Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Jussi-Pekka Lilja

Kirjastosovelluksen toteuttaminen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Ohjelmistotuotanto

Insinöörityö

12.11.2019

Tekijä Otsikko Sivumäärä Aika	Jussi-Pekka Lilja Kirjastosovelluksen toteuttaminen 25 sivua 12.11.2019
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Ohjelmistotuotanto
Ammatillinen pääaine	Ohjelmointi
Ohjaajat	Head of Departmen(ICT) Janne Salonen

Työn tarkoituksena oli kehittää järjestelmä, jolla voi etsiä ja lainata työpaikan kirjastossa olevia kirjoja. Aiemmin ei ollut kunnollista järjestelmää, ja jos tietty kirja oli lainassa, ei voinut olla varma, kenen hallussa se oli. Kirjoista ei myöskään ollut kunnollista listausta.

Järjestelmä toteutettiin verkkosovelluksella, jonka saisi helposti käyttöön mobiililaitteille, käyttöjärjestelmästä riippumatta.

Käyttöliittymän toteuttamiseen käytettiin suosittua javascript-kirjastoa React.js:ää. Käyttöliittymästä tehdään kutsuja rajapinnalle, joka palauttaa ja käsittelee tarvittavat tiedot käyttöliittymälle. Rajapinta toteutettiin Express.js:llä. Palvelinympäristössä käytössä oli Node.js, ja tietokannassa MySQL.

Avainsanat	React, Node.js, Express, MySQL

Author Title Number of Pages Date	Jussi-Pekka Lilja Building a library application 25 pages 12 October 2019
Degree	Bachelor of Engineering
Degree Programme	Ohjelmistotuotanto
Professional Major	Programming
Instructors	Janne Salonen Head of Department(ICT)

The purpose of the project was to create an application that you could use to search and loan the books that the company owned. There wasn't a real system before, so if a book was loaned to someone, there was uncertainty who had it. The books hadn't really been properly listed either.

The application was decided to be made as a web app, so that it could easily be used from mobile devices, regardless of the operating system.

The user interface was made using the popular Javascript library React.js. Requests are made from the UI to the Rest API, which formats the data and returns it to the UI. For the server, Node.js was used, and for the database, MySQL

Keywords	React, Node.js, Express, MySQL

Sisällys

Lyhenteet

CSS Cascading Style Sheets. Eräänlaiset tyyliohjeet selaimelle, jotka määrittävät kuinka selain näyttää sivun käyttäjälle

HTML Hypertext Markup Language. Standardisoitu kuvauskieli, jolla näytetään dokumentteja web-selaimessa.

SQL	Structured Query Language. Standardiosoitu kyselykieli, jota käytetään hakujen ja muutosten tekemiseen relaatiotietokannassa.
MySQL	Avoimen lähdekoodin SQL-järjestelmä.
REST	Representational State Transfer. http-protokollaan perustuva arkkitehtuuri, jonka avulla luodaan web-palveluita.
ECMA	European Computer Manufacturers Association. Järjestö jonka tarkoituksena on kehittää ja ylläpitää standardeja teknologian alalla.
DOM	Document Object Model. Kuvastaa dokumentin käyttöliittymää.
API	Application Programming Interface. Ohjelmointirajapinta, jonka avulla sovelluksen eri osat voivat keskustella keskenään.

1. Johdanto

Exove Oy on vuonna 2006 perustettu yritys, joka tekee nettisivuja erilaisille asiakkaille, yksityisille yrityksille sekä julkiselle sektorille. Teknologioina ovat yleisesti Drupal ja Wordpress, mutta projekteja on monenlaisia. Myös tässäkin työssä käytettyä React.js -kirjastoa on käytetty yrityksen projekteissa. Exovella ei tällä hetkellä ole omia tuotteita, vaan asiakas omistaa tuotetun koodin.

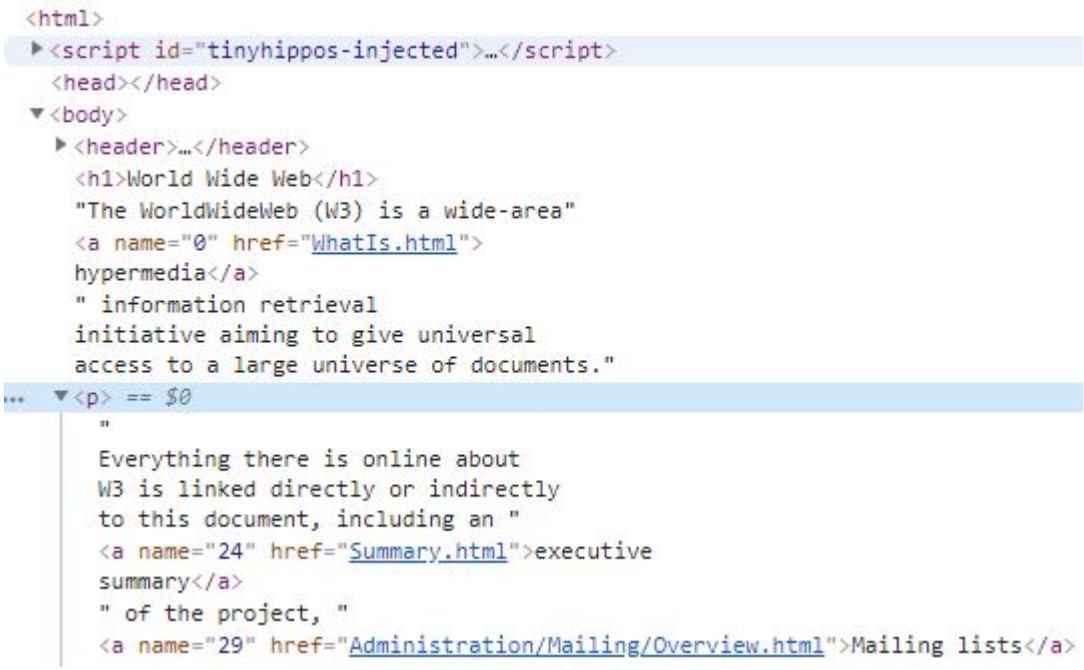
Exovella järjestetään usein koulutuksia työntekijöille, ja yrityksellä on myös laaja valikoima tietotekniikan alan kirjoja, joita työntekijät voivat lukea tai lainata. Kirjojen listaus on ollut puutteellinen, ja työn tarkoituksena on luoda järjestelmä, jolla valikoimaa voi selata helpommin, sekä myös lainata ja palauttaa teoksia. Haun voi tehdä kirjan tai kirjailijan nimen perusteella, sekä asiasanoilla etsimällä. Sovellusta on tarkoitus käyttää vain Exoven sisäisestä verkosta, tosin työntekijät voivat selata kirjoja myös kotoaan VPN-yhteyttä käyttäen.

Tarkoituksena oli myös perehtyä syvemmin sovelluskehitykseen, erityisesti Reactin ja oman REST API:n tekemiseen alusta asti. Tässä raportissa käsitellään sovelluksen tekemisessä käytettyjä teknologioita, hieman niiden historiaa, ja miten näitä teknologioita käytettiin sovelluksessa, sekä mitä ongelmia tuli vastaan, ja mitä näistä voidaan oppia seuraavaa projektia ajatellen. Lopussa mietitään myös jatkokehitysideoita sovellukselle.

2. Sovelluksen kehittämiseen käytetyt teknologiat

2.1. HTML

HTML (Hypertext Markup Language) on kuvauskieli dokumenteille joita näytetään selaimessa. Sillä voidaan kuvata mikä osa on dokumentin otsikko, ja mikä osa leipätekstiä. HTML koostuu elementeistä, joita voivat olla aiemmin mainitut otsikot, ja myös esimerkiksi lomakkeet, kuvat ja linkit.



1. Ensimmäisen HTML-sivun koodia. [5.]

Elementtien avulla voidaan kertoa selaimelle, miten sen tulisi käsitellä eri osia dokumentista, esimerkiksi näyttää otsikko elementti suuremmalla. HTML:ään sisällytetään myös tyylien määrittely (CSS) sekä skriptejä (nykyään käytännössä Javascript). Molempia käsitellään tarkemmin myöhemmin. Pelkkää HTML:ää käyttäen sivu näyttäisi karulta nykykäyttäjän silmille.

Vuonna 1989 fyysikko Tim Berners-Lee, joka työskenteli Cernille, teki ehdotuksen tavasta jakaa dokumentteja internetin kautta, käyttäen hypertekstiin perustuvaa järjestelmää. [6.] Ensimmäinen prototyyppi valmistui 1990, ja perustui aikaisempaan kuvauskieleen, SGML:ään (Standard Generalized Markup Language). Vaikka tarkoitus olikin käyttää HTML:ää tutkimustulosten jakamiseen, Bernes-Lee ei pitänyt ideoitaan yksityisenä, vaan kehotti muita osallistumaan kehitykseen. Tämä olikin yksi syy, miksi HTML perustui aiemmin standardisoituun kieleen.

HTML:n kehitys tapahtui pitkään akateemisissa piireissä, koska mikään isompi yritys ei ollut innokas ottamaan osaa kehitykseen. Kukaan ei ollut vielä varma, pystyisikö idealla tienaamaan rahaa. Kehitys kuitenkin jatkui, ja HTML jatkoi muotoutumista. Versio 2.0 julkaistiin heinäkuussa 1994, ja myöhemmin samana vuonna muodostettiin yhtiö nimeltä Netscape Communications Corporated, joka julkaisi Netscape-selaimen. Netscape ei ollut ensimmäinen selain, mutta sillä oli merkittävä rooli HTML:n esittelyssä isolle yleisölle, sekä myös HTML:n standardien muotoutumisessa. Jo seuraavana vuonna, 1995, Microsoft julkaisi oman selaimensa, Internet Explorerin, josta tuli suuri

kilpailija Netscapelle. Protokollia ja standardeja kehittämään perustettiin World Wide Web Consortium (W3C), joka yhä jatkaa toimintaansa [7.]. W3C:n sivuilta kehittäjät voivat hakea erilaista tietoa, ja yhtiö myös jatkaa erinäisten standardien kehittämistä, joista eräs tärkeimmistä viime vuosina on ollut saavutettavuus. [8.]

HTML5 on uusin versio HTML:stä. Se tuli saataville 2008, ja W3C:n suosituksiin vuonna 2014. Sen päämääränä on tukea multimedialementtejä, ja pitää koodi ihmisille helposti luettavissa. Muun muassa <video> ja <audio> elementit tulivat HTML5:n mukana. [9.]

Monet ohjelmistot myös tuottavat HTML:ää, esimerkiksi suositut WYSIWYG editorit (What you see is what you get), eli editorit joilla HTML:ää tuntematon käyttäjä voi muotoilla tekstiä, ja editori tuottaa tarvittavan HTML:n. Tuotetun koodin laatu ei kuitenkaan usein ole kovin hyvää, mikä hankaloittaa sen luettavuutta.

2.2. CSS

CSS eli Cascading Style Sheets, karkeasti käännettynä porrastetut tyyliarkit, on kuvailukieli, jolla kerrotaan selaimelle miten sisältö tulisi näyttää. HTML:n elementit ajavat osittain samaa asiaa, ja CSS:n sääntöjä voidaan myös sisällyttää niihin, mutta CSS voidaan erottaa HTML-elementeistä, ja välttää saman määrittelyn kirjoittamista elementteihin, mikä helpottaa kehitystä. CSS-tiedosto on myös yleensä erillinen HTML-tiedostosta, mikä helpottaa luettavuutta.

```
h1 {
    font-size: 20px;
}

.ad-section h1 {
    font-size: 25px;
    background: red;
}

.ad-section {
    background: green;
}
```

2. Esimerkki CSS-syntaksista

Kieli on itsessään melko yksinkertainen verrattuna moniin muihin. Siinä on käytössä porrastuvia sääntöjä, joiden avulla määritellään, mikä tyyli pätee mihinkin elementtiin. Säännöt voivat koskea elementtejä, tai elementtien ominaisuuksia, kuten luokkia. Elementillä voi olla monta luokkaa (class), ja luokkaa voi myös käyttää erilaisissa elementeissä. Luokka merkitään CSS:ssä pisteellä (.).

Kuvan 2. esimerkin mukaan kaikkien h1 elementtien fonttikoko on 20 pikseliä. Jos kuitenkin h1-elementti on elementin sisällä, jonka luokka on ”ad-section”, tuleekin kooksi 25 pikseliä. Kaikkien elementtien, joiden luokka on ad-section, taustaväriksi tulee vihreä. Syntaksi on siis yksinkertainen, mutta sivuston ja erilaisten tyylien tarpeiden kasvaessa myös CSS-tiedostot saattavat paisua, kun uusia ehtoja pitää luoda.

CSS:ää voi koodata myös edistyneemmillä kielillä, kuten SASS, SCSS ja LESS. Näihin on sisällytetty modernimpia ominaisuuksia, kuten muuttujia ja funktioita. Selain ei kuitenkaan lue suoraan näiden kielien tiedostoja, vaan ne yleensä kootaan CSS-tiedostoiksi. On myös saatavilla valmiita tyylikirjastoja, kuten Bootstrap, jotka sisältävät valmiita tyyliluokkia, joita voi kehittäjä voi käyttää omassa sivustossaan. Nämä eivät usein perustu pelkästään CSS:ään, vaan niissä on mukana myös Javascriptiä.

HTML itse ei sisältänyt tapaa muotoilla elementtejä ilmestyessään, ja ulkoasun muotoilu oli aluksi päätetty jättää selaimille. Kehittäjät ja käyttäjät kuitenkin harmittelivat, kun heillä ei ollut tapaa määritellä sivun ulkoasua. Tätä ongelmaa paikkaamaan alkoi Håkon Wium Lie, Cernin työntekijä. CSS esiteltiin kansainvälisessä konferenssissa 1994, ja kaupalliseen käyttöön sen otti Internet Explorer 1996, ja hieman myöhemmin Netscape Navigator. Netscape ei oikeastaan tukenut CSS:ää, vaan muunsi sen Javascriptiksi [10.].

2.3. Javascript

Javascriptillä on pitkä historia, ja nykyään se on käytössä 95.2% kaikista nettisivuista, eivätkä monet sivut edes toimi kunnolla, mikäli käyttäjän selain ei tue javascriptiä. [3.] Piilaaksossa tarjotuista työpaikoista javascript on kolmanneksi haetuin kieli, Pythonin ja Rubyn jälkeen. Näistä Ruby on menettämässä suosiotaan, kun taas javascript on yhä nousussa. [2.] Javascript sekoitetaan usein Javaan, mutta näillä kielillä on yhteistä lähinnä nimi, mikä ei kuitenkaan ole sattumaa. Javascriptin avulla selaimen toiminnasta saadaan paljon dynaamisempaa, kuin jos käytössä olisivat vain staattiset teknologiat. Sivun lataukset vähentyvät, ja dataa saadaan vaikka päivittymään reaaliajassa. Tosin Javascriptin lisääntyvä käyttö on luonut uusia turvallisuusuhkia, sekä osittain myös hi-

dastanut sivuja, kun sivulle mentäessä selain saattaa joutua lataamaan suuret määrät eri kirjastoja.

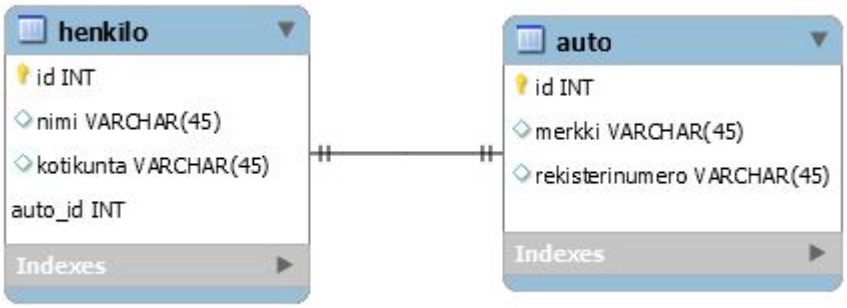
Javascriptin historia alkoi vuonna 1995. Eri selaimet kilpailivat markkina-asemasta, ja tuolloin suosittu ohjelmointikieli myös selaimilla oli Java. Java nähtiin kuitenkin monimutkaisena, suunnattuna ohjelmoijille, kun taas verkkosivuja tekivät ammattilaisten lisäksi myös amatöörit, harrastelijat sekä uusi ammattiryhmä, web-suunnittelijat. Netscape, tuona aikana suosittu selain, halusi uuden kielen, joka olisi helppo oppia sekä kevyt, ja tätä uutta kieltä palkattiin suunnittelemaan Brendan Eich. Eichin mukaan uuden kielen tuli olla ikään kuin pikkuveli Javalle, näyttää samalta, mutta kuitenkin olematta yhtä monimutkainen. Työllä oli tiukka aikataulu, koska se haluttiin mukaan Netscapen seuraavaan julkaisuun, ja Eich saikin työnsä valmiiksi 10 päivässä. Kuten monet muut kielet, Javascriptin perussyntaksi tuli C:stä, mutta se lainasi malleja myös muutamasta muusta kielestä. Javascriptin erikoisuus oli, että sitä ei tarvinnut erikseen koota ja ajaa kuten Javaa.

Javascriptiä ei kuitenkaan pitkään pidetty ”oikeana” ohjelmointikielenä, eikä sillä pitkään myöskään saatu tehtyä mitään kovin merkittävää, vain ärsyttäviä vilkkuvia elementtejä. Javascriptillä ei myöskään ollut pääsyä palvelimelle, joten sen kyky käsitellä ja näyttää tietoa oli rajoitettu. Ensimmäinen käännekohta tuli Microsoftin kehittäessä sähköpostiohjelma Outlook 2000:ta. Päämääränä oli saada sovelluksesta responsiivisempi, ilman että se söisi liikaa resursseja. Tätä tarkoitusta varten kehitettiin hankalasti nimetty komponentti XMLHttpRequest, jonka avulla Javascript pystyi hakemaan tietoa palvelimelta. Aluksi tämä komponentti toimi vain Windowsin koneilla ja Internet Explorerilla, molemmat Microsoftin tuotteita, mutta muut valmistajat alkoivat pian tehdä omia versioitaan komponentista. Kuuluisat käyttötapaukset komponentille olivat Googlen Gmail vuonna 2004 ja Google Maps vuonna 2005, jonka jälkeen myös Microsoft alkoi käyttämään komponenttia omassa selainpohjaisessa sähköpostiohelmassaan, Javascriptiin lisättiin näin asynkroninen elementti, eli käyttäjän ei tarvinnut joka kerta ladata sivua uudestaan, mikäli hän halusi uusimmat tiedot palvelimelta. [1.] [4.]

Nykyään Javascriptin standardisaatiota ylläpitää Ecma International. Standardin nimi on ECMAScript, ja siitä on useita editioita. Uusin edition on 10., nimeltään ECMAScript 2019, ja se julkaistiin kesäkuussa 2019. Alun perin Netscape oli lähettänyt Javascriptin standardisoitavaksi Ecma Internationalille 1996. Sen jälkeen muilla yhtiöillä oli omia versioitaan Ecmascriptistä, kuten Microsoftin JScript ja Adoben ActionScript Virtual Machine 2. Ecmascriptin 5. version yhteydessä 2009 nämä eri versiot päätyivät lopulta yhteen, sen jälkeen kun Oslossa järjestettiin tapaaminen vuonna 2008. MySQL ja tietokannat

Sovelluksia käytetään käsittelemään usein varastoitua tietoa, ja koska tänä päivänä pelkkä staattisten sivujen tuottaminen ei ole kannattavaa, pitää tämä tieto varastoida jonnekin. Tähän tarkoitukseen on kehitetty erilaisia tietokantaratkaisuja, sekä kieliä, joilla kannoissa olevaa tietoa voidaan käsitellä. Nykyään tietokannat voi karkeasti jakaa kahteen luokkaa, relaatiotietokannat ja ei-relaatiotietokannat (NoSQL). MySQL on relaatiokanta, jota käsitellään tarkemmin alempana. NoSQL taas ei ole, ja tieto säilötään siihen vapaammassa muodossa. NoSQL -nimike tuli käyttöön 2000-luvulla, tosin vastaavia toteutuksia oli ollut käytössä jo paljon aikaisemmin. Kuuluisa esimerkki NoSQL toteutuksesta on MongoDB. Koska työssä käytettiin MySQL:ää, ei NoSQL toteutuksiin perehdytä tämän enempää.

Relaatiotietokannoissa tieto on usein säilötty tauluihin, ja tämän tiedon muoto on usein tarkkaan määriteltyä. Taulujen sisällä tieto on kolumneissa ja riveissä, ja jokaisella rivillä on uniikki solu, jolla rivi voidaan tunnistaa taulun sisältä. Tämä on rivin ID tai pääavain (Primary key). Muut taulut saattavat sisältää viiteavaimia (Foreign key) toisen taulun pääavaimeen. Esimerkiksi voi olla kaksi taulua, Henkilö ja Auto. Henkilö taulussa voi olla kolumnit henkilön nimelle, ja kotikunnalle, sekä viittaus Auto-tauluun. Näin voidaan helposti tarkastella henkilöä ja tämän omistaman auton tietoja, ilman että kaikkea dataa tarvitsisi sisällyttää samalle riville, ja näin paisuttaa tarpeettomasti rivejä. Autoon voidaan viitata myös toisesta taulusta pelkän viiteavaimen avulla, ja näin tietoa ei myöskään tarvitse tarpeettomasti monistaa.



3. Esimerkki relaatiotauluista

IBM kehitti SQL:n alun perin 70-luvulla, ja eräs suosituimmista versioista nykypäivänä on MySQL, avoimen lähdekoodin ohjelmisto. Se on käytössä suosituilla alustoilla, kuten Wordpressillä, Drupalilla ja Joomlalla. Myös suositut sivut kuten Facebook, Youtube ja Twitter käyttävät MySQL:ää.

Mysql:n kehitti alun perin ruotsalainen yhtio nimeltä MySQL AB, ja se julkaistiin vuonna 1995. Vuonna 2010 teknologiayhtiö Oracle hankki MySQL:n, ja jatkaa yhä sen kehittämistä. Samana vuonnan, ennen hankintaa, alkuperäinen kehittäjä Michael Widenius teki oman haaransa, MariaDB:n.

MySQL:n heikkous on tiukka määrittely, jota voi olla myöhemmin hankala muuttaa datan määrän kasvaessa, mutta se on yhä suosituin ratkaisu teknologia jättien keskuudessa. NoSQL:n käyttö on kasvamassa, ja se lupaa paremmin skaalautuvia ratkaisuja.

3. Sovelluksen toteutus

3.1. Yleinen kuvaus

Sovelluksella voidaan käyttää kirjojen lainaukseen, ja kirjojen etsimiseen teoksen nimen, tekijän tai aiheen perusteella. Lainaus tehdään skannaamalla kirjan viivakoodi, tai vaihtoehtoisesti syöttämällä kirjan nimi.

Lainatessa kirjaa, käyttäjä skannaa sen, ja kirja tallennetaan käyttäjän nimen alle. Jos kirja on jo lainattu, annetaan virheviesti, tulevaisuudessa voidaan myös ehdottaa palauttamista. Mikäli skannaus ei toimi, annetaan myös käyttäjän etsiä kirjaa nimellä, ja lainata kirja sitä kautta. Kirjoilla on kappalemäärä, eikä eri painoksia tällä hetkellä erotella, vaan tietokannassa on määrä kirjoille.

Kirjan palauttaminen menee samalla tavalla kuin lainaaminen. Mikäli lainaajia on useampia, palautetaan lainaajien nimet, ja käyttäjä valitsee, kuka hän on.

Kirjoja lisätessä tarkistetaan, onko kirjaa olemassa. Mikäli ei, käyttäjä syöttää kirjan nimen, tekijän ja hakusanat ja tallentaa tiedot. Mikäli kirja on jo olemassa, lisätään kirjojen lukumäärää tietokannassa.

3.2. Käytetyt kirjastot ja kirjastojen hallinta

3.2.1. Paketinhallinta

Eri kirjastojen hallintaan käytettiin projektissa ohjelmaa nimeltä Yarn. Se toimii samalla tavalla kuin suosittu node package manager (npm), ja on myös Facebookin kehittämä. Molemmissa on pohjana package.json -tiedosto, jossa kerrotaan mitä kirjastoja sovellus käyttää. Kun ajetaan asennuskomento, haetaan tiedostossa määritellyt kirjastot node_modules kansioon, josta sovellus voi käyttää niitä.

Alun perin npm:n kanssa ei saatu tarkalleen tietoa riippuvuuksien versiosta. Kun yarnilla asennetaan kirjastoja, luodaan myös yarn.lock -tiedosto, johon tallennetaan kirjastojen versiot, ja sovelluksen rakenne. Näin saadaan helpommin replikoitua sovelluksen rakenne eri kehitystympäristöissä. Myös npm käyttää nykyään .lock -tiedostoa

3.2.2. React.js ja käyttöliittymä

React.js on Facebookin kehittämä Javascript kirjasto käyttöliittymän rakentamiseen. Sen avulla voi tehokkaasti muokata sivulla käyttäjälle esitettävää tietoa. Reactin eräitä keskeisimpiä ominaisuuksia ovat komponentit, virtuaalinen DOM, elinkaaren metodit ja JSX.

Komponentit kirjoitettaan useimmiten JSX:llä, tosin myös puhdasta Javascriptiä voidaan käyttää.

```
const name = 'Josh Perez';
const element = <h1 className=’header’>Hello, {name}</h1>;

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

Yllä elementti tallennetaan muuttujaan, joka sitten renderöidään reactin render() -funktiolla. JSX muistuttaa syntaksiltaan HTML:ää, vaikkakin attribuuttien nimeämisessä on pieniä eroja (HTML:ssä ’class’ ja JSX:ssä ’className’). Aaltosulkeiden ({}) sisään voidaan lisätä Javascriptiä, tässä tapauksessa muuttuja ’name’.

Komponentit voivat perustua luokkiin, tai olla funktionaalisia. Luokkapohjaiset olivat alkuperäinen tapa toteuttaa komponentteja, tosin tällä hetkellä molempia tuetaan Re-

actissa, ja niitä voi myös käyttää samassa sovelluksessa. Komponenteilla React-sovelluksesta saadaan modulaarinen, eli voidaan tehdä paljon uudelleenkäytettäviä osia.

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

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

Yllä esimerkki luokkapohjaisesta ja funktionaalisesta komponentista. Yllä olevat esimerkit tuottavat juuri saman HTML:n, tosin esimerkiksi selaimet jotka eivät tue ES6 syntaksia eivät myöskään suoraan tue luokkapohjaisia komponentteja.

Totetuksesta riippumatta komponentit tulee aina sisällyttää ylemmän elementin sisään, esimerkiksi divin. Komponentteja voidaan käyttää toisten komponenttien sisällä, samaan tyyliin kuin normaaleja HTML elementtejä. Niille voidaan myös antaa attribuutteeja.

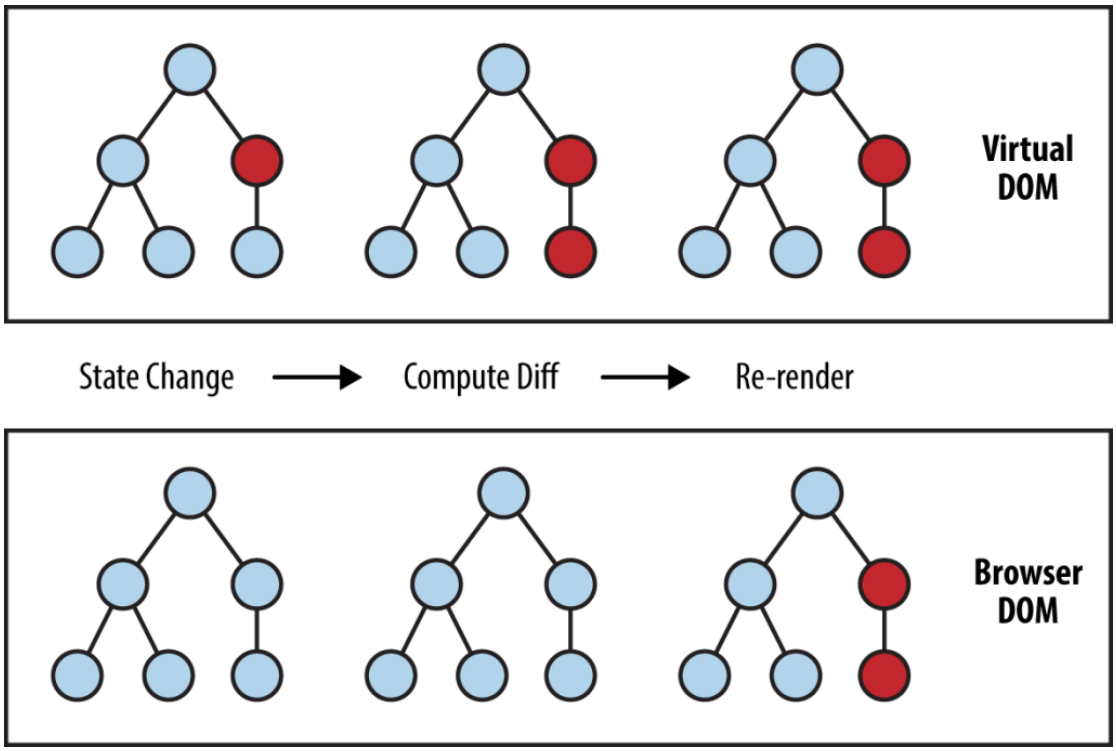
```
import Welcome from ‘./Welcome’;

function Hello (props) {
  return (
   <div>
    <p>This is the Hello component!</p>
    <Welcome name=’Teppo’ />
   </div>)
}
```

Tässä siis haetaan Welcome -komponentti, ja sisällytetään se Hello komponentin sisään. Tuloksena oleva HTML näyttäisi tältä:

```
<div>
 <p>This is the Hello component!</p>
 <h1>Hello, Teppo</h1>
</div>
```

Virtuaalinen DOM mahdollistaa, että dokumentti renderöidään mahdollisimman tehokkaasti. DOM (Document Object Model) kuvastaa dokumentin käyttöliittymää, ja aina kun datassa tapahtuu muutoksia, DOM pitää päivittää. Jos muutoksia tapahtuu paljon, ja dokumentti on suuri, voivat päivitykset viedä paljon resursseja.



4. Virtuaalinen DOM. [11.]

Virtuaalisessa DOMissa dokumentin rakenne on muistissa. Kun tieto päivittyy, päivitetään ensin virtuaalinen DOM, ja tämän jälkeen muutoksia verrataan oikeaan DOMiin. Näin voidaan päivittää vain osa DOMista, mikä säästää resursseja. Kuvan 4. esimerkin mukaisesti yksi punainen noodi on päivittynyt. Virtuaalinen DOM katsoo eron, ja päivittää uudestaan sen ja sen lapset. Lopuksi päivittynyt osio dokumentista liitetään oikeaan DOMiin. Reactissa jokaisella komponentilla on tila (state), jonka muuttumista React monitoroi. Kun komponentin tila muuttuu, renderöidään virtuaalinen DOM uudestaan, ja tästä muutokset valuvat oikeaan DOMiin. [11.]

Reactissa siis tärkeä konsepti on tilan ja elinkaaren käsittely. Reactissa on tarjolla erinäisiä funktioita joiden avulla kehittäjä voi vaikuttaa sovellukseen toimintaan, esimerkiksi silloin kun dokumentti alustetaan, tai kun komponentin tila päivittyy. Luokkapohjaisissa komponenteissa tilan päivittämiseen on tarjolla setState() -funktio, kun taas uudemmissa funktio-pohjaisissa komponenteissa on tarjolla erinäisiä ”koukkuja” (hooks).

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

function Example() {
```

```
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Esimerkissä on funktionaalinen komponentti, jolla on tilamuuttuja count. Sille taas on määritelty koukku setCount. Kun komponentissa olevaa nappia klikataan, kutsutaan setCount -funktiota, jossa count muuttujan tilaksi määritellään sen nykyinen arvo plus 1. Kun koukkua on kutsuttu, React tietää että komponentin tila on muuttunut, ja että tämä komponentti pitää renderöidä uudestaan.

```
import React, { useState, useEffect } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Toinen esimerkki koukuista on useEffect(). Sitä kutsutaan, kun komponentin tila päivittyy. Esimerkissä käytetään muuttujalle määriteltyä koukkua muuttamaan count muuttujan tilaa. Tämä saa siis komponentin renderöitymään uudelleen, ja kutsumaan useEffect hookkia. Tässä kehittäjä voi määritellä, mitä pitäisi tapahtua, kun komponentin tila päivittyy, esimerkin tapauksessa siis sivun titteliä päivitetään.

3.2.3. Rajapinta

APIa eli ohjelmointirajapinta tarjoaa pääsyn ohjelman osiin, joko ohjelman toiseen osaan tai täysin ulkopuoliseen sovellukseen. Rajapinta voi olla datarajapinta, joka palauttaa vain tietoa, eli sitä voi käyttää vain lukemiseen. Se voi olla myös toiminnallinen rajapinta, jota käyttämällä saa myös muokattua järjestelmässä olevaa tietoa. [12.] Monet

sivut tarjoavat rajapintoja kehittäjille, joko maksua vastaan tai ilmaiseksi, esimerkkinä Googlen erinäiset palvelut.

Sovelluksessa rajapinta on toteutettu käyttäen Express.js -kirjastoa. Expressin avulla määritellään erinäisiä reittejä, joihin käyttöliittymä voi tehdä http-kutsuja

```
const express = require('express')
const app = express()
const port = 3000

app.get('/hello', (req, res) => res.send('Hello World!'))

app.listen(port, () => console.log(`Example app listening on port ${port}!`))
```

Esimerkki Express sovelluksesta. Kun osoitteeseen /hello tehdään kutsu, palauttaa Express stringin ’Hello world’. Näin voidaan eristää kaikenlainen logiikka tiettyyn osaan sovelluksesta, ja käsitellä mahdollisimman vähän tietoa käyttöliittymässä. Esimerkissä voitaisiin myös tehdä kysely tietokantaan, joko muuttaa tai palauttaa tietoa sieltä.

3.2.4. Node.js ja palvelin

Javascript sai alkunsa käyttöliittymän puolella, mutta nykyään sitä voidaan käyttää myös selaimella. Perinteisesti suosittu kieli serverin puolella on ollut PHP, eikä Node.js olekaan sitä suoraan parempi. Node.js:n käyttäminen Javascript-projekteissa kuitenkin mahdollistaa saman kielen käytön eri alustoilla, käyttöliittymässä ja palvelimella. [18.] Node.js sopii hyvin sovelluksiin, jossa tietoa päivitetään reaaliajassa, ja sen on asynkroninen (PHP on oletuksena synkroninen, tosin sitäkin voidaan käyttää asynkronisesti).

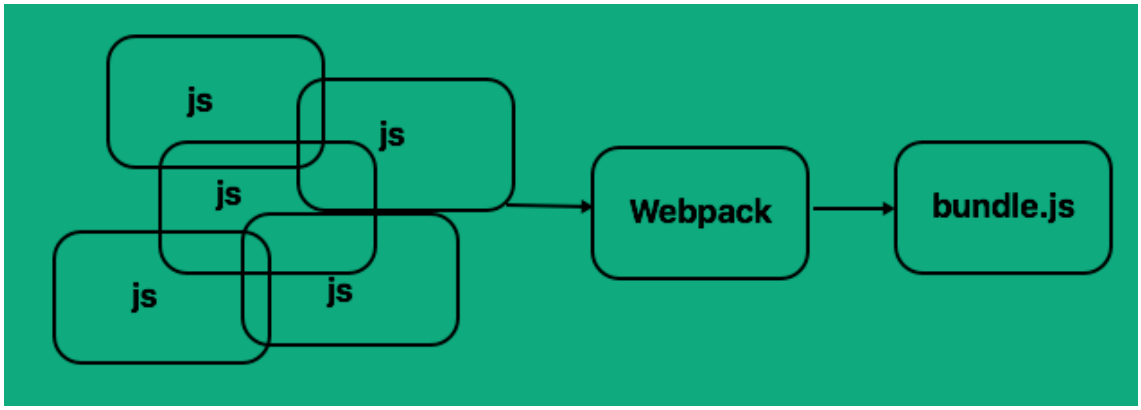
3.2.5. Webpack

Webpackillä voidaan hallita lukuisia eri riippuvuuksia, joita sovelluksella voi olla. Yhtään suuremmissa sovelluksissa riippuvuuksien hallinta voi olla hankalaa, ja jos ne ladataan väärässä järjestyksessä, sovellus ei toimi. Globaalien muuttujien käyttäminen voi myös tuottaa hankalasti löydettäviä virheitä.

Webpack luo näistä erinäisistä riippuvuuksista moduuleja, jotka on helppo lisätä dynaamisesti eri tiedostoihin, eikä latausjärjestyksestä tarvitse murehtia [14.]. Sovelluksessa moduuleja kutsutaan esimerkiksi seuraavilla tavoilla:

```
const module = require('module'); // tai const module = require('../module');
import module from 'module'; // tai import module from '../module';
```

Tässä projektissa, ylempää metodia käytetään palvelimella Node.js:n kanssa, ja alempaa metodia käyttöliittymän puolella Reactin kanssa. Node-moduleeja yleensä kutsutaan pelkästään nimellä, ja sovelluksen sisäisille tiedostoille annetaan polku tiedostoon. Import on uudemman ES6 Javascript-standardin mukainen. Tosin myös import yleensä muutetaan require muotoon, jotta se on yhteensopiva vanhempien selainten kanssa (Babelista lisää myöhemmin).

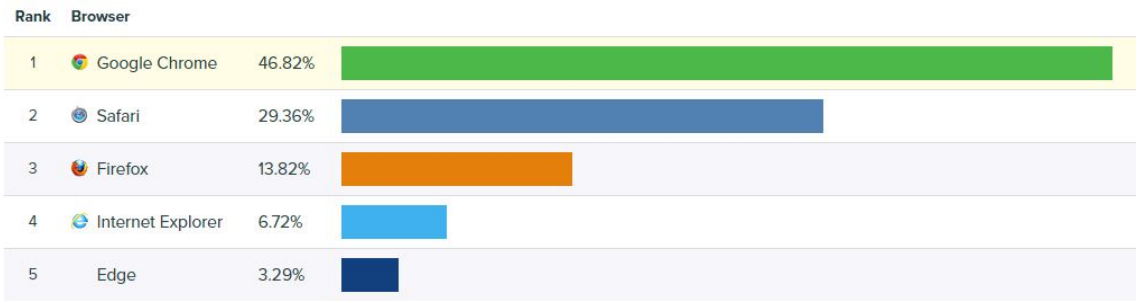


5. Webpack ja moduulien niputtaminen [14.]

Webpackin toinen ominaisuus on moodulien niputtaminen (bundling). Webpack koostaa tarvittavat moduulit yhdeksi tiedostoksi, ja näin riippuvuudeet voidaan ladata kerralla, mikä parantaa sovelluksen suorituskykyä.

3.2.6. Babel

Javascriptiin julkaistaan joka vuosi uusia ominaisuuksia, jotka helpottavat kehittäjien elämää ja parantavat syntaksia. Kuitenkin yhä on käytössä vanhoja selaimia, jotka eivät tue näitä uusia syntakseja. Koska koodin kirjoittaminen kahdesti olisi turhaa, on parempi muuntaa se sellaiseksi, jota kaikki selaimet ymmärtävät. Kuten alla olevasta kuvasta näkee, esimerkiksi Internet Explorer on yhä käytössä. Kyseisen version ominaisuudet on kuitenkin jäädytetty, eikä se tue Javascriptin uusimpia ominaisuuksia.



6. Selainten käyttö. https://www.stetic.com/market-share/browser/

Babel muuntaa Javascriptin versioon, jota tuetaan vanhemmissa selaimissa, jolloin kehittäjän ei tarvitse murehtia yhteensopivuusongelmista. Babel esimerkiksi muuntaa syntaksin, muuntaa uudet funktiot vanhemman version mukaiseksi koodiksi. [15.]

```
// Babel Input: ES2015 arrow function
[1, 2, 3].map((n) => n + 1);

// Babel Output: ES5 equivalent
[1, 2, 3].map(function(n) {
  return n + 1;
});
```

Esimerkissä käytetään alun perin ES5 yhteensopivaa nuolifunktiota, ja Babel muuntaa sen sellaiseksi, jota vanhemmat selaimet kuten Internet Explorer voi ymmärtää.

3.2.7. ESLint

ESLint on kirjasto, jonka avulla voidaan tunnistaa koodin rakenne, ja huomauttaa kehittäjälle, jos hän poikkeaa yleisistä käytännöistä. Tällä voidaan lisätä koodin luettavuutta, mikä auttaa erityisesti projekteissa, joissa työskentelee useampi kehittäjä (tai ymmärtämään koodia, jonka itse on kirjoittanut aiemmin), ja auttamaan kehittäjää virheiden välttämisessä [16.]. ESLinttiä voi itse konfiguroida haluamallaan tavalla, sekä myös käyttää suosittuja konfiguraatioita, kuten Airbnb:n ESLinttiä, jota myös tässä projektissa käytetään.

ESLintin voi integroida projektiin esimerkiksi niin, että syntaksi automaattisesti korjataan, tai niin että koodia ei saa sisällytettyä versionhallintaan ilman että se läpäisee tarkastuksia.

3.2.8. Create React App

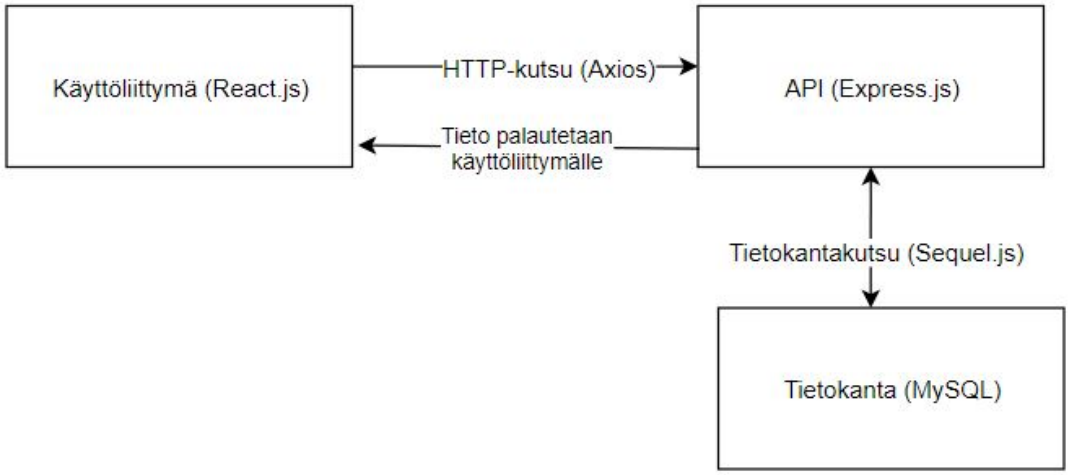
Create React App on yksi riippuvuus, joka sitoo yhteen monta muuta kirjastoa, ja varmistaa että ne ovat yhteensopivia. Ajamalla komento

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

saadaan luotua valmis React-projekti, ilman että tarvitsee ladata lukuisia paketteja erikseen. Tähän sisältyy mm. edellä mainitut Webpack, Babel ja ESLint, valmiiksi konfiguroituna React-projektille. Tämä saattaa osittain selittää Reactin suosiota; jo projektin aloitus on helppoa.

Projektin voi myös ejektoida, jolloin kehittäjä voi täysin muokata konfiguraation haluamakseen, mutta tämä vaatii syvällisempää osaamista, eikä normaalisti ole tarpeen.

3.3. Rakenne



7. Sovelluksen rakenne

Sovelluksen käyttöliittymänä on React, ja palvelimen puolella käytössä on Node.js, ja tietokantana MySQL. Rajapinta on toteutettu Express.js -kirjaston avulla. Kuvassa yllä esitetään sovelluksen eri osien suhteet, ja miten tieto niiden välillä kulkee.

Reactia käytetään lähinnä tiedon esittämiseen ja käyttäjän syöttämän tiedon vastaanottamiseen. Käyttöliittymästä tieto kulkee rajapinnalle http kutsujen kautta, ja tähän tarkoitukseen käytetään Axios -nimistä kirjastoa. Axiosilla voi tehdä POST ja GET -pyyntöjä, eli voi muuttaa tietoja tai lukea tietoja.

Oletuksena Reactilla tehdään yhden sivun applikaatioita, mutta tässä sovelluksessa käytetään perinteistä sivustorakennetta react-router-dom kirjaston avulla.

```
<BrowserRouter>
  <Switch>
    <Route exact path="/" component={Frontpage} />
    <Route exact path="/search" component={SearchView} />
    <Route exact path="/loan" component={Loan} />
    <Route exact path="/book/:code" component={Book} />
    <Route exact path="/return" component={ReturnBookView} />
    <Route exact path="/add" component={AddBooksView} />
    <Route component={Notfound} />
  </Switch>
</BrowserRouter>
```

Esimerkissä siis kaikki muut komponentit on kääritty BrowserRouter -komponentin sisään on kääritty Route -komponentit. Riippuen siitä, mikä osoite selaimessa on, tarjoillaan eri komponentti. Kun ollaan sivuston juuressa, eli ”/”, näytetään käyttäjälle Frontpage -komponentin sisältö. Kun taas ollaan ”/search” sivulla, näytetään SearchView -komponentti. Reitittimen avulla voidaan myös antaa komponentille parametreja, kuten näkyy ”/book/:code” -reitin kohdalla. Code on siis tietyn kirjan id, ja komponentissa näytetään id:n omaavan kirjan tiedot. Näin voidaan dynaamisesti käyttää samaan komponenttia, ja määritellä vain yksi reitti eri kirjoja varten.

Viivakoodien skannaamiseen käytetään Quagga.js nimistä kirjastoa. Se käyttää laitteen, tässä projektissa useimmiten kännykän, kameraa, ja tunnistaa viivakoodin. Kirjasto palauttaa viivakoodin numeromuodossa, sekä koodiin liittyviä tietoja. Kirjaston voi konfiguroida sen mukaan, minkälaisia viivakoodeja oletetaan skannattavan, esimerkiksi käytettävä standardi.

```
async function onDetected(result) {
  const { code } = result.codeResult;

  if (codeResults.length < 8) {
    codeResults.push(code);
    setCodeResults(codeResults);
    setScanningProgress(codeResults.length * 10);
  }
  else {
    const itemCode = _.head(_(codeResults)
      .countBy()
      .entries()
      .maxBy('[1]'));

    Quagga.stop();

    setItemCode(itemCode);
    setCodeResults(codeResults);

    await props.search({ code: itemCode });
  }
}
```

Yllä oleva esimerkki on koodi, kun kirjastolla tunnistetaan viivakoodi. Koska kamera ei ole paras tapa lukea viivakoodia, tehdään skannaus useamman kerran, ja sitten katsotaan mikä koodeista toistui useamman kerran.

3.3.1. Modulaarisuus

Sovelluksen osat, erityisesti käyttöliittymäpuolella Reactin kanssa, on tehty uudelleenkäytettäviksi. Hyvä ajatus on tehdä jokaisesta osasta koodia, joka toistetaan, oma komponenttinsa, mikäli mahdollista. Esimerkiksi pelkät otsikot voivat olla komponentteja, ja näin on helppo esimerkiksi vaihtaa otsikkojen tyyliä myöhemmin. Komponentit on tietenkin hyvä nimetä kuvaavasti.

Komponenttien käyttö lisää koodin luettavuutta, ja vähentää koodirivien määrää. Kun komponentti on sijoitettu toisen sisälle, ja nimetty osuvasti, voi toinen kehittäjä suoraan nähdä, mitä koodin on tarkoitus tehdä. Isommissa projekteissa myös eri kehittäjät voivat työskennellä eri komponenttien parissa, ja jakaa työn tehokkaammin.

```
  return (
    <div>
      <LoadingWithMessage loading={loading} errorMessage={errorMessage} />

      <BookForm
          formSubmit={addNewBookSubmit}
          submitText='Add a new item' disabled={!code}
          required={['name', 'keywords', 'author']}
       />

      <button type="button" onClick={() => setShowScanner(!showScanner)}>Tog-
gle scanner</button>

      {showScanner && <Scanner search={getCode} />}
    </div>
  );
```

Yllä olevassa koodiesimerkissä komponentin sisällä käytetään kolmea muuta komponenttia. LoadingWithmessage komponenttia käytetään ladattaessa sisältöä rajapinnasta. Kunnes sisältö on ladattu, käyttäjälle näytetään spinneri, tai virhetilanteessa komponentti voi myös näyttää virheviestin. Toinen komponentti on BookForm, jonka avulla voidaan syöttää tietoa kirjoista. Kolmas komponentti on skanneri, jolla voidaan skannata kirjojen viivakoodit. Esimerkiksi skanneri tekee aina saman toiminnon, skannaa viivakoodin, ja palauttaa numerosarjan, eikä välitä, mitä muut ohjelman osat tekevät.

3.3.2. Kehitysympäristö

Docker on joukko Docker ry:n kehittämiä tuotteita, joilla voidaan virtualisoida käyttöjärjestelmiä. Dockerilla nämä järjestelmät ovat konteissa (container), eristettynä muista järjestelmistä. Sovelluksessa esimerkiksi MySQL ja PHP ovat eri konteissa, mutta pystyvät kuitenkin keskustelemaan keskenään. Docker ry perustettiin vuonna 2010, ja ohjelmisto esiteltiin konferenssissa vuonna 2013.

Dockeria voidaan käyttää palvelinympäristössä, mutta se on myös kätevä kehityskäytössä. Dockeria käyttämällä voidaan varmistaa, että eri käyttäjien kehitysympäristöt vastaavat toisiaan. Kehitysympäristön konfiguraatiossa voidaan varmistaa, että esimerkiksi PHP versio on aina sama, ilman että uuden kehittäjän tarvitsee erikseen varmistaa sitä mistään.

Docker koostuu kolmesta komponentista: ohjelmistosta, objekteista ja rekistereistä. Dockerd on ohjelmisto, jota voidaan hallita komentorivin kautta Docker työkalulla. Objektit koostuvat kuvista (images), konteista (containers) ja palveluista (services). Kontti on siis eristetty osa ohjelmaa, joka saattaa esimerkiksi pyörittää MySQL:ää. Kuva on

luettava pohja, jonka perusteella voidaan rakentaa kontteja. Palvelulla voidaan eri ohjelmistot voivat keskustella keskenään. Rekisterissä taas säilytetään tiedot Dockerin kuvista.

Projektissa käytettiin dockeria kehitysympäristön luomiseen. Kontit määriteltiin käyttöliittymälle, rajapinnalle ja tietokannalle sekä tietokannan käyttöliittymälle. Nämä kontit ovat erillisiä, mutta voivat siis keskustella keskenään, koska on myös määritelty, että ne ovat samassa verkossa.

```
services:
  client:
    build: ./containers/client
    restart: always
    ports:
      - "3000:3000"
    volumes:
      - ./client:/client
    links:
      - api
    networks:
      - webappnetwork
```

Esimerkissä on määritys käyttöliittymän kontille. Ylemmässä pätkässä määritellään missä koodi on paikallisesti, ja sijainti kontissa, sekä mitkä portit kontille varataan, sekä että käyttöliittymä on yhteydessä rajapinna konttiin. Tämä on tiedostossa docker-compose.yml

```
# Use a lighter version of Node as a parentimage
FROM node:8
WORKDIR /client
# COPY package*.json /client/
RUN yarn
# COPY . /client/
EXPOSE 3000
CMD ["sh", "-c", "yarn && yarn start"]
```

Tässä koodissa määritellään, että käyttöliittymän kontin noden versio on 8, ja että kun kontti käynnistetään, ajetaan komennot yarn ja yarn start, eli asennetaan määritellyt moduulit ja käynnistetään sovellus.

Dockerin kanssa voidaan rakentaa kontit itse, tai käyttää valmista kuvaa. Alla on esimerkki tietokannan määrittelystä.

```
  mysql:
    image: mysql:5.7
    volumes:
      - ./mycustom.cnf:/etc/mysql/conf.d/custom.cnf
    ports:
      - "27017:27017"
    environment:
      - MYSQL_DATABASE=library
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_ALLOW_EMPTY_PASSWORD=yes
    restart: always
    networks:
      - webappnetwork
    command: ['mysqld', '--character-set-server=utf8mb4', '--collation-
server=utf8mb4_unicode_ci']
```

Esimerkissä on tietokannan määrittely. Tässä käytetään valmista kuvaa tietokannasta, sen sijaan että se olisi rakennettu paikallisesti.

Docker määrittelyjen avulla toinen kehittäjä voi nopeasti pystyttää samanlaisen kehitysympäristön kuin alkuperäinen tekijä, ja se on myös eristyksissä hänen omasta laitteestaan.

4. Jatkokehitysideoita

4.1. Kirjautuminen ja käyttäjänhallinta

Tällä hetkellä sovellukseen ei kirjauduta, vaan lainaajan on tarkoitus jättää nimi ja sähköposti, kun hän lainaa kirjan. Kirjautumisjärjestelmän avulla nämä tiedot voitaisiin saada suoraan tunnuksesta.

Kirjautumisjärjestelmän toteuttamisessa voitaisiin hyödyntää LDAP -järjestelmää, tai sitten Googlen kirjautumista. Googlen kirjautumisjärjestelmälle olisi paljon tukea, ja valmiita kirjastoja. Suunnitelmana on sallia sovelluksen käyttö vain sisäverkosta, joten toistaiseksi kirjautumisjärjestelmän toteuttaminen ei ole suuri prioriteetti.

4.2. Notifikaatiot

Toinen mahdollinen lisäominaisuus olisi huomautusten lähettäminen käyttäjille. Esimerkiksi käyttäjä voisi laittaa kirjan vahtiin, ja kun se palautetaan, saada siitä ilmoituksen sähköpostiin. Mahdollisesti myös käyttäjä voisi saada muistutuksen palauttaa kirja, jos se on ollut hänellä lainassa pitkään. Sakkojen antaminen ei tosin ole suunnitelmissa.

5. Johtopäätökset ja yhteenveto

Tarkoituksena oli kehittää sovellus, jolla työntekijät voisivat helposti lainata Exoven omistamia kirjoja, ja myös luetteloida kirjat.

React valittiin käyttöliittymän teknologiaksi, koska siitä oli kokemusta, sekä myös halua käyttää sitä uudessa projektissa. Se on myös suosittu teknologia, ja mikäli jonkun muun pitäisi mahdollisesti tulevaisuudessa jatkaa kehitystä, osaavia tekijöitä löytyy helpommin. Siihen löytyy myös paljon dokumentaatiota, sekä moduuleja. Koska projekti oli aloitettu aiemmin ja keskeytetty, oli Reactiin myös projektin kesken julkaistu uusia ominaisuuksia joita päätettiin käyttää ja kirjoittaa uudelleen vanhaa koodia (käytettiin funktionaalisia elementtejä luokkaelementtien sijaan).

Ennen kehityksen alkua piti selvittää, voisiko kameraa käyttää skannaukseen, ja kuinka kameran käyttöä tuetaan eri käyttöjärjestelmillä. Esimerkiksi iOS ei anna käyttää kameraa yhtä vapaasti kuin Android järjestelmät, ja skannerin tulisi olla käytettävissä mobiililla. Sovellus myös päätettiin tehdä web-applikaationa, että se saataisiin helposti käyttöön, joskin React-native oli myöskin harkinnassa.

Skannaus päätettiin toteuttaa viivakoodien skannaamisella, ja siihen löytyikin kirjasto, Quagga.js. QR-koodien käyttämistä harkittiin myös, ja ne ovatkin suositumpia, ja niiden skannaaminen olisi todennäköisesti myös luotettavampaa, mutta päädyttiin kuitenkin viivakoodeihin, koska ne ovat jo valmiina kaikissa kirjoissa, mikä helpottaa mahdollista käyttöönottoa tulevaisuudessa.

Kun skannausta oli ensin testattu ja toimittu sopivaksi, tuli seuraavaksi valita teknologiat. Palvelimella päädyttiin käyttämään Node.js:ää, koska kehityksessä haluttiin käyttää Javascriptiä, ja Node.js sopi muutenkin sovelluksen tarpeisiin. REST API:n kehit-

tämiseen valittiin Express.js, koska se on suosittu ja kevyt, joten sopii hyvin tällaiseen pieneen projektiin.

Myös kehitysympäristö haluttiin pystyttää, ja siihen valittiin Docker. Ympäristön pystyttäminen veikin yllättävästi aikaa, lähinnä siihen että käyttöliittymän ja palvelimen sai keskustelemaan keskenään, ilman kaikkien turvallisuuskysymysten huomiotta jättämistä.

Tämä projekti eroaa aikaisemmista projekteista siinä, että se on vain sisäiseen käyttöön, mikä helpottaa ja yksinkertaistaa monia asioita. Esimerkiksi yhteensopivuutta vanhempien selainten kanssa ei tarvitse testata, joskin suurempia ongelmia ei pitäisi olla, koska projektissa on käytössä Babel. Turvallisuus ei ole yhtä tärkeää, koska sen pitäisi toimia vain sisäverkosta, joskin peruskysymyksiä ei jätetä huomiotta. Esimerkiksi käyttäjälle näytetään vain tarvitut tiedot, eli ei esimerkiksi suoraan anneta tietokantakyselyn tulosta, vaan poimitaan vain olennaiset tiedot.

Lähteet

1. Charles Severance. 2012. Javascript: Designing a Language in 10 Days. Verkkodokumentti. <https://www.computer.org/csdl/magazine/co/2012/02/mco2012020007/13rRUy08MzA> Luettu 6.9.2019.

2. Alison DeNisco Rayome. 2018. Forget the most popular programming languages, here's what developers actually use. Verkkodokumentti . <https://www.techrepublic.com/article/forget-the-most-popular-programming-languages-heres-what-developers-actually-use/> Luettu 6.9.2019.

3. W3Techs.com. 2019. Usage statistics of JavaScript as client-side programming language on websites. Verkkodokumentti <https://w3techs.com/technologies/details/cp-javascript/all/all> Luettu 6.9.2019.

4. Matthew MacDonald. 2019. How Javascript Grew Up and Became a Real Language. Verkkodokumentti < https://medium.com/young-coder/how-javascript-grew-up-and-became-a-real-language-17a0b948b77f> Luettu 6.9.2019.

5. Ensimmäinen sivu joka käytti HTML:ää.. Verkkodokumentti < http://info.cern.ch/hypertext/WWW/TheProject.html> Luettu 6.9.2019.

6. Tim Berners-Lee. 1989. Information Management: A Proposal. Verkkodokumentti < https://medium.com/young-coder/how-javascript-grew-up-and-became-a-real-language-17a0b948b77f> Luettu 6.9.2019.

7. Addison Wesley Longman. 1998. A history of HTML. Verkkodokumentti < https://www.w3.org/People/Raggett/book4/ch02.html> Luettu 6.9.2019.

8. W3C mission statement. Verkkodokumentti < https://www.w3.org/Consortium/mission.html>

9. w3.org. 2014. HTML5 Differences from HTML4. Verkkodokumentti <https://www.w3.org/TR/2014/NOTE-html5-diff-20141209/>

10. Bert Bos. 2016. A brief history of CSS until 2016. Verkkodokumentti < https://www.w3.org/Style/CSS20/history.html> Luettu 6.9.2019

11. Adhithi Ravichandran. 2018. React Virtual DOM Explained in Simple English. Verkkodokumentti <https://programmingwithmosh.com/react/react-virtual-dom-explained/> Luettu 13.9.2019

12. Avoimen rajapinna määritelmä.2019. Verkkodokumentti < http://avoinrajapinta.fi/> Luettu 13.9.2019

13. Joe Barr. 2005. BitKeeper And Linux: The End Of The Road?. Verkkodokumentti <https://www.linux.com/news/bitkeeper-and-linux-end-road/> Luettu 20.9.2019

14. Sean Landsman. 2018. Webpack Tutorial: Understanding How it Works. Verkkodokumentti <https://medium.com/ag-grid/webpack-tutorial-understanding-how-it-works-f73dfa164f01> Luettu 28.9.2019

15. Babel dokumentaatio. 2019. What is Babel. Verkkodokumentti < https://babeljs.io/docs/en/index.html> Luettu 4.10.2019

16. ESLint dokumentaatio. 2019. Getting Started with ESLint. Verkkodokumentti < https://eslint.org/docs/user-guide/getting-started> Luettu 4.10.2019

17. Create React App dokumentaatio. 2019. Getting started. Verkkodokumentti < https://create-react-app.dev/docs/getting-started> Luettu 4.10.2019

18. Vishal Rambhiya. 2018. What is the difference between PHP and Node.js?. Verkkodokumentti < https://www.websoptimization.com/blog/what-is-the-difference-between-php-and-node-js/ > Luettu 4.10.2019