Joni Honkanen

ReactJS

Opinnäytetyö
Syksy 2017
SeAMK Tekniikka
Tietotekniikan tutkinto-ohjelma

SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Tekniikan yksikkö

Tutkinto-ohjelma: Tietotekniikka

Suuntautumisvaihtoehto: Sulautetut järjestelmät

Tekijä: Joni Honkanen

Työn nimi: ReactJS

Ohjaaja: Petteri Mäkelä

Vuosi: 2017 Sivumäärä: 43 Liitteiden lukumäärä: 0

JavaScript on saanut paljon suosiota viime vuosien aikana ja suosion mukana ovat myös tulleet useat JavaScriptiä käyttävät sovelluskehykset sekä kirjastot. Näistä voi olla vaikea valita itselleen sopivinta. Tässä opinnäytetyössä tutustutaan yhteen lupaavimmista JavaScript-kirjastoista, ReactJS-kirjastoon.

Opinnäytetyössä tutustuttiin React JavaScript -kirjastoon, sen rakenteeseen, ominaisuuksiin sekä käyttömahdollisuuksiin. Lisäksi perehdyttiin Redux JavaScript -kirjastoon ja siihen, kuinka se soveltuu käytettäväksi Reactin kanssa.

Työstä tuli kattava tietopaketti Reactista kiinnostuneille.

Avainsanat: React, Redux, JavaScript

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Technology

Degree programme: Information Technology

Specialisation: Embedded Systems

Author: Joni Honkanen

Title of thesis: ReactJs

Supervisor: Petteri Mäkelä

Year: 2017 Number of pages: 43

JavaScript is getting more popular every year and that is why there are many JavaScript frameworks and libraries available for different kind of development. It can be a hard task to choose the best framework to work with. This thesis focused on one of the most popular JavaScript-libraries from the past few years, which is called ReactJs.

The theoretical part of React includes information on the structure of React components, the lifecycle of the components and dataflow between the components. The study also introduces a part of a popular library that can be used with React to store application data to a single store. This library is called Redux. Redux has quite complicated architecture and one goal of this thesis was to explain how different parts of that architecture work. This thesis benefits people who are interested in React.js.

Keywords: React, Redux, JavaScript

SISÄLTÖ

Kuvaluettelo

Käytetyt termit ja lyhenteet

Deklaratiivinen	Ohjelmointityyli, jossa kerrotaan mihin tulokseen pyritään. Vastakohta imperatiiviselle ohjelmoinnille.
Elinkaari metodit	Komponentin eri vaiheissa suoritettavat toiminnot.
Front End	Käyttäjälle näkyvä osuus.
Imperatiivinen	Ohjelmointityyli, jossa ongelman ratkaisu selitetään vaihe kerralla.
JSX	HTML-kieltä muistuttava tapa kirjoittaa JavaScriptiä. Käytetään React-komponenttien luonnissa.
Komponentti	React-sovelluksen rakennuspalikka, josta sovellus koostuu.
MVC-arkkitehtuuri	Etenkin graafisissa käyttöliittymissä käytetty ohjelmistoarkkitehtuurityyli, jossa ohjelma jaetaan kolmeen osaan: malliin, näkymään sekä käsittelijään.
Properties	“Props” on data, joka tulee isäntäkomponentilta lapsikomponentille.
Renderointi	Kuvan muodostaminen.
Sovelluksen tila	Sovelluksen yhteinen data, joihin jokaisella komponentilla on pääsy.
SPA	Yhden sivun sovellus, käyttäjä pysyy samalla sivulla, vaikka sivun sisältö vaihtuisi.
Store	Säilö sovelluksen yhteiselle datalla.
Tila	Komponentin sisäinen yhteinen data.
Virtuaalinen DOM	Kopio DOM-mallista, tarkoituksena on verrata muutoksia alkuperäiseen DOM-malliin.

1 JOHDANTO

1.1 Työn tausta

Web-ohjelmointi kehittyy jatkuvasti, tämän vuoksi oikean sovelluskehyksen valitseminen projektiin voi olla vaikeaa. Suosituimpia JavaScript-sovelluskehyksiä ovat Angular.js, React.js, Vue.js, Ember.js sekä Meteor.js. Uuden sovelluskehyksen opettelu on aikaa vievää, vaikka ne pohjautuvatkin samaan kieleen, JavaScriptiin.

Sovelluskehykset nopeuttavat ohjelmistojen kehitystyötä tuomalla käytettäväksi valmiita toiminnollisuuksia, tämä vähentää kirjoitettavan koodin määrä. Opinnäytetyö on ajankohtainen kaikille, jotka harkitsevat ReactJS-kirjaston käyttöä tuleviin projekteihinsa tai haluavat tietää enemmän Reactin arkkitehtuurista sekä monipuolisuudesta.

1.2 Työn tavoite

Työn tavoitteena on esitellä React- sekä Redux JavaScript-kirjasto, näiden arkkitehtuuri sekä toiminta. Opinnäytetyössä käydään läpi yksityiskohtaisesti Reactin sekä Reduxin tärkeimmät termit, datan kulku sekä rakenne.

Lukijalle pitäisi selvitä, mikä React on, mihin sitä käytetään, mistä osioista React-sovellus koostuu, kuinka React toimii ja minkä vuoksi Redux sopii hyvin Reactin kanssa yhteen. Lisäksi lukijalle pitäisi muodostua kuva, kuinka monipuolisesti Reactia voidaan hyödyntää eri alustoille ohjelmoidessa.

1.3 Työn rakenne

Luvussa 2 esitellään eroavaisuus perinteisen sivuston sekä yhden sivun web-sovelluksen välillä. Lisäksi käydään läpi sovelluskehysten hyödyt web-sivujen suunnittelussa. Luvussa 3 kerrotaan Reactista, Reactin komponenteista, datan kulusta sekä React-sovelluskehityksestä eri alustoille. Luvussa 4 paneudutaan JSX:n käyttöön. Luku 5 käsittelee Redux-kirjaston ominaisuuksia. Luvussa 6 katsotaan, kuinka

React ja Redux toimivat yhdessä. Lopuksi on yhteenveto, jossa käydään opinnäytetyön tulokset ja hyödyt läpi.

2 YKSISIVUINEN WEB-SOVELLUS

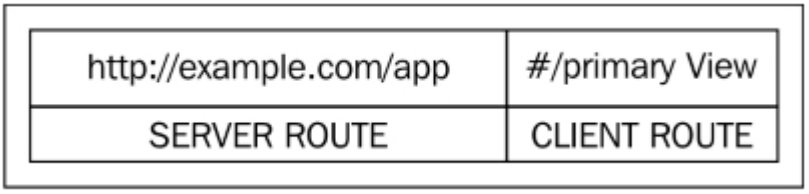
Verkkosivusto koostuu monista eri tiedostoista, joista selain muodostaa käyttäjälle näkyvän sivun. Verkkosivu on yleensä kirjoitettu HTML-kielellä, jota kaikki selaimet osaavat lukea. Sivu voi myös sisältää CSS-tyylimäärittelyjä sekä JavaScriptiä. CSS vastaa sivun tyylistä tuomalla esimerkiksi erilaisia fontteja ja värejä käytettäväksi. JavaScriptin tehtävä on taas tuoda sivulle dynaamista toiminnallisuutta.

Verkkosivun tiedostot sijaitsevat palvelimella. Käyttäjän pyrkiessä sivustolle selain pyytää palvelimelta HTML-dokumentin. Tämän jälkeen HTML-dokumentti muutetaan käyttäjälle näkyvään muotoon. Perinteisillä tekniikoilla tehtyjen sivujen navigointi toiselle sivulle tapahtuu lataamalla uusi HTML-dokumentti. Uuden HTML-tiedoston lataaminen palvelimelta tuo uuden sisällön sivulle, tämä toimenpide vie aina jonkin verran aikaa.

2.1 Single-page applications (SPA)

Single-page application on sovellus, jossa pystytään samalla www-sivulla, vaikka sivun sisältö vaihtuisi. Näitä sovelluksia käytetään selaimella, kuten muitakin vanhemman tekniikan verkkosivuja. Suurin näkyvä eroavaisuus perinteisen sekä SPA-sivun välillä on vähentynyt sivujen latausmäärä. (Code School [Viitattu 28.4.2017].)

Yksisivuinen sovellus tarjoaa samat ominaisuudet ja toiminnot, joita käytetään työpöytä- sekä natiivisovelluksissa. SPA-sovelluksissa vain yksi dokumentti ladataan selaimeen. Päädokumentin jälkeen muita resursseja, kuten skriptejä, tyylitiedostoja, dataa sekä muita ominaisuuksia, ladataan asynkronisesti näkyviin ilman että päädokumenttia muutetaan. Toisin sanoen sovelluksen elinkaaren ajan URL-osoitteen sisältö hash-merkin (#) edessä ei muutu, sillä selain ei pyydä palvelimelta muita sivuja näytettäväksi. (Vice & Horton [Viitattu 25.5.2017].) Kuviossa 1 esitellään SPA-sivuston url-osoitteen rakenne.



Kuva 1. SPA URL -esimerkki.
(Vice & Horton [Viitattu 25.5.2017]).

Sivun navigointi tapahtuu pääosin selaimen reitittimen kautta, joka reagoi URL-osoitteen muutoksiin hash-merkin jälkeen. Kun sovellus muuttaa URL-osoitteen sisältöä hash-merkin jälkeen, näkymäjärjestelmä muokkaa DOM-mallin eri näkymään. Kaikki muutokset ennen hash-merkkiä kuuluvat palvelimelle. Niiden muuttuessa ladataan myös uudet dokumentit näytettäväksi. (Vice & Horton [Viitattu 25.5.2017].)

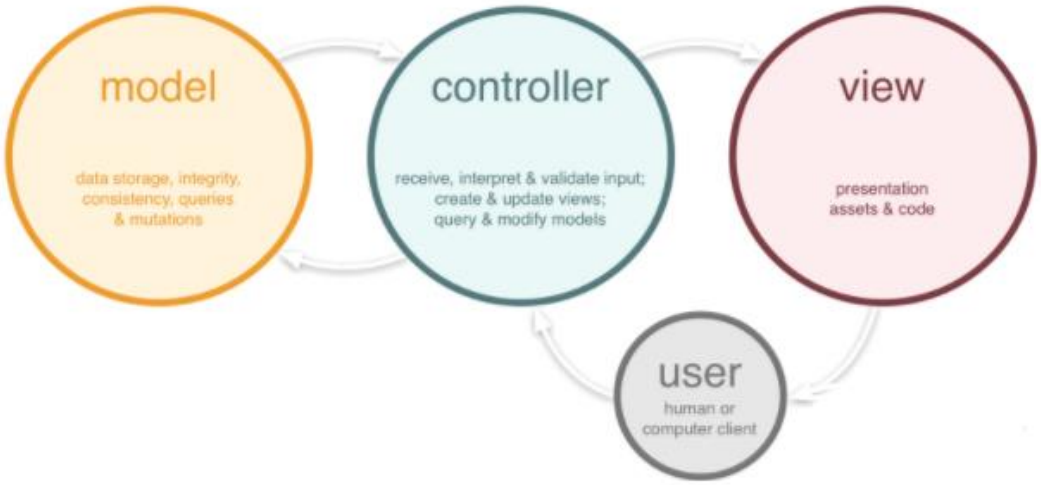
2.2 Sovelluskehykset

Sovelluskehyksellä tarkoitetaan sovellusalustaa, jonka ominaisuuksia hyödyntämällä sovelluksen kehitystyö on nopeampaa ja helpompaa. Sovelluskehykset tuovat valmiita toimintoja käytettäväksi, mikä vähentää koodauksen tarvetta. Lisäksi sovelluskehyksillä on omat suunnittelumallit, joita noudattaessa koodi pysyy yhtenäisempänä sekä helposti luettavana.

Mitä enemmän vuorovaikutteisuutta tapahtuu käyttäjän puolella, sitä enemmän tarvitaan JavaScript-koodia toteuttamaan kyseiset toiminnallisuudet. Mitä enemmän koodia kirjoitetaan, sitä tärkeämpää on koodin siisteys sekä ymmärrettävyys. Tähän tarkoitukseen sopii hyvin JavaScript-sovelluskehykset. Jokaisella sovelluskehyksellä on oma tapansa käsitellä eri ongelmia. Suosituimpia sovelluskehyksiä ovat Angular, React, Ember, Aurelia, Vue.js, Cycle.js sekä Backbone (Code School [Viitattu 28.4.2017].)

Moni sovelluskehys käyttää MVC-arkkitehtuuria web-ohjelmistojen suunnittelussa. MVC jakaa ohjelmiston logiikan eri osioihin. Näin ohjelman suunnittelu ja koodaaminen ovat hallitumpaa. MVC koostuu kolmesta osasta, jotka näkyvät kuvassa 2: malli, näkymä sekä käsittelijä. Malli (Model) kuvastaa ohjelmiston dataa. Näkymä

(View) on kaikki, mitä käyttäjä näkee sovellusta käyttäessään. Käsittelijä (Controller) mahdollistaa datan tuonnin näkymälle, sekä käyttäjien toiminnot. (MVC Architecture [Viitattu 4.9.2017].)



Kuva 2. MVC-osien vuorovaikutus.
(MVC [Viitattu 4.9.2017].)

Sopivan sovelluskehyksen löytäminen voi olla vaikeaa, sillä jokaisella sovelluskehyksellä on sekä hyötyjä että haittoja. Yhteistä sovelluskehyksille on kuitenkin se, että ne käyttävät JavaScriptiä. (Code School [Viitattu 28.4.2017].)

3 REACT.JS

3.1 Mikä on React.js

React, jota kutsutaan myös nimellä React.js tai ReactJS, on avoimen lähdekoodin JavaScript-kirjasto, jonka tarkoitus on muuttaa ohjelmassa käytettävä data käyttäjälle näkyvään muotoon. Reactia pidetään MVC-arkkitehtuurin näkymänä (view). Malli (model) ja käsittelijä (controller) vaatii toisen sovelluskehyksen käyttöä. Reactin kanssa tähän suositellaan Reduxia. (Singh & Bhatt 2016, 7-8.)

React on front-end-kirjasto, jonka kehitti Facebook vuonna 2011. React sallii käyttäjiensä luoda uudelleen käytettäviä HTML-tyylisiä komponentteja, joista sovellus koostuu. React käyttää virtuaalista DOM-mallia. Kun sovelluksen data muuttuu, Reactin virtuaalinen DOM vertaa muutoksia nykyiseen ja päivittää pelkästään muuttuneet komponentit. Tällöin ei tarvitse päivittää koko käyttöliittymää, vaan pelkästään tarvittavat komponentit. Tämä tekee Reactista hyvin nopean. (Codemurai [Viitattu 1.5.2017].)

Reactia käyttävät muun muassa suuret yritykset, kuten Facebook, Instagram, Netflix, Alibaba, Yahoo, E-bay, Khan-Academy, AirBnB, Sony ja Atlassian (Sonpatki & AM [Viitattu 8.7.2017]).

React käyttää deklaratiivista ohjelmointia eli selittävän ohjelmoinnin ajatusmallia. Se on yksi niistä syistä, mikä tekee Reactin käytöstä tehokasta. Reactin käyttämisen kannalta on tärkeää ymmärtää, mitä deklaratiivinen ohjelmointi tarkoittaa ja mikä on sen eroavaisuus imperatiiviseen ohjelmointiin. Imperatiivinen ohjelmointi kuvastaa kuinka asiat toimii vaihe vaiheelta, deklaratiivinen ohjelmointi kertoo, mitä halutaan saavuttaa. (Bertoli 2017, 8.)

Bertolin (2017, 8) mukaan tosielämän esimerkki imperatiivisesta ohjelmoinnista olisi mennä baariin ja käskeä baarimikkoa seuraavasti:

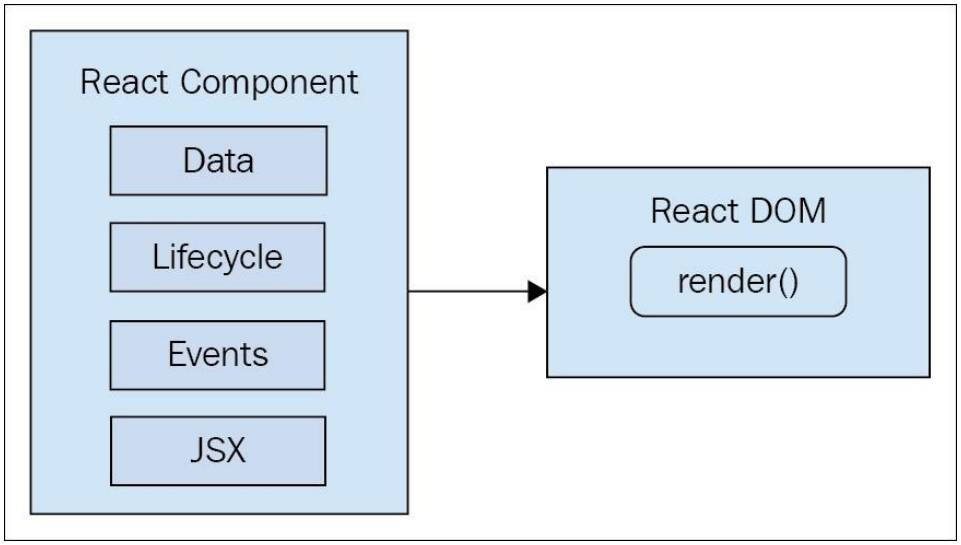
- Ota lasi hyllyltä
- Pistä lasi hanan alle
- Vedä hanasta niin kauan, että lasi täyttyy

- Anna minulle lasi.

Betroli (2017, 8) kertoo että, deklaratiivisena vastaava olisi sanoa baarimikolle: ”Kalja kiitos”. Deklaratiivinen lähestymistapa olettaa, että baarimikko tietää kuinka kalja tarjoillaan.

React on jaettu kahteen suurempaan ohjelmointirajapintaan, ReactDOM-malliin sekä React-komponenttiin. ReactDOM on vastuussa React-komponentin renderöimisestä selaimeen kuvan 3 mukaisesti. Boduchin ([Viitattu 26.7.2017]) mukaan React-komponentti voi sisältää seuraavia asioita:

1. Data: Arvoja jotka tulevat mm. muilta komponenteilta sekä tietokannoista.
2. Elinkaari: Komponentin elinkaaren aikana olevia toimintoja.
3. Tapahtumat: Kuvastaa käyttäjän toimintoja
4. JSX: Kuvaa komponentin rakennetta HTML-tyylisesti.

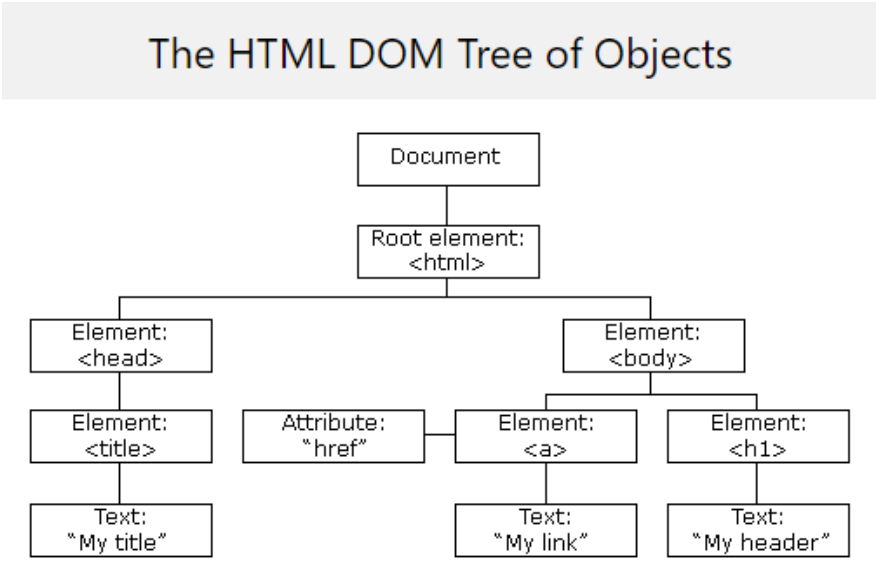


Kuva 3. React-komponentin lähettäminen ReactDOM-malliin.
(Boduch [Viitattu 26.7.2017]).

3.2 Virtuaalinen DOM

Document Object Model (DOM) on ohjelmointirajapinta (Application programming interface, API) HTML-dokumenteille. Kun selainta pyydetään renderöimään HTML-

tiedosto, joudutaan HTML-tiedosto jäsentelemään ensin DOM-malliksi ja vasta sen jälkeen se näkyy selaimessa. (Robbestad [Viitattu 9.7.2017].) Kuva 4 esittää DOM-mallia sekä näyttää sen puumaisen rakenteen, joka koostuu HTML-elementeistä.



Kuva 4. Esimerkki DOM-mallista.
(DOM [Viitattu 5.9.2017]).

DOM on tunnetusti hidas. Mikäli DOM-mallin muutokset tapahtuvat vain harvoin, hitaus ei ole haitaksi. Suurissa ja monimutkaisissa ohjelmissa DOM-mallin muokkaamiseen ja uudelleen rakentamiseen saattaa mennä huomattavasti aikaa. Reactin ratkaisu tähän on pitää jäljitelmä DOM-mallista muistissa ja tehdä muutokset siihen. Tätä kutsutaan virtuaaliseksi DOM-malliksi. Kun tarvittavat muutokset on tehty muistiin, React pystyy tekemään muutokset todelliseen DOM-malliin mahdollisimman yksinkertaisesti. Tämä lähestymistapa mahdollistaa nopeat peräkkäiset muutokset ja parantaa ohjelman tehokkuutta. (Masiello & Friedmann 2017 [Viitattu 2.9.2017].)

3.3 Komponentit

Komponentti voi olla JavaScript-funktio tai -objekti, joka muutetaan HTML-koodiksi. React-sovellus koostuu monista erilaisista komponenteista. Komponentti voi myös

sisältää lapsikomponentteja. React-sovellus on pääosin yksi iso komponentti, joka sisältää useita lapsikomponentteja. (Grider [15.4.2017].)

Isäntäkomponentti voi lähettää arvojaan lapselleen. Tätä komponenttien välistä dataa kutsutaan ”properties”-nimellä (props). Kun lapsikomponentti vastaanottaa datan isännältä, dataa pystyy muokkaamaan uudelleen ja lähettämään eteenpäin muille lapsikomponenteille. (Bertoli 2017, 107.)

Komponentti voi olla joko funktionaalinen (functional component) tai luokallinen (class component). Funktionaalisia komponentteja käytetään staattisen datan esittämiseen, kuten kuvan, joka pysyy aina samanlaisena. Luokallisella komponentilla on enemmän toimintoja kuin funktionaalisella. Näitä ovat muun muassa muuttuvan datan esittäminen sekä komponentin elinkaaren tunnistaminen (Lifecycle methods). (Grider [Viitattu 20.4.2017])

Stephen Grider (2017) listaa suurimmat eroavaisuudet kyseisille komponenteille seuraavasti:

Luokallinen komponentti

- Käyttää muuttuvaa dataa
- Käsittelee kaiken muuttuvan datan (datan nouto, käyttäjä toiminnot)
- Tietää milloin tulee renderöitäväksi (elinkaarimetodit)
- Suuritöinen.

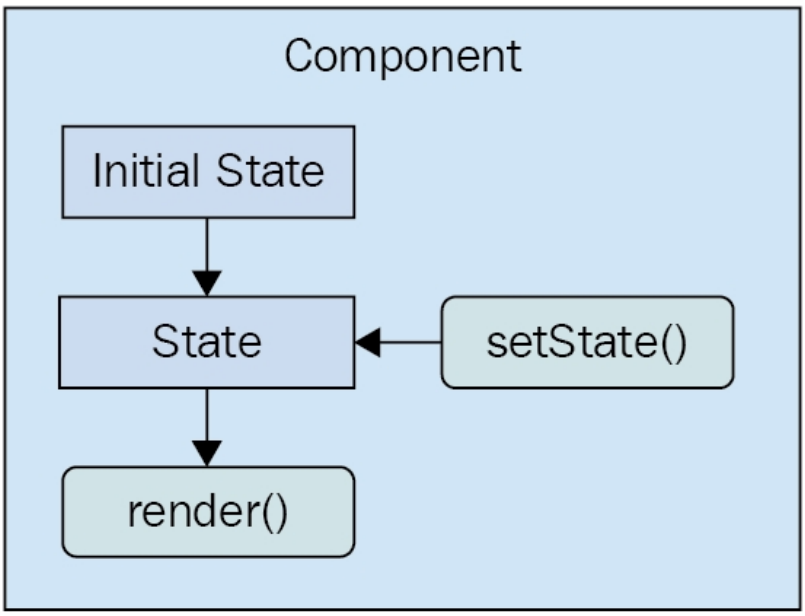
Funktionaalinen komponentti

- Käytetään esittämään staattista dataa
- Ei pysty noutamaan dataa
- Helppo kirjoittaa.

3.3.1 Tutustuminen komponentin tilaan

Tila (state) on yksi Reactin vaikeimmista, mutta tärkeimmistä aiheista. Tila on JavaScript-objekti, jota käytetään tallentamaan ja vastaamaan käyttäjien tapahtumiin. Jokaisella class-pohjaisella komponentilla voi olla monia eri tiloja. Tilan muuttuessa komponentti sekä komponentin lapset renderöidään uudelleen. (Grider

[15.4.2017].) Kuvasta 5 näkyy, kuinka tilan arvojen päivitys aiheuttaa komponentin uudelleen renderöimisen.



Kuva 5. Komponentin tila.
(Boduch [Viitattu 26.7.2017]).

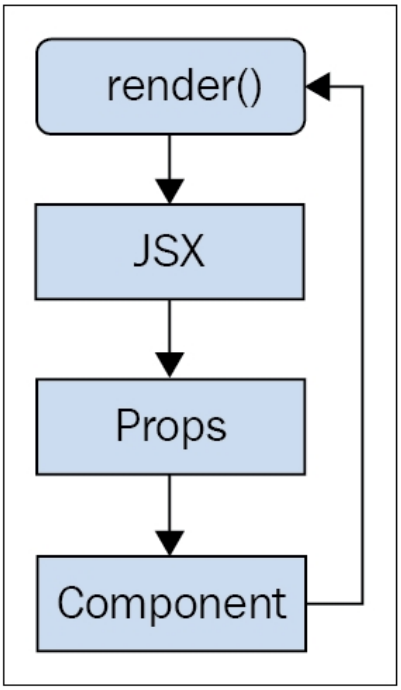
Tilallisen komponentin voi määritellä kahdella eri tapaa, joko käyttämällä React.createClass- tai extends React.Component-komentoja. Molemmilla tavoilla pääsee samaan lopputulokseen, vaikka niiden rakenteet hieman eroavatkin toisistaan. Suurimmat eroavaisuudet ovat tilojen ja ominaisuuksien luomisessa. (Bertoli 2017, 51-53.)

Komponentin elinkaaren aikana komponentin tilaa pystyy muokkaamaan useita kertoja. Joka kerta tilan muuttuessa React renderöi uudelleen komponentin näkyviin uusin arvoin. Kun setState-metodia kutsutaan tuomaan uusi tila, tila liitetään yhteen nykyisen tilan kanssa. (Bertoli 2017, 61-62.)

3.3.2 Properties

Ominaisuuksia (properties, props) käytetään datan lähettämiseen React-komponenteille. Nämä arvot annetaan vain kerran komponentin renderöinnin jälkeen. Suurin eroavaisuus komponentin tilan sekä propertien välillä on se, että tilan arvo

voi muuttua renderoimisen jälkeen, kun propertien arvot pysyvät samana. (Boduch [Viitattu 26.7.2017]). Kuva 6 kertoo renderöinnin jälkeisen järjestyksen.



Kuva 6. Propertien renderointi.
(Boduch [Viitattu 26.7.2017]).

3.3.3 Elinkaarimetodit

React-komponentin elinkaaren aikana on kolme eri vaihetta, jolloin voi suorittaa eri elinkaaritoimintoja (kuva 7). Komponentti voi olla kiinnittymässä (mounting), kiinnittynyt (mounted) tai irtaantumassa (unmounting). Jokaisen vaiheen aikana on eri toimenpiteitä, joita voi suorittaa. (Vepsäläinen 2017, 37-39.)

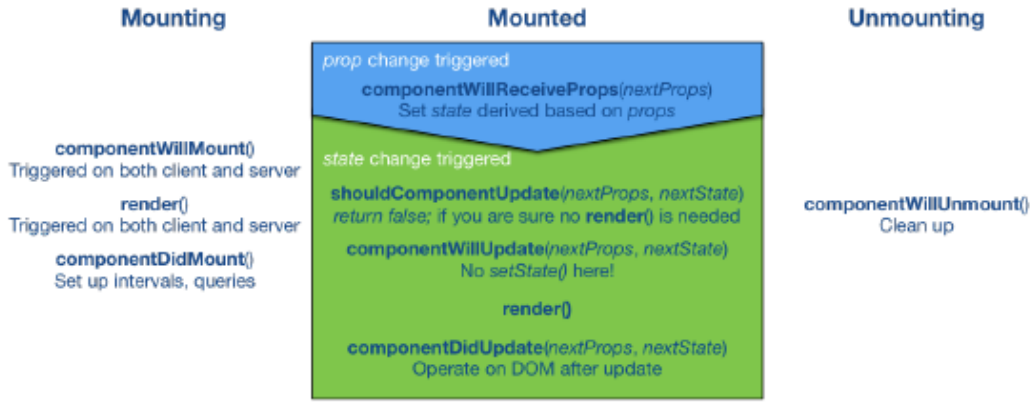
Vepsäläinen (2017, 38-39) kertoo komponentin kiinnittymisvaiheen mahdollisista toiminnoista seuraavasti:

- componentWillMount() laukaistaan kerran ennen komponentin renderöintiä.
- render() päivittää komponentin tilan. Se suoritetaan kerran huolimatta tilan muutoksista.
- componentDidMount() suoritetaan ensimmäisen renderöinnin jälkeen.

Kun komponentti on kiinnitetty, on mahdollista päivittää komponenttia seuraavan järjestyksen mukaisesti: componentWillReceiveProps(), ShouldComponentUpdate(), componentWillUpdate(), render() sekä componentDidUpdate(). (Vepsäläinen 2017, 37-39.)

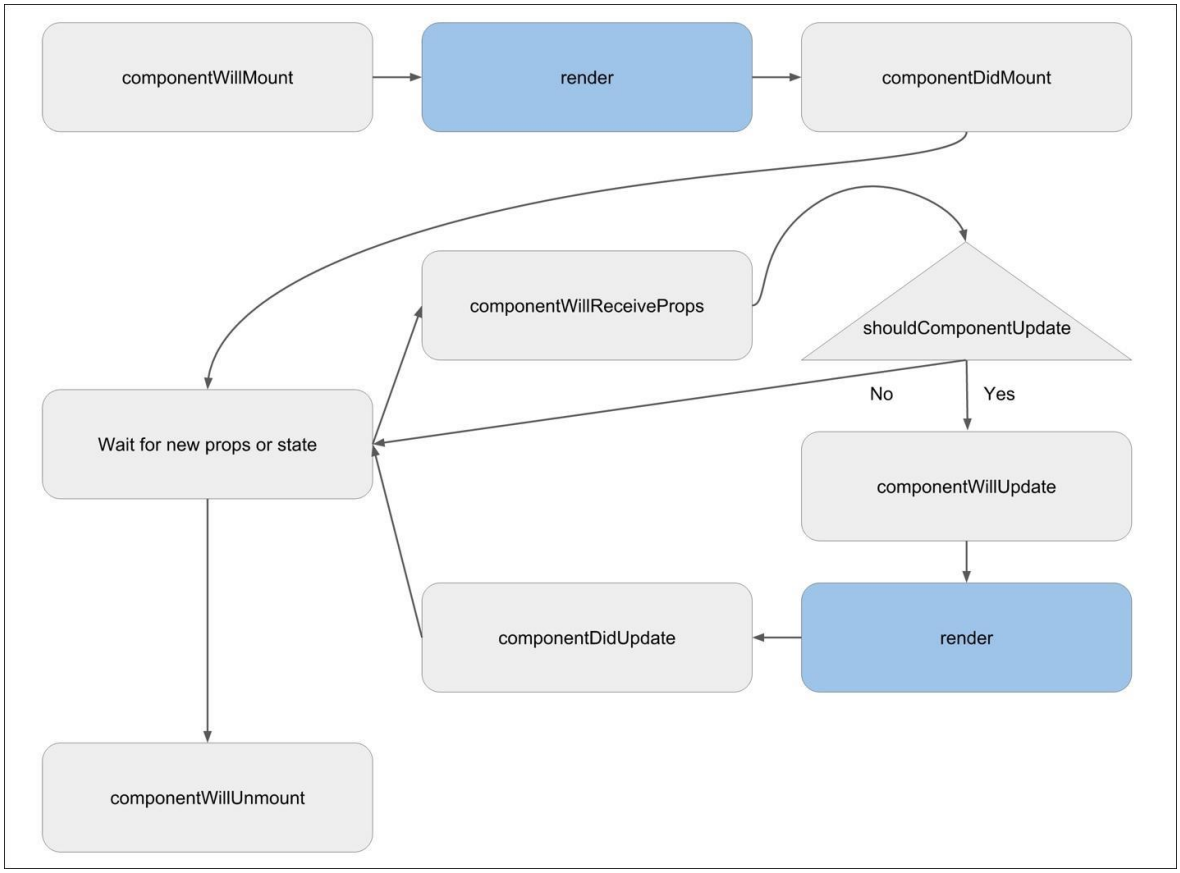
ComponentWillReceiveProps() suoritetaan, kun komponentti saa uudet arvot (props). Näillä arvoilla voi esimerkiksi päivittää komponentin tilan. ShouldComponentUpdate() mahdollistaa renderöinnin optimoinnin. Tällöin tarkistetaan, onko komponenttia välttämättä tarve päivittää. ComponentWillUpdate() suoritetaan ennen komponentin päivittämistä, sekä ennen uudelleen renderöintiä. Render() päivittää komponentin uusilla arvoilla. ComponentDidUpdate() suoritetaan uudelleen renderöinnin jälkeen. (Vepsäläinen 2017, 37-39.)

Kun komponentti irroitetaan DOM-mallista, voi suorittaa componentWillUnmout-toiminnon. Tällöin on ideaalinen hetki nollata komponentin arvot. (Vepsäläinen 2017, 37-39.)



Kuva 7. Komponentin elinkaari.
(Vepsäläinen 2017, 38).

Kuvassa 8 näkyy elinkaaritoimintojen logiikka.



Kuva 8. Komponentin elinkaari toiminnot.
(Masiello & Friedmann 2017 [Viitattu 2.9.2017]).

Tilattomilla funktionaalisilla komponenteille ei ole kyseisiä elinkaaritoimintoja, sillä funktionaalisia komponentteja käytetään vain staattisen datan esittämiseen (Bertoli 2017, 59).

3.4 React Native

Puhelinsovellukset voivat olla joko natiiveja tai hybridisovelluksia. Natiiveilla sovelluksilla tarkoitetaan yksilöityjä sovelluksia tietyille alustoille (Android, iOS, Windows). Yleisin kieli Android-ohjelmien tekoon on Java. iOS-ohjelmien kieli on joko Objective-C tai Swift.

Hybridisovellukset käyttävät web-tekniikoita (HTML, CSS, JavaScript) ja tarvitsevat toimiakseen WebView-alustan, joka muuntaa näkymän puhelimelle sopivaksi. Suosituimpia ohjelmia hybridisovellusten tekoon ovat Cordova sekä Xamarin.

Natiivisovellukset tuntuvat usein nopeilta ja responsiivisilta verrattuna hybridisovelluksiin. Hybridisovelluksissa animaatiot, skrollaus sekä näppäimistön käyttö voi olla myös tökkivää ja hidasta. Useimmiten natiivisovellus antaa paremman käyttäjäkokemuksen kuin hybridisovellus. Hybridisovelluksen paremmuus onkin sen kehittämisessä, sillä web-tekniikoita käyttäen saa toimivan sovelluksen nopeasti kehitettyä sekä Android että iOS-laitteille. (Novick [Viitattu 1.9.2017].)

React Native mahdollistaa natiivien sovellusten kehittämisen Android- ja iOS-laitteille, joten yhden ohjelmointikielen osaaminen riittää molempien tekoon (Masiello & Friedmann [Viitattu 2.9.2017]).

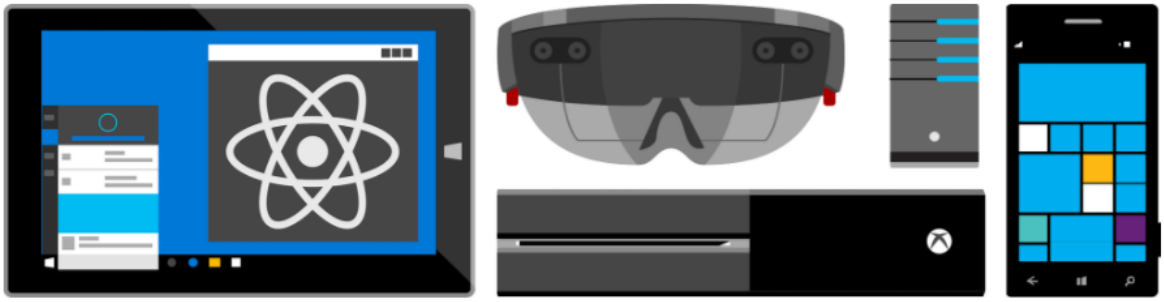
React Native käyttää samaa deklaratiivista ohjelmointitapaa käyttöliittymien tekoon kuin React web-ohjelmoinnissa. React Native mahdollistaa nopean sovellusten kehittämisen, sillä muutosten jälkeen tarvitsee vain päivittää sovellus. Muilla tekniikoilla joutuu sovelluksen aina rakentamaan uudelleen, jolloin kuluu huomattavasti enemmän aikaa React Nativeen verrattuna. Suurempana eroavaisuutena Reactin sekä React Nativen välinä on se ettei React Nativessa ole virtuaalista DOM-mallia. (Masiello & Friedmann [Viitattu 2.9.2017].)

React Native on sovelluskehys, kun taas React on JavaScript-kirjasto. Kun Reactilla aloittaa projektin, sitä joutuu mahdollisesti täydentämään eri JavaScript-kirjastoilla. React Nativessa suurin osa asioista projektin tekoon on jo valmiina. Tämän vuoksi siihen ei välttämättä tarvitse lisätä uusia kirjastoja. React Nativen sovelluksia voi testata Androi-emulaattorilla tai puhelimella. (React Native 2 [Viitattu 15.5.2017.)

3.5 React-työpöytäsovellukset

Reactilla pystyy tekemään selain-ja puhelinsovellusten lisäksi myös työpöytäsovelluksia Windowsille ja MacOS-käyttöjärjestälmälle. Suosituin ohjelma tähän tarkoitukseen on Electron. Electron on sovelluskehys natiivien työpöytäsovellusten tekoon web-teknologioilla. (Electron [Viitattu 14.9.2017]).

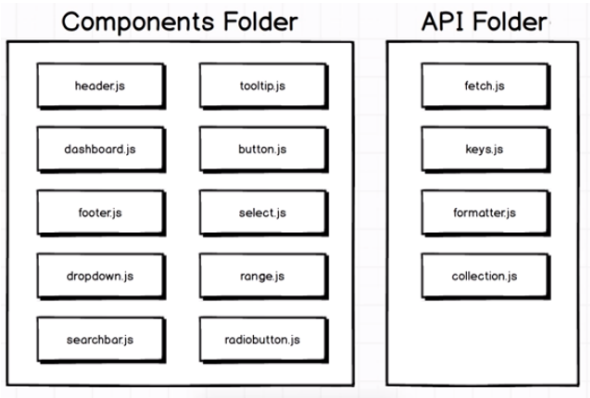
Microsoft mahdollistaa UWP-sovellusten (Universal Windows Platform) kehittämisen React Nativella. Tähän kuuluvat laitteet jotka käyttävät Windows-käyttöjärjestelmää (Windows 10, Windows 10 Mobile). Näitä ovat esimerkiksi Microsoftin tabletit, puhelimet, tietokoneet sekä Xbox.



Kuva 9. Microsoftin laitteet.
(Microsoft [Viitattu 14.9.2017]).

3.6 Webpack ja Node.js

Perinteisiin web-sivuihin verrattuna yhden sivun web-sovellukset sisältävät huomattavasti enemmän JavaScriptiä. Lisäksi suuret JavaScript-tiedostot ovat muuttunut useiksi pienemmiksi JavaScript-tiedostoiksi. Näitä tiedostoja kutsutaan moduuleiksi. JavaScript-tiedostojen pilkkominen pienemmiksi helpottaa ja nopeuttaa oikean koodin löytämisen verrattuna yhteen suureen tiedostoon. (Grider [Viitattu 2.6.2017].) Kuvassa 10 on esimerkki järkevästä tavasta pilkkoa suuri tiedosto pienempiin moduuleihin ja nimetä ne käyttötarkoituksen mukaisesti.



Kuva 10. Esimerkki kansioista ja niiden sisällä olevista moduuleista.
(Grider [Viitattu 2.6.2017]).

Useiden moduulien ongelmaksi tulee niiden lataaminen selaimelle. Monen eri tiedoston lataaminen on hitaampaa kuin yhden ison tiedoston. Lisäksi ongelmaksi tulee eri moduulien suhteet toisiin moduuleihin ja niiden latausjärjestys. Mitä vähemmän tiedostoja tarvitsee ladata kerralla, sen paremmin sivusto toimii. (Grider [Viitattu 2.6.2017].)

3.6.1 Webpack

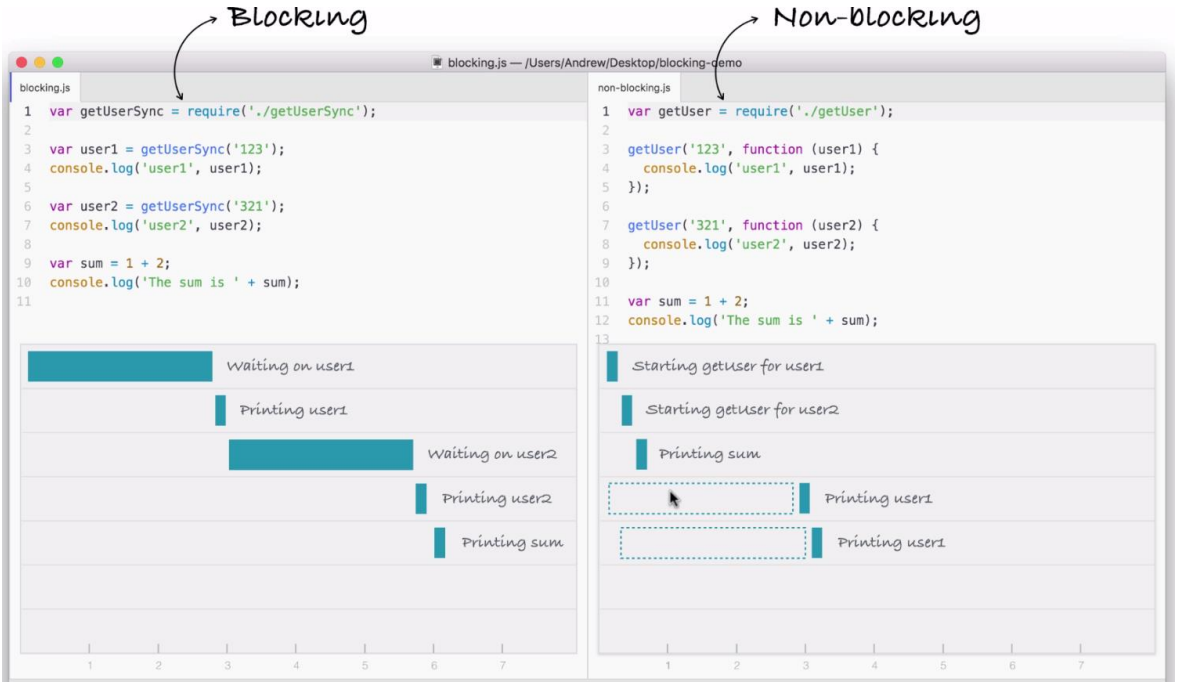
Webpackin tarkoitus on ottaa kaikki pienet JavaScript-moduulit ja yhdistää ne yhdeksi isoksi JavaScript-tiedostoksi. Lisäksi Webpack pitää huolen, että jokainen moduuli ladataan oikeassa järjestyksessä. Tämä on webpackin päätarkoitus ja se ratkaisee aiemmin mainitut ongelmat. (Grider [Viitattu 2.6.2017].)

3.6.2 Node.js

Node mahdollistaa JavaScriptin käytön palvelimissa (backend) tuoden samanlaisia ominaisuuksia kuten Javalla, Pythonilla tai PHP-kielellä. Näitä ominaisuuksia on esimerkiksi tiedostojen luonti sekä poistaminen, tietokantaan yhteyden ottaminen sekä web-palvelimen luominen. Node käyttää Chrome’s V8 JavaScript -moottoria, joka muuntaa JavaScriptin erittäin tehokkaasti tietokoneelle ymmärrettävään muotoon. (Mead & Percival [Viitattu 2.6.2017].)

Node.js käyttää tapahtumaperäistä, tukkiutumatonta I/O-mallia, joka tekee siitä kevyen sekä tehokkaan. Tukkiutumaton I/O mahdollistaa usean eri käyttäjän pyynnöt palvelimelle ilman että pyynnöt hidastuisivat merkittävästi muista pyynnöistä. Tukkiutuva I/O pystyy tekemään vain yhden pyynnön kerralla, jonka vuoksi se on käytössä hitaampi (kuva 11). (Mead & Percival [Viitattu 2.6.2017].)

Tässä yhteydessä I/O (input/output) tarkoittaa palvelimen suorittamia toimintoja. Näitä toimintoja voi olla esimerkiksi: tietokantaan kirjoittaminen, tietokannasta lukeminen, tiedostojen päivittäminen tiedostojärjestelmään tai http-pyyntö toiseen webserveriin. (Mead & Percival [Viitattu 2.6.2017].)



Kuva 11. Tukkiutuvan ja tukkiutumattoman I/O:n ajan käyttö.
(Mead & Percival [Viitattu 2.6.2017]).

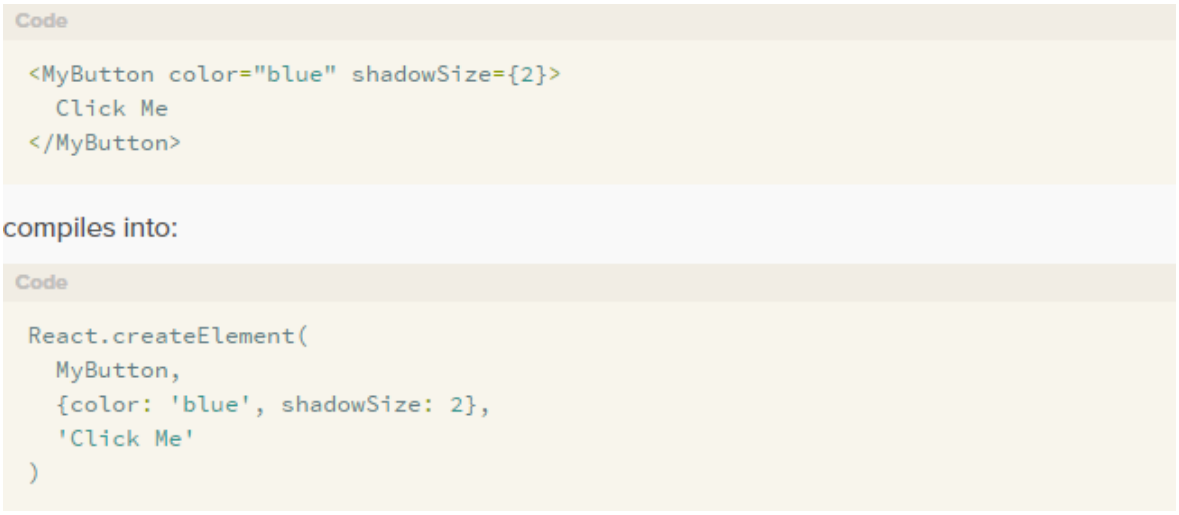
Noden tarjoama package manager / npm on yksi suurimmista palveluista JavaScript-moduulien lataamiselle (Node.js [Viitattu 2.6.2017]). Npm-yhteisö kehittää jatkuvasti uusia JavaScript-kirjastoja, joita muut voivat hyödyntää projekteissaan. Valmiiden kirjastojen käyttö nopeuttaa huomattavasti kehitystyötä, sillä se vähentää kirjoitetun koodin määrää. (Mead & Percival [Viitattu 2.6.2017].)

4 JSX

4.1 Mikä on JSX

JSX on JavaScript-syntaksi, joka muistuttaa XML-ohjelmointia. Sitä käytetään komponenttien luontiin Reactissa. JSX muistuttaa pitkälti HTML-kieltä pienillä eroavaisuuksilla. Vaikka JSX muistuttaa HTML-kieltä se on kuitenkin JavaScriptiä. (Sengupta, Singhal & Corvalan [Viitattu 15.7.2017].)

React sallii kaksi eri tapaa määritellä elementtejä (kuva 12). Ensimmäinen tapa on käyttää JavaScript-funktioita ja toinen tapa on käyttää JSX-syntaksia. Sen tarjoamat aloitus- ja lopetustunnisteet auttavat luomaan komponentille puumaisen rakenteen. (Bertoli 2017, 20-21.) React toimii myös ilman JSX-syntaksia, mutta sen käyttö tekee siitä helposti luettavan sekä kirjoitettavan. JSX-syntaksia käyttäen React-komponentista tulee samaan tapaan hyvin järjestelty kuten HTML-elementistä. (AM & Sonpatki [Viitattu 8.7.2017])



Kuva 12. React-elementti tehtynä JSX-syntaksina sekä ilman.
(JSX [Viitattu 8.9.2017]).

JSX-syntaksia suositellaan käytettäväksi kuvailemaan, miltä käyttöliittymän kuuluisi näyttää. JSX-syntaksin sisälle voi kirjoittaa normaalia JavaScriptiä käyttämällä aaltosulkuja ({…}) kuvan 13 mukaisesti. JSX kannattaa pilkkoa monelle riville luetta-

vuuden vuoksi. Suositeltavaa on myös lisätä sen ympärille kaarisulut kuvan 13 mukaisesti. JSX-syntaksia voi myös käyttää if-lauseiden sekä for-silmukoiden sisällä. (React [Viitattu 28.4.2017].)

Code

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Kuva 13. JavaScriptin käyttö JSX-syntaksin sisällä.
(React [Viitattu 28.4.2017]).

4.2 Babel

Käyttääkseen JSX-kieltä (ja joitain ES2015 ominaisuuksia) on asennettava Babel. Syynä on se, että halutaan käyttää toimintoja, joita ei ole vielä lisätty selaimiin. Nämä ominaisuudet tekevät koodista siistimmän ohjelmoijille, mutta selaimet eivät osaa suorittaa niitä. Ratkaisu on kirjoittaa koodit käyttäen JSX-syntaksia sekä Es2015-koodia ja muuntaa ne ES5-muotoon. ES5 on standardi jota nykyiset selaimet käyttävät. Babel kääntää ES2015-koodin ES5-JavaScriptiksi, sekä muuntaa JSX-syntaksin JavaScript-funktioiksi. Babelin käyttö on melko yksinkertaista. Babelin saa asennettua käyttäen komentoa ”npm install – global babel-cli”. (Bertoli 2017, 21.)

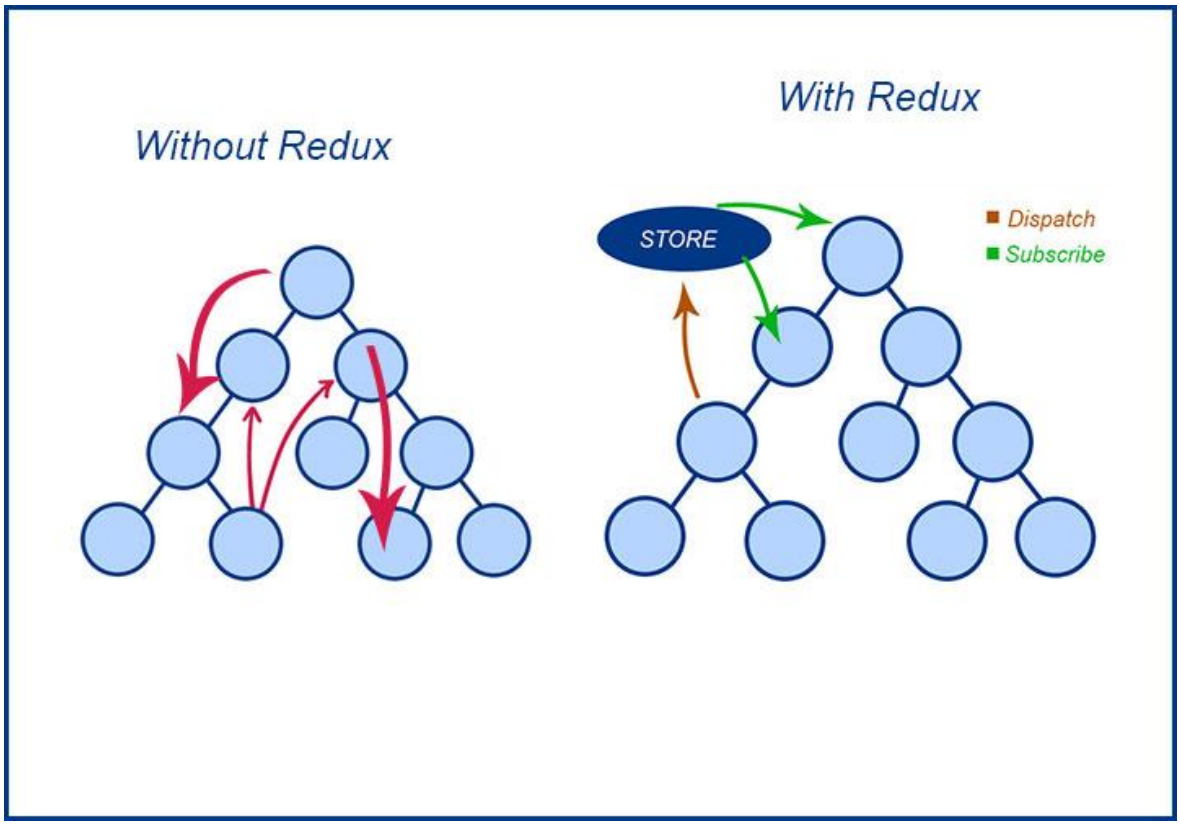
5 REDUX.JS

5.1 Mikä on Redux

Redux on erittäin suosittu kirjasto, jota ohjelmoijat käyttävät etenkin React-sovelluksissa. GitHubissa Reduxia kuvaillaan itseään ennustavana tilasäilönä JavaScript-sovelluksille. Sen sijaan että komponenteilla olisi oma tila, Redux tarjoaa koko React-sovelluksen yhteisen tilan kuvan 14 mallin mukaisesti. Jokainen komponentti pystyy muokkaamaan ja laajentamaan sovelluksen yhteistä tilaa. (Kho [Viitattu 3.6.2017].)

Sovelluksen voi tehdä pitkälle käyttämällä pelkästään komponenttien sisäisiä tiloja. Ongelma muodostuu silloin, kun halutaan jakaa kyseinen tilan toisen komponentin kanssa. Tätä helpottamaan on kehitetty erilaisia tilojen hallintatapoja. Flux-arkkitehtuuri oli ensimmäinen kunnollinen ratkaisu tähän ongelmaan. Se sallii ohjelman muokkaamisen käyttämällä toimintoja (actions), tiloja (stores) sekä näkymiä (views). Redux käyttää Fluxin toimintaideaa ja laittaa sen tiettyyn muottiin, jolloin sen käytöstä tulee vieläkin tehokkaampaa. (Vepsäläinen 2017, 64.) Suurimpana eroavaisuutena Fluxin sekä Reduxin välillä on tilasäilöjen määrä. Reduxissa käytetään yhtä tilasäiliötä, kun Fluxissa niitä voi olla monia. Redux on ladattava JavaScript-kirjasto, kun taas Flux on ohjelmointitapa. (Singh & Bhatt 2016, 114-115.)

Redux yksinkertaistaa tilojen käyttöä. Sovelluksen yhteiseen varastoon säilötään tiloja, joihin eri komponentit pääsevät käsiksi. Reduxia pystyy myös käyttämään muiden JavaScript-kirjastojen ja sovelluskehysten kanssa, kuten AngularJs- ja jQuery-kehyksen kanssa. (Singh & Bhatt 2016, 111-113.)



Kuva 14. Redux-säilö.
(Singh & Bhatt 2016, 113).

5.2 Redux-peruskäsitteet

Reduxia voi kuvailla kolmella pääperiaatteella koskien sovelluksen tilaa:

- yksi lähde datalle
- tila on vain luettava
- muutokset pystyvät tekemään vain puhtailla funktioilla

5.2.1 Tilapuu

Sovelluksen yhteiset tilat on säilötty yhteen tilapuuhun objektiksi. Tämä helpottaa ja nopeuttaa ohjelman kehitystä sekä testausta. Reduxin säilö toimii välittäjänä kaikille tilamuutoksille. Säilö toimii myös välikätenä kahden eri komponentin väliselle kommunikaatiolle (Singh & Bhatt 2016, 115). Toisin kuin React, joka säilöö jokaiseen komponenttiin oman tilan, Redux tallentaa tähän yhteen tilapuuhun

kaiken datan, jonka sovellus tarvitsee (Kho [Viitattu 3.6.2017]). Kuva 15 on esimerkki Reduxin käyttämästä tilapuusta, joka voi sisältää monia eri objekteja.

```
console.log(store.getState())

/* Prints
{
  visibilityFilter: 'SHOW_ALL',
  todos: [
    {
      text: 'Consider using Redux',
      completed: true,
    },
    {
      text: 'Keep all state in a single tree',
      completed: false
    }
  ]
}
*/
```

Kuva 15. Esimerkki Reduxin tilapuusta.
(Redux [Viitattu 2.5.2017]).

5.2.2 Vain luettava tila

React-sovelluksessa komponentit eivät pysty muuttamaan sovelluksen yhteistä tilaa, vaan tilan päivitykset tehdään lähettämällä haluttu muutos toimintojen (actions) kautta (Singh & Bhatt 2016, 115). Tällöin nykyisen tilan tiedot sekä päivitykset siirretään uuteen tilaan. Samalla muodostuu myös uusi tila objekti. Kuvassa 16 on esimerkki toiminnosta.

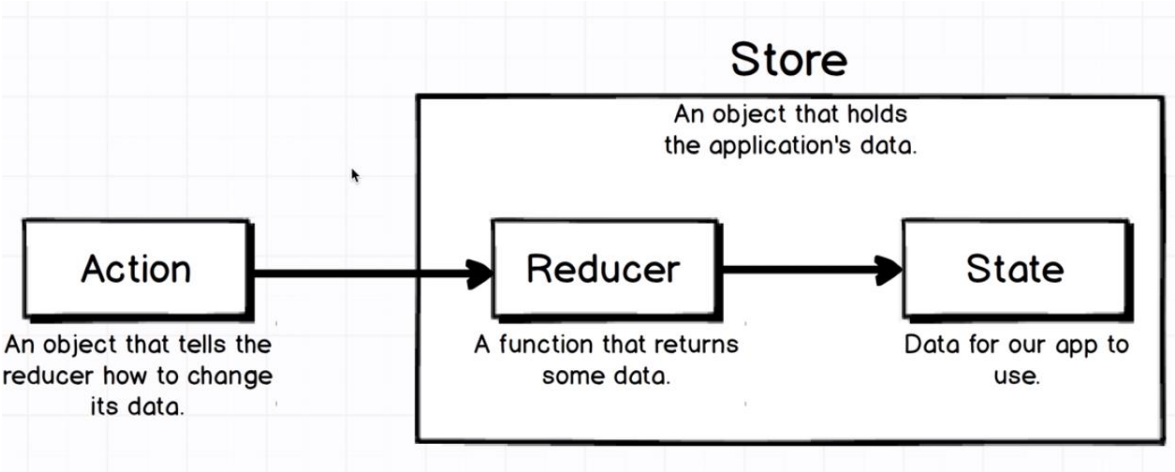
```
store.dispatch({
  type: 'COMPLETE_TODO',
  index: 1
})

store.dispatch({
  type: 'SET_VISIBILITY_FILTER',
  filter: 'SHOW_COMPLETED'
})
```

Kuva 16. Esimerkki toiminnosta
(Redux [Viitattu 2.5.2017]).

5.2.3 Muutokset tapahtuvat puhtaiden funktioiden avulla

Puhdas funktio on funktio, jolla ei ole mitään sivuvaikutuksia (kuten tiedoston lukeminen tai http-pyyntö). Puhtaiden funktioiden arvon voi aina ennustaa. Joka kerta kun sama funktio suoritetaan samoilla arvoilla, pysyvät myös funktion tuottamat loppuarvot samoina. Epäpuhtaissa funktioissa arvot saattavat muuttua, kun sama funktio suoritetaan uudestaan. Reduxissa käytetään vain puhtaita funktioita. (Dyl & Przeorski 2017, 21.)

Reduxin vain luettavaa-tilaa pystyy muokata toimien avulla. Reduxin reducerit ovat puhtaita funktioita, jotka suorittavat kyseisen muutoksen. Kaikki tilan muutokset säilötään Reduxin säilöön. (Kho [Viitattu 3.6.2017].) Kuva 17 näyttää toimintajärjestyksen ennen sovelluksen tilan päivittymistä.



Kuva 17. Tilan päivittäminen toiminnon avulla.
(Grider [Viitattu 20.4.2017]).

5.3 Pääkäsitteet

Reduxin käytön kannalta on välttämätöntä tietää muutama käsite. Nämä käsitteet ovat actions, reducers sekä store ja näiden toimintaan tutustutaan tässä kappaleessa.

5.3.1 Actions ja Action Creators

Toiminnot ”actions” ovat yksinkertaisia objekteja, jotka sisältävät informaation Reduxin säilöön lähetettävästä datasta. Reduxin säilöllä ei ole muuta tapaa saada dataa kuin toimintojen kautta. Toiminto vaatii aina tyyppiarvon, ”type”. Tämä kovakoodattu arvo, kuvaa millainen toiminto suoritetaan. (Kho [Viitattu 3.6.2017].) Tyyppiarvon lisäksi objektille voi antaa muitakin arvoja. Suositeltavaa on kuitenkin lähettää niin vähän dataa kuin mahdollista. (Actions, [Viitattu 11.9.2017].)

Toiminto koostuu kahdesta osasta, action creatorista sekä actionista. Action creator on funktio, joka palauttaa objektin. Tämän jälkeen objekti lähetetään aina kaikille reducereille. Tätä action creatorin lähettämää objektia kutsutaan actioniksi. (Grider [Viitattu 20.4.2017].) Kuvassa 18 on esimerkki toiminnosta, siinä näkyy action creator -funktio sekä sen sisällä oleva action-objekti.

```
const ADD_TASK = 'ADD_TASK';

export function addTask(taskName) {
  return {
    type: ADD_TASK,
    taskName: taskName
  }
}
```

Kuva 18. Esimerkki toiminnosta.

Actionit ja action creatorsit ovat sovelluksen tilan muutoksia varten. Actionin aktivoi yleensä käyttäjä. Näitä tapahtumia voi olla esimerkiksi nappulan painaminen, syöttökenttään kirjoittaminen tai sivun lataaminen. Grider (2016) kertoo action toimintojen elinkaaren menevän seuraavasti:

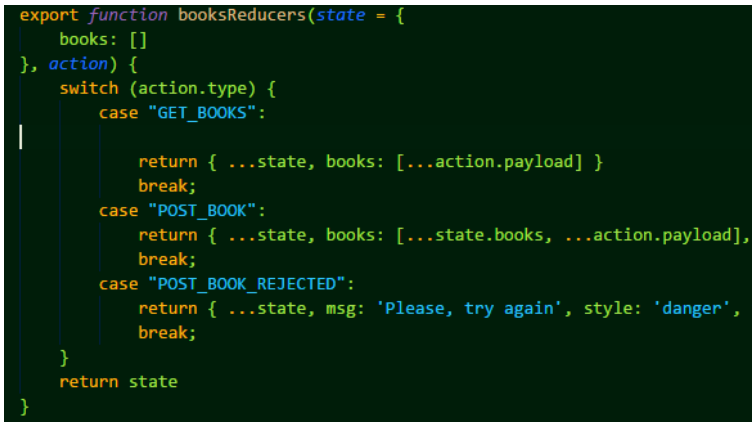
1. Käyttäjä suorittaa toiminnon
2. Toiminto aktivoi action creatorin
3. Action creator palauttaa objektin eli actionin
4. Objekti läheteään reducereille
5. Reducer palauttaa tietyn tilan, riippuen action creatorin sisällöstä
6. Reducerin palauttama tila siirtyy sovelluksen tilaksi
7. Sovelluksen tila lähetetään Reactin komponenteille
8. Tarvittavat komponentit renderöidään uudelleen.

5.3.2 Reducers

Reduxin reducer-funktio ottaa sovelluksen nykyisen tilan ja päivittää siihen actionin kautta tulevat uudet arvot. Tämän jälkeen päivitetty tila on käytettävissä koko sovelluksen laajuisesti. Kho (Kho [Viitattu 3.6.2017]) painottaa kirjassaan reducer-funktioissa kahta asiaa:

- Sovelluksen tila ei koskaan muutu, sen sijaan kopio tilasta palautetaan muuttuneiden arvojen kanssa.
- Aina kun action-toiminto laukaistaan, jokainen toiminto lähetetään kaikille reducereille. Aiemmin esitelty toiminnon tyyppiarvo ratkaisee sen, tuleeko toiminto käytettäväksi vai ei.

Reducerit tehdään yleensä kuvan 19 mukaisesti switch-case-muotoon. Jos aiemmin esitelty toiminnon tyyppiarvo vastaa reducerin switch-case-tapauksen arvoa, siirtyvät toiminnon arvot reducerille. (Grider [15.4.2017]).

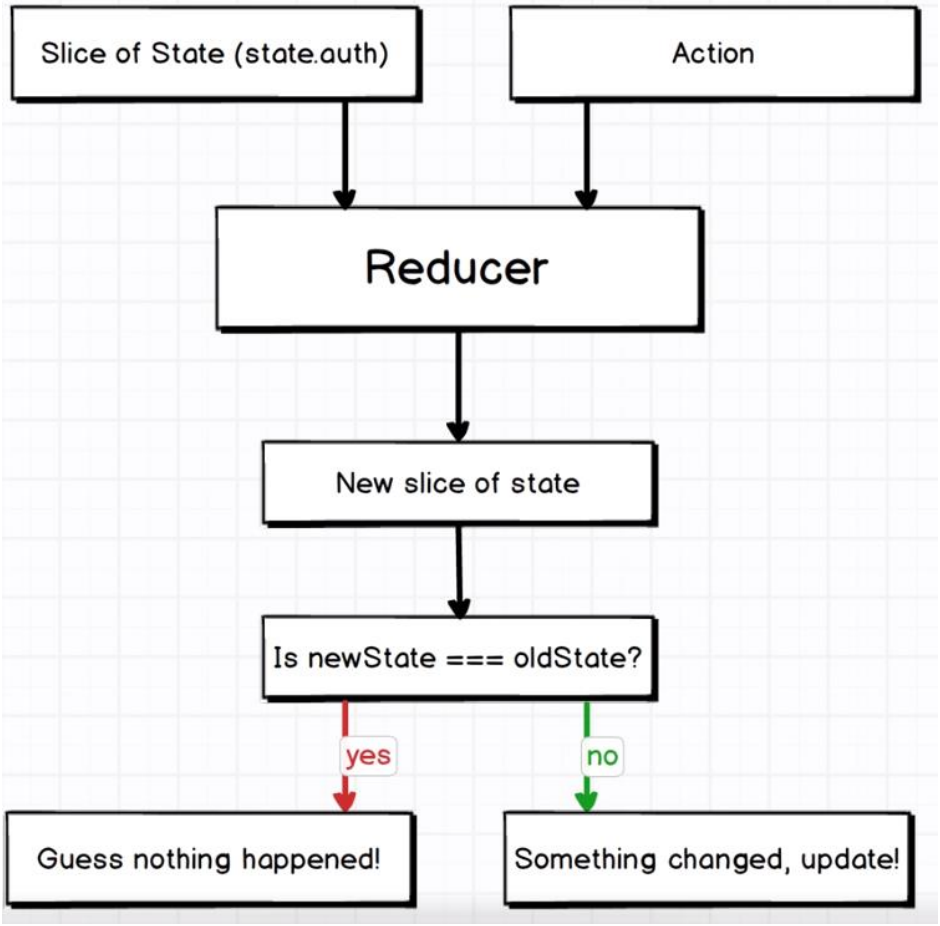


```
export function booksReducers(state = {
    books: []
}, action) {
    switch (action.type) {
        case "GET_BOOKS":

            return { ...state, books: [...action.payload] }
            break;
        case "POST_BOOK":
            return { ...state, books: [...state.books, ...action.payload],
            break;
        case "POST_BOOK_REJECTED":
            return { ...state, msg: 'Please, try again', style: 'danger',
            break;
    }
    return state
}
```

Kuva 19. Reducerin rakenne.

5.3.3 Sovelluksen tila

Reduxin tila on sovelluksen yhteinen data, joka on tallennettu yhteen tilapuuhun. Näitä datoja voi olla esimerkiksi syötetyt arvot tekstikentissä, tieto käyttäjän kirjautumisesta, mitkä kuvat ovat näkyvissä, ja niin edelleen. Sovelluksen kaikki yhteinen data muodostaa sovelluksen tilan. (Grider [15.4.2017].) Reducer-funktion aktivoituessa Redux katsoo tilan vanhan ja uuden arvon. Jos uusi tila on identtinen vanhan tilan kanssa, mitään ei tapahdu. Jos tilojen välillä on eroavaisuutta, tila päivitetään ja uusi tila lähetetään komponenteille. (Grider [Viitattu 20.4.2017].) Kuva 20 havainnollistaa edellä mainitun asian.



Kuva 20. Tilojen vertailu.
(Grider [Viitattu 20.4.2017]).

5.3.4 Store

Redux-sovelluksessa on vain yksi tilasäilö (store). Tänne on säilötty sovelluksen yhteinen data, joihin kaikilla komponenteilla on pääsy. Storen sisältöä pystyy muokkaamaan aiemmin mainituilla actioneilla sekä reducereilla (Singh & Bhatt 2016, 124.) Reduxin (Redux 2 [Viitattu 12.5.2017]) kotisivuilla lukee storen tehtävistä seuraavasti:

- Säilöö sovelluksen tilan
- Sallii pääsyn sovelluksen tilaan
- Sallii sovelluksen tilan päivittämisen.

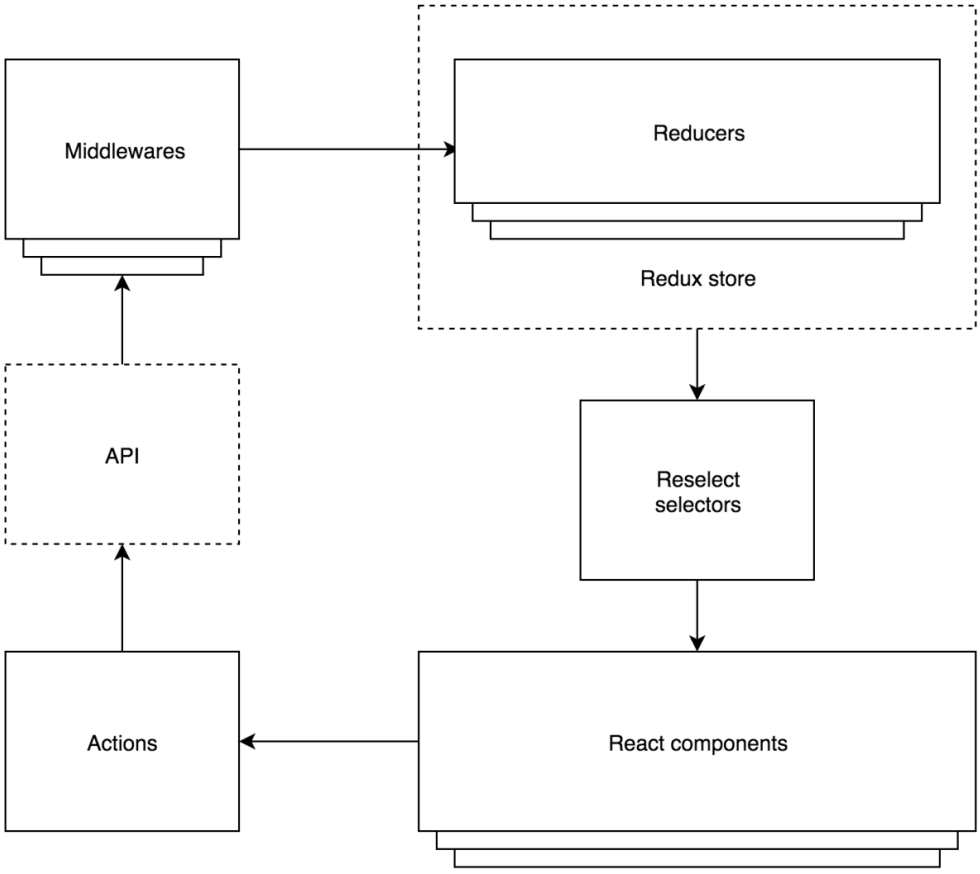
5.4 Redux-arkkitehtuuri

Kuten aikaisemmin mainittiin, Redux on ominut Fluxin rakenteen, joten sillä on myös vastaavanlainen arkkitehtuuri. Tilan muutokset lähetetään storeen toimintojen sekä reducereiden kautta. Lisäksi store mahdollistaa kommunikoinnin komponenttien välillä. (Singh & Bhatt 2016, 116.)

Actionin sekä reducerin välissä on mahdollista käyttää kolmannen osapuolen laajennuksia, tätä kohtaa kutsutaan Reduxin middlewareksi. Middlewaressa voi esimerkiksi kirjata dataa, ilmoittaa kaatumisia tai tehdä API-pyyntöjä. (Redux 3 [Viitattu 12.5.2017].)

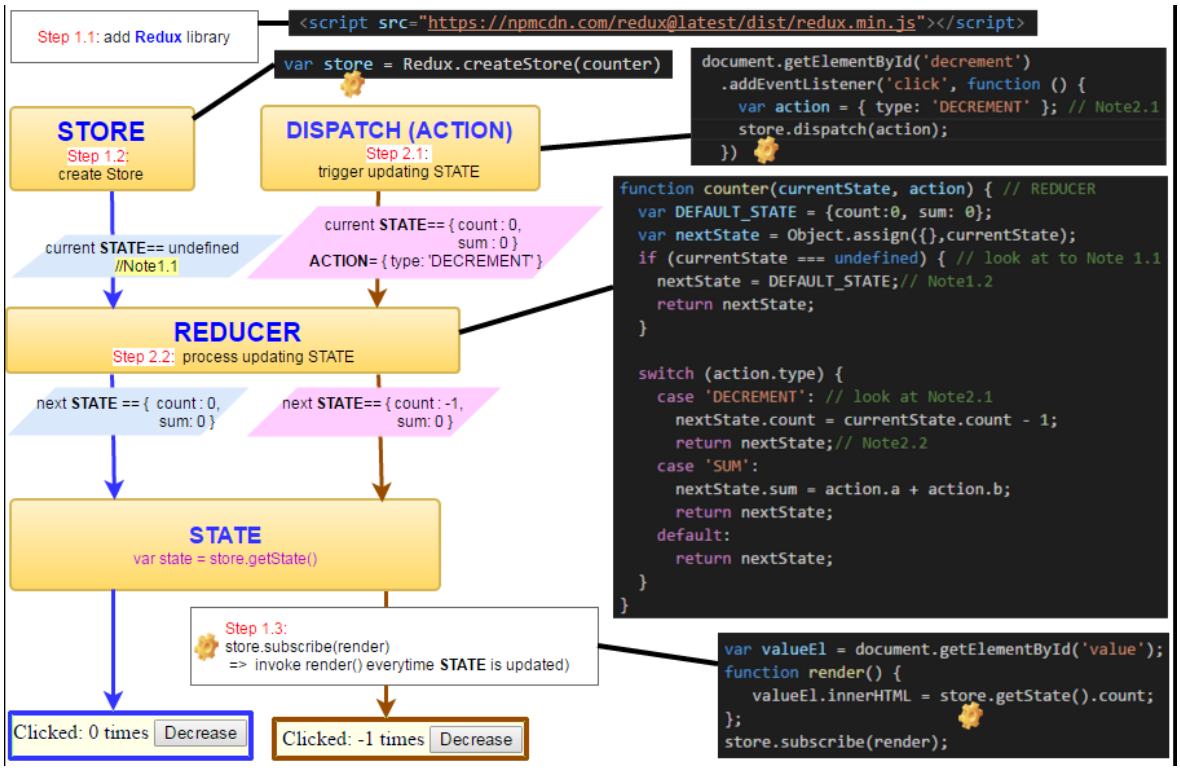
Middleware on funktio, jonka kautta kaikki actionit joutuvat kulkemaan. Riippuen actionin tyypistä, middleware voi päästää actionin jatkamaan matkaa, muuttaa actionin arvoja tai pysäyttää sen kulun. Middleware mahdollistaa monien erilaisten tapahtumien aktivoinnin ennen reducereita. Middlewareja voi olla monia peräkkäin. (Grider [15.4.2017].)

Kuvassa 21 on esitelty Reduxin arkkitehtuuri. Käyttäjä aktivoi tapahtuman kuvan oikeassa alakulmassa. Tapahtuma käy jokaisen middlewaren lävitse. Tämän jälkeen tapahtuma lähetetään kaikille reducereille, joissa tarkistetaan, onko toiminto heille sopiva. Reducerit päivittävät toiminnolta tulevien arvojen mukaan sovelluksen tilan. Tämän jälkeen uusi data lähetetään komponenteille, mikä näkyy käyttäjälle näkymän päivittymisenä. (Singh & Bhatt 2016, 118.)



Kuva 21. Redux-arkkitehtuuri.
(Singh & Bhatt 2016, 117).

Kuva 22 esittää kuinka yksinkertaisimmillaan Redux otetaan käyttöön. Leonardo Daniel (2016) jakaa videossaan Reduxin kahteen eri vaiheeseen. Vaiheessa yksi olevat kohdat määritellään vain kertaalleen, riippumatta siitä kuinka iso ja monimutkainen sovellus on. Tähän vaiheeseen kuuluu Rudex-kirjaston lisääminen projektiin sekä storen luonti. Vaiheessa kaksi kuuluvat actionit sekä reducerit, sillä näitä voi olla projektissa monia.



Kuva 22. Vaiheet 1.1–1.3 määritetään kertaalleen, 2.1–2.2 mahdollisesti useamman kerran.
(Daniel [Viitattu 17.5.2017]).

6 REACT-REDUX

6.1 Reactin sekä Reduxin yhdistäminen

Tomasellon (2017) mukaan react-komponentit voivat olla joko säiliökomponentteja (smart components/containers) tai esityksellisiä komponentteja (dumb components/presentational components) riippuen siitä, onko komponentti yhdistettynä Reduxin luomaan sovelluksen tilaan. Säiliökomponentit saavat dataa sovelluksen yhteisestä tilasta. Esityksellisillä komponenteilla ei pysty vastaanottamaan tai lähettämään muutoksia Reduxin säilöön.

Jotta Reduxia pystyy käyttämään Reactin kanssa, täytyy Reduxin tilan kulkuun lisätä yksi vaihe, provider. Provider tulee koko React sovelluksen isännäksi, jolloin se pystyy lähettämään sovelluksen tiloja säilöstä React-komponenteille. (Tomasello 2017.)

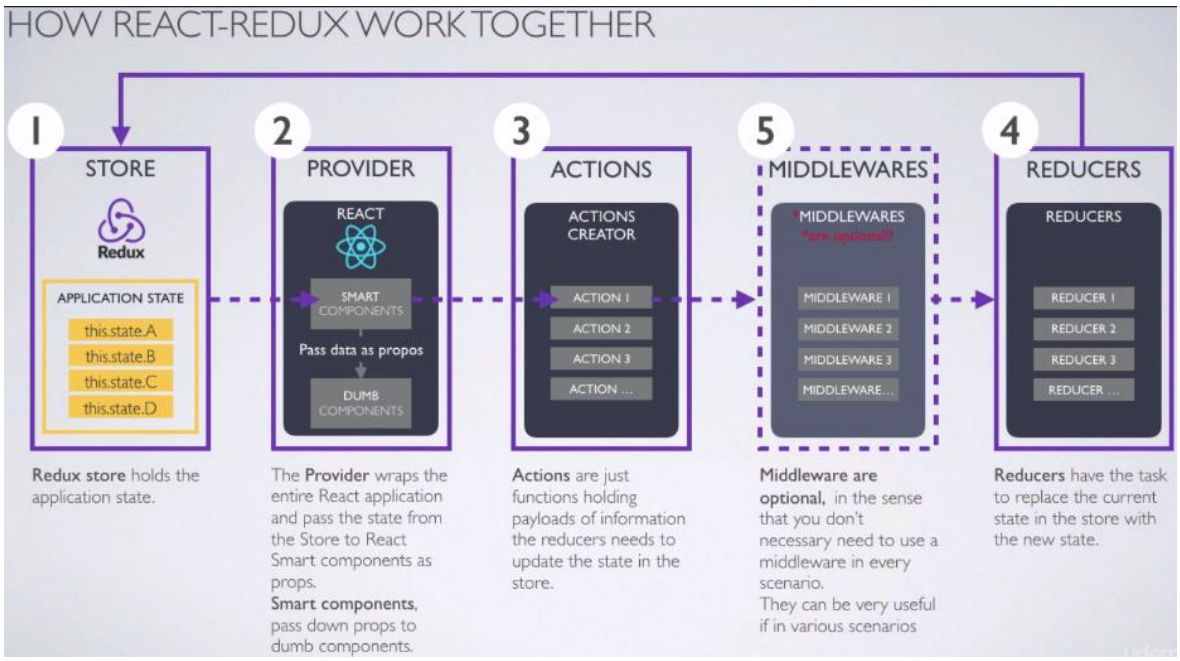
```
ReactDOM.render(
  <Provider store={createStoreWithMiddleware(reducers)}>
    <App />
  </Provider>
  , document.querySelector('.container'));
```

Kuva 23. Provider käärii sisälleen Reactin pääkomponentin.

Suurin osa Reactin komponenteista on esityksellisiä komponentteja. Reduxiin kytkemistä varten tarvitsee tehdä muutamia säiliökomponentteja. Nämä komponentit pystyvät noutamaan dataa säilöstä sekä päivittämään sovelluksen tiloja. Kyseiset komponentit pystyvät muuttamaan sovelluksen tilaa toimintojen avulla. (React-Redux [Viitattu 11.9.2017].)

React-Redux-kirjaston mukana tulee connect()-funktio, joka sallii säiliökomponentin yhdistämisen Reduxin tilaan. Connect()-funktio mahdollistaa mapStateToProps()- sekä mapDispatchToProps()-funktioiden käytön. MapStateToProps()-funktiota käytetään sovelluksen tilan tuomiseen säiliökomponentille. MapDispatchToProps()-funktiota käytetään sovelluksen tilan päivittämiseen. Tämä aktivoi toiminnon, joka lähetetään reducereille. (React-Redux [Viitattu 11.9.2017].)

Kuvassa 24 näkyy, kuinka React-Redux toimii yhdessä sekä kuinka data kulkee sovelluksen sisällä tilaa päivittäessä. Kohta 1 edustaa Reduxin tilasäiliöä, johon sovelluksen tilat on säilötty. Kohdassa 2 esitetään, kuinka provider mahdollistaa datan lähettämisen säiliöstä komponenteille. Kohdassa 3 säiliökomponentti on aktivoinut toiminnon, jonka tarkoituksena on päivittää sovelluksen tila. Kohdassa 4 toiminto on edennyt reducerille, jossa tilan päivitys toteutuu. Tämän jälkeen uusi tila on käytettävissä komponenteille.



Kuva 24. Datan kulku React-Redux-sovelluksessa.
(Daniel [Viitattu 17.5.2017]).

7 Yhteenveto ja pohdinta

Työn tavoitteena oli tutustua viime vuosina suosituksi tulleeseen React JavaScript -kirjastoon. Työssä käytiin läpi, kuinka deklaratiivinen sekä imperatiivinen ohjelmointi eroavat toisistaan. Lisäksi tutustuttiin komponenttipohjaiseen sovelluksen tekoon. Työ esitteli myös Redux JavaScript -kirjaston, jolla dataa pystyy säilömään sovelluksen yhteisteen säilöön.

React ja Redux sisältävät paljon monimutkaisia käsitteitä ja termejä. Näitä käytiin opinnäytetyössä läpi perusteellisesti käyttäen apuna monia eri lähteitä sekä kuvia. Lukijalle pitäisi jäädä kattava käsitys Reactin ja Reduxin arkkitehtuurista, datan kulusta sekä toiminnoista.

Suurimpana haasteena opinnäytetyössä oli lähteiden löytäminen, sekä vaikeiden käsitteiden selittäminen. Monissa eri lähteissä Reactista ei oltu kerrottu paljoakaan.

Kirjoittajalla oli entuudestaan enemmän kokemusta jQuery JavaScript -kirjastosta sekä PHP-kielestä. Siirtyminen Reactiin oli melko vaativa, sillä jQuery sekä React eroavat toisistaan paljon. Opinnäytetyön tekeminen auttoi syventymään Reactiin niin teoriassa kuin käytännössä. Samalla React Native tuli tutummaksi mobiilisovellusten tekemiseen.

LÄHTEET

Actions. Ei päiväystä. Actions. [Verkkosivu]. Redux.js.org. [Viitattu 11.9.2017]. Saatavana: http://redux.js.org/docs/basics/Actions.html

AM, V. & Sonpatki, P. 2016. [Verkkokirja]. ReactJS by Example – Build Modern Web Applications with React. Birmingham UK: Packt Publishing Ltd. [Viitattu 8.7.2017]. Saatavana Packt-palvelusta. Vaatii käyttöoikeuden.

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

Boduch, A. 2017. React and React Native. [Verkkokirja]. Birmingham UK: Packt Publishing Ltd. [Viitattu 26.7.2017]. Saatavana Packt-palvelusta. Vaatii käyttöoikeuden.

Code School. Ei päiväystä. Beginner's guide to web development. [Verkkojulkaisu]. Orlando: Code School LLC. [Viitattu 28.4.2017]. Saatavana: http://courseware.codeschool.com/beginners_guide/CodeSchool-BeginnersGuideToWebDevelopment.pdf

Codemurai. Ei päiväystä. Learn Programming. [Puhelinsovellus]. Zenva Pty Ltd. [Viitattu 1.5.2017].

Daniel, L. 2016. React Redux React-Router: From Beginner to Paid Professional. [Video]. San Francisco: Udemy, Inc. [Viitattu 17.5.2017]. Saatavana: https://www.udemy.com/react-redux-react-router/learn/v4/t/lecture/5575444?start=0. Vaatii käyttöoikeuden.

DOM. Ei päiväystä. JavaScript HTML DOM [Verkkosivu]. w3schools. [Viitattu 5.9.2017]. Saatavana: https://www.w3schools.com/js/js_htmldom.asp

Dyl, T. & Przeorski, K. 2017. Mastering Full Stack React Web Development. Birmingham UK: Packt Publishing Ltd.

Electron. Ei päiväystä. Build cross platform desktop apps with JavaScript, HTML, and CSS [Verkkosivu]. Electron. [Viitattu 14.9.2017]. Saatavana: https://electron.atom.io/

Grider, S. 2016. Modern React with Redux. [video]. San Francisco: Udemy, Inc. [Viitattu 15.4.2017]. Saatavana: https://www.udemy.com/react-redux/learn/v4/t/lecture/4288062?start=0. Vaatii käyttöoikeuden.

Grider, S. 2017. The Complete React Native and Redux Course. [video]. San Francisco: Udemy, Inc. [Viitattu 20.4.2017]. Saatavana: https://www.udemy.com/the-complete-react-native-and-redux-course/learn/v4/t/lecture/5744002?start=0. Vaatii käyttöoikeuden.

Grider, S. 2017. Webpack 2: The Complete Developer's Guide. [video]. San Francisco: Udemy, Inc. [Viitattu 2.6.2017]. Saatavana: https://www.udemy.com/webpack-2-the-complete-developers-guide/learn/v4/t/lecture/6296178?start=0. Vaatii käyttöoikeuden.

JSX. Ei päiväystä. JSX In Depth [Verkkosivu]. React. [Viitattu 8.9.2017]. Saatavana: https://facebook.github.io/react/docs/jsx-in-depth.html

Kho, R. 2017. React Native By Example. [Verkkokirja]. Birmingham UK: Packt Publishing Ltd. [Viitattu 3.6.2017]. Saatavana Packt-palvelusta. Vaatii käyttöoikeuden.

Masiello, E. & Friedmann, J. 2017. Mastering React Native. [Verkkokirja]. Birmingham UK: Packt Publishing Ltd. [Viitattu 2.9.2017]. Saatavana Packt-palvelusta. Vaatii käyttöoikeuden.

Mead, A. & Percival, R. 2017. The Complete Node.js Developer Course (2nd Edition). [Video]. San Francisco: Udemy, Inc. [Viitattu 2.6.2017]. Saatavana: https://www.udemy.com/the-complete-nodejs-developer-course-2. Vaatii käyttöoikeuden.

Microsoft. Ei päiväystä. React Native plugin for Universal Windows Platform (UWP). [Verkkosivu]. GitHub Inc. [Viitattu 14.9.2017]. Saatavana: https://github.com/Microsoft/react-native-windows

MVC Architecture. Ei päiväystä. AngularJS – MVC Architecture. [Verkkosivu]. Tutorials point. [Viitattu 4.9.2017]. Saatavana: https://www.tutorialspoint.com/angularjs/angularjs_mvc_architecture.htm

MVC. 25.2.2015. Programming in Java Using the MVC Architecture. [Verkkosivu]. C# Corner. [Viitattu 4.9.2017]. Saatavana: http://www.c-sharpcorner.com/UploadFile/201fc1/programming-in-java-using-the-mvc-architecture/

Node.js. Ei päiväystä. Node.js [Verkkosivu]. Node.js Foundation. [Viitattu 2.6.2017]. Saatavana: https://nodejs.org/en/

Novick, V. 2017. [Verkkokirja]. React Native - Building Mobile Apps with JavaScript. Birmingham UK: Packt Publishing Ltd. [Viitattu 1.9.2017]. Saatavana Packt-palvelusta. Vaatii käyttöoikeuden.

React Native 2. 27.12.2016. What are the main differences between ReactJS and React-Native? [Verkkosivu]. Medium [Viitattu 15.5.2017]. Saatavana:

https://medium.com/@alexmngn/from-reactjs-to-react-native-what-are-the-main-differences-between-both-d6e8e88ebf24

React Native. Ei päiväystä. React Native. [Verkkosivu]. [Viitattu 15.5.2017]. Saatavana: https://facebook.github.io/react-native/

React. Ei päiväystä. Introducing JSX. [Verkkosivu]. [Viitattu 28.4.2017]. Saatavana: https://facebook.github.io/react/docs/introducing-jsx.html

React-Redux. Ei päiväystä. Usage with React. [Verkkosivu]. Redux.js.org. [Viitattu 11.9.2017]. Saatavana: http://redux.js.org/docs/basics/UsageWithReact.html

Redux 2. Ei päiväystä. Store. [Verkkosivu]. Redux.js.org. [Viitattu 12.5.2017]. Saatavana: http://redux.js.org/docs/basics/Store.html

Redux 3. Ei päiväystä. Middleware. [Verkkosivu]. Redux.js.org. [Viitattu 12.5.2017]. Saatavana: http://redux.js.org/docs/advanced/Middleware.html

Redux. Ei päiväystä. Three Principles. [Verkkosivu]. Redux.js.org. [Viitattu 2.5.2017]. Saatavana: http://redux.js.org/docs/introduction/ThreePrinciples.html#single-source-of-truth

Robbestad, S.A. 2017. [Verkkokirja]. ReactJS Blueprints. Birmingham UK: Packt Publishing Ltd. [Viitattu 9.7.2017]. Saatavana Packt-palvelusta. Vaatii käyttöoikeuden.

Sengupta, D. Singhal, M. & Corvalan, D. 2016. Getting Starged with React. [Verkkokirja]. Birmingham UK: Packt Publishing Ltd. [Viitattu 15.7.2017] Saatavana Packt-palvelusta. Vaatii käyttöoikeuden.

Singh, H. & Bhatt, M. 2016. Learning Web Development with React and Bootstrap. Birmingham UK: Packt Publishing Ltd.

Tomasello, M. 2017. Full stack Universal React with Redux, Express and MongoDB. [Video]. San Francisco: Udemy, Inc. [Viitattu 16.5.2017]. Saatavana: https://www.udemy.com/full-stack-universal-react-with-redux-express-and-mongodb/learn/v4/t/lecture/7015032. Vaatii käyttöoikeuden.

Vepsäläinen, J. 2017. SurviveJS – React. 2.5.7. British Columbia, Canada: Leanpub.

Vice, R & Horton, A. 2016. Mastering React. [Verkkokirja]. Birmingham UK: Packt Publishing Ltd. [Viitattu 25.5.2017]. Saatavana Packt-palvelusta. Vaatii käyttöoikeuden.