACTJS-KIRJASTO

2.1 Yleistä

ReactJS on näkymäkerrosta käsittelevä komponenttipohjainen JavaScript-kirjasto, jota käytetään tehokkaan ja nopean käyttöliittymän rakentamiseen. ReactJS on kirjasto, joka yhdistää JavaScriptin nopeuden ja uuden menetelmän web-sivustojen renderointiin, joka tekee siitä dynaamisen ja reagoivan erityisesti käyttäjän syöttöä varten. React-kirjaston kehitti Facebookin ohjelmistoinsinööri Jordan Walke vuonna 2011. (Altexsoft 2017.)

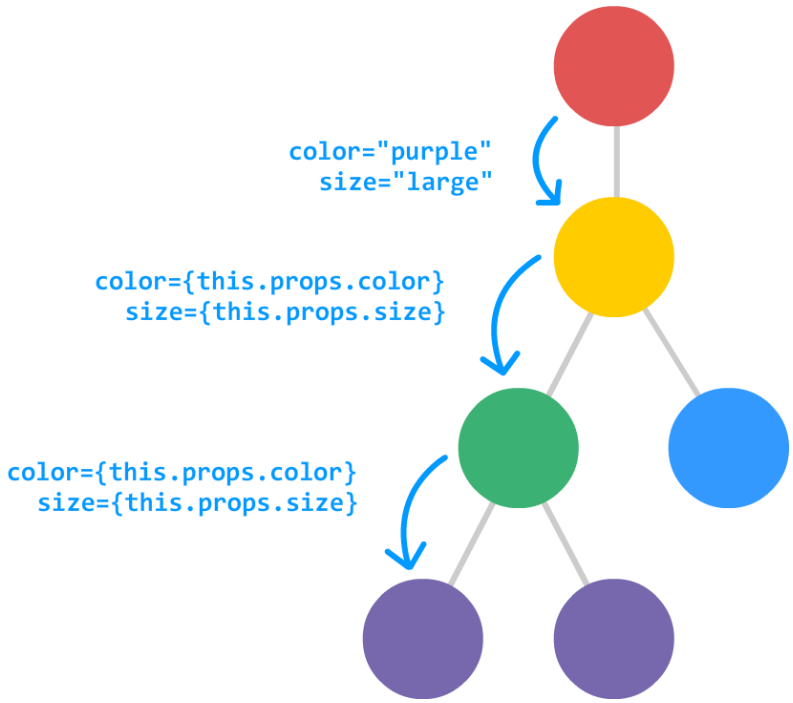
ReactJS on muuttanut ajattelutapaa web-sovelluksien sekä käyttöliittymien kehityksestä. Yksi sen ainutlaatuisista ominaisuuksista on se, että se ei pelkästään toimi asiakaspuolella, vaan sitä voidaan myös renderoida palvelinpuolelle, ja ne voivat toimia yhdessä eli dataa voidaan välittää näiden kahden kesken. ReactJS käyttää myös konseptia, jota kutsutaan nimellä virtual DOM. Kun sovellukseen tulee muutoksia, virtual DOM päivittää vain muutetut komponentit. Tämä ominaisuus tekee DOM-manipulaatiosta nopean. (Wheeler 2014.)

React-kirjastossa on pääasiassa kahdenlaista dataa nimeltään props ja state. Tärkein ero näissä kahdessa on se, että state on yksityinen, joten sitä voidaan muuttaa vain itse komponentin sisällä. Props data on ulkoista, joten niitä ei kontrolloida komponenteilla, vaan niitä siirretään korkeammalla hierarkiassa olevista komponenteista, jotka myös hallitsevat dataa. (Borgen [Viitattu 29.3.2018].) Lyhyesti sanottuna ReactJS on JavaScript-kirjasto, joka pakottaa kehittäjän ajattelemaan koodaamista komponenttien mukaisesti (Survivejs [Viitattu 4.4.2018]).

2.1.1 Props

Ominaisuuksilla (properties, props) komponentit voivat olla yhteydessä toisiinsa. React-kirjastossa propsit etenevät alaspäin isäntäkomponentista lapsikomponentteihin. Tämän takia React-kirjaston tietovirtaa kutsutaan niin sanotusti yksisuuntaiseksi. On myös mahdollista, että on olemassa oletusominaisuuksia, joten ominaisuuksia voidaan asettaa, vaikka isäntäkomponentti ei siirtäisi niitä eteenpäin. (Wynne-McHardy [Viitattu 1.4.2018].)

Alla olevassa kuvassa 1 nähdään, kuinka propsit color ja size etenevät hierarkiassa alaspäin isäntäkomponentista lapsikomponentteihin.



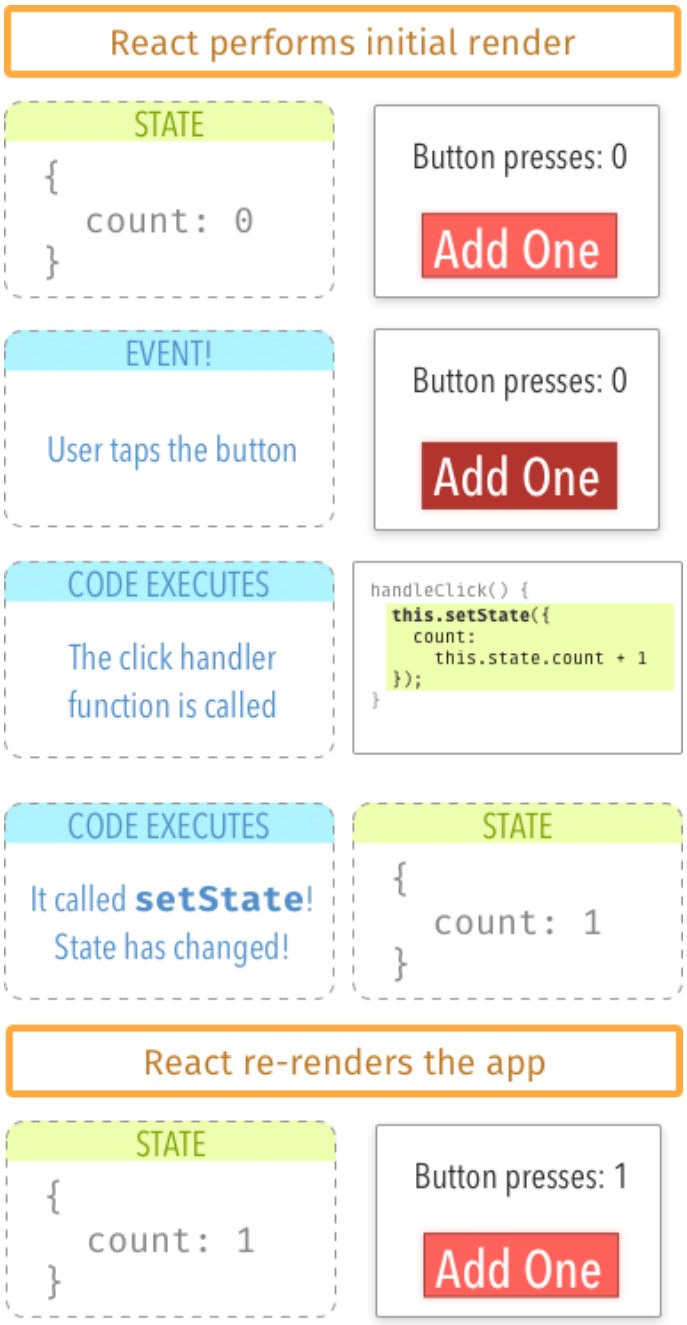
Kuva 1. Propertien toiminta.
(Kirupa 2016).

2.1.2 State

Jokaisen React-komponentin keskiössä on sen tila (state). Se on objekti, joka määrittää, miten kyseinen komponentti renderoi ja käyttäytyy. Toisin sanoen, tila on se minkä avulla voidaan luoda komponentteja, jotka ovat dynaamisia ja vuorovaikutteisia. (Simons [Viitattu 4.4.2018].)

Pääasiassa tilaa käytetään muuttuvaa dataa varten. Tämä on erityisen hyödyllinen käyttäjän syöttöä ajatellen. Jos käsitellään esimerkiksi hakupalkkeja: käyttäjä kirjoittaa tiedon hakuun, ja tila päivittää komponentissa sen mitä käyttäjä näkee. (Wynne-McHardy [Viitattu 4.4.2018].)

Kuvassa 2 nähdään esimerkki komponentin tilan renderoinnista yksinkertaisen laskurin muodossa. Ensin alkutila asetetaan nollaan, josta laskuri lähtee laskemaan ylöspäin. Käyttäjä painaa nappia, josta seuraa handleClick-tapahtuma, joka lisää laskurin numeroa yhdellä. Tämä lisäys suoritetaan setState-metodilla, joka muuttaa tilaa. ReactJS renderoi sovelluksen uudelleen tilan vaihtuessa, jonka jälkeen laskurin numero on lisääntynyt yhdellä. (Ceddia 2016.)



Kuva 2. Tilan toimintaperiaate.
(Ceddia 2016).

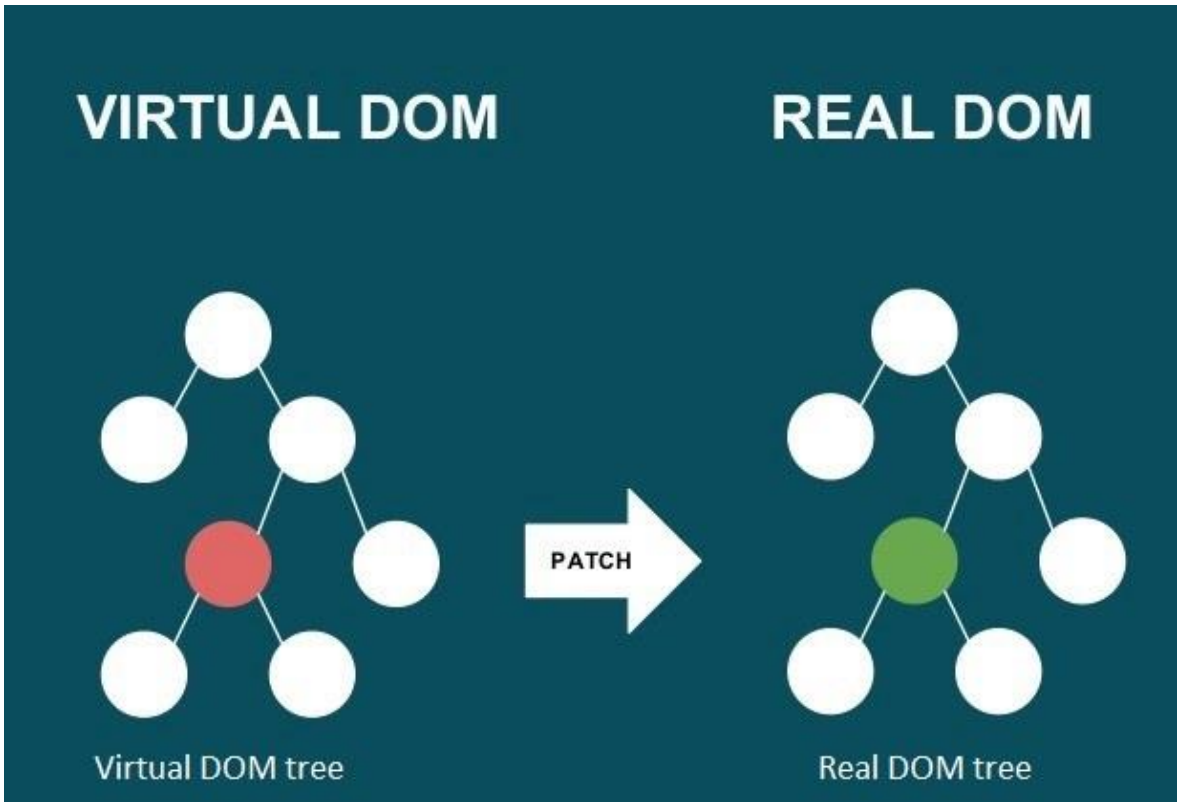
2.1.3 Virtual DOM

Yksi ohjelmoinnin perusongelmista on se, miten voidaan käsitellä tiloja (state). Oletetaan, että käyttöliittymän kehityksessä halutaan näyttää samaa dataa useissa eri paikoissa. ReactJS ratkaisee tämän ongelman virtual DOM-mallilla. Kun komponentin tila muuttuu, se päivittyy ensin virtual DOM-malliin. Virtual DOM on olemassa todellisen DOM-mallin tai jonkin muun renderoitavan kohteen päällä. Tämä ratkaisee ongelmat tilojen käsittelyssä tehokkaasti. Aina kun niihin tehdään muutoksia, React löytää parhaimman tavan tehdä muutoksia alla olevaan DOM-rakenteeseen. (Survivejs [Viitattu 4.4.2018].)

React suorittaa DOM-mallin siirtymisen lineaarisesti. Facebookin dokumentaatiosta nähdään, että virtual DOM-mallin algoritmi on rakennettu kahdesta olettamuksesta, jotka takaavat prosessin nopeuden:

- Kaksi elementtiä, joilla on erilainen tyyppi luo erilaisia puita.
- Kehittäjät voivat viivata avain-propsilla, mitkä elementit pysyvät stabiileina renderoinneissa. (Bas 2017.)

Näiden olettamuksien perusteella React ottaa sisäänrakennetun mallin DOM-puusta eli virtual DOM-mallista ja vertaa sitä todelliseen DOM-malliin, joka alkaa kahdesta juurielementistä. Jos juurielementeillä on erilaisia tyyppejä, React luo uuden puun ja kaikki aiemmat DOM-solmut poistetaan. Komponentit, jotka sijoittuvat juuritasolle ja siitä alaspäin, suorittavat elinkaarimetodin componentWillUnmount(). Tämän jälkeen React luo uuden puun ja komponentit suorittavat componentWillMount()- ja componentDidMount()-elinkaarimetodit. On myös tärkeää ymmärtää, että kun DOM-solmuja poistetaan, solmujen sisällä olevat tilat (state) poistuvat. Tätä prosessin periaatetta voidaan seurata kuvasta 3. (Bas 2017.)



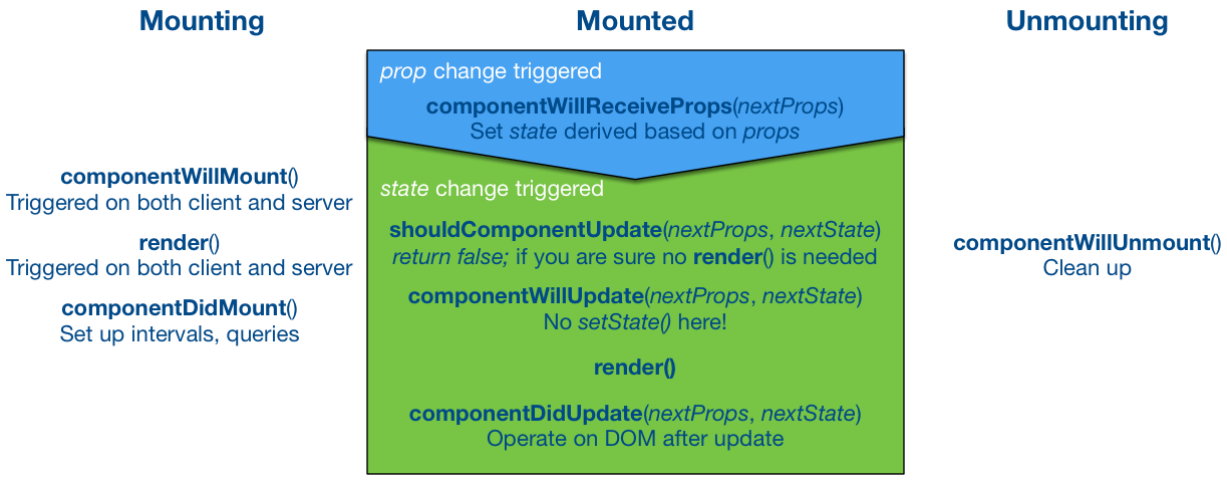
Kuva 3. Virtuaalipuut
(Bas 2017).

DOM-mallin käsittely edellä kuvatulla tavalla johtaa suorituskyvyn paranemiseen. DOM-mallin manuaalinen käsittely on yleensä tehotonta ja sitä on vaikea optimoida. React-kirjaston DOM-manipulaation ansiosta voidaan säästää paljon aikaa ja vaivaa. (Survivejs [Viitattu 4.4.2018].)

2.2 Elinkaarimetodit

Jokaisella komponentilla on useita elinkaarimetodeita, joita voidaan ajaa tietyn aikavälein koodin suorittamisen aikana. Metodeja, joiden etuliitteenä on will, kutsutaan juuri ennen kuin jotain tapahtuu. Metodeita, joiden etuliitteeseen kuuluu did, kutsutaan heti sen jälkeen, kun jotain on tapahtunut. (React [Viitattu 5.4.2018].)

Kuvasta 4 nähdään elinkaarimetodien kolme eri vaihetta: kiinnitys (mounting), kiinnitetty (mounted) ja irrotus (unmounting). Jokaisella näistä on omia liittyviä metodeja.



Kuva 4. Komponentin elinkaarimetodit.
(Survivejs [Viitattu 6.4.2018].)

Kiinnityksessä (mounting) kutsutaan seuraavia metodeja, kun komponentti luodaan ja lisätään DOM-malliin:

- Constructor() eli konstruktori React-komponentille. Tämä kutsutaan ennen kuin se kiinnitetään. React-komponentin alaluokan konstruktorin toteutuksessa tulisi kutsua super(props)-metodi ennen mitään muuta lausuntoa. Muuten propsia ei ole määritelty konstruktorissa, mikä voi johtaa virheilmoituksiin.
- Render() päivittää komponentin tilan. Tämä suoritetaan kerran tilan muutoksesta huolimatta.
- ComponentWillMount() käynnistyy ennen mitä tahansa renderointia. Yksi tapa käyttää tätä on ladata dataa asynkronisesti ja pakottaa renderointi setState-metodin kautta. Näin render()-metodi havaitsee päivitetyn tilan ja se suoritetaan vain kerran huolimatta tilan muutoksista. Tämä käynnistyy silloin, kun se renderoidaan palvelimella.
- ComponentDidMount() käynnistyy alustavan renderoinnin jälkeen ja tämän kautta käyttäjä pääsee käsiksi DOM:iin. Tätä koukkua (hook) voidaan käyttää esimerkiksi jQuery-liitännäisen pakkauksessa komponentin sisälle. Metodi ei käynnisty silloin, kun se renderoidaan palvelimella. (Survivejs [Viitattu 6.4.2018].)

Kiinnitetty (mounted)-metodin voi laukaista kun muutoksia tapahtuu props- tai state-tiloissa. Näitä seuraavia metodeja voidaan kutsua, kun komponentteja renderoidaan uudelleen:

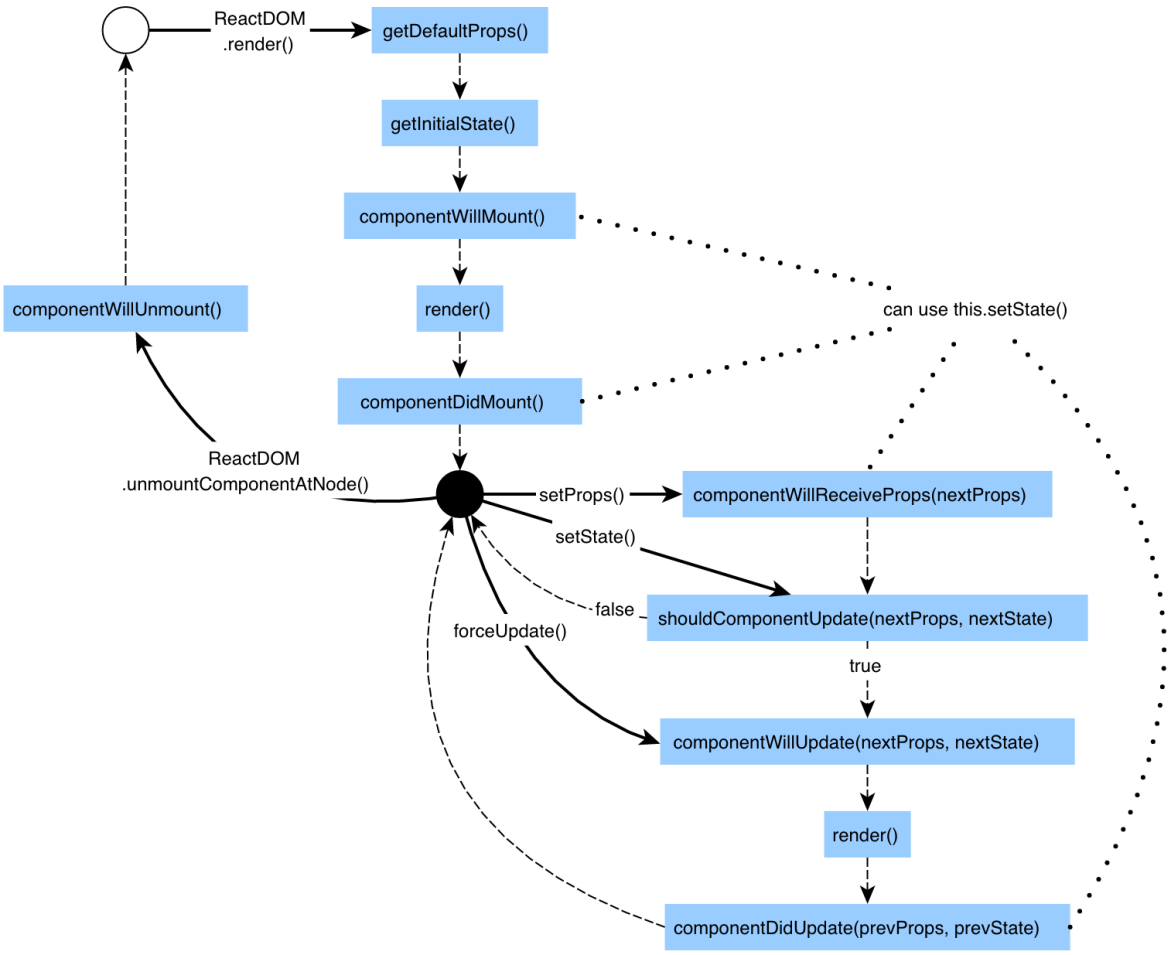
- ComponentWillReceiveProps() käynnistyy, kun komponentti vastaanottaa uusia ominaisuuksia (props). Käyttäjä voi esimerkiksi muokata komponentin tilaa vastaanotetun propsin perusteella.
- ShouldComponentUpdate()-metodin avulla voidaan optimoida renderointi. Jos tarkistetaan props sekä state ja huomataan, ettei päivityksiä tarvita, tämä palautetaan takaisin falsena.
- ComponentWillUpdate() käynnistyy shouldComponentUpdate()-metodin jälkeen ja ennen render()-metodia. Tässä ei ole mahdollista käyttää setState()-metodia, mutta voidaan esimerkiksi asettaa luokan (class) props.
- ComponentDidUpdate() käynnistyy renderoinnin jälkeen. Käyttäjä voi täällä muokata DOM-mallia. Tätä voidaan myös hyödyntää muiden koodien mukauttamisessa Reactin kanssa. (Survivejs [Viitattu 6.4.2018].)

Irroituksessa (unmounting) metodia kutsutaan, kun komponenttia poistetaan DOM-mallista:

- ComponentWillUnmount() käynnistyy, kun komponentti irrotetaan DOM-mallista. Tämä on hyvä tilaisuus suorittaa puhdistuksia, kuten ajastimien tai omien erilaisten DOM-elementtien poistoja. (Survivejs [Viitattu 6.4.2018].)

Virheenkäsittelyssä (error handling) käytetään metodia componentDidCatch() kun renderoinnin aikana, elinkaaressa tai minkä tahansa lapsikomponentin konstruktorissa tapahtuu virhe (Survivejs [Viitattu 6.4.2018]).

Kuvassa 5 nähdään elinkaarien toiminta React-sovelluksessa.



Kuva 5. Elinkaarien toiminnot.
(McGinnis 2016.)

2.3 Single-page application (SPA)

Single-page application on sovellus, joka lataa yhden HTML-sivun ja kaikki sen tarvitsemat resurssit (JavaScript ja CSS), joita sovellus käyttää toimiakseen. Sovelluksen ideana on se, että pysytään samalla www-sivulla, vaikka sivun sisältö muuttuisi. Kaikki vuorovaikutukset sivun tai sivujen kanssa eivät vaadi edestakaista käyntiä palvelimella, mikä tarkoittaa, että sivua ei tarvitse ladata uudelleen. (React [Viitattu 6.4.2018].)

Single-page-sovelluksen periaate on siis se, että käyttäjän tarvitsee navigoida jonnekin vain kerran. Selaimessa DOM renderoidaan kokonaan vain yhden kerran. Tämän jälkeen JavaScript suorittaa taustalla minkä tahansa toimenpiteen palvelimella

ja muuttaa näkymää tarpeen mukaan. Kirjoitushetkellä SPA käyttää JavaScript-kehyksiä, jotka mahdollistavat tämän tekniikan, kuten esimerkiksi mallinnuksen ja reitityksen. Toteuttamiseen käytetään paljon AJAX-menetelmää, mutta nykyään SPA-sovelluksia voidaan toteuttaa muillakin tavoilla. (Sotelo 2014.)

2.4 Node.js

Tässä työssä sivuston luomiseen käytetään Node.js-ohjelmistoa. Node.js on JavaScript runtime-ympäristö, joka on rakennettu Chromen V8-JavaScript-moottorille. Se käyttää tapahtumavetoista non-blocking I/O mallia, joka tekee siitä kevyen ja tehokkaan. (Node.js [Viitattu 7.4.2018].)

Tämä mahdollistaa JavaScript-ohjelmoinnin muilla alustoilla kuten verkkoselaimilla ja tarjoaa monia ohjelmointirajapintoja. Asynkronisilla tapahtumilla ajettu Node on suunniteltu rakentamaan skaalautuvia verkko-ohjelmia. (Node.js [Viitattu 8.4.2018].)

Node.js-ohjelmiston pakettiekosysteemi NPM on maailman suurin avoimien lähdekirjastojen ekosysteemi. NPM-työkalua hallitaan yleisesti komentoriviltä ja se tallentaa konfiguraation package.json-nimiseen tiedostoon. Tiedostoon tallennetaan esimerkiksi sovelluksen riippuvuuksien tietoja. (Heller 2017.)

2.5 React-kirjaston asennus

Asennuksessa käytetään Windowsin komentokehotetta. Aluksi luodaan C-asemalle tiedosto projektia varten ja asennetaan React-kirjasto yhdellä komennolla, joka kokoaa kaikki tarvittavat paketit käyttöä varten (kuva 6).

```
C:\Projekti>npm install -g  create-react-app
```

Kuva 6. React-kirjaston asennus

Kun kaikki paketit on asennettu, luodaan itse projekti React-sivua varten ja annetaan tälle nimi, joka on tässä tapauksessa reactprojekti (kuva 7). Tämän komennon suorittamisessa saattaa kestää muutamia minuutteja.

```
C:\Projekti>create-react-app reactprojekti
```

Kuva 7. React-projektin luonti

Seuraavaksi siirrytään Projekti-kansiossa sijaitsevaan reactprojekti-kansioon, josta käynnistetään kyseinen projekti npm start -komennolla (kuva 8).

```
C:\Projekti\reactprojekti>npm start
```

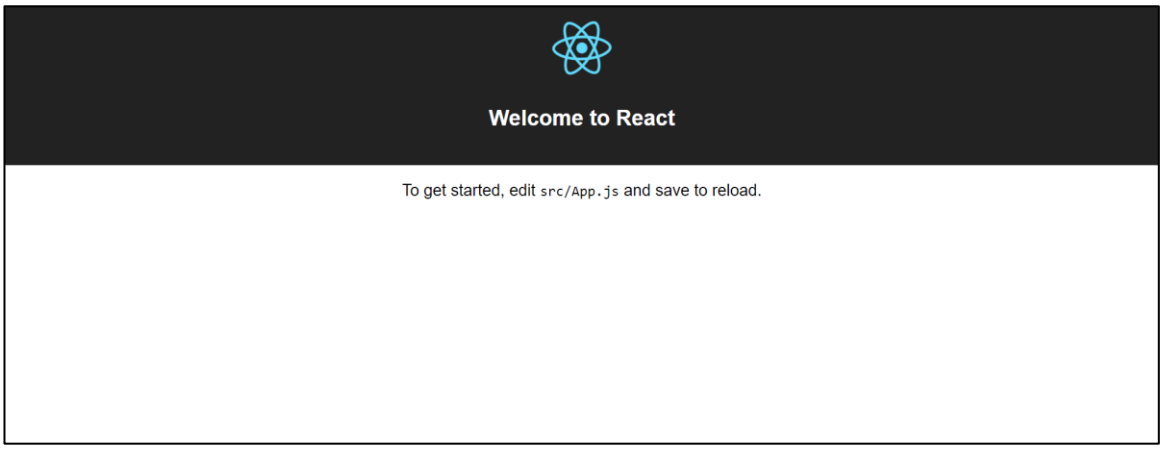
Kuva 8. React-sovelluksen käynnistys

Komennon suorittamisen jälkeen saadaan kuvassa 9 nähtävissä oleva ilmoitus. React-sivu on luotu kohteeseen http://localhost:3000, johon sovellus voidaan koodata.

```
Compiled successfully!

The app is running at:

  http://localhost:3000/

Note that the development build is not optimized.
To create a production build, use npm run build.
```

Kuva 9. React-sovellus toiminnassa

Kuvassa 10 on esitetty React-sivun oletusnäkymä, joka sisältää headerissa sijaitsevan React-logon ja tervehdyksen.



Kuva 10. React-pohjaisen sivun oletusnäkymä

3 OHJELMOINTIRAJAPINTOJEN KÄYTTÖ

3.1 PEPPI-ohjelmointirajapinta

Peppi on avoin SOA-menetelmin rakennettu järjestelmäkokonaisuus erilaisten kouluorganisaatioiden opetus- ja koulutusprosesseja varten. Se yhdistää opiskelijoiden, opettajien sekä opintohallinnon tehtävät samaan järjestelmäkokonaisuuteen. Peppi-järjestelmän kehittäjänä toimii Peppi-konsortio, joka on hankintayksikköstatuksen omaavien eri koulutusorganisaatioiden yhteenliittymä ja kehittäjäyhteisö. (Peppi-konsortio [Viitattu 8.4.2018].)

PEPPI-rajapintoja kutsutaan käyttämällä GET- ja POST-pyyntöjä kohteesta riippuen. POST-pyynnöissä parametrit annetaan aina JSON-muodossa, ja GET-pyynnöissä syötetään haettavan kohteen tunniste (ID). Kieli valitaan antamalla haluttu kieli parametreissa header ja query. Oletuskielenä toimii suomi (fi), mutta vaihtoehtoisesti sisällön voi pyytää myös englanniksi (en). JSON-muodossa päivämäärät kirjoitetaan aina muotoon ”yyyy-MM-dd’T’HH:mm” esimerkiksi ”2018-12-31T09:30”. Haettaviin tietoihin kuuluvat tilat/varaukset, opintojaksot, opintojaksototeutukset ja opetussuunnitelmat. (Peppi [Viitattu 8.4.2018].)

POST-pyynnöllä luodaan uusi resurssi palvelimelle. GET-pyynnöllä haetaan resurssi palvelimelta. Seuraavissa luvuissa on esitetty esimerkit näiden pyyntöjen tekemisestä PEPPI-järjestelmään.

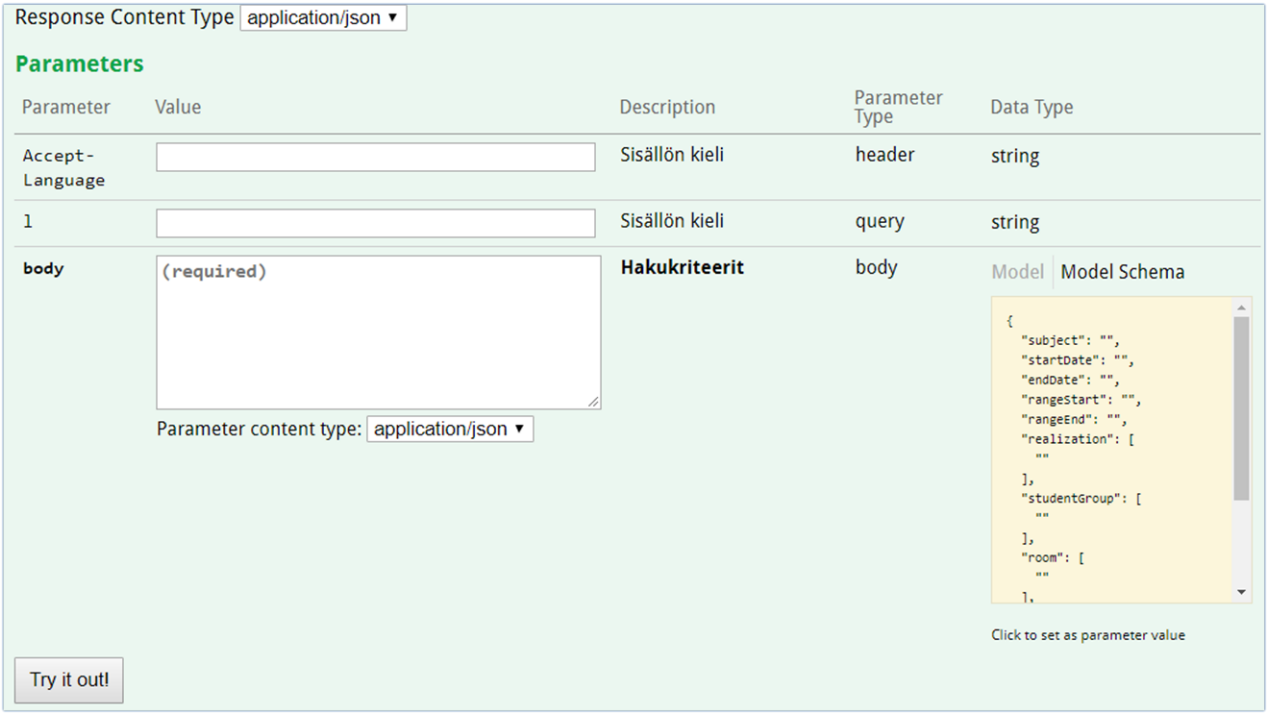
3.1.1 POST-pyyntö

POST-pyynnöillä voidaan luoda uusia resursseja palvelimelle. Uusia resursseja luodaan tiedostoista, jotka lähetetään POST-metodin kanssa. POST-metodin suorittama funktio määritetään palvelimella ja se yleensä riippuu URI-tunnisteesta. (Fielding ym. 1999, 53.)

POST-pyynnön ominaisuuksia ovat seuraavat:

- POST-pyynnön yhteydessä lähetetään dataa viestirunkoon eli bodyyn. Tämän mukana on yleensä myös ylätunnisteita eli headereita, kuten Accept-Language ja Content-Type.
- POST-metodilla voidaan siirtää enemmän dataa palvelimeen.
- POST-pyynnöt eivät jää välimuistiin. (Paul 2017.)

Kuvassa 11 nähdään PEPPI-rajapinnan POST-pyyntömenetelmän parametri-ikkuna, johon voidaan syöttää kieli sekä mahdollisia parametreja kyselyä varten.



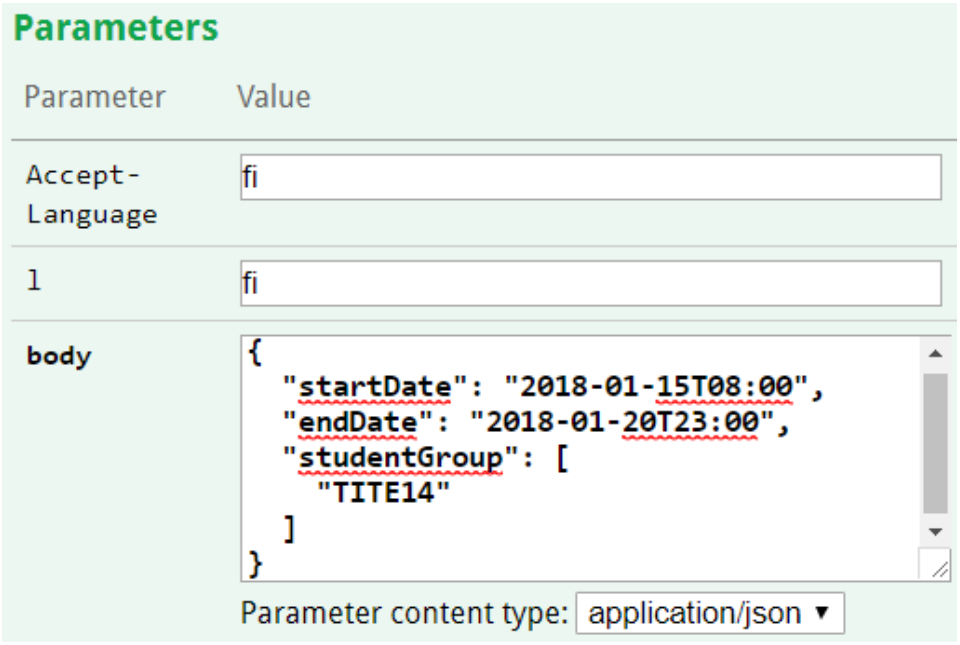
Kuva 11. POST-pyynnön parametri-ikkuna

Kuvassa 12 nähdään kaikki mahdolliset PEPPI-kyselyn parametrit, joita voidaan käyttää POST-pyynnön viestirungossa eli bodyssa.

```
{
  "subject": "",
  "startDate": "",
  "endDate": "",
  "rangeStart": "",
  "rangeEnd": "",
  "realization": [
    ""
  ],
  "studentGroup": [
    ""
  ],
  "room": [
    ""
  ],
  "building": [
    ""
  ],
  "from": 0,
  "size": 0
}
```

Kuva 12. Parametrit kyselyä varten

Kuvassa 13 nähdään esimerkki ryhmän TITE14 viikon 3 lukujärjestyksen hausta.



Kuva 13. Esimerkki POST-pyynnöstä

Kuvassa 14 on esitetty POST-pyynnön vastaus.

Kuva 14. POST-pyynnön vastaus

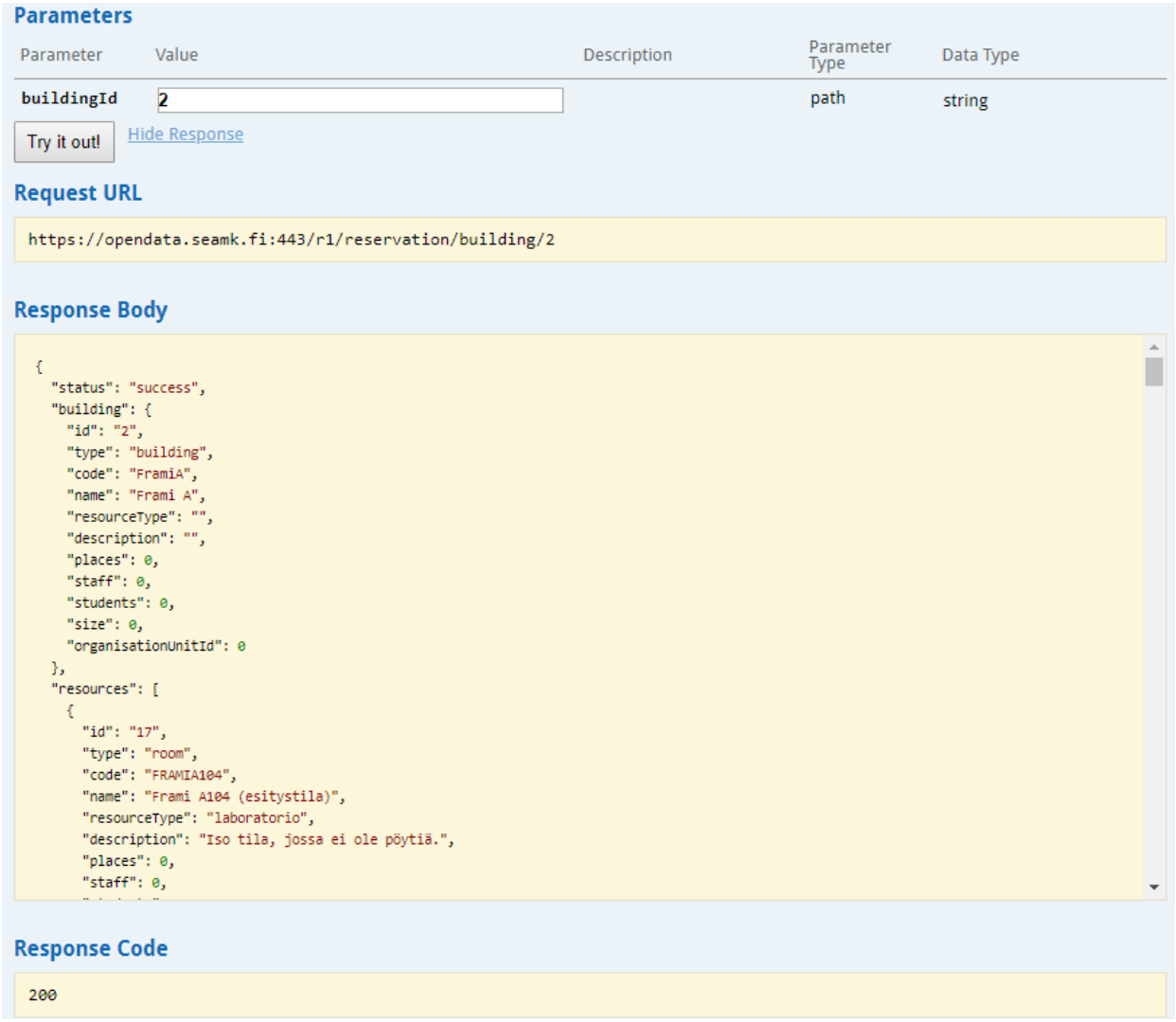
3.1.2 GET-pyyntö

GET-pyynnöillä haetaan resursseja palvelimelta. Haettavia resursseja ei muuteta, vaan ne haetaan samanlaisena kuin ne ovat palvelimessa. Haettavia resursseja määritellään URI-tunnisteessa. (Fielding ym. 1999, 52.)

GET-pyynnön ominaisuuksia ovat seuraavat:

- GET-pyynnön yhteydessä pyyntöparametri sisältyy URL-merkkijonoon.
- GET-metodilla voidaan välittää vain rajallinen määrä dataa.
- GET-pyynnön jäävät välimuistiin ja ne voidaan lisätä kirjanmerkkeihin. GET-metodeita ei siis kannata käyttää luottamuksellisten tietojen kanssa. (Paul 2017.)

Kuvassa 15 tarkastellaan esimerkkinä hakua Framin A-rakennuksen kaikista tiloista, jotka saadaan syöttämällä kyseisen rakennuksen tunnistusnumero, joka on 2.

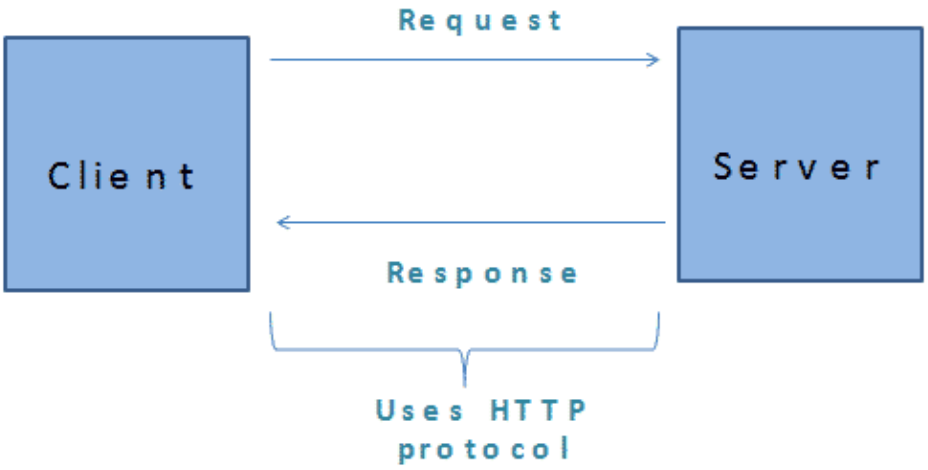


Kuva 15. GET-pyynnön esimerkki ja vastaus

3.2 HTTP-pyyntöesimerkkejä

HTTP (Hypertext Transfer Protocol) on sovelluskerroksen tiedonsiirtoprotokolla. Se on suunniteltu web-selainten ja web-palvelimien väliseen viestintään, mutta sitä voidaan myös käyttää muihin tarkoituksiin. HTTP perustuu asiakas-palvelin-malliin, eli asiakas avaa yhteyden pyynnön tekemiseen ja odottaa, kunnes se saa vastauksen. (MDN Web Docs 2018b.)

HTTP on tilaton protokolla, mikä tarkoittaa sitä, että palvelin ei säilytä mitään dataa kahden pyynnön välillä. Vaikka protokolla yleensä perustuu TCP/IP-kerrokseen, sitä voidaan käyttää millä tahansa luotettavalla siirtokerroksella. (MDN Web Docs 2018b.)



Kuva 16. Siirtoprotokollan periaate.
(TekTutorialsHub [Viitattu: 10.4.2018].)

3.2.1 jQuery AJAX

AJAX-tekniikalla voidaan pyytää dataa palvelimelta, sekä päivittää osia verkkosivulla. Siihen kuuluu sisäänrakennettu XMLHttpRequest-objekti, jolla voidaan pyytää dataa verkkopalvelimesta. jQuery AJAX on yksi suosituimmista tavoista käyttää HTTP-pyyntöjä. (Six revisions 2010.)

Esimerkkinä käytetään Framin A-rakennuksen kaikkien tilojen haku GET-pyyntömetodilla. jQuery AJAX-tekniikan käyttöä varten asennetaan jQuery Windows-komentokehotteessa (kuva 17) ja liitetään se App.js-tiedostoon. Lisäksi liitetään Todos-komponentti, johon sisältyy datan käsittely ja tulostus (kuva 18).

```
C:\Projekti\reactprojekti>npm install jquery --save
```

Kuva 17. jQueryn asennus

```
import React, { Component } from 'react';
import './App.css';

import $ from 'jquery';
import Todos from "./Components/Todos"
```

Kuva 18. jQueryn ja komponenttien liittäminen App.js-tiedostoon

Ensimmäisenä app.js-tiedostossa luodaan constructor-metodi, jossa huolehditaan pääasiassa kahdesta asiasta. Kun constructor-metodia on kutsuttu, käytetään super()-metodia, jonka avulla voidaan välittää propseja lapsikomponentteihin. Tämän jälkeen asetetaan tila. Kun asetetaan alkutilaa, sen tulisi olla tyhjä. Esimerkiksi tässä tilanteessa on tyhjä lista, johon asetetaan haettu data. Kun pyyntömetodia käytetään, asetetaan uusi tila datalle, joka on haettu.

Haussa asetetaan parametrit, kuten haun tyyppi, url sekä datan tyyppi. Rajapinnan käyttöön tarvitaan oma API-avain, jolla päästään rajapinnan tietoihin. Haun yhteydessä käytetään Basic Authorization -metodia, jolla voidaan syöttää käyttäjänimi ja salasana ylätunnisteeseen eli headeriin kun suoritetaan hakupyyntö. Onnistuneen haun jälkeen asetetaan tila todos-alkutilalle, johon siirretään haettu data. Virhetilanteen tapahtuessa voidaan asettaa ilmoitus konsoliin. Näitä askeleita seurataan kuvassa 19.

```
class App extends Component {
  constructor(){
    super();
    this.state = {
      todos:[]
    }
  }
  getTodos(){
    $.ajax({
      type: "GET",
      url: 'https://opendata.seamk.fi/r1/reservation/building/2',
      dataType:'json',
      headers: {
        Authorization: "Basic " + btoa("username" + ":" + "password")
      },
      cache: false,
      success: function(data){
        this.setState({todos: data.resources ? data.resources : []}, function(){
          console.log(this.state);
        });
      }.bind(this),
      error: function(xhr, status, err){
        console.log(err);
      }
    });
  }
```

Kuva 19. Konstruktori sekä jQuery-haun esimerkki

GET-pyynnön vastausta voidaan tarkastella konsolissa, jossa seurataan vastaan-otettua dataa (kuva 20).

▼ todos: Array(36)
▶ 0: {id: "17", type: "room", code: "FRAMIA104", name: "Frami A104 (esitystila)", resourceType: "laboratorio", …}
▶ 1: {id: "25", type: "room", code: "FRAMIA130.1", name: "Frami A130.1 (Käyttötekniikka)", resourceType: "laboratorio", …}
▶ 2: {id: "26", type: "room", code: "FRAMIA130.3", name: "Frami A130.3 (robotiikan labra 25)", resourceType: "laboratorio", …}
▶ 3: {id: "27", type: "room", code: "FRAMIA140.1", name: "Frami A140.1 (automaatiotekniikan labra 24)", resourceType: "laboratorio", …}
▶ 4: {id: "28", type: "room", code: "FRAMIA140.2", name: "Frami A140.2 (atk 30)", resourceType: "atk-luokka", …}
▶ 5: {id: "29", type: "room", code: "FRAMIA140.4", name: "Frami A140.4 (materiaalitekniikan labra)", resourceType: "laboratorio", …}
▶ 6: {id: "30", type: "room", code: "FRAMIA150.7", name: "Frami A150.7 (virtuaalikeskus CAVE)", resourceType: "laboratorio", …}
▶ 7: {id: "31", type: "room", code: "FRAMIA210.2", name: "Frami A210.2 (teoria 56)", resourceType: "teorialuokka", …}
▶ 8: {id: "32", type: "room", code: "FRAMIA210.3", name: "Frami A210.3 (atk 35)", resourceType: "atk-luokka", …}
▶ 9: {id: "33", type: "room", code: "FRAMIA220.3", name: "Frami A220.3 (sul.labra 26)", resourceType: "laboratorio", …}
▶ 10: {id: "34", type: "room", code: "FRAMIA220.5", name: "Frami A220.5 (ele.labra 26)", resourceType: "laboratorio", …}
▶ 11: {id: "35", type: "room", code: "FRAMIA240.2", name: "Frami A240.2 (atk 35)", resourceType: "atk-luokka", …}
▶ 12: {id: "36", type: "room", code: "FRAMIA240.4", name: "Frami A240.4 (atk 35)", resourceType: "atk-luokka", …}
▶ 13: {id: "37", type: "room", code: "FRAMIA250.1", name: "Frami A250.1 (teoria 56)", resourceType: "teorialuokka", …}
▶ 14: {id: "38", type: "room", code: "FRAMIA320.2", name: "Frami A320.2 (kielistudio 36)", resourceType: "atk-luokka", …}
▶ 15: {id: "39", type: "room", code: "FRAMIA320.6", name: "Frami A320.6 (fysiikan labra 4)", resourceType: "laboratorio", …}
▶ 16: {id: "40", type: "room", code: "FRAMIA320.7", name: "Frami A320.7/8 (teoria 56)", resourceType: "teorialuokka", …}
▶ 17: {id: "41", type: "room", code: "FRAMIA340.1", name: "Frami A340.1 (atk 35)", resourceType: "atk-luokka", …}

Kuva 20. jQuery GET -pyynnön konsolin loki

Todos.js-komponentissa voidaan kartoittaa haettua dataa. Tässä tilanteessa käytetään map()-funktiota, jolla voidaan käydä läpi haettu data ja poimia tietoa avaimien perusteella. Tässä esimerkissä haetaan kaikki data, jonka avaimena on name. Lisäksi liitetään myös TodoItem.js-komponentti, jossa käsitellään kartoitettua dataa (kuva 21).

```
import React, { Component } from 'react';
import TodoItem from './TodoItem';

class Todos extends Component {
  render() {
    let todoItems;
    if(this.props.todos){
      todoItems = this.props.todos.map(todo => {
        return (
          <TodoItem key={todo.name} todo={todo} />
        );
      });
    }
    return (
      <div className="Todos">
        <h3>Tulokset:</h3>
          {todoItems}
      </div>
    );
  }
}
export default Todos;
```

Kuva 21. Datan kartoitus Todos.js-komponentissa

TodoItem.js-komponentissa voidaan käsitellä kartoitettua dataa, kuten nähdään kuvasta 22.

```
import React, { Component } from 'react';

class TodoItem extends Component {
  render() {
    return (
      <li className="Todo">
        <strong>{this.props.todo.name}</strong>
      </li>
    );
  }
}
export default TodoItem;
```

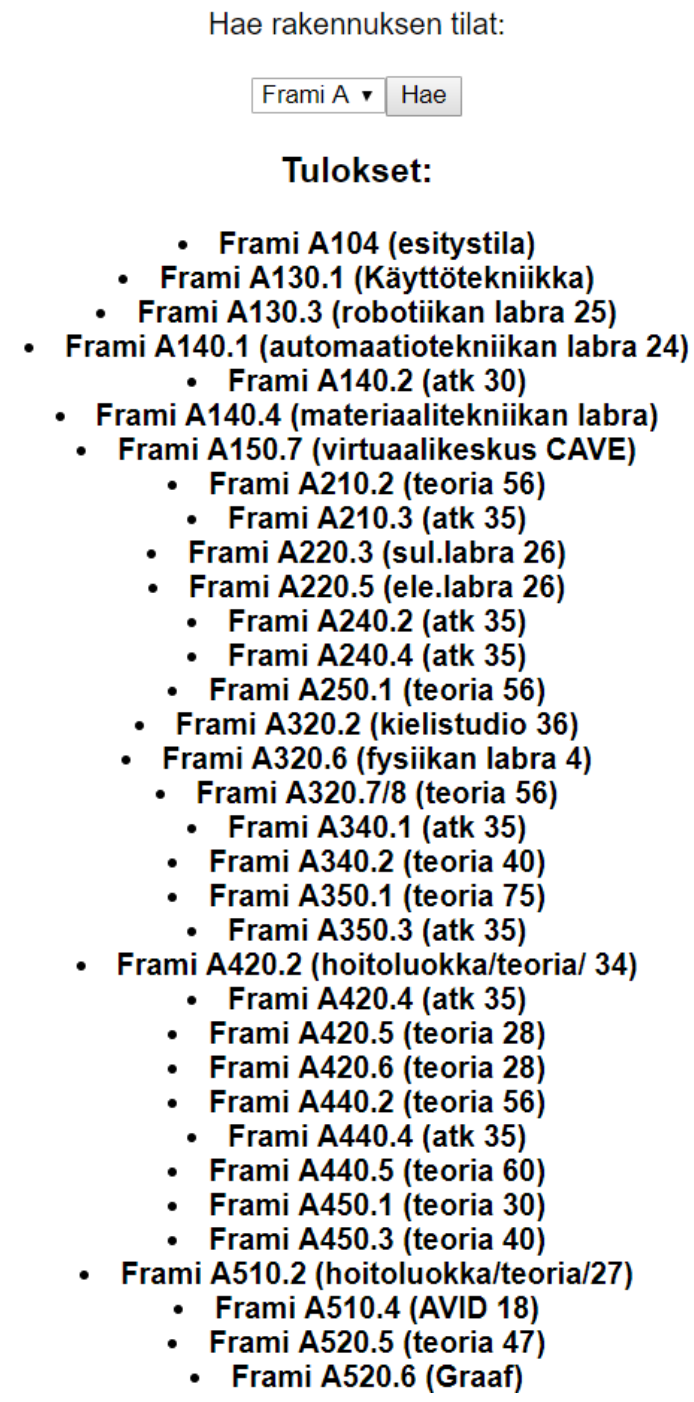
Kuva 22. Kartoitetun datan käsittely TodoItem.js-komponentissa

Seuraavaksi luodaan elinkaarimetodit ja lopuksi render-metodi, jonne tulostetaan kartoitettu data (kuva 23).

```
  componentDidMount(){
    this.getTodos();
  }

  render() {
    return (
      <div className="App">
        <Titles />
        <Form />
        <Todos todos={this.state.todos}/>
      </div>
    );
  }
}
export default App;
```

Kuva 23. Elinkaarimetodit ja renderointi

Näin sivulle saadaan tulostettua Framin A-rakennuksen tilat, jotka nähdään kuvassa 24.



Kuva 24. jQuery GET -pyynnön tulostus

3.2.2 Axios

Axios on promise-pohjainen HTTP-client, joka toimii selaimessa sekä node.js-ympäristössä. Siinä on yksi API-rajapinta, joka käsittelee XMLHttpRequesteja ja solmujen HTTP-käyttöliittymää. Sen lisäksi Axios pakkaa pyyntöjä käyttämällä polyfillia, joka on suunniteltu käyttämään ES6-version uutta promise-syntaksia. (Bernardes 2015.)

Esimerkkinä käytetään HTTP POST -pyyntömetodia, jolla haetaan ryhmän viikoittaisen kurssin aiheet. Ensimmäisenä asennetaan Axios komentokehotteessa (kuva 25) ja liitetään se App.js-tiedostoon (kuva 26).

```
C:\Projekti\reactprojekti>npm install axios --save
```

Kuva 25. Axioksen asennus

```
import axios from 'axios';
```

Kuva 26. Axioksen liittäminen App.js-tiedostoon

Ensimmäisenä luodaan constructor-metodi. Kun constructor-metodia on kutsuttu, käytetään super()-metodia, jolla välitetään propseja lapsikomponentteihin. Tämän jälkeen asetetaan alkutila. Kun pyyntömetodia käytetään, asetetaan uusi tila datalle setState-metodilla. Myös getTodos()-funktio sidotaan.

POST-pyynnön mukaan lisätään kyselyparametrit (postData) viestirunkoon eli bodyyn, jotta saadaan haluttu data. Tämän jälkeen määritellään ylätunnisteet eli headerit, joihin voi syöttää Basic Authentication, jotta rajapintoihin pääsy olisi mahdollista.

Seuraavaksi suoritetaan axios.post-komento, johon kuuluu url, josta dataa haetaan sekä kyselyparametrit ja headerit. Tämän jälkeen asetetaan uusi tila todos-alkutilalle setState-metodilla. Nämä askeleet nähdään kuvassa 27.

```
class App extends Component {
  constructor(props){
    super(props);
    this.state = {
      todos:[]
    }
    this.getTodos = this.getTodos.bind(this);
  }

getTodos(){
  var postData = {
    "startDate": "2018-02-05T08:00",
    "endDate": "2018-02-10T23:00",
    "studentGroup": ["TITE14"]
  };
  let axiosConfig = {
    headers: {
      'Content-Type': 'application/json',
      'Authorization': "Basic " + btoa("username" + ":" + "password")
    }
  };
  axios.post('https://opendata.seamk.fi:443/r1/reservation/search?l=fi', postData, axiosConfig)
  .then((response) => {
    console.log(response);
    if (response.data.reservations) {
      this.setState({todos: response.data.reservations})
      console.log(response.data.reservations);
    }
  })
  .catch((err) => {
    console.log("Error: ", err);
  })
}
```

Kuva 27. Axios POST -pyynnön esimerkki

Konsolissa voidaan seurata haettua dataa, joka nähdään kuvassa 28.

```
subject: "Opinnäytetyö KC00F99999-127", modifiedDate: "2018-02-06T06:03:41", startDate: "2018-02-06T11:00:00", endDate: "2018-02-06T11:45:00", …}
subject: "Technical English A800BD63-3002", modifiedDate: "2018-02-06T06:03:41", startDate: "2018-02-06T12:15:00", endDate: "2018-02-06T14:00:00", …}
subject: "Virtualisointiympäristöt KL04BT53033-3001", modifiedDate: "2018-02-06T06:03:40", startDate: "2018-02-06T14:15:00", endDate: "2018-02-06T16:00:00", …}
subject: "Java EE KL04BTOHTE2-3001", modifiedDate: "2018-02-06T06:03:28", startDate: "2018-02-06T14:15:00", endDate: "2018-02-06T16:00:00", …}
subject: "Java EE KL04BTOHTE2-3001", modifiedDate: "2018-02-07T06:03:31", startDate: "2018-02-07T08:00:00", endDate: "2018-02-07T10:45:00", …}
subject: "Langaton tiedonsiirto KL04BT42041-3002", modifiedDate: "2018-02-07T06:03:42", startDate: "2018-02-07T11:00:00", endDate: "2018-02-07T14:00:00", …}
subject: "Langaton tiedonsiirto KL04BT42041-3002", modifiedDate: "2018-02-08T06:03:40", startDate: "2018-02-08T11:00:00", endDate: "2018-02-08T14:00:00", …}
subject: "Technical English / tuntimuutos", modifiedDate: "2018-02-08T12:50:41", startDate: "2018-02-08T14:15:00", endDate: "2018-02-08T16:00:00", …}
subject: "Tehoelektroniikka KL25AB10110-3004", modifiedDate: "2018-02-08T12:50:34", startDate: "2018-02-08T16:15:00", endDate: "2018-02-08T18:00:00", …}
subject: "Virtualisointiympäristöt KL04BT53033-3001", modifiedDate: "2018-02-09T06:03:39", startDate: "2018-02-09T11:00:00", endDate: "2018-02-09T14:00:00", …}
```

Kuva 28. Axios POST -pyynnön konsolin loki

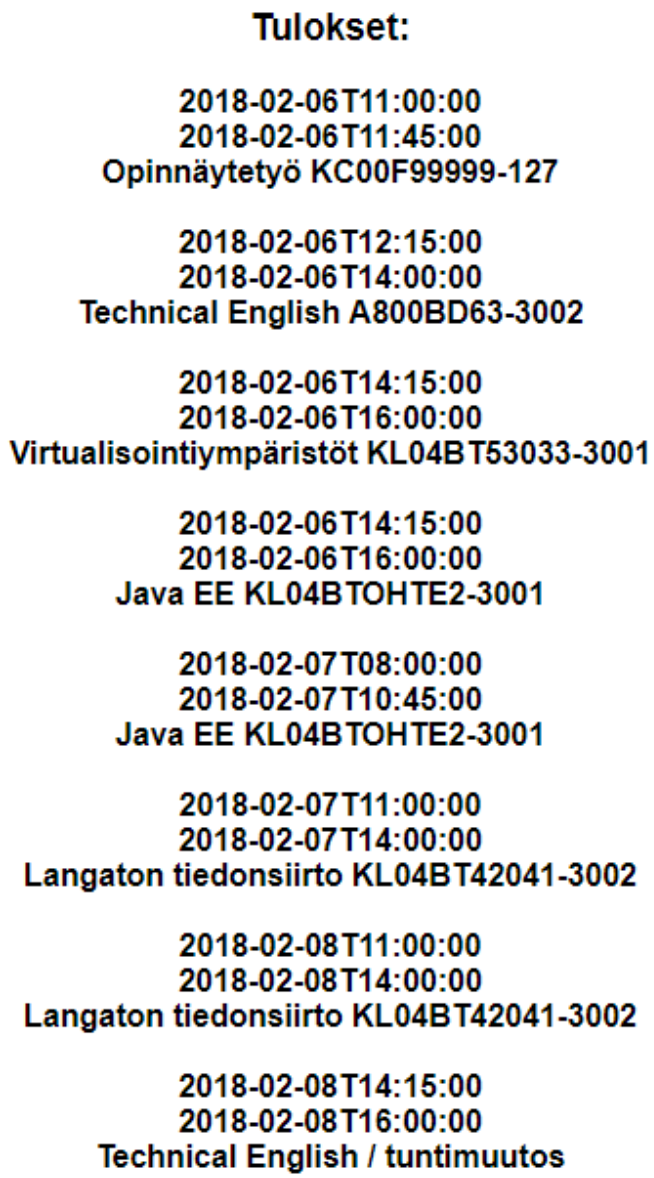
TodoItem.js-komponentissa kartoitetaan dataa. Komponentissa valitaan subject, startDate ja endDate tulostamista varten. Tämä nähdään kuvassa 29.

```
import React, { Component } from 'react';

class TodoItem extends Component {
  render() {
    return (
      <div className="Todo">
        <strong>{this.props.todo.startDate}</strong>
        <br/>
        <strong>{this.props.todo.endDate}</strong>
        <br/>
        <strong>{this.props.todo.subject}</strong>
        <br/><br/>
      </div>
    );
  }
}
export default TodoItem;
```

Kuva 29. Axios TodoItem.js -komponentti

Lopuksi käydään läpi elinkaarimetodit ja renderoidaan kohteet (kuva 24).

Näin sivulle saadaan tulostettua TITE14-ryhmän viikon kurssien aiheet sekä niiden ajankohdat, jotka nähdään kuvassa 30.



Kuva 30. Axios POST -pyynnön tulostus

3.2.3 Fetch

Fetch on sisäänrakennettu JavaScript ES6 promise -pohjainen API, joka tekee XMLHttpRequestista yksinkertaisen ja helpon datan hakuun GET- sekä POST-pyynnöillä. Siitä on nopeasti tulossa standardikeino käyttää HTTP-pyyntöjä. (MDN Web Docs 2018b.)

Fetch API tarjoaa JavaScript-käyttöliittymän HTTP-pyyntöjen sekä vastausten käsittelyyn. Se tarjoaa myös globaalin fetch()-metodin, joka mahdollistaa helpon ja loogisen tavan hakea resursseja asynkronisesti verkosta. (MDN Web Docs 2018b.)

Fetch eroaa mm. AJAX-tekniikasta seuraavilla tavoilla:

- Fetch()-metodista palautettu promise ei tule hylätyksi HTTP-protokollan virhetilasta johtuen, vaikka vastaus olisikin HTTP 404 tai 500. Se ratkaisee tilan normaalisti ja hylkää promisen vain, jos tapahtuu verkon toimintahäiriö tai jos jokin estää pyynnön suorittamista.
- Oletuksena fetch ei lähetä tai vastaanota yhtään evästeitä palvelimelta, mikä johtaa vahvistamattomiin pyyntöihin, jos sivusto turvautuu käyttäjän istunnon ylläpitämiseen. Evästeiden lähettämistä varten on asetettava ”credentials”-asetus, jonka avulla voidaan lähettää automaattisesti evästeitä nykyiseen verkkotunnukseen. (MDN Web Docs 2018.)

Esimerkkinä käytetään kaikkien SeAMK:n rakennuksien hakeminen fetchin GET-pyynnöllä.

Aluksi asetettiin konstruktorissa error, isLoaded ja alkutila. Tämän jälkeen määritellään ylätunnisteet ja suoritetaan fetch()-funktio, johon kuuluu pyynnön metodin asetus (GET) ja ylätunnisteiden yhdistäminen kyselyyn.

Seuraavaksi luetaan ja jäsennetään data JSON-muotoon json()-metodilla ja muutetaan tilaa setState-metodilla. Lopuksi asetetaan virheilmoitus. Nämä askeleet nähdään kuvassa 31.

```
class App extends Component {
  constructor(props){
    super(props);
    this.state = {
      error: null,
      isLoaded: false,
      todos:[]
    }
  }
  getTodos(){
    let myHeaders = {
      'Content-Type': 'application/json',
      'Authorization': "Basic " + btoa("username" + ":" + "password")
    };
    fetch('https://opendata.seamk.fi:443/r1/reservation/building', {
      method: "GET",
      headers: myHeaders
    })
    .then(res => res.json())
    .then(
      (result) => {
        this.setState({isLoaded: true,
        todos: result.resources ? result.resources : []
      });
        console.log(this.state);
      },
      (error) => {
        this.setState({
          isLoaded: true,
          error
        });
      }
    )
  }
```

Kuva 31. Fetch GET -pyynnön esimerkki

Haettu data kaikista rakennuksista näkyy konsolin lokissa, joka nähdään kuvassa 32.

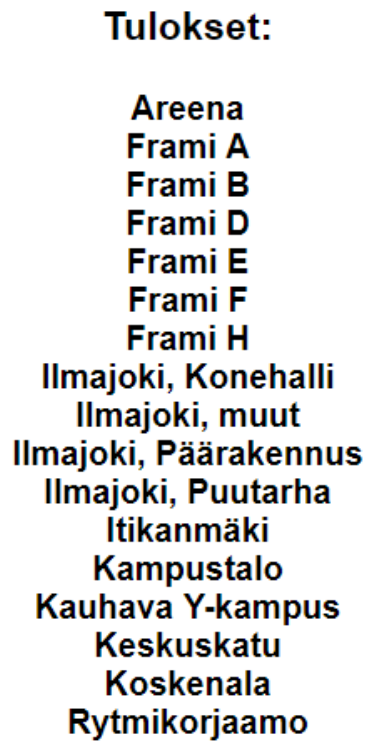

```
isLoaded: true
todos: Array(17)
▶ 0: {id: "4175", type: "building", code: "Areena", name: "Areena", resourceType: "", …}
▶ 1: {id: "2", type: "building", code: "FramiA", name: "Frami A", resourceType: "", …}
▶ 2: {id: "13", type: "building", code: "FramiB", name: "Frami B", resourceType: "", …}
▶ 3: {id: "4174", type: "building", code: "FramiD", name: "Frami D", resourceType: "", …}
▶ 4: {id: "3", type: "building", code: "FramiE", name: "Frami E", resourceType: "", …}
▶ 5: {id: "4", type: "building", code: "FramiF", name: "Frami F", resourceType: "", …}
▶ 6: {id: "5", type: "building", code: "FramiH", name: "Frami H", resourceType: "", …}
▶ 7: {id: "9", type: "building", code: "IlmajokiKh", name: "Ilmajoki, Konehalli", resourceType: "", …}
▶ 8: {id: "11", type: "building", code: "IlmajokiMuut", name: "Ilmajoki, muut", resourceType: "", …}
▶ 9: {id: "8", type: "building", code: "IlmajokiPr", name: "Ilmajoki, Päärakennus", resourceType: "", …}
▶ 10: {id: "10", type: "building", code: "IlmajokiPt", name: "Ilmajoki, Puutarha", resourceType: "", …}
▶ 11: {id: "6", type: "building", code: "Itikanmaki", name: "Itikanmäki", resourceType: "", …}
▶ 12: {id: "7", type: "building", code: "Kampustalo", name: "Kampustalo", resourceType: "", …}
▶ 13: {id: "614", type: "building", code: "Kauhava", name: "Kauhava Y-kampus", resourceType: "", …}
▶ 14: {id: "16", type: "building", code: "Keskuskatu", name: "Keskuskatu", resourceType: "", …}
▶ 15: {id: "15", type: "building", code: "Koskenala", name: "Koskenala", resourceType: "", …}
▶ 16: {id: "12", type: "building", code: "Rytmikorjaamo", name: "Rytmikorjaamo", resourceType: "", …}
```


Kuva 32. Fetch GET -pyynnön konsolin loki

Todos.js-komponentissa kartoitetaan kerätty data, kuten aiemmissa esimerkeissä. TodoItem.js-komponentissa valitaan dataa avaimen perusteella, mikä on tässä esimerkissä name-avain (kuva 22).

Lopuksi käydään läpi elinkaarimetodi ja renderoidaan komponentit tulostusta varten (kuva 23).

Näin tulostetaan sivulle SeAMK:n kaikki rakennukset, jotka näkyvät kuvassa 33.



Kuva 33. Fetch GET -pyynnön tulostus

4 YHTEENVETO

Insinöörityön tavoitteena oli tutustua JavaScript-kielen avoimen lähdekoodin front-end-kirjastoon nimeltään ReactJS, joka on käyttäjien keskuudessa hyvin suosittu. Työssä käytiin myös läpi erilaisia HTTP-pyyntömenetelmiä, joita voidaan käyttää React-kirjaston kanssa. Menetelmiä kokeiltiin SeAMK:n avoimilla PEPPI-rajapinnoilla.

Työssä perehdyttiin React-kirjaston perusteisiin, kuten elinkaarimetodien toimintoihin, virtual DOM-malliin sekä SPA-sovellukseen. HTTP-metodeista käytiin läpi GET- ja POST-pyynnöt, joita voidaan käyttää datan hakuun rajapinnasta kolmella eri menetelmällä, jotka olivat jQuery AJAX, Axios ja Fetch.

Suurimpana haasteena oli kokonaan uuden kirjaston opettelu, josta kirjoittajalla ei ollut aiempaa kokemusta. Tämä aihe valittiin, koska React-kirjastosta on tullut hyvin yleinen tekniikka. Kirjaston opettelu hyödyttää paljon ja se on tärkeää myös oman osaamisen kannalta etenkin front-end-puolella.

Tämä opinnäytetyö antoi hyvän pohjan työssä käytetyille tekniikoille. Tulevaisuudessa työn tekijän on helpompi hallita näitä työkaluja uusissa projekteissa ja lisätä kokemusta kyseisillä tekniikoilla.

LÄHTEET

Altexsoft. 2017. The Good and the Bad of ReactJS and React Native. [Verkkosivu]. Altexsoft. [Viitattu: 4.4.2018]. Saatavana: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-reactjs-and-react-native/

Bas, R. 2017. React Reconciliation. [Verkkosivu]. Medium. [Viitattu 16.4.2018]. Saatavana: https://medium.com/@ryanbas21/react-reconciliation-7075e3f07437

Bernardes, M. 2015. How to Use Axios as Your HTTP Client. [Verkkosivu]. CodeHeaven. [Viitattu 11.4.2018]. Saatavana: http://codeheaven.io/how-to-use-axios-as-your-http-client/

Borgen, P. Ei päiväystä. Learn React.JS in 8 minutes. [Verkkosivu]. Medium. [Viitattu 29.3.2018]. Saatavana: https://medium.com/learning-new-stuff/learn-react-js-in-7-min-92a1ef023003

Ceddia, D. 2016. A Visual Guide to State in React. [Verkkosivu]. [Viitattu 15.4.2018]. Saatavana: https://daveceddia.com/visual-guide-to-state-in-react/

Fielding, R., Gettys, J., Mogul, J.C., Nielsen, H.F., Masinter, L., Leach, P.J. & Berners-Lee, T. 1999. Hypertext Transfer Protocol – HTTP/1.1. IETF. [Viitattu 10.4.2018]. Saatavana: https://tools.ietf.org/html/rfc2616

Heller, M. 2017. What is Node.js? The JavaScript runtime explained. [Verkkosivu]. InfoWorld. [Viitattu 9.4.2018]. Saatavana: https://www.infoworld.com/article/3210589/node-js/what-is-nodejs-javascript-runtime-explained.html

Kirupa. 2016. Transferring Properties. [Verkkosivu]. Kirupa. [Viitattu 1.4.2018]. Saatavana: https://www.kirupa.com/react/transferring_properties.htm

McGinnis, T. 2016. An Introduction to Life Cycle Events in React. [Verkkosivu]. [Viitattu 6.4.2018]. Saatavana: https://tylermcginnis.com/an-introduction-to-life-cycle-events-in-react-js/

MDN Web Docs. 2018a. HTTP. [Verkkosivu]. MDN Web Docs. [Viitattu 9.4.2018]. Saatavana: https://developer.mozilla.org/en-US/docs/Web/HTTP

MDN Web Docs. 2018b. Using Fetch. [Verkkosivu]. MDN Web Docs. [Viitattu 12.4.2018]. Saatavana: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

Node.js. Ei päiväystä. Node.js. [Verkkosivu]. Node.js Foundation. [Viitattu 7.4.2018]. Saatavana: https://nodejs.org/en/

Node.js. Ei päiväystä. About Node.js. [Verkkosivu]. Node.js Foundation. [Viitattu 8.4.2018]. Saatavana: https://nodejs.org/en/about/

Paul, J. 2017. What is GET and POST method in HTTP and HTTPS Protocol. [Verkkosivu]. Javarevisited. [Viitattu 17.4.2018]. Saatavana: http://javarevisited.blogspot.fi/2012/03/get-post-method-in-http-and-https.html

Peppi. Ei päiväystä. Peppi Open Data. [Verkkosivu]. Peppi-konsortio. [Viitattu 8.4.2018]. Saatavana: https://opendata.seamk.fi/peppiopendatadoc/

Peppi-konsortio. Ei päiväystä. Etusivu. [Verkkosivu]. Peppi-konsortio. [Viitattu 8.4.2018]. Saatavana: http://www.peppi-konsortio.fi/

React. Ei päiväystä. Glossary of React Terms. [Verkkosivu]. Facebook Open Source. [Viitattu 6.4.2018]. Saatavana: https://reactjs.org/docs/glossary.html#single-page-application

React. Ei päiväystä. React Component. [Verkkosivu]. Facebook Open Source. [Viitattu 5.4.2018]. Saatavana: https://reactjs.org/docs/react-component.html

Simons, E. Ei päiväystä. Understanding "state" in React Components. [Verkkosivu]. Thinkster.io. [Viitattu 4.4.2018]. Saatavana: https://thinkster.io/tutorials/understanding-react-state

Six revisions. 2010. The Power of jQuery with Ajax. [Verkkosivu]. WebpageFX. [Viitattu: 11.4.2018]. Saatavana: https://www.webpagefx.com/blog/web-design/the-power-of-jquery-with-ajax/

Sotelo, C. 2014. Evolution of the Single Page Application. [Verkkosivu]. Paislee.io. [Viitattu 7.4.2018]. Saatavana: http://paislee.io/evolution-of-the-single-page-application/

Survivejs. Ei päiväystä. Introduction to React. [Verkkosivu]. Survivejs. [Viitattu 4.4.2018]. Saatavana: https://survivejs.com/react/getting-started/introduction-to-react/

Survivejs. Ei päiväystä. Understanding React Components. [Verkkosivu]. Survivejs. [Viitattu 6.4.2018]. Saatavana: https://survivejs.com/react/getting-started/understanding-react-components

TekTutorialsHub. Ei päiväystä. HTTP GET and POST methods in HTTP protocol. [Verkkosivu]. TekTutorialsHub. [Viitattu 10.4.2018]. Saatavana: https://www.tektutorialshub.com/http-get-and-post-methods/

Wheeler, K. 2014. Learning React.js: Getting Started and Concepts. [Verkkosivu]. Scotch.io. [Viitattu 29.3.2018]. Saatavana: https://scotch.io/tutorials/learning-react-getting-started-and-concepts

Wynne-McHardy, R. Ei päiväystä. Understanding State and Props in React. [Verkkosivu]. Hackernoon. [Viitattu 1.4.2018]. Saatavana: https://hackernoon.com/understanding-state-and-props-in-react-94bc09232b9c